The LuaTEX-ja package

The LuaTEX-ja project team

20240714.0 (July 14, 2024)

Contents

This documentation is far from complete. It may have many grammatical (and contextual) errors. Also, several parts are written in Japanese only.

Part I
User's manual

1 Introduction

The LuaTEX-ja package is a macro package for typesetting high-quality Japanese documents when using LuaTEX.

1.1 Backgrounds

Traditionally, ASCII pTEX, an extension of TEX, and its derivatives are used to typeset Japanese documents in TEX. pTEX is an engine extension of TEX: so it can produce high-quality Japanese documents without using very complicated macros. But this point is a mixed blessing: pTEX is left behind from other extensions of TEX, especially ε-TEX and pdfTEX, and from changes about Japanese processing in computers (*e.g.*, the UTF-8 encoding).

Recently extensions of pTEX, namely upTEX (Unicode-implementation of pTEX) and ε-pTEX (merging of pTEX and ε-TEX extension), have developed to fill those gaps to some extent, but gaps still exist.

However, the appearance of LuaTEX changed the whole situation. With using Lua "callbacks", users can customize the internal processing of LuaTEX. So there is no need to modify sources of engines to support Japanese typesetting: to do this, we only have to write Lua scripts for appropriate callbacks.

1.2 Major changes from pTEX

The LuaTEX-ja package is under much influence of pTEX engine. The initial target of development was to implement features of pTEX. However, implementing all feature of pTEX is impossible, since all process of LuaTEX-ja must be implemented only by Lua and TEX macros. Hence *LuaTEX-ja is not a just porting of pTEX; unnatural specifications/behaviors of pTEX were not adopted.*

The followings are major changes from pTEX. For more detailed information, see Part III or other sections of this manual.

■Command names pTEX addes several primitives, such as `\kanjiskip`, `\prebreakpenalty`, and `\ifydir`. They can be used as follows:

```
\kanjiskip=10pt  \dimen0=kanjiskip
\tbaselineshift=0.1zw
\dimen0=\tbaselineshift
\prebreakpenalty`あ=100
\ifydir ... \fi
```

However, we cannot use them under LuaTEX-ja. Instead of them, we have to write as the following.

```
\ltjsetparameter{kanjiskip=10pt}  \dimen0=\ltjgetparameter{kanjiskip}
\ltjsetparameter{talbaselineshift=0.1\zw}
\dimen0=\ltjgetparameter{talbaselineshift}
\ltjsetparameter{prebreakpenalty={`あ,100}}
\ifnum\ltjgetparameter{direction}=4 ... \fi
```

Note that pTEX adds new two useful units, namely `zw` and `zh`. As shown above, *they are changed to* `\zw` *and* `\zh` *respectively* in LuaTEX-ja.[1]

■Linebreak after a Japanese character In pTEX, a line break after Japanese character is ignored (and doesn't yield a space), since line breaks (in source files) are permitted almost everywhere in Japanese texts. However, LuaTEX-ja doesn't have this feature completely, because of a specification of LuaTEX. For the detail, see Section 15.

[1]LuaTEX-ja 20200127.0 introduces `\ltj@zw` and `\ltj@zh`, which are copy of `\zw` and `\zh`.

■**Spaces related to Japanese characters** The insertion process of glues/kerns between two Japanese characters and between a Japanese character and other characters (we refer glues/kerns of both kinds as **JAglue**) is rewritten from scratch.

- As LuaTeX's internal ligature handling is *node-based* (*e.g.*, `of{}fice` doesn't prevent ligatures), the insertion process of **JAglue** is now *node-based*.
- Furthermore, nodes between two characters which have no effects in line break (*e.g.*, `\special` node) and kerns from italic correction are ignored in the insertion process.
- *Caution: due to above two points, many methods which did for the dividing the process of the insertion of **JAglue** in pTeX are not effective anymore.* In concrete terms, the following two methods are not effective anymore:

  `ちょ{}っと` `ちょ\/っと`

  If you want to do so, please put an empty horizontal box (hbox) between it instead:

  `ちょ\hbox{}っと`

- In the process, two Japanese fonts which only differ in their "real" fonts are identified.

■**Directions** From version 20150420.0, LuaTeX-ja supports vertical writing. We implement this feature by using callbacks of LuaTeX; so it must *not* be confused with Ω-style direction support of LuaTeX itself. Due to implementation, the dimension returned by `\wd`, `\ht`, or `\dp` depends on the content of the register *only*. This is major difference with pTeX.

■**`\discretionary`** Japanese characters in discretionary break (`\discretionary`) is not supported.

■**Greek and Cyrillic letters, and ISO 8859-1 symbols** By default, LuaTeX-ja uses Japanese fonts to typeset Greek and Cyrillic letters, To change this behavior, put `\ltjsetparameter{jacharrange={-2,-3}}` in the preamble. For the detailed description, see Subsection 4.1.

From version 20150906.0, characters which belongs both ISO 8859-1 and JIS X 0208, such as ¶ and §, are now typeset in alphabetic fonts.

1.3 Notations

In this document, the following terms and notations are used:

- Characters are classified into following two types. Note that the classification can be customized by a user (see Subsection 4.1).
  - **JAchar**: standing for characters which is used in Japanese typesetting, such as Hiragana, Katakana, Kanji, and other Japanese punctuation marks.
  - **ALchar**: standing for all other characters like latin alphabets.

  We say *alphabetic fonts* for fonts used in **ALchar**, and *Japanese fonts* for fonts used in **JAchar**.
- A word in a sans-serif font with underline (like prebreakpenalty) means an internal parameter for Japanese typesetting, and it is used as a key in `\ltjsetparameter` command.
- A word in a sens-serif font without underline (like fontspec) means a package or a class of LaTeX.
- In this document, natural numbers start from zero. ω denotes the set of all natural numbers which can be used in TeX.

1.4 About the project

■Project Wiki Project Wiki is under construction.

- https://github.com/luatexja/luatexja/wiki/Home(en) (English)
- https://github.com/luatexja/luatexja/wiki (Japanese)
- https://github.com/luatexja/luatexja/wiki/Home(zh) (Chinese)

This project is hosted by GitHub.

■Members

- Hironori KITAGAWA
- Yusuke KUROKI
- Tomoaki HONDA
- Kazuki MAEDA
- Noriyuki ABE
- Shuzaburo SAITO
- Takayuki YATO
- Munehiro YAMAMOTO
- MA Qiyuan

2 Getting Started

2.1 Installation

The following packages are needed for the LuaTeX-ja package.

- LuaTeX 1.10.0 (or later) (DVI output (`\outputmode=0` is not supported.)
- recent luaotfload (v3.1 or later recommended)
- adobemapping (Adobe cmap and pdfmapping files)
- etoolbox (if you want to use LuaTeX-ja with LaTeX 2_ε)
- ltxcmds, pdftexcmds
- fontspec v2.7c (or later)
- *Harano Aji fonts* (`https://github.com/trueroad/HaranoAjiFonts`)
  More specifically, HaranoAjiMincho-Regular and HaranoAjiGothic-Medium.

Now LuaTeX-ja is available from CTAN (in the `macros/luatex/generic/luatexja` directory), and the following distributions:

- TeX Live (in `texmf-dist/tex/luatex/luatexja`)
- MiKTeX (in `luatexja.tar.xz`)

Harano Aji fonts are also available in these distributions (`haranoaji` in TeX Live and MiKTeX).

■HarfBuzz and LuaTeX-ja Using LuaTeX-ja with LuaHBTeX(LuaTeX integrated with HarfBuzz) is not well tested. Maybe documents can typeset without an error, but with unwanted results (especially, vertical typesetting and `\CID`).

Especially, *We don't recommend defining a Japanese font with HarfBuzz*, by specifying `Renderer=Harfbuzz` etc. (fontspec) or mode=harf (otherwise).

■Manual installation

1. Download the source, by one of the following method. At the present, LuaTeX-ja has no *stable* release.
   - Clone the Git repository by
     ```
     $ git clone https://github.com/luatexja/luatexja.git
     ```
   - Download the `zip` archive of HEAD in the `master` branch from
     `https://github.com/luatexja/luatexja/archive/refs/heads/master.zip`.

   Note that the `master` branch, and hence the archive in CTAN, are not updated frequently; the forefront of development is not the `master` branch.
2. Extract the archive. You will see `src/` and several other sub-directories. But only the contents in `src/` are needed to work LuaTeX-ja.
3. If you downloaded this package from CTAN, you have to run following commands to generate classes:
   ```
   $ cd src
   $ lualatex ltjclasses.ins
   $ lualatex ltjsclasses.ins
   $ lualatex ltjltxdoc.ins
   ```
4. Copy all the contents of `src/` into one of your `TEXMF` tree. `TEXMF/tex/luatex/luatexja/` is an example location. If you cloned entire Git repository, making a symbolic link of `src/` instead copying is also good.
5. If `mktexlsr` is needed to update the file name database, make it so.

2.2 Cautions

For changes from pTeX, see Subsection 1.2.

- The encoding of your source file must be UTF-8. Other encodings, such as EUC-JP or Shift-JIS, are not supported.
- LuaTeX-ja is very slower than pTeX, and uses a lot of memory.
- Note that when LuaTeX-ja is loaded in plain LuaTeX, we cannot use color specification on font loading, such as

```
\font\hoge=lmroman10-regular.otf:color=FF0000 % \font primitive
```

  This is because codes for shifting baseline in math mode (LuaTeX-ja) collide with and prevents loading codes for font color (luaotfload) in these environments. *We recommend to use LaTeX 2020-02-02 (or later)*, since we can avoid this collision in there.

2.3 Using in plain TeX

To use LuaTeX-ja in plain TeX, simply put the following at the beginning of the document:

```
\input luatexja.sty
```

This does minimal settings (like `ptex.tex`) for typesetting Japanese documents:

- The following 12 Japanese fonts are preloaded:

direction	**classification**	**font name**	**"10 pt"**	**"7 pt"**	**"5 pt"**
yoko (horizontal)	*mincho*	HaranoAjiMincho-Regular	`\tenmin`	`\sevenmin`	`\fivemin`
	gothic	HaranoAjiGothic-Medium	`\tengt`	`\sevengt`	`\fivegt`
tate (vertical)	*mincho*	HaranoAjiMincho-Regular	`\tentmin`	`\seventmin`	`\fivetmin`
	gothic	HaranoAjiGothic-Medium	`\tentgt`	`\seventgt`	`\fivetgt`

  - The "default" Japanese fonts (and JFMs for them) can be modified by defining `\ltj@stdmcfont` etc. *before* one inputs `luatexja.sty` (Subsection 8.3).
  - A character in an alphabetic font is generally smaller than a Japanese font in the same size. So actual size specification of these Japanese fonts is in fact smaller than that of alphabetic fonts, namely scaled by 0.962216.

- The amount of glue that are inserted between a **JAchar** and an **ALchar** (the parameter xkanjiskip) is set to

$$(0.25 \cdot 0.962216 \cdot 10\,\mathrm{pt})^{+1\,\mathrm{pt}}_{-1\,\mathrm{pt}} = 2.40554\,\mathrm{pt}^{+1\,\mathrm{pt}}_{-1\,\mathrm{pt}}.$$

2.4 Using in LaTeX

Using in LaTeX 2_ε is basically same. To set up the minimal environment for Japanese, you only have to load `luatexja.sty`:

```
\usepackage{luatexja}
```

It also does minimal settings (counterparts in pLaTeX are `plfonts.dtx` and `pldefs.ltx`).

- Font encodings for Japanese fonts are `JY3` (for horizontal direction) and `JT3` (for vertical direction).
- Traditionally, Japanese documents use only two families: *mincho* (明朝体) and *gothic* (**ゴシック体**). *mincho* is used in the main text, while *gothic* is used in the headings or for emphasis.

classification	commands		family
mincho (明朝体)	`\textmc{...}`	`{\mcfamily ...}`	`\mcdefault`
gothic (ゴシック体)	`\textgt{...}`	`{\gtfamily ...}`	`\gtdefault`
(Japanese counterpart for typewriter font)	—	—	`\jttdefault`

Here `\jttdefault` specifies the Japanese font family in `\verb` or `verbatim` environment, and its default value is `\mcdefault` (mincho family).[2] LuaTEX-ja does not define commands to only switch current Japanese font family to `\jttdefault`.

- By default, the following fonts are used for these two families.

classification	family	`\mdseries`	`\bfseries`	scale
mincho (明朝体)	`mc`	HaranoAjiMincho-Regular	HaranoAjiGothic-Medium	0.962216
gothic (ゴシック体)	`gt`	HaranoAjiGothic-Medium	HaranoAjiGothic-Medium	0.962216

- Note that the bold series (series `bx` or `b`) in both family are same as the medium series of gothic family. There is no italic nor slanted shape for these `mc` and `gt`.
- From version 20181102.0, one can specifies `disablejfam` option at loading LuaTEX-ja. This option prevents loading a patch for LATEX, which are needed to support Japanese characters in math mode.

