

# 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 L $\TeX$  is called pL $\TeX$  when running on p $\TeX$ / $\varepsilon$ -p $\TeX$ , and called upL $\TeX$  when running on up $\TeX$ / $\varepsilon$ -up $\TeX$ .

## Purpose of this document

This document is written for developers of  $\TeX$ /L $\TeX$ , who aim to support p $\TeX$ /pL $\TeX$  and its variants up $\TeX$ /upL $\TeX$ . 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@texjp.org](mailto:issue@texjp.org)

# Contents

<b>I</b>	<b>Brief introduction</b>	<b>4</b>
1	p $\TeX$ and its variants	4
2	Eminent characteristics of p $\TeX$ family	4
3	Compatibility with Western $\TeX$	5
4	$\LaTeX$ on p $\TeX$ /up $\TeX$ — p $\LaTeX$ /up $\LaTeX$	5
<b>II</b>	<b>Details</b>	<b>6</b>
<b>5</b>	<b>Output format — DVI</b>	<b>6</b>
5.1	Extensions of DVI format in p $\TeX$ family . . . . .	7
5.2	DVI drivers with Japanese support . . . . .	8
5.2.1	Using <i>dvipdfmx</i> . . . . .	8
5.2.2	Using <i>dvips</i> . . . . .	8
<b>6</b>	<b>Programming on p<math>\TeX</math> family</b>	<b>8</b>
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 $\TeX$ additions (available in p $\TeX$ , up $\TeX$ , $\varepsilon$ -p $\TeX$ , $\varepsilon$ -up $\TeX$ ) . . . . .	9
6.3.2	p $\TeX$ additions (available in p $\TeX$ , up $\TeX$ , $\varepsilon$ -p $\TeX$ , $\varepsilon$ -up $\TeX$ ) . . . . .	10
6.3.3	up $\TeX$ additions (available in up $\TeX$ , $\varepsilon$ -up $\TeX$ ) . . . . .	11
6.3.4	$\varepsilon$ -p $\TeX$ additions (available in $\varepsilon$ -p $\TeX$ , $\varepsilon$ -up $\TeX$ ) . . . . .	12
6.3.5	$\varepsilon$ -up $\TeX$ additions (available in $\varepsilon$ -up $\TeX$ ) . . . . .	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 p $\TeX$ . . . . .	14
6.6.2	Detecting up $\TeX$ . . . . .	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 $\TeX$ in DVI mode . . . . .	18

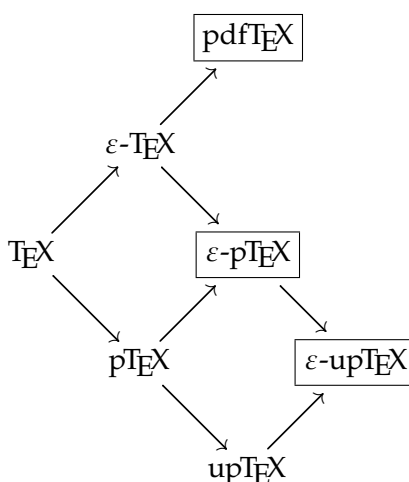
6.8	Recommendation for file encoding . . . . .	18
6.9	Input handling . . . . .	18
6.10	Japanese tokens . . . . .	18
<b>7</b>	<b>Basic introduction to Japanese typesetting</b>	<b>18</b>
7.1	Automatic insertion of glue and penalties . . . . .	19
7.2	Japanese fonts . . . . .	19
<b>8</b>	<b>Other strange beasts</b>	<b>19</b>
8.1	Internal kanji encodings . . . . .	19

## Part I

# Brief introduction

## 1 p $\TeX$ and its variants

The figure below shows the relationship between engines.



p $\TeX$  is an old Japanese-specific extension of  $\TeX$ 82, which aims to support proper typesetting 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.

$\epsilon$ -p $\TeX$  and  $\epsilon$ -up $\TeX$  are  $\epsilon$ - $\TeX$  extensions of p $\TeX$  and up $\TeX$  respectively. In the current release, some extensions derived from pdf $\TeX$  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 `\catcode` (`\kcatcode`) 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 T<sub>E</sub>X

pT<sub>E</sub>X/upT<sub>E</sub>X are almost upward compatible with Knuthian T<sub>E</sub>X, 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.

