Guide to pTEX for developers unfamiliar with Japanese

Japanese TEX Development Community*

version p4.1.1, June 15, 2024

pTEX and its variants, upTEX, ε-pTEX and ε-upTEX, are all TEX engines with native Japanese support. Its output is always a DVI file, which can be processed by several DVI drivers with Japanese support including *dvips* and *dvipdfmx*. Formats based on LATEX is called pLATEX when running on pTEX/ε-pTEX, and called upLATEX when running on upTEX/ε-upTEX.

Purpose of this document

This document is written for developers of TEX/LATEX, who aim to support pTEX/pLATEX and its variants upTEX/upLATEX. Knowledge of the followings are assumed:

- Basic knowledge of Western TEX (Knuthian TEX, ε-TEX and pdfTEX),
- ... and its programming conventions.

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

> Note: This English 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`

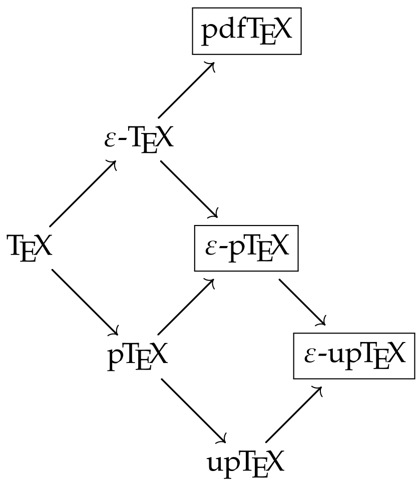
Contents

Part I

Brief introduction

1 pTeX and its variants

The figure below shows the relationship between engines.



pTeX is an old Japanese-specific extension of TeX82, which aims to support proper typesetting 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.

ε-pTeX and ε-upTeX are ε-TeX extensions of pTeX and upTeX respectively. In the current release, some extensions derived from pdfTeX and Ω 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 `\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 TEX

pTEX/upTEX 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.

ε-pTEX/ε-upTEX are almost upward compatible with ε-TEX, however, input handling is similar to pTEX/upTEX. It does not pass the e-TRIP test. That said, please note that "raw ε-TEX" is unavailable anymore in TEX Live and derived distributions; they provide a command `etex` only as "DVI mode of pdfTEX." You should note that ε-pTEX/ε-upTEX are *not* upward compatible with DVI mode of pdfTEX, which will be discussed later in Section 6.7.

There is no advantage to choose pTEX/upTEX over ε-pTEX/ε-upTEX, so we focus mainly on ε-pTEX/ε-upTEX.

4 LATEX on pTEX/upTEX — pLATEX/upLATEX

Format based on LATEX is called pLATEX when running on pTEX, and called upLATEX when running on upTEX. When building the format, `platex.ltx` (pLATEX) or `uplatex.ltx` (upLATEX) loads `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, pLATEX/upLATEX are almost upward compatible with original LATEX, except for the followings:

- Order of float objects; in pLATEX/upLATEX, ⟨*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 engine difference (Japanese handling, absence of pdfTEX features, etc). In recent versions of TEX Live and its derivatives, the default engines of pLATEX and upLATEX are as follows:

Date	pLATEX	upLATEX
TEX Live 2010	pTEX	—
TEX Live 2011	ε-pTEX	—
TEX Live 2012–2022, 2023 initial	ε-pTEX	ε-upTEX
Since 2023-06-01	ε-upTEX in legacy (see below)	ε-upTEX

The command `platex` started ε-pTEX (not pTEX) with preloaded format `platex.fmt` in TEX Live 2022. Since 2023-06-01, pLATEX has switched its engine from ε-pTEX to ε-upTEX in "legacy-encoding-compatibility mode." It means that additional primitives of (ε-)upTEX is available also on pLATEX, but the internal code of Japanese characters is still non-Unicode to keep the backward compatibility. Further information can be found in Section 6.6.2.