  Without `disablejfam` option, one can typeset Japanese characters in math mode as `$あ$` (see Page 10) as before. Japanese characters in math mode are typeset by the font family `mc`.
- If you use the beamer class with the default font theme (which uses sans serif fonts) and with LuaTEX-ja, you might want to change default Japanese fonts to the gothic family. The following line changes the default Japanese font family to it:

```
\renewcommand{\kanjifamilydefault}{\gtdefault}
```

However, above settings are not sufficient for Japanese-based documents. To typeset Japanese-based documents, you are better to use class files other than `article.cls`, `book.cls`, and so on. At the present, LuaTEX-ja has the counterparts of jclasses (standard classes in pLATEX) and jsclasses (classes by Haruhiko Okumura), namely, ltjclasses[3] and ltjsclasses[4].

Original jsclasses use `\mag` primitive to set the main document font size. However, LuaTEX does not support `\mag` in PDF output, so ltjsclasses uses the `nomag*` option[5] by default to set the main font size. If this causes some unexpected behavior, specify `nomag` option in `\documentclass`.

■geometry package and classes for vertical writing It is well-known that the geometry package produces the following error, when classes for vertical writing is used:

```
! Incompatible direction list can't be unboxed.
\@begindvi ->\unvbox \@begindvibox
                                  \global \let \@begindvi \@empty
```

Now, LuaTEX-ja automatically applies the patch lltjp-geometry to the geometry package, when the direction of the document is *tate* (vertical writing). This patch lltjp-geometry also can be used in pLATEX; for the detail, please refer `lltjp-geometry.pdf` (Japanese).

3 Changing Fonts

3.1 plain TEX and LATEX 2$_\varepsilon$

■plain TEX To change Japanese fonts in plain TEX, you must use the command `\jfont` and `\tfont`. So please see Subsection 8.1.

[2] When ltjsclasses classes are used, or luatexja-fontspec (or luatexja-preset) is loaded with `match` option, `\ttfamily` changes the current Japanese font amily to `\jttdefault`. These classes and packages also redefine `\jttdefault` to `\gtdefault` (*gothic* family).

[3] `ltjarticle.cls`, `ltjbook.cls`, `ltjreport.cls`, `ltjtarticle.cls`, `ltjtbook.cls`, `ltjtreport.cls`. The latter `ltjt*.cls` are for vertically written Japanese documents.

[4] `ltjsarticle.cls`, `ltjsbook.cls`, `ltjsreport.cls`, `ltjskiyou.cls`.

[5] Same effect as the BXjscls classes (by Takayuki Yato) and jsclasses. However, these classes uses only TEX code, but ltjsclasses uses Lua code.

■LaTeX 2ε (NFSS2) For LaTeX 2ε, LuaTeX-ja adopted most of the font selection system of pLaTeX 2ε (in `plfonts.dtx`).

	encoding	family	series	shape	selection
Alphabetic fonts	`\romanencoding`	`\romanfamily`	`\romanseries`	`\romanshape`	`\useroman`
Japanese fonts	`\kanjiencoding`	`\kanjifamily`	`\kanjiseries`	`\kanjishape`	`\usekanji`
both	—	–	`\fontseries`	`\fontshape`*	—
auto select	`\fontencoding`	`\fontfamily`	—	—	`\usefont`

- `\fontfamily`, `\fontseries`, and `\fontshape` try to change attributes of Japanese fonts, as well as those of alphabetic fonts. Of course, `\selectfont` is needed to select current text fonts.

  Note that `\fontshape` always changes current alphabetic font shape, but it does *not* change current Japanese font shape if the target shape is unavailable for current Japanese encoding/family/series. For the detail, see Subsection 11.2.

- `\fontencoding{⟨encoding⟩}` changes the encoding of alphabetic fonts or Japanese fonts depending on the argument. For example, `\fontencoding{JY3}` changes the encoding of Japanese fonts to `JY3`, and `\fontencoding{T1}` changes the encoding of alphabetic fonts to `T1`. `\fontfamily` also changes the current Japanese font family, the current alphabetic font family, *or both*. For the detail, see Subsection 11.2.

- For defining a Japanese font family, use `\DeclareKanjiFamily` instead of `\DeclareFontFamily`. (In previous version of LuaTeX-ja, using `\DeclareFontFamily` didn't cause any problem. But this no longer applies the current version.)

- Defining a Japanese font shape can be done by usual `\DeclareFontShape`:

```
\DeclareFontShape{JY3}{mc}{b}{n}{<-> s*HaranoAjiMincho--Bold:jfm=ujis;-kern}{}
    % Harano Aji Mincho Bold
```

■Japanese characters in math mode Since pTeX supports Japanese characters in math mode, there are sources like the following:

```
1 $f_{高温}$~($f_{\text{high temperature}}$).
2 \[ y=(x-1)^2+2\quad よって\quad y>0 \]
3 $5\in 素:=\{\,p\in\mathbb N:\text{$p$ is a
      prime}\,\}$.
```

$f_{高温}$ ($f_{\text{high temperature}}$).

$$y=(x-1)^2+2\quad よって\quad y>0$$

$5\in 素:=\{\,p\in\mathbb N:\text{p is a prime}\,\}$.

We (the project members of LuaTeX-ja) think that using Japanese characters in math mode are allowed if and only if these are used as identifiers. In this point of view,

- The lines 1 and 2 above are not correct, since "高温" in above is used as a textual label, and "よって" is used as a conjunction.
- However, the line 3 is correct, since "素" is used as an identifier.

Hence, in our opinion, the above input should be corrected as:

```
$f_{\text{高温}}$~%
($f_{\text{high temperature}}$).
\[ y=(x-1)^2+2\quad
  \mathrel{\mbox{よって}}\quad y>0 \]
$5\in 素:=\{\,p\in\mathbb N:\text{$p$ is a
      prime}\,\}$.
```

$f_{\text{高温}}$ ($f_{\text{high temperature}}$).

$$y=(x-1)^2+2\quad \mathrel{\mbox{よって}}\quad y>0$$

$5\in 素:=\{\,p\in\mathbb N:\text{p is a prime}\,\}$.

We also believe that using Japanese characters as identifiers is rare, hence we don't describe how to change Japanese fonts in math mode in this chapter. For the method, please see Subsection 8.6.

When LuaTeX-ja is loaded with `disablejfam` option, one cannot write Japanese characters in math mode as `$素$`. At that case, one have to use `\mbox` (or `\text` in the amsmath package).

Table 1. Commands of luatexja-fontspec

Japanese fonts	`\jfontspec`	`\setmainjfont`	`\setsansjfont`	`\setmonojfont`
Alphabetic fonts	`\fontspec`	`\setmainfont`	`\setsansfont`	`\setmonofont`
Japanese fonts	`\newjfontfamily`	`\renewjfontfamily`	`\setjfontfamily`	`\providejfontfamily`
Alphabetic fonts	`\newfontfamily`	`\renewfontfamily`	`\setfontfamily`	`\providefontfamily`
Japanese fonts	`\newjfontface`	`\renewjfontface`	`\setjfontface`	`\providejfontface`
Alphabetic fonts	`\newfontface`	`\renewfontface`	`\setfontface`	`\providefontface`
Japanese fonts	`\defaultjfontfeatures`	`\addjfontfeatures`		
Alphabetic fonts	`\defaultfontfeatures`	`\addfontfeatures`		

3.2 luatexja-fontspec package

To use the functionality of the fontspec package to Japanese fonts, it is needed to load the luatexja-fontspec package in the preamble, as follows:

`\usepackage[⟨options⟩]{luatexja-fontspec}`

This luatexja-fontspec package automatically loads luatexja and fontspec packages, if needed.

In the luatexja-fontspec package, several commands are defined as counterparts of original commands in the fontspec package (see Table 1):

The package option of luatexja-fontspec are the followings:

`match`
: If this option is specified, usual family-changing commands such as `\rmfamily`, `\textrm`, `\sffamily`, … also change Japanese font family.

`pass=`⟨*options*⟩
: *(Obsoleted)* Specify options ⟨*options*⟩ which will be passed to the fontspec package.

`scale=`⟨*float*⟩
: Override the ratio of the font size of Japanese fonts to that of alphabetic fonts. The default value is determined as follows:

  - The value of `\Cjascale` is used, if this control sequence is already defined.
  - It is calculated automatically from the current Japanese font at the loading of the package, if `\Cjascale` is not defined.

  `\Cjascale` is defined in ltjclasses and ltjsclasses.

All other options listed above are simply passed to the fontspec package. This means that two lines below are equivalent, for example.

```
\usepackage[no-math]{fontspec}\usepackage{luatexja-fontspec}
\usepackage[no-math]{luatexja-fontspec}
```

Note that kerning information in a font is not used (that is, `kern` feature is set off) by default in these seven (or eight) commands. This is because of the compatibility with previous versions of LuaTeX-ja (see 8.1).

Below is an example of `\jfontspec`.

```
\jfontspec[CJKShape=NLC]{HaranoAjiMincho-Regular}
JIS~X~0213:2004→辻鯵\par
\jfontspec[CJKShape=JIS1990]{HaranoAjiMincho-Regular}
JIS~X~0208-1990→辻鯵\par
\jfontspec[CJKShape=JIS1978]{HaranoAjiMincho-Regular}
JIS~C~6226-1978→辻鯵
```

JIS X 0213:2004 →辻鯵
JIS X 0208-1990 →辻鯵
JIS C 6226-1978 →辻鯵

3.3 Presets of Japanese fonts

With luatexja-preset package, one use one of "preset" to simplify Japanese font setting. For details of package options, and those of each presets, please see Subsecion 13.6. The following presets are defined:

```
haranoaji, hiragino-pro, hiragino-pron, ipa, ipa-hg, ipaex, ipaex-hg,
kozuka-pr6, kozuka-pr6n, kozuka-pro, moga-mobo, moga-mobo-ex, bizud,
morisawa-pr6n, morisawa-pro, ms, ms-hg, noembed, noto-otc, noto-otf, noto,
noto-jp, sourcehan, sourcehan-jp, ume, yu-osx, yu-win, yu-win10
```

For example, this document loads luatexja-preset package by

```
\usepackage[haranoaji]{luatexja-preset}
```

which means that Harano Aji fonts will be used in this document.

3.4 \CID, \UTF, and macros in japanese-otf package

Under pLaTeX, japanese-otf package (developed by Shuzaburo Saito) is used for typesetting characters which is in Adobe-Japan1-6 CID but not in JIS X 0208. Since this package is widely used, LuaTeX-ja supports some of functions in the japanese-otf package, as an external package luatexja-otf.

```
森\UTF{9DD7}外と\CID{13966}田百\UTF{9592}とが
\UTF{9AD9}島屋に\\
\CID{7652}飾区の\CID{13706}野家，
\CID{1481}城市，葛西駅，\\
高崎と\CID{8705}\UTF{FA11}，濱と\ajMayuHama\\
\aj半角{カタカナ}\ajKakko3\ajMaruYobi{2}%
\ajLig{令和}\ajLig{〇問}\ajJIS
```

森鷗外と内田百閒とが髙島屋に
葛飾区の𠮷野家，葛城市，葛西駅，
高崎と髙﨑，濱と濵
ｶﾀｶﾅ⑶㊊㋿㉄〄

4 Changing Internal Parameters

There are many internal parameters in LuaTeX-ja. And due to the behavior of LuaTeX, most of them are not stored as internal register of TeX, but as an original storage system in LuaTeX-ja. Hence, to assign or acquire those parameters, you have to use commands `\ltjsetparameter` and `\ltjgetparameter`.

4.1 Range of JAchars

LuaTeX-ja divides the Unicode codespace `U+0080–U+10FFFF` into *character ranges*, numbered 1 to 217. The grouping can be (globally) customized by `\ltjdefcharrange`. The next line adds whole characters in Supplementary Ideographic Plane and the character "漢" to the character range 100.

```
\ltjdefcharrange{100}{"20000-"2FFFF,`漢}
```

A character can belong to only one character range. For example, whole SIP belong to the range 4 in the default setting of LuaTeX-ja, and if one executes the above line, then SIP will belong to the range 100 and be removed from the range 4.

The distinction between **ALchar** and **JAchar** is performed by character ranges. This can be edited by setting the jacharrange parameter. For example, the code below is just the default setting of LuaTeX-ja, and it sets

- a character which belongs character ranges 1, 4, 5, and 8 is **ALchar**,
- a character which belongs character ranges 2, 3, 6, 7, and 9 is **JAchar**.

```
\ltjsetparameter{jacharrange={-1, +2, +3, -4, -5, +6, +7, -8, +9}}
```

The argument to jacharrange parameter is a list of non-zero integer. Negative integer $-n$ in the list means that "each character in the range n is an **ALchar**", and positive integer $+n$ means that "… is a **JAchar**".