$\varepsilon$ -pT<sub>E</sub>X/ $\varepsilon$ -upT<sub>E</sub>X are almost upward compatible with  $\varepsilon$ -T<sub>E</sub>X, however, input handling is similar to pT<sub>E</sub>X/upT<sub>E</sub>X. It does not pass the e-TRIP test. That said, please note that “raw  $\varepsilon$ -T<sub>E</sub>X” is unavailable anymore in T<sub>E</sub>X Live and derived distributions; they provide a command `etex` only as “DVI mode of pdfT<sub>E</sub>X.” You should note that  $\varepsilon$ -pT<sub>E</sub>X/ $\varepsilon$ -upT<sub>E</sub>X are *not* upward compatible with DVI mode of pdfT<sub>E</sub>X, which will be discussed later in Section 6.7.

There is no advantage to choose pT<sub>E</sub>X/upT<sub>E</sub>X over  $\varepsilon$ -pT<sub>E</sub>X/ $\varepsilon$ -upT<sub>E</sub>X, so we focus mainly on  $\varepsilon$ -pT<sub>E</sub>X/ $\varepsilon$ -upT<sub>E</sub>X.

### 4 L<sup>A</sup>T<sub>E</sub>X on pT<sub>E</sub>X/upT<sub>E</sub>X — pL<sup>A</sup>T<sub>E</sub>X/upL<sup>A</sup>T<sub>E</sub>X

Formats based on L<sup>A</sup>T<sub>E</sub>X is called pL<sup>A</sup>T<sub>E</sub>X when running on pT<sub>E</sub>X/ $\varepsilon$ -pT<sub>E</sub>X, and called upL<sup>A</sup>T<sub>E</sub>X when running on upT<sub>E</sub>X/ $\varepsilon$ -upT<sub>E</sub>X. In recent versions (around 2011) of T<sub>E</sub>X Live and its derivatives, the default engines of pL<sup>A</sup>T<sub>E</sub>X and upL<sup>A</sup>T<sub>E</sub>X are  $\varepsilon$ -pT<sub>E</sub>X and  $\varepsilon$ -upT<sub>E</sub>X. That is, the command `platex` starts  $\varepsilon$ -pT<sub>E</sub>X (not pT<sub>E</sub>X) with preloaded format `platex.fmt`.

In the kernel level (`platex.ltx` and `uplatex.ltx`), pL<sup>A</sup>T<sub>E</sub>X and upL<sup>A</sup>T<sub>E</sub>X 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, pL<sup>A</sup>T<sub>E</sub>X/upL<sup>A</sup>T<sub>E</sub>X are almost upward compatible with original L<sup>A</sup>T<sub>E</sub>X, except for the followings:

- Order of float objects; in pL<sup>A</sup>T<sub>E</sub>X/upL<sup>A</sup>T<sub>E</sub>X,  $\langle$ *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 absence of pdfT<sub>E</sub>X features.

## 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 L $\TeX$  in DVI mode.

- The DVI format is, as its name suggests, inherently driver-independent. However, some L $\TeX$  packages (`graphicx`, `color`, `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 `dvips`. When you choose to process the resulting DVI file with a driver other than `dvips` (e.g. `dvipdfmx`) after running L $\TeX$ , you need to pass a proper driver option (e.g. `[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.<sup>1</sup>

- 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<sup>2</sup>:

```
LaTeX Warning: Unused global option(s):  
[dvipdfmx].
```

## 5.1 Extensions of DVI format in pTeX family

The DVI format output by pTeX 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 [1] but never used in TeX82, can come out.

- `set2` (129): Used to typeset a Japanese character with 2-byte code (both pTeX and upTeX).
- `set3` (130): Used to typeset a Japanese character with 3-byte code (upTeX only).

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

In addition, pTeX/upTeX defines one additional DVI command.

- `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 `dir` (255) appears at least once in a single pTeX/upTeX DVI, the `post_post` table of postamble contains ID = 3.

---

<sup>1</sup>The fact that there had been a mismatch in option names (`[dvi.pdfm]` vs. `[dvi.pdfmx]`) between packages may also have been part of it; `geometry` did not understand `[dvi.pdfmx]` option until 2018!