Part II

Details

5 Output format — DVI

The output of pTEX family is always a DVI file. This is in contrast to the mainstream of pdfTEX 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 `\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 LATEX, 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.[1]

- 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[2]:

```
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.

[1] 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!

[2] Since LATEX 2$_\varepsilon$ 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 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 regsiters 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 Ω, is used as a marker of a change file `fam256.ch`.[3]

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 ε-pTeX and ε-upTeX, a change file `fam256.ch`, which is derived from Ω, extends the upper limit to 256. As a consequence, ε-pTeX and ε-upTeX allows 256 math fonts, each of which can contain 256 characters.[4]

For pLaTeX/upLaTeX users to use more than 16 math fonts, it is necessary to use macros which exploit Ω-derived primitives such as `\omathchar`. Recent (u)pLaTeX (since 2016/11/29) partially supports this, and the maximum number of math alphabets that can be defined by `\DeclareMathAlphabet` is extended to 256 (`\e@mathgroup@top`) without needing any extension package. However, symbol fonts are restricted to 16 as `\DeclareMathSymbol` etc still use the standard `\mathchar` etc. A simple solution to use more symbol fonts as well as math alphabets is to load a package mathfam256[5] 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 pTeX additions (available in pTeX, upTeX, ε-pTeX, ε-upTeX)

- ▶ `\autospacing`
- ▶ `\autoxspacing`

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

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

[5] `https://www.ctan.org/pkg/mathfam256`

- `\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)

- `\ptexminorversion` — 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)
- `\scriptscriptbaselineshiftfactor` — New primitive since p3.7 (TeX Live 2016)
- `\showmode`
- `\sjis`
- `\tate`
- `\tbaselineshift`
- `\textbaselineshiftfactor` — New primitive since p3.7 (TeX Live 2016)
- `\tfont`
- `\tojis` — New primitive since p4.1.0 (TeX Live 2023)
- `\toucs` — New primitive since p3.10.0 (TeX Live 2022)
- `\ucs` — Imported from upTeX, since p3.10.0 (TeX Live 2022)[6]
- `\xkanjiskip`
- `\xspcode`
- `\ybaselineshift`
- `\yoko`
- `H`
- `Q`
- `zh`
- `zw`

6.3.2 upTeX additions (available in upTeX, ε-upTeX)

- `\disablecjktoken`
- `\enablecjktoken`
- `\forcecjktoken`

[6] The primitive \ucs was part of "upTeX 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 ε-pTEX additions (available in ε-pTEX, ε-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)
- `\lastnodefont` — New primitive since 220214 (TEX Live 2022)
- `\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 (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-upTEX additions (available in ε-upTEX)

- `\currentcjktoken` — New primitive since 191112 (TEX Live 2020)

6.3.5 Other cross-engine additions

SyncTEX extension (available in pTEX, upTEX, ε-pTEX, ε-upTEX):

- `\synctex`

TeX Live additions (available in pTeX, upTeX, ε-pTeX, ε-upTeX):

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

6.4 Omitted primitives and unsupported features

Compared to Knuthian TeX and ε-TeX, some primitives and extensions are omitted due to conflict with Japanese handling.

- The encTeX extension, including the primitives `\mubyte` etc., is unavailable.
- The MLTeX extension, such as `\charsubdef`, is not enabled by default. It becomes available with the command-line option `-mltex`, but not well-tested.

6.5 Behavior of Western TeX primitives

Here we provide some notes on behavior of Knuthian TeX and ε-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."

`\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`, `\defaulthyphenchar`, `\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 (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` ⟨*character code*⟩=⟨*character*⟩

► `\if` ⟨*token*$_1$⟩ ⟨*token*$_2$⟩, `\ifcat` ⟨*token*$_1$⟩ ⟨*token*$_2$⟩

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 is almost upward compatible with pTEX, 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 `\...cjktoken` primitive is defined.

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

This can be called "encoding detection" of upTEX.

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. This can be called "legacy-encoding-compatibility mode" of upTEX, where the `\kcatcode` difference remains but the internal encoding difference disappears. This method is used in building `platex.fmt` on ε-upTEX, since 2023-06-01. Therefore, to distinguish upLATEX from pLATEX, "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 `\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 upTeX/ε-upTeX)

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 ε-pTeX/ε-upTeX)

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 ε-pTeX/ε-upTeX)

6.6.4 Creating a Japanese character token with a specified code

Short version:

- With ε-pTeX 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 pTeX family.

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