Note that characters `U+0000–U+007F` are always treated as an **ALchar** (this cannot be customized).

Table 2. Characters in predefined character range 8.

§ (U+00A7)	Section Sign	¨ (U+00A8)	Diaeresis
° (U+00B0)	Degree sign	± (U+00B1)	Plus-minus sign
´ (U+00B4)	Spacing acute	¶ (U+00B6)	Paragraph sign
× (U+00D7)	Multiplication sign	÷ (U+00F7)	Division Sign

Table 3. Unicode blocks in predefined character range 1.

U+0080–U+00FF	Latin-1 Supplement	U+0100–U+017F	Latin Extended-A
U+0180–U+024F	Latin Extended-B	U+0250–U+02AF	IPA Extensions
U+02B0–U+02FF	Spacing Modifier Letters	U+0300–U+036F	Combining Diacritical Marks
U+1E00–U+1EFF	Latin Extended Additional		

Table 4. Unicode blocks in predefined character range 3.

U+2070–U+209F	Superscripts and Subscripts		
U+20A0–U+20CF	Currency Symbols	U+20D0–U+20FF	Comb. Diacritical Marks for Symbols
U+2100–U+214F	Letterlike Symbols	U+2150–U+218F	Number Forms
U+2190–U+21FF	Arrows	U+2200–U+22FF	Mathematical Operators
U+2300–U+23FF	Miscellaneous Technical	U+2400–U+243F	Control Pictures
U+2500–U+257F	Box Drawing	U+2580–U+259F	Block Elements
U+25A0–U+25FF	Geometric Shapes	U+2600–U+26FF	Miscellaneous Symbols
U+2700–U+27BF	Dingbats	U+2900–U+297F	Supplemental Arrows-B
U+2980–U+29FF	Misc. Math Symbols-B	U+2B00–U+2BFF	Misc. Symbols and Arrows

■**Default character ranges** LuaTEX-ja predefines nine character ranges for convenience. They are determined from the following data:

- Blocks in Unicode 12.0.0.
- The `Adobe-Japan1-UCS2` mapping between a CID Adobe-Japan1- and Unicode.
- The PXbase bundle for upTEX by Takayuki Yato.

Now we describe these nine ranges. The superscript "J" or "A" after the number shows whether each character in the range is treated as **JAchar**s or not by default. These settings are similar to the `prefercjk` settings defined in `PXbase` bundle. Any characters equal to or above `U+0080` which does not belong to these eight ranges belongs to the character range 217.

Range 8^A The intersection of the upper half of ISO 8859-1 (Latin-1 Supplement) and JIS X 0208 (a basic character set for Japanese). The character list is indicated in Table 2.

Range 1^A Latin characters that some of them are included in Adobe-Japan1-7. This range consists of the Unicode ranges indicated in Table 3, *except characters in the range 8 above.*

Range 2^J Greek and Cyrillic letters. JIS X 0208 (hence most of Japanese fonts) has some of these characters.

- `U+0370–U+03FF`: Greek and Coptic
- `U+0400–U+04FF`: Cyrillic
- `U+1F00–U+1FFF`: Greek Extended

Range 3^J Miscellaneous symbols. The block list is indicated in Table 4.

Range 9^J The intersection of the "General Punctuation" block (`U+2000–U+206F`) and Adobe-Japan1-7 character collection. This character range characters in Table 5.

Range 4^A Characters usually not in Japanese fonts. This range consists of almost all Unicode blocks which are not in other predefined ranges. Hence, instead of showing the block list, we put the definition of this range itself.

```
\ltjdefcharrange{4}{%
   "500-"10FF, "1200-"1DFF, "2440-"245F, "27C0-"28FF, "2A00-"2AFF,
  "2C00-"2E7F, "4DC0-"4DFF, "A4D0-"A95F, "A980-"ABFF, "E000-"F8FF,
```

Table 5. Characters in predefined character range 9.

	(U+2002)	En space	‐	(U+2010)	Hyphen
‑	(U+2011)	Non-breaking hyphen	–	(U+2013)	En dash
—	(U+2014)	Em dash	―	(U+2015)	Horizontal bar
‖	(U+2016)	Double vertical line	‘	(U+2018)	Left single quotation mark
’	(U+2019)	Right single quotation mark	‚	(U+201A)	Single low-9 quotation mark
“	(U+201C)	Left double quotation mark	”	(U+201D)	Right double quotation mark
„	(U+201E)	Double low-9 quotation mark	†	(U+2020)	Dagger
‡	(U+2021)	Double dagger	•	(U+2022)	Bullet
‥	(U+2025)	Two dot leader	…	(U+2026)	Horizontal ellipsis
‰	(U+2030)	Per mille sign	′	(U+2032)	Prime
″	(U+2033)	Double prime	‹	(U+2039)	Single left-pointing angle quot.
›	(U+203A)	Single right-pointing angle quot.	※	(U+203B)	Reference mark
‼	(U+203C)	Double exclamation mark	‾	(U+203E)	Overline
‿	(U+203F)	Undertie	⁂	(U+2042)	Asterism
⁄	(U+2044)	Fraction slash	⁇	(U+2047)	Double question mark
⁈	(U+2048)	Question exclamation mark	⁉	(U+2049)	Exclamation question mark
⁑	(U+2051)	Two asterisks aligned vertically			

Table 6. Unicode blocks in predefined character range 6.

`U+2460–U+24FF`	Enclosed Alphanumerics	`U+2E80–U+2EFF`	CJK Radicals Supplement
`U+3000–U+303F`	CJK Symbols and Punctuation	`U+3040–U+309F`	Hiragana
`U+30A0–U+30FF`	Katakana	`U+3190–U+319F`	Kanbun
`U+31F0–U+31FF`	Katakana Phonetic Extensions	`U+3200–U+32FF`	Enclosed CJK Letters and Months
`U+3300–U+33FF`	CJK Compatibility	`U+3400–U+4DBF`	CJK Unified Ideographs Ext-A
`U+4E00–U+9FFF`	CJK Unified Ideographs	`U+F900–U+FAFF`	CJK Compatibility Ideographs
`U+FE10–U+FE1F`	Vertical Forms	`U+FE30–U+FE4F`	CJK Compatibility Forms
`U+FE50–U+FE6F`	Small Form Variants	`U+FF00–U+FFEF`	Halfwidth and Fullwidth Forms
`U+1B000–U+1B0FF`	Kana Supplement	`U+1B100–U+1B12F`	Kana Extended-A
`U+1F100–U+1F1FF`	Enclosed Alphanumeric Supp.	`U+1F200–U+1F2FF`	Enclosed Ideographic Supp.
`U+20000–U+2FFFF`	(Supp. Ideographic Plane)	`U+30000–U+3FFFF`	(Tert. Ideographic Plane)
`U+E0100–U+E01EF`	Variation Selectors Supp.		

Table 7. Unicode blocks in predefined character range 7.

`U+1100–U+11FF`	Hangul Jamo	`U+2F00–U+2FDF`	Kangxi Radicals
`U+2FF0–U+2FFF`	Ideographic Description Characters	`U+3100–U+312F`	Bopomofo
`U+3130–U+318F`	Hangul Compatibility Jamo	`U+31A0–U+31BF`	Bopomofo Extended
`U+31C0–U+31EF`	CJK Strokes	`U+A000–U+A48F`	Yi Syllables
`U+A490–U+A4CF`	Yi Radicals	`U+A960–U+A97F`	Hangul Jamo Extended-A
`U+AC00–U+D7AF`	Hangul Syllables	`U+D7B0–U+D7FF`	Hangul Jamo Extended-B

```
  "FB00-"FE0F, "FE20-"FE2F, "FE70-"FEFF, "10000-"1AFFF, "1B170-"1F0FF,
  "1F300-"1FFFF, ... (and characters in U+2000–U+206F which are not in range 9)
} % non-Japanese
```

Range 5^A Surrogates and Supplementary Private Use Areas.

Range 6^J Characters used in Japanese. The block list is indicated in Table 6.

Range 7^J Characters used in CJK languages, but not included in Adobe-Japan1-7. The block list is indicated in Table 7.

■Notes on `U+0080–U+00FF` You should treat characters in texttttU+0080–U+00FF as **ALchar**, when you use traditional 8-bit fonts, such as the marvosym package.

For example, `\Frowny` which is provided by the marvosym package has the same codepoint as § (U+00A7). Hence, as previous versions of LuaTeX-ja, if these characters are treated as JAchars, then `\Frowny` produces “ § ” (in a Japanese font).

To avoid such situations, the default setting of LuaTEX-ja is changed in version 20150906.0 so that all characters U+0080–U+00FF are treated as **ALchar**.

If you want to output a character as **ALchar** and **JAchar** regardless the range setting, you can use \ltjalchar and \ltjjachar respectively, as the following example.

```
\gtfamily\large % default, ALchar, JAchar
¶, \ltjalchar`¶, \ltjjachar`¶\\   % default: ALchar
α, \ltjalchar`α, \ltjjachar`α   % default: JAchar
```

¶, ¶, ¶
α, α, α

4.2 kanjiskip and xkanjiskip

JAglue is divided into the following three categories:

- Glues/kerns specified in JFM. If \inhibitglue is issued around a **JAchar**, this glue will not be inserted at the place.
- The default glue which inserted between two **JAchar**s (kanjiskip).
- The default glue which inserted between a **JAchar** and an **ALchar** (xkanjiskip).

The value (a skip) of kanjiskip or xkanjiskip can be changed as the following. Note that only their values *at the end of a paragraph or a hbox are adopted in the whole paragraph or the whole hbox.*

```
\ltjsetparameter{kanjiskip={0pt plus 0.4pt minus 0.4pt},
                 xkanjiskip={0.25\zw plus 1pt minus 1pt}}
```

Here \zw is a internal dimension which stores fullwidth of the current Japanese font. This \zw can be used as the unit zw in pTEX.

The value of these parameter can be get by \ltjgetparameter. Note that the result by \ltjgetparameter is *not* the internal quantities, but *a string* (hence \the cannot be prefixed).

```
kanjiskip: \ltjgetparameter{kanjiskip},\\
xkanjiskip: \ltjgetparameter{xkanjiskip}
```

kanjiskip: 0.0pt plus 0.4pt minus 0.5pt,
xkanjiskip: 2.40553pt plus 1.0pt minus 1.0pt

It may occur that JFM contains the data of "ideal width of kanjiskip" and/or "ideal width of xkanjiskip". To use these data from JFM, set the value of kanjiskip or xkanjiskip to \maxdimen (these "ideal width" cannot be retrived by \ltjgetparameter).

4.3 Insertion setting of xkanjiskip

It is not desirable that xkanjiskip is inserted into every boundary between **JAchar**s and **ALchar**s. For example, xkanjiskip should not be inserted after opening parenthesis (*e.g.*, compare "(あ" and "(あ"). LuaTEX-ja can control whether xkanjiskip can be inserted before/after a character, by changing jaxspmode for **JAchar**s and alxspmode parameters **ALchar**s respectively.

```
1 \ltjsetparameter{jaxspmode={`あ,preonly},
     alxspmode={`\!,postonly}}
2 pあq い!う
```

p あq い! う

The second argument preonly means that the insertion of xkanjiskip is allowed before this character, but not after. the other possible values are postonly, allow, and inhibit.

jaxspmode and alxspmode use a same table to store the parameters on the current version. Therefore, line 1 in the code above can be rewritten as follows:

```
\ltjsetparameter{alxspmode={`あ,preonly}, jaxspmode={`\!,postonly}}
```

One can use also numbers to specify these two parameters (see Subsection 9.1).

If you want to enable/disable all insertions of kanjiskip and xkanjiskip, set autospacing and autoxspacing parameters to true/false, respectively.

4.4 Shifting the baseline

To make a match between a Japanese font and an alphabetic font, sometimes shifting of the baseline of one of the pair is needed. In pTEX, this is achieved by setting `\ybaselineshift` (or `\tbaselineshift`) to a non-zero length (the baseline of **ALchar** is shifted below). However, for documents whose main language is not Japanese, it is good to shift the baseline of Japanese fonts, but not that of alphabetic fonts. Because of this, LuaTEX-ja can independently set the shifting amount of the baseline of alphabetic fonts and that of Japanese fonts.