<sup>2</sup>Since L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> 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 *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 p $\TeX$  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  $\TeX$  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:

```
\ifx\eTeXversion\undefined
% Knuthian TeX, pTeX, upTeX:
% 256 registers, 1 mark
```



```

\else
  \ifx\omathchar\undefined
    % e-TeX, pdfTeX (in extended mode):
    % 32768 registers, 32768 mark classes
  \else
    % e-pTeX, e-upTeX (in extended mode):
    % 65536 registers, 65536 mark classes
  \fi
\fi

```

Here a primitive `\omathchar`, which is derived from  $\Omega$ , is used as a marker of a change file `fam256.ch`.<sup>3</sup>

## 6.2 Number of math families

In `pTeX` and `upTeX`, the number of math fonts is restricted to 16, each of which can contain 256 characters (same as Knuthian `TeX`). In  `$\varepsilon$ -pTeX` and  `$\varepsilon$ -upTeX`, a change file `fam256.ch`, which is derived from  $\Omega$ , extends the upper limit to 256. As a consequence,  `$\varepsilon$ -pTeX` and  `$\varepsilon$ -upTeX` allows 256 math fonts, each of which can contain 256 characters.<sup>4</sup>

For `pLATeX`/`upLATeX` users to use more than 16 math fonts, it is necessary to use macros which exploit  $\Omega$ -derived primitives such as `\omathchar`. Recent (u)pL<sup>A</sup>TeX (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 `mathfam256`<sup>5</sup> 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 SyncTeX additions (available in `pTeX`, `upTeX`, `$\varepsilon$ -pTeX`, `$\varepsilon$ -upTeX`)

In the standard build of `TeX Live`, SyncTeX extension is unavailable in Knuthian `TeX`; however, it is enabled in `pTeX` family.

---

<sup>3</sup>There is another `pTeX`-derived engine named `pTeX-ng` (or Asiatic `pTeX`) <https://github.com/clerkma/ptex-ng>; it is based on  `$\varepsilon$ -TeX` and `upTeX`, but currently does not adopt `fam256.ch` so it has the same number of registers and mark classes as  `$\varepsilon$ -TeX`.

<sup>4</sup> $\Omega$  allows 256 math fonts, each of which can contain 65536 characters.

<sup>5</sup><https://www.ctan.org/pkg/mathfam256>

- ▶ `\synctex`

### 6.3.2 pTeX additions (available in pTeX, upTeX, $\varepsilon$ -pTeX, $\varepsilon$ -upTeX)

- ▶ `\autospacing`
- ▶ `\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`
- ▶ `\ptexminorversion` — New primitive since p3.8.0 (T<sub>E</sub>X Live 2018)
- ▶ `\ptexrevision` — New primitive since p3.8.0 (T<sub>E</sub>X Live 2018)
- ▶ `\ptexversion` — New primitive since p3.8.0 (T<sub>E</sub>X Live 2018)
- ▶ `\scriptbaselineshiftfactor` — New primitive since p3.7 (T<sub>E</sub>X Live 2016)
- ▶ `\scriptscriptbaselineshiftfactor` — New primitive since p3.7 (T<sub>E</sub>X Live 2016)
- ▶ `\showmode`
- ▶ `\sjis`
- ▶ `\tate`
- ▶ `\tbaselineshift`
- ▶ `\textbaselineshiftfactor` — New primitive since p3.7 (T<sub>E</sub>X Live 2016)
- ▶ `\tfont`
- ▶ `\xkanjiskip`
- ▶ `\xspcode`
- ▶ `\ybaselineshift`
- ▶ `\yoko`
- ▶ `H`
- ▶ `Q`
- ▶ `zh`
- ▶ `zw`

### 6.3.3 upT<sub>E</sub>X additions (available in upT<sub>E</sub>X, $\varepsilon$ -upT<sub>E</sub>X)

- ▶ `\disablecjktoken`
- ▶ `\enablecjktoken`
- ▶ `\forcecjktoken`
- ▶ `\kchar`
- ▶ `\kchardef`
- ▶ `\ucs`
- ▶ `\uptexrevision` — New primitive since u1.23 (T<sub>E</sub>X Live 2018)
- ▶ `\uptexversion` — New primitive since u1.23 (T<sub>E</sub>X Live 2018)