```
\begingroup
\lccode`\?=\mycount
\lowercase{\endgroup \def\X{?}}
```

defines `\X` which expands to a character number `\mycount` while the `\catcode` of `?` (12) is preserved. However, the trick cannot be applied to Japanese characters, since pTeX 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. ε-pTEX 191112 or later (TEX Live 2020) provides expandable primitives `\Uchar` and `\Ucharcat`, which are derived from XeTEX. Regardless of their names, and unlike XeTEX or LuaTEX, these primitives do *not* necessarily take a Unicode value as an argument. These primitives in ε-pTEX and ε-upTEX take a valid character code (see 8.1) 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 ε-pTEX, 16–19 for ε-upTEX.

► `\Ucharcat` ⟨*character code*⟩ ⟨*category code*⟩

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

- With ε-pTEX:
  - 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-pTEX/ε-upTEX are *not* upward compatible with DVI mode of pdfTEX, which is available as the `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: `\pdfoutput`, `\pdfinfo`, `\pdfobj` etc.[7]
- 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

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 pTEX family, depending on the configuration and runtime options. To make this possible, pTEX 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 pTEX 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 pTEX/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:

  ```
  \ifx\epTeXinputencoding\undefined \else \epTeXinputencoding utf8 \fi
  ```

  it will help pTEX family to read forcibly in UTF-8, so it becomes *always* safe for 2016–.

[7] ε-pTEX/ε-upTEX has primitives `\pdfpagewidth` and `\pdfpageheight`; this is just because they were convenient for implementing `\pdfsavepos`, and their behavior is somewhat different from that of pdfTEX. Also note that ε-pTEX/ε-upTEX does not have `\pdfhorigin` and `\pdfvorigin`.

- If you aim to support broader legacy environment of pTeX 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 pTeX 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).

- upTeX default: always properly reads UTF-8 and ISO-2022-JP.
- pTeX default: always properly reads ISO-2022-JP. It also properly reads UTF-8 almost always[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 (ε-)pTeX, off for (ε-)upTeX, 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 (ε-)(u)pTeX. For Unix it was not implemented yet.
- In TeX Live 2017 and older, the default input encoding of (ε-)pTeX was Shift-JIS for Windows, UTF-8 for Unix.
- The primitive `\epTeXinputencoding` was added to ε-pTeX/ε-upTeX in TeX Live 2016. Older versions does not have it.
- Very old distributions by ASCII Corporation (–2009) supported only legacy encodings; UTF-8 was not allowed.

[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.

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[9].)

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 latter two are specific to pTEX family, which are defined in a JFM (Japanese TEX font metric) format.[10]

While typesetting, pTEX family automatically switches between these 3 fonts, depending on the character code and the writing direction:

[9] To be precise, it passes through a code conversion; however, this is an identity conversion which has no effect because upTEX defaults to internal Unicode.

[10] 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`).

- 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 pTeX 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 iniTeX mode. Once a valid Japanese font is selected, there is no way of selecting "Japanese `\nullfont`" to discard all characters.

Moreover, pTeX and friends assume that each Japanese font (except "Japanese `\nullfont`" in iniTeX 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 Japanese 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.

The former is the same as Knuthian TeX, but the latter is a problem.

In upTeX (default internal Unicode mode), the range is very simple:

$$c \geq 0$$

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

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

where

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

Therefore, the overall range of ⟨*character code*⟩ is *not* continuous. This is similar for "legacy-encoding-compatibility mode" of upTeX.

To check whether an integer is a valid Japanese character code or not, you can use `\iffontchar` with ε-pTeX 190709 or later (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[11] Japanese font containing all valid Japanese character code (see 7.2).

Note that pTEX (not including upTEX with internal Unicode) does not support typesetting characters outside JIS X 0208, which is a subset of accepted range of ⟨*character code*⟩ described above. To check if an integer is in the range of JIS X 0208, you can use `\toucs` with pTEX p3.10.0 or later (TEX Live 2022):

```
\ifnum\toucs\mycount>0
  % \mycount is in the range of JIS X 0208
\fi
```

The primitive `\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 pTEX formats (e.g. plain pTEX, pLATEX) is loaded.

[12] Defined in `jisx0208.h` of `ptexenc` library.

References

[1] TUG DVI Standards Working Group, *The DVI Driver Standard, Level 0*. `https://ctan.org/pkg/dvistd`

[2] Victor Eijkhout, *TEX by Topic, A TEXnician's Reference*, Addison-Wesley, 1992. `https://www.eijkhout.net/texbytopic/texbytopic.html`

Index