	Horizontal writing (*yoko* direction) etc.	**Vertical writing(*tate* direction)**
Alphabetic fonts	yalbaselineshift parameter	talbaselineshift parameter
Japanese fonts	yjabaselineshift parameter	tjabaselineshift parameter

Here the horizontal line in the below example is the baseline of a line.

```
\vrule width 150pt height 0.2pt depth 0.2pt \hskip-120pt
\ltjsetparameter{yjabaselineshift=0pt, yalbaselineshift=0pt}abcあいう
\ltjsetparameter{yjabaselineshift=5pt, yalbaselineshift=2pt}abcあいう
```

abc あいう abc あいう

There is an interesting side-effect: characters in different size can be vertically aligned center in a line, by setting two parameters appropriately. The following is an example (beware the value is not well tuned):

```
\vrule width 150pt height4.417pt depth-4.217pt%
\kern-150pt
\large xyz漢字
{\scriptsize
  \ltjsetparameter{yjabaselineshift=-1.757pt,
   yalbaselineshift=-1.757pt}
  漢字xyzあいう
}あいうabc
```

xyz 漢字漢字 xyz あいうあいう abc

Note that setting positive yalbaselineshift or talbaselineshift parameters does not increase the depth of one-letter *syllable p* of **Alchar**, if its left-protrusion (`\lpcode`) and right-protrusion (`\rpcode`) are both non-zero. This is because

- These two parameters are implemented by setting `yoffset` field of a glyph node, and this does not increase the depth of the glyph.
- To cope with the above situation, LuaTEX-ja automatically supplies a rule in every syllable.
- However, we cannot use this "supplying a rule" method if a syllable comprises just one letter whose `\lpcode` and `\rpcode` are both non-zero.

This problem does not apply for yjabaselineshift nor tjabaselineshift, becuse a *JAchar* is encapsulated by a horizontal box if needed.

4.5 *kinsoku* parameters and OpenType features

Among parameters which related to Japanese word-wrapping process (*kinsoku shori*),

jaxspmode, alxspmode, prebreakpenalty, postbreakpenalty and kcatcode

are stored by each character codes.

OpenType font features are ignored in these parameters. For example, a fullwidth katakana "ア" on line 10 in the below input is replaced to its halfwidth variant "ｱ", by `hwid` feature. However, the penalty inserted after it is 10 which is the postbreakpenalty of "ア", not 20.

```
\ltjsetparameter{postbreakpenalty={`ア, 10}}
\ltjsetparameter{postbreakpenalty={`ｱ, 20}}

\newcommand\showpostpena[1]{%
  \leavevmode\setbox0=\hbox{#1\hbox{}}%
  \unhbox0\setbox0=\lastbox\the\lastpenalty}

\showpostpena{ア},
\showpostpena{ｱ},
{\addjfontfeatures{CharacterWidth=Half}\showpostpena{ア}}
```

ア 10, ｱ 20, ア 10

Part II

Reference

5 \catcode in LuaTeX-ja

5.1 Preliminaries: \kcatcode in pTeX and upTeX

In pTeX and upTeX, the value of `\kcatcode` determines whether a Japanese character can be used in a control word. For the detail, see Table 8.

`\kcatcode` can be set by a row of JIS X 0208 in pTeX, and generally by a Unicode block[6] in upTeX. So characters which can be used in a control word slightly differ between pTeX and upTeX.

5.2 Case of LuaTeX-ja

The role of `\kcatcode` in pTeX and upTeX can be divided into the following four kinds, and LuaTeX-ja can control these four kinds separately:

- *Distinction between **JAchar** or **ALchar*** is controlled by the character range, see Subsection 4.1.
- *Whether the character can be used in a control word* is controlled by setting `\catcode` to 11 (enabled) or 12 (disabled), as usual.
- *Whether jcharwidowpenalty can be inserted before the character* is controlled by the lowermost bit of the kcatcode parameter.
- *Linebreak after a **JAchar*** does not produce a space.

Default setting of `\catcode` of Unicode characters with LuaTeX is slightly inconvenient for pTeX users to shifting to LuaTeX-ja, because several fullwidth characters which can be used in a control word with pTeX, such as “１” (FULLWIDTH DIGIT ONE), cannnot be used in a control word with LuaTeX. Hence, LuaTeX-ja changes the `\catcode` of some characters—whose line breaking class is “ID” (Ideographic) in UAX #14—, to allow these characters in the control word.

5.3 Non-kanji characters in a control word

Because the engine differ, so non-kanji JIS X 0208 characters which can be used in a control word differ in pTeX, in upTeX, and in LuaTeX-ja. Table 9 shows the difference. Except for three characters ・, ゛, and ‘゜, LuaTeX-ja admits more characters in a control word than upTeX.

Difference becomes larger, if we consider non-kanji JIS X 0213 characters. For the detail, see `https://github.com/h-kitagawa/kct`.

6 Directions

LuaTeX supports four Ω-style directions: `TLT`, `TRT`, `RTT` and `LTL`. However, neither directions are not well-suited for typesetting Japanese vertically, hence we implemented vertical writing by rotating `TLT`-box by 90 degrees.

LuaTeX-ja supports four directions, as shown in Table 10. The second column (*yoko* direction) is just horizontal writing, and the third column (*tate* direction) is vertical writing. The fourth column (*dtou* direction) is actually a hidden feature of pTeX. We implemented this for debugging purpose. The fifth column (*utod* direction) corresponds the “`tate (math) direction`” of pTeX.

Directions can be changed by `\yoko`, `\tate`, `\dtou`, `\utod`, only when the current list is null. These commands cannot be executed in unrestricted horizontal modes, nor math modes. The direction of a math formula is changed to *utod*, when the direction outside the math formula is *tate* (vertical writing).

[6] upTeX divides `U+FF00–U+FFEF` (Halfwidth and Fullwidth Forms) into three subblocks, and `\kcatcode` can be set by a subblock.

Table 8. `\kcatcode` in upTEX

`\kcatcode`	meaning	control word	widow penalty	linebreak
15	non-cjk	(treated as usual LATEX)		
16	kanji	Y	Y	ignored
17	kana	Y	Y	ignored
18	other	N	N	ignored
19	hangul	Y	Y	space

Table 9. Difference of the set of non-kanji JIS X 0208 characters which can be used in a control word

	row	col.	pTEX	upTEX	LuaTEX-ja
・ (U+30FB)	1	6	N	Y	N
゛ (U+309B)	1	11	N	Y	N
゜ (U+309C)	1	12	N	Y	N
｀ (U+FF40)	1	14	N	N	Y
＾ (U+FF3E)	1	16	N	N	Y
￣ (U+FFE3)	1	17	N	N	Y
＿ (U+FF3F)	1	18	N	N	Y
ヽ (U+30FD)	1	19	N	Y	Y
ヾ (U+30FE)	1	20	N	Y	Y
ゝ (U+309D)	1	21	N	Y	Y
ゞ (U+309E)	1	22	N	Y	Y
〃 (U+3003)	1	23	N	N	Y
仝 (U+4EDD)	1	24	N	Y	Y
々 (U+3005)	1	25	N	N	Y
〆 (U+3006)	1	26	N	N	Y
〇 (U+3007)	1	27	N	N	Y
ー (U+30FC)	1	28	N	Y	Y

	row	col.	pTEX	upTEX	LuaTEX-ja
／ (U+FF0F)	1	31	N	N	Y
＼ (U+FF3C)	1	32	N	N	Y
～ (U+FF5E)	1	33	N	N	Y
｜ (U+FF5C)	1	35	N	N	Y
＋ (U+FF0B)	1	60	N	N	Y
＝ (U+FF1D)	1	65	N	N	Y
＜ (U+FF1C)	1	67	N	N	Y
＞ (U+FF1E)	1	68	N	N	Y
＃ (U+FF03)	1	84	N	N	Y
＆ (U+FF06)	1	85	N	N	Y
＊ (U+FF0A)	1	86	N	N	Y
＠ (U+FF20)	1	87	N	N	Y
〒 (U+3012)	2	9	N	N	Y
〓 (U+3013)	2	14	N	N	Y
￢ (U+FFE2)	2	44	N	N	Y
Å (U+212B)	2	82	N	N	Y
Greek letters (row 6)			Y	N	Y
Cyrillic letters (row 7)			N	N	Y

6.1 Boxes in different direction

As in pTEX, one can use boxes of different direction in one document. The below is an example.

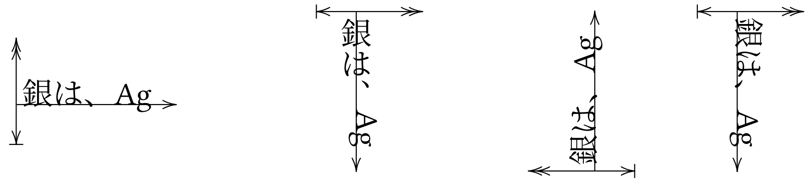
```
ここは横組%      yoko
\hbox{\tate %    tate
  \hbox{縦組}%   tate
  の中に
  \hbox{\yoko 横組の内容}% yoko
  を挿入する
}
また横組に戻る% yoko
```

ここは横組 縦組の中に横組の内容を挿入する また横組に戻る

Table 11 shows how a box is arranged when the direction inside the box and that outside the box differ.

■ **`\wd` and direction** In pTEX, `\wd`, `\ht`, `\dp` means the dimensions of a box register *with respact to the current direction*. This means that the value of `\wd0` etc. might differ when the current direction is different, even if `\box0` stores the same box. However, this no longer applies in LuaTEX-ja.

Table 10. Directions supported by LuaTeX-ja

	horizontal (*yoko* direction)	vertical (*tate* direction)	*dtou* direction	*utod* direction
Commands	`\yoko`	`\tate`	`\dtou`	`\utod`
Beginning of the page	Top	Right	Left	Right
Beginning of the line	Left	Top	Bottom	Top
Used Japanese font	horizontal (`\jfont`)	vertical (`\tfont`)	horizontal (90° rotated)	
Example				
(Notation used in Ω)	TLT	RTR, RTT	LBL	RTR

```
\setbox0=\hbox to 20pt{foo}
\the\wd0,~\hbox{\tate\vrule\the\wd0}
\wd0=100pt
\the\wd0,~\hbox{\tate \the\wd0}
```

20.0pt, 20.0pt 100.0pt, 100.0pt

To access box dimensions *with respect to current direction*, one have to use the following commands instead of `\wd` wtc.

`\ltjgetwd`⟨*num*⟩, `\ltjgetht`⟨*num*⟩, `\ltjgetdp`⟨*num*⟩

These commands return *an internal dimension* of `\box`⟨*num*⟩ with respect to the current direction. One can use these in `\dimexpr` primitive, as the followings.

```
\dimexpr 2\ltjgetwd42-3pt\relax, \the\ltjgetwd1701
```

The following is an example.

```
\parindent0pt
\setbox32767=\hbox{\yoko よこぐみ}
\fboxsep=0mm\fbox{\copy32767}
\vbox{\hsize=20mm
\yoko YOKO \the\ltjgetwd32767, \\
  \the\ltjgetht32767, \\ \the\ltjgetdp32767.}
\vbox{\hsize=20mm\raggedleft
\tate TATE \the\ltjgetwd32767, \\
  \the\ltjgetht32767, \\ \the\ltjgetdp32767.}
\vbox{\hsize=20mm\raggedleft
\dtou DTOU \the\ltjgetwd32767, \\
  \the\ltjgetht32767, \\ \the\ltjgetdp32767.}
```

よこぐみ

YOKO 38.48877pt, 8.46753pt, 1.15466pt.

TATE 9.6222pt, 19.24438pt, 19.24438pt.

DTOU 9.6222pt, 38.48877pt, 0.0pt.

`\ltjsetwd`⟨*num*⟩=⟨*dimen*⟩, `\ltjsetht`⟨*num*⟩=⟨*dimen*⟩, `\ltjsetdp`⟨*num*⟩=⟨*dimen*⟩

These commands set the dimension of `\box`⟨*num*⟩. One does not need to group the argument ⟨*num*⟩; four calls of `\ltjsetwd` below have the same meaning.