### 6.3.4 $\varepsilon$ -pTeX additions (available in $\varepsilon$ -pTeX, $\varepsilon$ -upTeX)

- ▶ `\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)
- ▶ `\lastnodesubtype` — New primitive since 180226 (TeX Live 2018)
- ▶ `\odelcode`
- ▶ `\odelimiter`
- ▶ `\omathaccent`
- ▶ `\omathchar`
- ▶ `\omathchardef`
- ▶ `\omathcode`
- ▶ `\oradical`
- ▶ `\pagefistretch`
- ▶ `\pdfcreationdate` — New primitive since 130605 (TeX Live 2014)
- ▶ `\pdfelapsedtime` — New primitive since 161114 (TeX Live 2017)
- ▶ `\pdffiledump` — New primitive since 140506 (TeX Live 2015)
- ▶ `\pdffilemoddate` — New primitive since 130605 (TeX Live 2014)
- ▶ `\pdffilesize` — 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 (T<sub>E</sub>X Live 2016)
- ▶ `\pdfrandomseed` — New primitive since 161114 (T<sub>E</sub>X Live 2017)
- ▶ `\pdfresettimer` — New primitive since 161114 (T<sub>E</sub>X Live 2017)
- ▶ `\pdfsavepos`
- ▶ `\pdfsetrandomseed` — New primitive since 161114 (T<sub>E</sub>X Live 2017)
- ▶ `\pdfshellescape` — New primitive since 141108 (T<sub>E</sub>X Live 2015)
- ▶ `\pdfstrcmp`
- ▶ `\pdfuniformdeviate` — New primitive since 161114 (T<sub>E</sub>X Live 2017)
- ▶ `\readpapersizespecial` — New primitive since 180901 (T<sub>E</sub>X Live 2019)
- ▶ `\Uchar` — New primitive since 191112 (T<sub>E</sub>X Live 2020)
- ▶ `\Ucharcat` — New primitive since 191112 (T<sub>E</sub>X Live 2020)
- ▶ `\vfi`
- ▶ `fi`

### 6.3.5 $\varepsilon$ -upT<sub>E</sub>X additions (available in $\varepsilon$ -upT<sub>E</sub>X)

- ▶ `\currentcjktoken` — New primitive since 191112 (T<sub>E</sub>X Live 2020)

## 6.4 Omitted primitives and unsupported features

Compared to Knuthian T<sub>E</sub>X and  $\varepsilon$ -T<sub>E</sub>X, some primitives are omitted due to conflict with Japanese handling. One is encT<sub>E</sub>X extension, such as `\mubyte`. The MLT<sub>E</sub>X extension, such as `\charsubdef`, is included but not well-tested.

## 6.5 Behavior of Western T<sub>E</sub>X primitives

Here we provide some notes on behavior of Knuthian T<sub>E</sub>X and  $\varepsilon$ -T<sub>E</sub>X primitives when used within pT<sub>E</sub>X 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.”

`\catcode`, `\sfcode`, `\mathcode`, `\delcode`, `\lccode`, `\uccode`.

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

`\endlinechar, \newlinechar, \escapechar, \defaultshyphenchar, \defaultskewchar.`

## 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 `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 pTeX

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

```
\ifx\kanjiskip\undefined
\else
% pTeX / upTeX / e-pTeX / e-upTeX
\fi
```

## 6.6.2 Detecting upTeX

upTeX/ $\varepsilon$ -upTeX are almost upward compatible with pTeX/ $\varepsilon$ -pTeX respectively, 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 kanji encoding, for direct use of its huge character set.

The first difference can be detected by checking if `\...cjktoken` primitive is defined.

```
\ifx\enablecjktoken\undefined
\else
% upTeX/e-upTeX
\fi
```

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

```
\ifx\kanjiskip\undefined
\else
\ifnum\jis"2121="3000
% upTeX/e-upTeX with internal Unicode
\else
% pTeX/e-pTeX
% or, upTeX/e-upTeX with internal EUC-JP or Shift-JIS
\fi
\fi
```

Please note that the format-build setting of `-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.