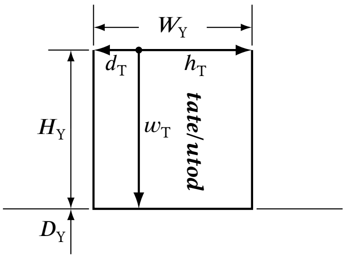
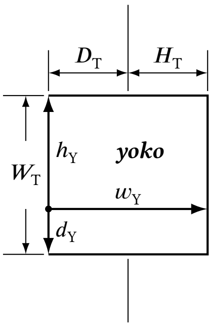
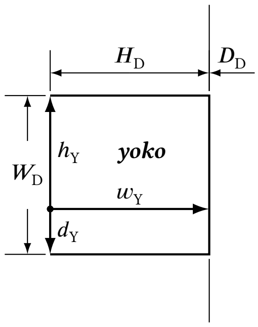
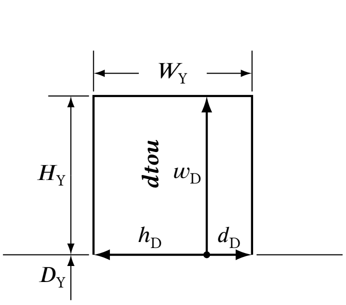
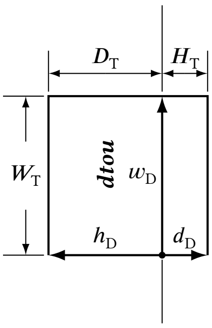
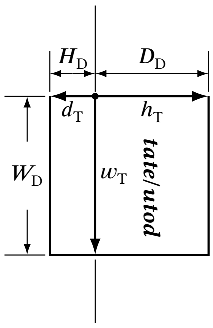
```
\ltjsetwd42 20pt,  \ltjsetwd42=20pt,  \ltjsetwd=42 20pt,  \ltjsetwd=42=20pt
```

6.2 Getting current direction

The direction parameter returns the current direction, and the boxdir parameter (with the argument ⟨*num*⟩) returns the direction of a box register `\box`⟨*num*⟩. The returned value of these parameters are a *string*:

Direction	*yoko*	*tate*	*dtou*	*utod*	(empty)
Returned value	4	3	1	11	0

Table 11. Boxes in different direction

typeset in *yoko* direction	**typeset in *tate* or *utod* direction**	**typeset in *dtou* direction**
		
$W_\mathrm{Y} = h_\mathrm{T} + d_\mathrm{T}$, $H_\mathrm{Y} = w_\mathrm{T}$, $D_\mathrm{Y} = 0\,\mathrm{pt}$	$W_\mathrm{T} = h_\mathrm{Y} + d_\mathrm{Y}$, $H_\mathrm{T} = w_\mathrm{Y}/2$, $D_\mathrm{T} = w_\mathrm{Y}/2$	$W_\mathrm{D} = h_\mathrm{Y} + d_\mathrm{Y}$, $H_\mathrm{D} = w_\mathrm{Y}$, $D_\mathrm{D} = 0\,\mathrm{pt}$
		
$W_\mathrm{Y} = h_\mathrm{D} + d_\mathrm{D}$, $H_\mathrm{Y} = w_\mathrm{D}$, $D_\mathrm{Y} = 0\,\mathrm{pt}$	$W_\mathrm{T} = h_\mathrm{D} + d_\mathrm{D}$, $H_\mathrm{T} = d_\mathrm{D}$, $D_\mathrm{T} = h_\mathrm{D}$	$W_\mathrm{D} = w_\mathrm{T}$, $H_\mathrm{D} = d_\mathrm{T}$, $D_\mathrm{D} = h_\mathrm{T}$

```
\leavevmode\def\DIR{\ltjgetparameter{direction}}
\hbox{\yoko \DIR}, \hbox{\tate\DIR},
\hbox{\dtou\DIR}, \hbox{\utod\DIR},
\hbox{\tate$\hbox{tate math: \DIR}$}

\setbox2=\hbox{\tate}\ltjgetparameter{boxdir}{2}
```

4, 3, 1, 11, tate math: 11
3

7 Redefined primitives by LuaTeX-ja

The following primitives are redefined by LuaTeX-ja (using `\protected\def`), for supporting Japanese typesetting and multiple directions:

`\/`

`\unhbox`⟨*num*⟩, `\unvbox`⟨*num*⟩, `\unhcopy`⟨*num*⟩, `\unvcopy`⟨*num*⟩

`\vadjust{`⟨*material*⟩`}`

`\insert`⟨*number*⟩`{`⟨*material*⟩`}`

`\lastbox`

`\raise`⟨*dimen*⟩⟨*box*⟩, `\lower`⟨*dimen*⟩⟨*box*⟩, `\moveleft`⟨*dimen*⟩⟨*box*⟩, `\moveright`⟨*dimen*⟩⟨*box*⟩,

`\split`⟨*number*⟩`to`⟨*dimen*⟩, `\vcenter{`⟨*material*⟩`}`

```
\makeatletter\scriptsize\ttfamily
\meaning\vadjust         \\ % current
\meaning\ltj@@vadjust    \\ % LuaTeX-ja
\meaning\ltj@@orig@vadjust % original
```

```
luacall 45
luacall 45
\vadjust
```

Figure 1. Redefining `\vadjust` primitive by LuaTeX-ja

```
\makeatletter
\def\ltj@stop@overwrite@primitive{\insert\vadjust\/\unhbox\vcenter\fontseries}
\makeatother
  %% Keep the meaning of \insert, \vadjust, \/, \unhbox and \vcenter.
  %% \fontseries will still be redefined by \LuaTeX-ja, because it is not primitive.
\usepackage{luatexja}
...
\usepackage{breqn}
...
\makeatletter
\ltj@overwrite@primitive\expandafter{\insert\vadjust\/\unhbox\vcenter}
\makeatother
  %% Redefine \insert, \vadjust, \/, \unhbox and \vcenter.
```

Figure 2. `\ltj@stop@overwrite@primitive` and `\ltj@overwrite@primitive`

On each primitive `\`⟨*primitive*⟩ in the list above, its meaning just before loading LuaTeX-ja is backed up into `\ltj@@orig@`⟨*primitive*⟩, and the meaning after redefinition by LuaTeX-ja is stored in `\ltj@@`⟨*primitive*⟩. For example, Figure 1 shows the situation of `\vadjust` primitive.

7.1 Suppressing redefinitions

Sometimes redefining primitives by LuaTeX-ja causes a problem. For example, the breqn package (v0.98k) assumes that `\vadjust` and `\insert` have their primitive meanings. So, this package cannot by loaded after LuaTeX-ja by default.

LuaTeX-ja version 20210517.0 has features for that problem. Namely:

- Primitives which is listed in `\ltj@stop@overwrite@primitive` are retain their meanings at just before loading LuaTeX-ja.
- After loading LuaTeX-ja, one can specify primitives to `\ltj@overwrite@primitive`, to redefine them by LuaTeX-ja.

See Figure 2 for an example.

8 Font Metric and Japanese Font

8.1 `\jfont`

To load a font as a Japanese font (for horizontal direction), you must use the `\jfont` instead of `\font`, while `\jfont` admits the same syntax used in `\font`. LuaTeX-ja automatically loads luaotfload package, so TrueType/OpenType fonts with features can be used for Japanese fonts:

```
\jfont\tradmc={IPAexMincho:script=latn;%
  +trad;-kern;jfm=ujis} at 14pt
\tradmc 当／体／医／区
```

當／體／醫／區

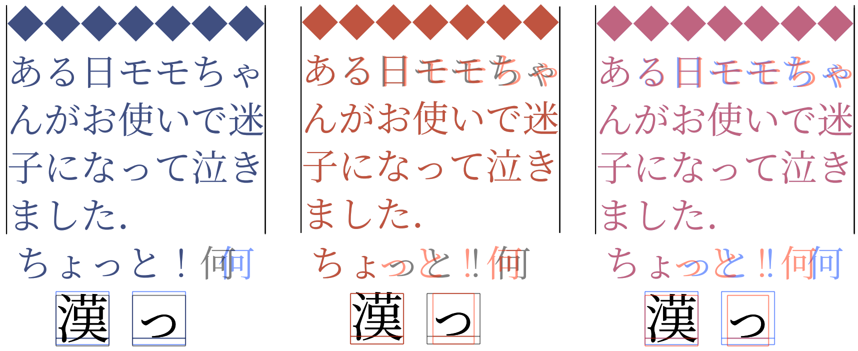
It is required to specify a (horizontal) *JFM* in at each calling of `\jfont`. A JFM is a Lua script which contains measurements of characters and glues/kerns that are automatically inserted for Japanese typesetting. The structure of JFM will be described in the next subsection.

```
\ltjsetparameter{differentjfm=both}
\jfont\F=HaranoAjiMincho-Regular:jfm=ujis
\jfont\G=HaranoAjiGothic-Medium:jfm=ujis
\jfont\H=HaranoAjiGothic-Medium:jfm=ujis;jfmvar=hoge
\F ）{\G 【】}（ % halfwidth space
   ）{\H 『』}（ % fullwidth space

ほげ，{\G 「ほげ」}（ほげ）\par
ほげ，{\H 「ほげ」}（ほげ）% pTeX-like

\ltjsetparameter{differentjfm=paverage}
```

）【】（）『』（
ほげ，**「ほげ」**（ほげ）
ほげ，　**「ほげ」**　（ほげ）

Figure 3. Example of `jfmvar` key

Table 12. Differences between horizontal JFMs shipped with LuaTEX-ja



(Blue: `jfm-ujis.lua`, Black: `jfm-jis.lua`, Red: `jfm-min.lua`)

Note that the defined control sequence (`\tradmc` in the example above) using `\jfont` is not a *font_def* token, but a macro. Hence the input like `\fontname\tradmc` causes a error. We denote control sequences which are defined in `\jfont` by ⟨*jfont_cs*⟩.

■**Specifying JFM** The general scheme for specifying a JFM is the following:

`\jfont`⟨*jfont_cs*⟩`=...;jfm=`⟨*JFM name*⟩`[/{`⟨*JFM features*⟩`}];...;[jfmvar=`⟨*identifier*⟩`];...`

⟨*JFM name*⟩ The name of a (horizontal) JFM. LuaTEX-ja searches and loads `jfm-`⟨*JFM name*⟩`.lua` [7].

⟨*JFM features*⟩ An optional comma-separated list of JFM options. Enclosing braces (`{}`) are optional, but this does not escape any characters. The contents of this list can be accessed by a table `luatexja.jfont.jfm_feature` from a JFM, at its loading. See Figure 4 for an example.

Note that any JFM files which is shipped with LuaTEX-ja does not use this feature.

⟨*identifier*⟩ An optional string.

LuaTeX-ja "does not distinguish" two Japanese fonts which uses same JFM and are the same size. Here "uses same JFM" means that all of ⟨*JFM name*⟩, ⟨*JFM features*⟩ and ⟨*identifier*⟩ of two fonts agree.

For example, The first "）" and "【" in Figure 3 are typeset in different real fonts. However, because they use the same JFMs and their size are same, LuaTEX-ja inserts penalties, glues and kerns as if these two character are typeset in a same font. Namely, the glue between these characters is halfwidth, as in "）【". However, this does not applies with `\F` and `\H` in Figure 3, because their ⟨*identifier*⟩ are different.

■**Horizontal JFMs** The following horizontal JFMs are shipped with LuaTEX-ja.

[7] When LuaTEX-ja (version 20230409 or later) is used under LATEX 2_ε, LuaTEX-ja searches a JFM also in directories which are specified in `\input@path`.

\A: (nil)
\B: [ps] = false, [kern] = “0.5”, [kana] = true,
\C: [down] = “0.2”, [kern] = “0.5”,
\D: [down] = “0.2”, [kern] = “0.5”,

	\A	\B	\C	\D
\A	あ漢イ字	あ漢 イ 字	あ漢 イ 字	あ漢 イ 字
\B	あ 漢 イ字	あ 漢イ 字	あ 漢 イ 字	あ 漢 イ 字
\C	あ 漢 イ字	あ 漢 イ 字	あ 漢イ 字	あ 漢イ 字
\D	あ 漢 イ字	あ 漢 イ 字	あ 漢イ 字	あ 漢イ 字

```
\small\ltjsetparameter{differentjfm=both}\tabcolsep=.5\zw
  % \printjfmfeat is defined in the source of this document
\jfont\A=HaranoAjiMincho-Regular:jfm=testf                    at 9pt \printjfmfeat\A
\jfont\B=HaranoAjiMincho-Bold:jfm=testf/kern=0.5,-ps,+kana    at 9pt \printjfmfeat\B
\jfont\C=HaranoAjiGothic-Regular:jfm=testf/kern=0.5,down=0.2 at 9pt \printjfmfeat\C
\jfont\D=HaranoAjiGothic-Bold:jfm=testf/down=0.2,kern=0.5     at 9pt \printjfmfeat\D
\def\TEST#1{\string#1&{#1あ漢}{\A イ字}&{#1あ漢}{\B イ字}&{#1あ漢}{\C イ字}&{#1あ漢}{\D イ字}}
\vspace{-4\baselineskip}\hfill\ttfamily
\begin{tabular}{lllll}
  &\string\A&\string\B&\string\C&\string\D\\\TEST\A\\\TEST\B\\\TEST\C\\\TEST\D\\
\end{tabular}
  % No space between ``漢'' and ``イ'' iff two Japanese fonts uses same JFM
\ltjsetparameter{differentjfm=paverage}
```

Figure 4. Example of JFM features

jfm-ujis.lua A standard horizontal JFM of LuaTeX-ja. This file is based on `upnmlminr-h.tfm`, a metric for UTF/OTF package that is used in upTeX. When you are going to use the luatexja-otf package, you should use this JFM.

jfm-jis.lua A counterpart for `jis.tfm`, “JIS font metric” which is widely used in pTeX. A major difference between `jfm-ujis.lua` and this `jfm-jis.lua` is that most characters under `jfm-ujis.lua` are square-shaped, while that under `jfm-jis.lua` are horizontal rectangles.

jfm-min.lua A counterpart for `min10.tfm`, which is one of the default Japanese font metric shipped with pTeX.

jfm-prop.lua A JFM for proportional typesetting. This JFM doesn’t have any information of character dimension (width, height, depth), nor glues/kerns information.

jfm-propw.lua Another JFM for proportional typesetting. In contrast to `jfm-prop.lua`, this JFM has informations of character height and depth.

See Table 12 for the difference among `jfm-ujis.lua`, `jfm-jis.lua`, `jfm-min.lua`.

■Using kerning information in a font Some fonts have information for inter-glyph spacing. LuaTeX-ja 20140324.0 or later treats kerning spaces like an italic correction; any glue and/or kern from the JFM and a kerning space from the font can coexist. See Figure 5 for detail.

At version 20220411.0, defaults Japanese fonts which are defined at the loading of LuaTeX-ja, ltjclasses, and ltjsclasses do not insert font-derived kerning spaces by default. This is because standard JFMs do not expect font-derived kerning spaces between Japanese characters.

Also note that in `\setmainjfont` etc. which are provided by luatexja-fontspec package, kerning option is set *off* (`Kerning=Off`) by default. This means the following two lines have the same meaning:

```
\setmainjfont{HaranoAjiMincho-Regular}
\setmainjfont[Kerning=Off]{HaranoAjiMincho-Regular}
```

■`extend` and `slant` The following setting can be specified as OpenType font features:

`extend=`⟨*extend*⟩ expand the font horizontally by ⟨*extend*⟩.

`slant=`⟨*slant*⟩ slant the font.

|ダイナミックダイクマ|
|ダイナミックダイクマ|
|ダイナミックダイクマ|
|ダイナミックダイクマ|

|ダ イ ナ ミ ッ ク ダ イ ク マ|
|ダ イ ナ ミ ッ ク ダ イ ク マ|
|ダイナミックダイクマ|
|ダ イ ナ ミ ッ ク ダ イ ク マ|

```
\newcommand\test{\vrule ダイナミックダイクマ\vrule\\}
\jfont\KMFW = HaranoAjiMincho-Regular:jfm=prop;-kern at 17.28pt
\jfont\KMFK = HaranoAjiMincho-Regular:jfm=prop at 17.28pt % kern is activated
\jfont\KMPW = HaranoAjiMincho-Regular:jfm=prop;script=dflt;+palt;-kern at 17.28pt
\jfont\KMPK = HaranoAjiMincho-Regular:jfm=prop;script=dflt;+palt;+kern at 17.28pt
\begin{multicols}{2}
\ltjsetparameter{kanjiskip=0pt}
{\KMFW\test \KMFK\test \KMPW\test \KMPK\test}

\ltjsetparameter{kanjiskip=3pt}
{\KMFW\test \KMFK\test \KMPW\test \KMPK\test}
\end{multicols}
```

Figure 5. Kerning information and kanjiskip

```
\leavevmode
\ltjsetparameter{kanjiskip=0pt plus 3\zw}
\vrule\hbox to 15\zw{あ「い」う，えお}\vrule\\
\jfont\G=HaranoAjiMincho-Regular%
  :jfm=ujis;-ltjksp at \zw
\G\leavevmode%
\vrule\hbox to 15\zw{あ「い」う，えお}\vrule
```

|あ　　「い」　　う，　　え　　お|
|あ「い」う，え　　　　　　　お|

Figure 6. `ltjksp` “feature”

Note that LuaTEX-ja doesn’t adjust JFMs by these `extend` and `slant` settings; one have to write new JFMs on purpose. For example, the following example uses the standard JFM `jfm-ujis.lua`, hence the letterspacing and the width of italic corrections are not correct:

```
\jfont\E=HaranoAjiMincho-Regular:extend=1.5;jfm=ujis;-kern
\jfont\S=HaranoAjiMincho-Regular:slant=1;jfm=ujis;-kern
\E あいうえお \S あいう\/ABC
```

あいうえおあいうABC

■`ltjksp` “feature” `kanjiskip_natural`, `kanjiskip_stretch`, `kanjiskip_shrink` keys (Page ??) makes tha LuaTEX-ja inserts not only a glue which is specified by a JFM, and also the natural width/stretch part/shrink part of kanjiskip. This functionality is disabled by `-ltjksp` specification, as shown in Figure 6.

■`ltjpci` “feature” By default, The luaotfload package (since v3.19) normalizes Unicode sequences to NFC. However, this normalization converts CJK compatibility ideographs to their canonical equivalents, such as “神” (`U+FA19`) to “神”. One can use variation selectors, but old fonts does not support them.

So, *LuaTEX-ja now protects CJK compatibility ideographs from processing by the luaotfload package by defualt.* This functionality is disabled by `-ltjpci` specification, as shown in Figure 7.

8.2 `\tfont`

`\tfont` loads a font as a Japanese font for vertical direction. This command admits the same syntax as in `\font` and `\jfont`. A font defined by `\tfont` differs the following points from that by `\jfont`:

```
\def\TEST{\leavevmode\char"FA10\char"FA12\char"FA15
  \char"FA19.か\char"3099.は\char"309A.\par}
\jfont\A=HaranoAjiMincho-Regular:jfm=ujis; at 15pt
\A\TEST % default
\jfont\G=HaranoAjiMincho-Regular:jfm=ujis;-ltjpci at 15pt
\G\TEST % ltjpci off
\jfont\H=HaranoAjiMincho-Regular:jfm=ujis;-normalize at 15pt
\H\TEST % normalization off
```

塚晴凞神.が.ぱ.
塚晴凞神.が.ぱ.
塚晴凞神.か゛.は゜.

Figure 7. `ltjpci` “feature”

```
\jfont\X=[HaranoAjiMincho-Regular.otf]:jfm=ujis
\tfont\U=[HaranoAjiMincho-Regular.otf]:jfm=ujisv
\tfont\V=[HaranoAjiMincho-Regular.otf]:jfm=ujisv;jpotf
\def\TEST#1#2{\leavevmode\hbox{#1#2\string#2 “引用，と句読点．”}}
\ttfamily\centering\TEST\yoko\X \quad \TEST\tate\U \quad \TEST\tate\V
```

\X “引用，と句読点．” \U “引用，と句読点．” \V 〝引用、と句読点。〟

Figure 8. `jpotf` “feature”

- OpenType Feature `vrt2`[8] is automatically activated, unless `vert` and//or `vrt2` features are explicitly activated or deactivated (as the second line in the example below).

```
\tfont\S=HaranoAjiMincho-Regular:jfm=ujisv % vrt2 is automatically activated
\tfont\T=HaranoAjiMincho-Regular:jfm=ujisv;-vert % vert and vrt2 are not activated
\tfont\U=file:ipaexm.ttf:jfm=ujisv
    % vert is automatically activated, since this font does not have vrt2
```

- Sometimes `vert` and/or `vrt2` are not activated while one specified activation of these feature. This is because the font does not define these features in current combination of script tag and language system identifier.

  In this situation, LuaTeX-ja performs all replacements which is defined in `vert` feature for *some* scripts for *some* languages.

- `\tfont` uses a vertical JFM instead of a horizontal JFM. LuaTeX-ja ships following vertical JFMs:

  `jfm-ujisv.lua` A standard vertical JFM in LuaTeX-ja. This JFM is based on `upnmlminr-v.tfm`, a metric for UTF/OTF package that is used in upTeX.

  `jfm-tmin.lua` A counterpart for `tmin10.tfm`, which is one of the default Japanese font metric shipped with pTeX.

- If `vert` and/or `vrt2` features are activated, one can specify `jpotf` to additional substitutions. By default, it substitutes ideographic comma/period for fullwidth comma/period, and double prime quotation marks for double quotation marks (See Figure 8). One can customize substitutions by lua function `luatexja.jfont.register_vert_replace` (see Japanese version of this manual).

[8] If the font does not define the `vrt2` feature, `vert` is used instead.

8.3 Default Japanese fonts and JFMs

If following commands are defined at loading LuaTeX-ja package, these change default Japanese fonts and JFMs for them:

`\ltj@stdmcfont` The default Japanese font for the mincho family.

`\ltj@stdgtfont` The default Japanese font for the gothic family.

`\ltj@stdyokojfm` The default JFM for horizontal direction.

`\ltj@stdtatejfm` The default JFM for vertical direction.

For example,

```
\def\ltj@stdmcfont{IPAMincho}
\def\ltj@stdgtfont{IPAGothic}
```

makes that IPA Mincho and IPA Gothic will be used as default Japanese fonts, instead of Harano Aji fonts.

This feature is intended for classes which use special JFMs[9]. It is recommended to use `\luatexja-preset` or `\luatexja-fontspec` package to select standard fonts in ordinary LaTeX sources.

For compatibility with earlier versions, LuaTeX-ja reads `luatexja.cfg` automatically if it is found by LuaTeX. One should not overuse this `luatexja.cfg`; it will overwrite the definition of `\ltj@stdmcfont` and others.

8.4 Prefix psft

Besides "`file`" and "`name`" prefixes which are introduced in the luaotfload package, LuaTeX-ja adds "`psft`" prefix in `\jfont` (and `\font`), to specify a "name-only" Japanese font which will not be embedded to PDF. Note that these non-embedded fonts under current LuaTeX has Identity-H encoding, and this violates the standard ISO32000-1:2008 ([10]).

OpenType font features, such as "+jp90", have no meaning in name-only fonts using "psft" prefix, because we can't expect what fonts are actually used by the PDF reader. Note that `extend` and `slant` settings (see above) are supported with `psft` prefix, because they are only simple linear transformations.

■**cid key** The default font defined by using `psft` prefix is for Japanese typesetting; it is Adobe-Japan1-7 CID-keyed font. One can specify `cid` key to use other CID-keyed non-embedded fonts for Chinese or Korean typesetting.

```
\jfont\testJ={psft:Ryumin-Light:cid=Adobe-Japan1-7;jfm=jis}    % Japanese
\jfont\testD={psft:Ryumin-Light:jfm=jis}                       % default: Adobe-Japan1-7
\jfont\testC={psft:AdobeMingStd-Light:cid=Adobe-CNS1-7;jfm=jis}% Traditional Chinese
\jfont\testG={psft:SimSun:cid=Adobe-GB1-6;jfm=jis}             % Simplified Chinese
\jfont\testK={psft:Batang:cid=Adobe-Korea1-2;jfm=jis}          % Korean
\jfont\testKR={psft:SourceHanSerifAKR9:cid=Adobe-KR-9;jfm=jis} % Korean
```

Note that the code above specifies `jfm-jis.lua`, which is for Japanese fonts, as JFM for Chinese and Korean fonts.

At present, LuaTeX-ja supports only 5 values written in the sample code above. Specifying other values, e.g.,

```
\jfont\test={psft:Ryumin-Light:cid=Adobe-Japan2;jfm=jis}
```

produces the following error:

```
! Package luatexja Error: bad cid key `Adobe-Japan2'.

See the luatexja package documentation for explanation.
Type  H <return>  for immediate help.
<to be read again>
                   \par
```

[9] This is because commands has `@` in their names.

```
1.78

? h
I couldn't find any non-embedded font information for the CID
`Adobe-Japan2'. For now, I'll use `Adobe-Japan1-6'.
Please contact the LuaTeX-ja project team.
?
```

8.5 Structure of a JFM file

A JFM file is a Lua script which has only one function call:

```
luatexja.jfont.define_jfm { ... }
```

Real data are stored in the table which indicated above by `{ ... }`. So, the rest of this subsection are devoted to describe the structure of this table. Note that all lengths in a JFM file are floating-point numbers in design-size unit.

`version=`⟨*version*⟩ (optional, default value is 1)

The version JFM. Currently 1, 2, and, 3 are supported

`dir=`⟨*direction*⟩ (required)

The direction of JFM. `'yoko'` (horizontal) or `'tate'` (vertical) are supported.

`zw=`⟨*length*⟩ (required)

The amount of the length of the "full-width".

`zh=`⟨*length*⟩ (required)

The amount of the "full-height" (height + depth).

`kanjiskip={`⟨*natural*⟩`,` ⟨*stretch*⟩`,` ⟨*shrink*⟩`}` (optional)

This field specifies the "ideal" amount of kanjiskip. As noted in Subsection 4.2, if the parameter kanjiskip is `\maxdimen`, the value specified in this field is actually used (if this field is not specified in JFM, it is regarded as 0 pt). Note that ⟨*stretch*⟩ and ⟨*shrink*⟩ fields are in design-size unit too.

`xkanjiskip={`⟨*natural*⟩`,` ⟨*stretch*⟩`,` ⟨*shrink*⟩`}` (optional)

Like the `kanjiskip` field, this field specifies the "ideal" amount of xkanjiskip.