## 6.6.3 Defining large integer constants

According to [2] (Section 3.3),

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

Here is the list of primitives which can be used for this purpose in 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  $\text{upT}_{\text{E}}\text{X}$ / $\varepsilon\text{-upT}_{\text{E}}\text{X}$ )  
 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  $\varepsilon\text{-pT}_{\text{E}}\text{X}$ / $\varepsilon\text{-upT}_{\text{E}}\text{X}$ )  
 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 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  $\varepsilon\text{-pT}_{\text{E}}\text{X}$ / $\varepsilon\text{-upT}_{\text{E}}\text{X}$ )
- (optional) 256–2147483647: `\chardef` (only for  $\text{upT}_{\text{E}}\text{X}$ / $\varepsilon\text{-upT}_{\text{E}}\text{X}$ )

#### 6.6.4 Creating a Japanese character token with a specified code

Short version:

- With  $\varepsilon\text{-pT}_{\text{E}}\text{X}$  191112 or later ( $\text{T}_{\text{E}}\text{X}$  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  $\text{pT}_{\text{E}}\text{X}$  family.



Short note on the “`\lowercase trick`”: to create a character token with a specified code value between 0–255 with Knuthian  $\text{T}_{\text{E}}\text{X}$ , 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  $\text{p}\text{T}\text{E}\text{X}$  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.  $\varepsilon$ - $\text{p}\text{T}\text{E}\text{X}$  191112 or later ( $\text{T}\text{E}\text{X}$  Live 2020) provides expandable primitives `\Uchar` and `\Ucharcat`, which are derived from  $\text{X}\text{Y}\text{T}\text{E}\text{X}$ . Regardless of their names, and unlike  $\text{X}\text{Y}\text{T}\text{E}\text{X}$  or  $\text{L}\text{u}\text{a}\text{T}\text{E}\text{X}$ , these primitives do *not* necessarily take a Unicode value as an argument. These primitives in  $\varepsilon$ - $\text{p}\text{T}\text{E}\text{X}$  and  $\varepsilon$ - $\text{u}\text{p}\text{T}\text{E}\text{X}$  take a valid character code (see 8.1) based on the internal kanji 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  $\varepsilon$ - $\text{p}\text{T}\text{E}\text{X}$ , 16–19 for  $\varepsilon$ - $\text{u}\text{p}\text{T}\text{E}\text{X}$ .

##### ► `\Ucharcat` *<character code>* *<category code>*

Expands to a character token with slot *<character code>* and *<category code>* specified.

- With  $\varepsilon$ - $\text{p}\text{T}\text{E}\text{X}$ :

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

## 6.7 Difference from pdf $\TeX$ in DVI mode

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

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

- All primitives specific to PDF output: `\pdfoutput`, `\pdfinfo`, `\pdfobj` etc.<sup>6</sup>
- All primitives related to micro-typography: `\pdffontexpand`, `\pdfprotrudechars`, etc.
- Some primitives related to handling of strings: `\pdfescapestring`, `\pdfescapehex` etc.

## 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.

---

<sup>6</sup> $\varepsilon$ -p $\TeX$ / $\varepsilon$ -up $\TeX$  has primitives `\pdfpagewidth` and `\pdfpageheight`; this is just because they were convenient for implementing `\pdfsavepos`, and their behavior is somewhat different from that of pdf $\TeX$ . Also note that  $\varepsilon$ -p $\TeX$ / $\varepsilon$ -up $\TeX$  does not have `\pdfhorigin` and `\pdfvorigin`.

## 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.<sup>7</sup>

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

In Knuthian  $\TeX$ , the primitive `\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 `\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 `\nullfont`” to discard all characters.

Moreover, p $\TeX$  and friends assume that each Japanese font (except “Japanese `\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 *character code* is a union of the following two:

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

---

<sup>7</sup>A JFM is a modified version of the standard TFM. It can be created by `(u)pPLtoTF`, and decoded by `(u)pTFtoPL`. Please also refer to the man pages of these programs (`ppltotf.man1.pdf` and `ptftopl.man1.pdf`).

The former is the same as Knuthian  $\text{\TeX}$ , but the latter is a problem. In  $\text{\upTeX}/\text{\varepsilon-upTeX}$  with `-kanji-internal=uptex` (default on), the range is very simple:

$$c \geq 0$$

However in  $\text{\pTeX}/\text{\varepsilon-pTeX}$ , only legacy encodings (EUC-JP as `eu`, or Shift-JIS as `sjis`) are available for `-kanji-internal`. In this case, the range can be represented as follows:

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

where

$$\begin{cases} C_1 = C_2 = \{\text{"a1}, \dots, \text{"fe}\} & (\text{eu}), \\ C_1 = \{\text{"81}, \dots, \text{"9f}\} \cup \{\text{"e0}, \dots, \text{"fc}\}, C_2 = \{\text{"40}, \dots, \text{"7e}\} \cup \{\text{"80}, \dots, \text{"fc}\} & (\text{sjis}). \end{cases}$$

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 `\iffontchar` with  $\text{\varepsilon-pTeX}$  190709 or later ( $\text{\TeX}$  Live 2020). Suppose a count register `\mycount` stores an integer, you can do it as follows:

```
\iffontchar\jfont\mycount
  % \mycount is a valid Japanese character code
\fi
```

Here the primitive `\jfont` is used merely as a representative non-empty<sup>8</sup> Japanese font containing all valid Japanese character code (see 7.2).

---

<sup>8</sup>This assumption is always safe after one of the standard  $\text{\pTeX}$  formats (e.g. plain  $\text{\pTeX}$ ,  $\text{\pLaTeX}$ ) is loaded.

## References

- [1] TUG DVI Standards Working Group, *The DVI Driver Standard, Level 0*.  
<https://ctan.org/pkg/dvistd>
- [2] Victor Eijkhout, *T<sub>E</sub>X by Topic, A T<sub>E</sub>Xnician's Reference*, Addison-Wesley, 1992.  
<https://www.eijkhout.net/texbytopic/texbytopic.html>

## Index

Symbols	
\accent	14
\autospacing	10
\autoxspacing	10
\char	14
\chardef	14, 16
\currentcjktoken	13
\currentspacingmode	12
\currentxspacingmode	12
\disablecjktoken	11
\disinhibitglue	10
\dtou	10
\enablecjktoken	11
\epTeXinputencoding	12
\epTeXversion	12
\euc	10
\expanded	12
\font	14
\fontdimen	14
\fontname	14
\forcecjktoken	11
\hfi	12
\if	14
\ifcat	14
\ifdbox	10
\ifddir	10
\ifincsname	12
\ifjfont	10
\ifmbox	10
\ifmdir	10
\ifpdfprimitive	12
\iftbox	10
\iftdir	10
\iftfont	10
\ifybox	10
\ifydir	10
\inhibitglue	10
\inhibitxspcode	10
\jcharwidowpenalty	10
\jfam	10
\jfont	10
\jis	10
\kanjiskip	10
\kansuji	10
\kansujichar	10
\kcatcode	10
\kchar	11
\kchardef	11, 16
\kuten	10
\lastnodechar	12
\lastnodesubtype	12
\mathchardef	16
\noautospacing	10
\noautoxspacing	11
\odelcode	12
\odelimiter	12
\omathaccent	12
\omathchar	12
\omathchardef	12, 16
\omathcode	12
\oradical	12
\pagefistretch	12
\pdfcreationdate	12
\pdfelapsedtime	12
\pdffiledump	12
\pdffilemoddate	12
\pdffilesize	12
\pdflastxpos	12
\pdflastypos	12
\pdfmdfivesum	12
\pdfnormaldeviate	12
\pdfpageheight	12
\pdfpagewidth	12
\pdfprimitive	13

\pdfrandomseed .....	13		
\pdfresettimer .....	13	zh .....	11
\pdfsavepos .....	13	zw .....	11
\pdfsetrandomseed .....	13		
\pdfshellescape .....	13		
\pdfstrcmp .....	13		
\pdfuniformdeviate .....	13		
\postbreakpenalty .....	11		
\prebreakpenalty .....	11		
\ptexminorversion .....	11		
\ptexrevision .....	11		
\ptexversion .....	11		
\readpapersizespecial .....	13		
\scriptbaselineshiftfactor .....	11		
\scriptscriptbaselineshiftfactor ..	11		
\showmode .....	11		
\sjis .....	11		
\synctex .....	10		
\tate .....	11		
\tbaselineshift .....	11		
\textbaselineshiftfactor .....	11		
\tfont .....	11		
\Uchar .....	13,17		
\Ucharcat .....	13,17		
\ucs .....	11		
\uptexrevision .....	11		
\uptexversion .....	11		
\vfi .....	13		
\xkanjiskip .....	11		
\xspcode .....	11		
\ybaselineshift .....	11		
\yoko .....	11		

**F**

fi .....	13
----------	----

**H**

H .....	11
---------	----

**Q**

Q .....	11
---------	----