■**Character classes** Besides from above fields, a JFM file have several sub-tables those indices are natural numbers. The table indexed by $i \in \omega$ stores information of *character class* i. At least, the character class 0 is always present, so each JFM file must have a sub-table whose index is `[0]`. Each sub-table (its numerical index is denoted by i) has the following fields:

`chars={`⟨*character*⟩`, ...}` (required except character class 0)

This field is a list of characters which are in this character type i. This field is optional if $i = 0$, since all **JAchar** which do not belong any character classes other than 0 are in the character class 0 (hence, the character class 0 contains most of **JAchar**s). In the list, character(s) can be specified in the following form:

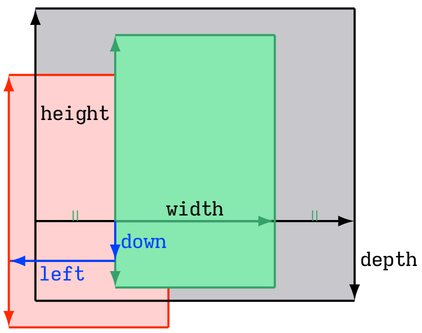
- a Unicode code point
- the character itself (as a Lua string, like `'あ'`)
- a string like `'あ*'` (the character followed by an asterisk)
- several "imaginary" characters (We will describe these later.)

`width=`⟨*length*⟩`, height=`⟨*length*⟩`, depth=`⟨*length*⟩`, italic=`⟨*length*⟩ (required)

Specify the width of characters in character class i, the height, the depth and the amount of italic correction. All characters in character class i are regarded that its width, height, and depth are as values of these fields. The default values are shown in Table 13.

Direction of JFM	'yoko' (horizontal)	'tate' (vertical)
width field	the width of the "real" glyph	
height field	the height of the "real" glyph	0.0
depth field	the depth of the "real" glyph	0.0
italic field	0.0	

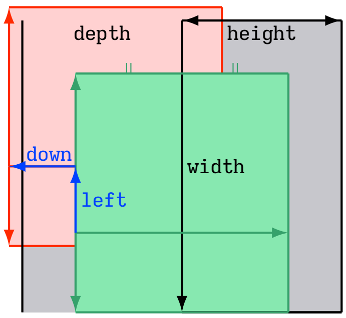
Table 13. Default values of width field and other fields



Consider a Japanese character node which belongs to a character class whose the align field is 'middle'.

- The black rectangle is the imaginary body of the node. Its width, height, and depth are specified by JFM.
- Since the align field is 'middle', the "real" glyph is centered horizontally (the green rectangle) first.
- Furthermore, the glyph is shifted according to values of fields left and down. The ultimate position of the real glyph is indicated by the red rectangle.

Figure 9. The position of the real glyph (horizontal Japanese fonts)



(...)

Figure 10. The position of the real glyph (vertical Japanese fonts)

left=⟨*length*⟩, down=⟨*length*⟩, align=⟨*align*⟩

These fields are for adjusting the position of the "real" glyph. Legal values of align field are 'left', 'middle', and 'right'. If one of these 3 fields are omitted, left and down are treated as 0, and align field is treated as 'left'. The effects of these 3 fields are indicated in Figure 9 and Figure 10.

In most cases, left and down fields are 0, while it is not uncommon that the align field is 'middle' or 'right'. For example, setting the align field to 'right' is practically needed when the current character class is the class for opening delimiters'.

kern={[*j*]=⟨*kern*⟩, [*j*′]={⟨*kern*⟩, [ratio=⟨*ratio*⟩]}, ...}

glue={[*j*]={⟨*width*⟩, ⟨*stretch*⟩, ⟨*shrink*⟩, [ratio=⟨*ratio*⟩, ...]}, ...}

Specifies the amount of kern or glue which will be inserted between characters in character class *i* and those in character class *j*.

⟨*ratio*⟩ specifies how much the glue is originated in the "right" character. It is a real number between 0 and 1, and treated as 0.5 if omitted. For example, The width of a glue between an ideographic full stop "。" and a fullwidth middle dot "・" is three-fourth of fullwidth, namely halfwidth from the ideographic full stop, and quarter-width from the fullwidth middle dot. In this case, we specify ⟨*ratio*⟩ to $0.25/(0.5+0.25)=1/3$.

In case of glue, one can specify following additional keys in each [*j*] subtable:

priority=⟨*priority*⟩ An integer in $[-4, 3]$ (treated as 0 if omitted), or a pair of these integers {⟨*stretch*⟩, ⟨*shrink*⟩} (version 2 or later). This is used only in line adjustment with priority by luatexja-adjust (see Subsection 13.3). Higher value means the glue is easy to stretch, and is also easy to shrink.

kanjiskip_natural=⟨*num*⟩, kanjiskip_stretch=⟨*num*⟩, kanjiskip_shrink=⟨*num*⟩

These keys specifies the amount of the natural width of kanjiskip(the stretch/shrink part, respectively) which will be inserted in addition to the original JFM glue. Default values of them are all 0.

As an example, in `jfm-ujis.lua`, the standard JFM in horizontal writing, we have

- Between an ordinal letter "あ" and an ideographic opening bracket, we have a glue whose natural part and shrink part are both half-width, while its stretch part is zero. However, this glue also can be stretched as much as the stretch part of kanjiskip times the value of `kanjiskip_stretch` key (1 in this case).
- Between an ideographic closeing brackets (including the ideographic comma "，") and an ordinal letter (including an **ALchar** "f"), we have the same glue. Again, this glue also can be stretched as much as the stretch part of kanjiskip times the value of `kanjiskip_stretch` key (1 in this case).
- Between an ideographic opening bracket and an ordinal letter and between an ordinal letter and an ideographic closing bracket, we have a glue whose natural part and stretch part are both zero, while its shrink part as much as the shrink part of kanjiskip.

Hence we have the following result:

```
\leavevmode\let\V=\vrule
\ltjsetparameter{kanjiskip=0pt plus 5\zw}
\ltjsetparameter{xkanjiskip=0pt plus 0.5\zw}
\V\hbox spread 7\zw{aあ「い」う，えお」f}\V

\vrule\hbox{aあ「い」う，えお」f}\V\par
\ltjsetparameter{kanjiskip=0pt minus \zw}
\V\hbox spread -2.5\zw{aあ「い」う，えお」f}\V
```

|aあ　「い」　う，　え　お」　f|
|aあ「い」う，えお」f|
|aあ「い」う，えお」f|

end_stretch=⟨*kern*⟩, end_shrink=⟨*kern*⟩ (optional, version 1 only)

end_adjust={⟨*kern*⟩, ⟨*kern*⟩, ...} (optional, version 2 or later)

round_threshold=⟨*float*⟩ (optional, version 3 or later, only available in character class 0)

■Character to character classes We explain how the character class of a character is determined, using `jfm-test.lua` which contains the following:

```
[0] = {
   chars = { '漢' },
   align = 'left', left = 0.0, down = 0.0,
   width = 1.0, height = 0.88, depth = 0.12, italic=0.0,
},
[2000] = {
   chars = { '。', 'ﾋ' },
   align = 'left', left = 0.0, down = 0.0,
   width = 0.5, height = 0.88, depth = 0.12, italic=0.0,
},
```

Now consider the following input/output:

```
\jfont\a=IPAexMincho:jfm=test;+hwid
\setbox0\hbox{\a ヒ漢}\the\wd0
```

15.0pt

Now we look why the above source outputs 15 pt.

1. The character "ヒ" is converted to its half width form "ﾋ" by `hwid` feature.
2. According to the JFM, the character class of "ﾋ" is 2000, hence its width is halfwidth.

3. The character class of “漢” is zero, hence its width is fullwidth.

4. Hence the width of \hbox equals to 15 pt.

This example shows that the character class of a character is generally determined *after applying font features by luaotfload.*

However, if the class determined by the glyph after application of features is zero, LuaTEX-ja adopts the class determined by the glyph *before* application of features. The following input is an example.

```
\jfont\a=HaranoAjiMincho-Regular:jfm=test;+vert
\a 漢。\inhibitglue 漢
```

漢 漢

Here, the character class of the ideographic full stop “。” (`U+3002`) is determined as follows:

1. As the case of “ヒ”, the ideographic full stop “。” is converted to its vertical form “︒” (`U+FE12`) by `vert` feature.

2. The character class of “︒”, according to the JFM is *zero*.

3. However, LuaTEX-ja remembers that this “︒” is obtained from “。” by font features. The character class of “。” is *non-zero value*, namely, 2000.

4. Hence the ideographic full stop “。” in above belongs the character class 2000.

■**Imaginary characters** As described before, one can specify several “imaginary characters” in `chars` field. The most of these characters are regarded as the characters of class 0 in pTEX. As a result, LuaTEX-ja can control typesetting finer than pTEX. The following is the list of imaginary characters:

`'boxbdd'`
: The beginning/ending of a hbox, and the beginning of a noindented (i.e., began by `\noindent`) paragraph. If a hbox *b* begins (resp. ends) a glue or kern between this “charater” and a **JAchar**, **JAglue** won’t be inserted before(resp. after) the hbox *b*. kanjiskip and xkanjiskip around a hbox.

`'parbdd'`
: The beginning of an (indented) paragraph.

`'jcharbdd'`
: A boundary between **JAchar** and anything else.

`'alchar'`, `'nox_alchar'`
: (version 3 or later) A boundary between **JAchar** and **ALchar**.

`'glue'`
: (version 3 or later) A boundary between **JAchar**, and, a glue or kern.

−1 The left/right boundary of an inline math formula.

■**Porting JFM from pTEX** See Japanese version of this manual.

8.6 Math font family

TEX handles fonts in math formulas by 16 font families[10], and each family has three fonts: `\textfont`, `\scriptfont` and `\scriptscriptfont`.

LuaTEX-ja’s handling of Japanese fonts in math formulas is similar; Table 14 shows counterparts to TEX’s primitives for math font families. There is no relation between the value of `\fam` and that of `\jfam`; with appropriate settings, one can set both `\fam` and `\jfam` to the same value. Here ⟨*jfont_cs*⟩ in the argument of jatextfont etc. is a control sequence which is defined by `\jfont`, i.e., a *horizontal* Japanese font.

[10] Omega, Aleph, LuaTEX and ε-(u)pTEX can handles 256 families, but an external package is needed to support this in plain TEX and LATEX.

Table 14. Commands for Japanese math fonts

Japanese fonts	alphabetic fonts
`\jfam` $\in [0, 256)$	`\fam`
jatextfont `={`⟨*jfam*⟩`,`⟨*jfont_cs*⟩`}`	`\textfont`⟨*fam*⟩`=`⟨*font_cs*⟩
jascriptfont `={`⟨*jfam*⟩`,`⟨*jfont_cs*⟩`}`	`\scriptfont`⟨*fam*⟩`=`⟨*font_cs*⟩
jascriptscriptfont `={`⟨*jfam*⟩`,`⟨*jfont_cs*⟩`}`	`\scriptscriptfont`⟨*fam*⟩`=`⟨*font_cs*⟩

8.7 Callbacks

LuaTEX-ja also has several callbacks. These callbacks can be accessed via `luatexbase.add_to_callback` function and so on, as other callbacks.

`luatexja.load_jfm` callback

With this callback, one can overwrite JFMs. This callback is called when a new JFM is loaded.

```
function (<table> jfm_info, <string> jfm_name)
  return <table> new_jfm_info
end
```

The argument `jfm_info` contains a table similar to the table in a JFM file, except this argument has `chars` field which contains character codes whose character class is not 0.

An example of this callback is the `ltjarticle` class, with forcefully assigning character class 0 to `'parbdd'` in the JFM `jfm-min.lua`.

`luatexja.define_jfont` callback

This callback and the next callback form a pair, and you can assign characters which do not have fixed code points in Unicode to non-zero character classes. This `luatexja.define_font` callback is called just when new Japanese font is loaded.

```
function (<table> jfont_info, <number> font_number)
  return <table> new_jfont_info
end
```

`jfont_info` has the following fields, *which may not overwritten by a user*:

`size` The font size specified at `\jfont` in scaled points (1 sp $= 2^{-16}$ pt).

`zw`, `zh`, `kanjiskip`, `xkanjiskip` These are scaled value of those specified by the JFM, by the font size.

`jfm` The internal number of the JFM.

`var` The value of `jfmvar` key, which is specified at `\jfont`. The default value is the empty string.

`chars` The mapping table from character codes to its character classes.
The specification `[i].chars={`⟨*character*⟩`, ...}` in the JFM will be stored in this field as `chars={[`⟨*character*⟩`]=`i`, ...}`.

`char_type` For $i \in \omega$, `char_type[`i`]` is information of characters whose class is i, and has the following fields:

- `width`, `height`, `depth`, `italic`, `down`, `left` are just scaled value of those specified by the JFM, by the font size.
- `align` is a number which is determined from `align` field in the JFM:

$$\begin{cases} 1 & (\texttt{'right'} \text{ in JFM}), \\ 0.5 & (\texttt{'middle'} \text{ in JFM}), \\ 0 & (\text{otherwise}). \end{cases}$$

For $i, j \in \omega$, `char_type[`i`][`j`]` stores a kern or a glue which will be inserted between character class i and class j.

The returned table new_jfont_info also should include these fields, but you are free to add more fields (to use them in the luatexja.find_char_class callback). The font_number is a font number.

A good example of this and the next callbacks is the luatexja-otf package, supporting "AJ1-xxx" form for Adobe-Japan1 CID characters in a JFM. This callback doesn't replace any code of LuaTeX-ja.

luatexja.find_char_class callback

This callback is called just when LuaTeX-ja is trying to determine which character class a character chr_code belongs. A function used in this callback should be in the following form:

```
function (<number> char_class, <table> jfont_info, <number> char_code)
  if char_class~=0 then return char_class
  else
    ....
    return (<number> new_char_class or 0)
  end
end
```

The argument char_class is the result of LuaTeX-ja's default routine or previous function calls in this callback, hence this argument may not be 0. Moreover, the returned new_char_class should be as same as char_class when char_class is not 0, otherwise you will overwrite the LuaTeX-ja's default routine.

luatexja.set_width callback

This callback is called when LuaTeX-ja is trying to encapsule a **JAchar** *glyph_node*, to adjust its dimension and position.

```
function (<table> shift_info, <table> jfont_info, <table> char_type)
  return <table> new_shift_info
end
```

The argument shift_info and the returned new_shift_info have down and left fields, which are the amount of shifting down/left the character in a scaled point.

A good example is test/valign.lua. After loading this file, the vertical position of glyphs is automatically adjusted; the ratio (height ： depth) of glyphs is adjusted to be that of letters in the character class 0. For example, suppose that

- The setting of the JFM: (height) $= 88x$, (depth) $= 12x$ (the standard values of Japanese OpenType fonts);
- The value of the real font: (height) $= 28y$, (depth) $= 5y$ (the standard values of Japanese TrueType fonts).

Then, the position of glyphs is shifted up by

$$\frac{88x}{88x+12x}(28y+5y) - 28y = \frac{26}{25}y = 1.04y.$$

9 Parameters

9.1 \ltjsetparameter

As described before, \ltjsetparameter and \ltjgetparameter are commands for accessing most parameters of LuaTeX-ja. One of the main reason that LuaTeX-ja didn't adopted the syntax similar to that of pTeX (*e.g.*, \prebreakpenalty`） =10000) is the position of hpack_filter callback in the source of LuaTeX, see Section 14.

\ltjsetparameter and \ltjglobalsetparameter are commands for assigning parameters. These take one argument which is a key-value list. The difference between these two commands is the scope of assignment; \ltjsetparameter does a local assignment and \ltjglobalsetparameter does a global one by default. They also obey the value of \globaldefs, like other assignments.

The following is the list of parameters which can be specified by the \ltjsetparameter command. [\cs] indicates the counterpart in pTeX, and symbols beside each parameter has the following meaning:

- "∗" : values at the end of a paragraph or a hbox are adopted in the whole paragraph or the whole hbox.
- " † ": assignments are always global.

jcharwidowpenalty =⟨*penalty*⟩* [\jcharwidowpenalty]

Penalty value for suppressing orphans. This penalty is inserted just after the last **JAchar** which is not regarded as a (Japanese) punctuation mark.

kcatcode ={⟨*char_code*⟩,⟨*natural number*⟩}*

An additional attributes which each character whose character code is ⟨*char_code*⟩ has. At version 20120506.0 or later, the lowermost bit of ⟨*natural number*⟩ indicates whether the character is considered as a punctuation mark (see the description of jcharwidowpenalty above).

prebreakpenalty ={⟨*char_code*⟩,⟨*penalty*⟩}* [\prebreakpenalty]

Set a penalty which is inserted automatically before the character ⟨*char_code*⟩, to prevent a line starts from this character. For example, a line cannot started with one of closing brackets "〛", so LuaTEX-ja sets

```
\ltjsetparameter{prebreakpenalty={`〛,10000}}
```

by default.

pTEX has following restrictions on \prebreakpenalty and \postbreakpenalty, but they don't exist in LuaTEX-ja:

- Both \prebreakpenalty and \postbreakpenalty cannot be set for the same character.
- We can set \prebreakpenalty and \postbreakpenalty up to 256 characters.

postbreakpenalty ={⟨*char_code*⟩,⟨*penalty*⟩}* [\postbreakpenalty]

Set a penalty which is inserted automatically after the character ⟨*char_code*⟩, to prevent a line ends with this character.

jatextfont ={⟨*jfam*⟩,⟨*jfont_cs*⟩}* [\textfont in TEX]

jascriptfont ={⟨*jfam*⟩,⟨*jfont_cs*⟩}* [\scriptfont in TEX]

jascriptscriptfont ={⟨*jfam*⟩,⟨*jfont_cs*⟩}* [\scriptscriptfont in TEX]

yjabaselineshift =⟨*dimen*⟩

yalbaselineshift =⟨*dimen*⟩ [\ybaselineshift]

tjabaselineshift =⟨*dimen*⟩

talbaselineshift =⟨*dimen*⟩ [\tbaselineshift]

jaxspmode ={⟨*char_code*⟩,⟨*mode*⟩}*

Set whether inserting xkanjiskip is allowed before/after a **JAchar** whose character code is ⟨*char_code*⟩. The followings are allowed for ⟨*mode*⟩:

0, inhibit Insertion of xkanjiskip is inhibited before the character, nor after the character.

1, preonly Insertion of xkanjiskip is allowed before the character, but not after.

2, postonly Insertion of xkanjiskip is allowed after the character, but not before.

3, allow Insertion of xkanjiskip is allowed both before the character and after the character. This is the default value.

This parameter is similar to the \inhibitxspcode primitive of pTEX, but not compatible with \inhibitxspcode.

alxspmode ={⟨*char_code*⟩,⟨*mode*⟩}* [\xspcode]

Set whether inserting xkanjiskip is allowed before/after a **ALchar** whose character code is ⟨*char_code*⟩. The followings are allowed for ⟨*mode*⟩:

0, inhibit Insertion of xkanjiskip is inhibited before the character, nor after the character.

1, preonly Insertion of xkanjiskip is allowed before the character, but not after.

2, postonly Insertion of xkanjiskip is allowed after the character, but not before.

3, allow Insertion of xkanjiskip is allowed before the character and after the character. This is the default value.

Note that parameters jaxspmode and alxspmode share a common table, hence these two parameters are synonyms of each other.

autospacing =⟨*bool*⟩ [`\autospacing`]

autoxspacing =⟨*bool*⟩ [`\autoxspacing`]

kanjiskip =⟨*skip*⟩* [`\kanjiskip`]

The default glue which inserted between two **JAchar**s. Changing current Japanese font does not alter this parameter, as pTEX.

If the natural width of this parameter is `\maxdimen`, LuaTEX-ja uses the value which is specified in the JFM for current Japanese font (See Subsection 8.5).

xkanjiskip =⟨*skip*⟩* [`\xkanjiskip`]

The default glue which inserted between a **JAchar** and an **ALchar**. Changing current font does not alter this parameter, as pTEX.

As kanjiskip, if the natural width of this parameter is `\maxdimen`, LuaTEX-ja uses the value which is specified in the JFM for current Japanese font (See Subsection 8.5).

differentjfm =⟨*mode*⟩†

Specify how glues/kerns between two **JAchar**s whose JFM (or size) are different. The allowed arguments are the followings:

`average`, `both`, `large`, `small`, `pleft`, `pright`, `paverage`

The default value is `paverage`. …

jacharrange =⟨*ranges*⟩

kansujichar ={⟨*digit*⟩, ⟨*char_code*⟩}* [`\kansujichar`]

direction =⟨*dir*⟩ (always local)

Assigning to this parameter has the same effect as `\yoko` (if ⟨*dir*⟩ = 4), `\tate` (if ⟨*dir*⟩ = 3), `\dtou` (if ⟨*dir*⟩ = 1) or `\utod` (if ⟨*dir*⟩ = 11). If the argument ⟨*dir*⟩ is not one of 4, 3, 1 nor 11, the behavior of this assignment is undefined.

9.2 `\ltjgetparameter`

`\ltjgetparameter` is a control sequence for acquiring parameters. It always takes a parameter name as first argument.

```
\ltjgetparameter{differentjfm},
\ltjgetparameter{autospacing},
\ltjgetparameter{kanjiskip},
\ltjgetparameter{prebreakpenalty}{`）}.
```

paverage, 1, 0.0pt plus 0.4pt minus 0.5pt, 10000.

The return value of `\ltjgetparameter` *is always a string*, which is outputted by `tex.write()`. Hence any character other than space “ ” (`U+0020`) has the category code 12 (other), while the space has 10 (space).

- If first argument is one of the following, no additional argument is needed.

  jcharwidowpenalty, yjabaselineshift, yalbaselineshift, autospacing, autoxspacing, kanjiskip, xkanjiskip, differentjfm, direction

Note that `\ltjgetparameter{autospacing}` and `\ltjgetparameter{autoxspacing}` returns 1 or 0, not `true` nor `false`.

- If first argument is one of the following, an additional argument—a character code, for example—is needed.

  kcatcode, prebreakpenalty, postbreakpenalty, jaxspmode, alxspmode

  `\ltjgetparameter{jaxspmode}{...}` and `\ltjgetparameter{alxspmode}{...}` returns 0, 1, 2, or 3, instead of `preonly` etc.

- `\ltjgetparameter{jacharrange}{`⟨*range*⟩`}` returns `0` if "characters which belong to the character range ⟨*range*⟩ are **JAchar**", 1 if "... are **ALchar**". Although there is no character range −1, specifying −1 to ⟨*range*⟩ does not cause an error (returns 1).

- For an integer ⟨*digit*⟩ between 0 and 9, `\ltjgetparameter{kansujichar}{`⟨*digit*⟩`}` returns the character code of the result of `\kansuji`⟨*digit*⟩.

- `\ltjgetparameter{adjustdir}` returns a integer which represents the direction of the surrounding vertical list. As direction, the return value 1 means *dtou* direction, 3 means *tate* direction (vertical typesetting), and 4 means *yoko* direction (horizontal typesetting).

- For an integer ⟨*register*⟩ between 0 and 65535, `\ltjgetparameter{boxdir}{`⟨*register*⟩`}` returns the direction of `\box`⟨*register*⟩. If this box register is void, the returned value is zero.

- The following parameter names *cannot be specified* in `\ltjgetparameter`.

  jatextfont, jascriptfont, jascriptscriptfont, jacharrange

- `\ltjgetparameter{chartorange}{`⟨*char_code*⟩`}` returns the range number which ⟨*char_code*⟩ belongs to (although there is no parameter named "chartorange").

  If ⟨*char_code*⟩ is between 0 and 127, this ⟨*char_code*⟩ does not belong to any character range. In this case, `\ltjgetparameter{chartorange}{`⟨*char_code*⟩`}` returns −1.

  Hence, one can know whether ⟨*char_code*⟩ is **JAchar** or not by the following:

  ```
  \ltjgetparameter{jacharrange}{\ltjgetparameter{chartorange}{⟨char_code⟩}}
                                                    % 0 if JAchar, 1 if ALchar
  ```

- Because the returned value is string, the following conditionals do not work if kanjiskip (or xkanjiskip) has the stretch part or the shrink part.

  ```
  \ifdim\ltjgetparameter{kanjiskip}>\z@ ... \fi
  \ifdim\ltjgetparameter{xkanjiskip}>\z@ ... \fi
  ```

  The correct way is using a temporary register.

  ```
  \@tempskipa=\ltjgetparameter{kanjiskip} \ifdim\@tempskipa>\z@ ... \fi
  \@tempskipa=\ltjgetparameter{xkanjiskip}\ifdim\@tempskipa>\z@ ... \fi
  ```

9.3 Alternative Commands to `\ltjsetparameter`

The basic method to set parameters of LuaTeX-ja is to use `\ltjsetparameter` or `\ltjglobalsetparameter`. However, these commands are slow, because they parse a key-value list, so several alternative commands are used in LuaTeX-ja. *This subsection is not for general LuaTeX-ja users.*

■Setting kanjiskip or xkanjiskip In ltjsclasses, every size-changing command such as `\Large` changes `\kanjiskip` and `\xkanjiskip`. But a simple implementation, as the code below, is slow since two key-value lists are parsed by `\ltjsetparameter`:

```
\ltjsetparameter{kanjiskip=0\zw plus .1\zw minus .01\zw}
\@tempskipa=\ltjgetparameter{xkanjiskip}
\ifdim\@tempskipa>\z@
  \if@slide
    \ltjsetparameter{xkanjiskip=0.1em}
  \else
    \ltjsetparameter{xkanjiskip=0.25em plus 0.15em minus 0.06em}
  \fi
\fi
```

Hence, LuaTeX-ja defines more primitive commands, namely `\ltj@setpar@global`, `\ltjsetkanjiskip`, and `\ltjsetxkanjiskip`. Here

```
\ltj@setpar@global\ltjsetkanjiskip 10pt
```

and `\ltjsetparameter{kanjiskip=10pt}` has the same effect. The actual code of ltjsclasses is shown below:

```
\ltj@setpar@global
\ltjsetkanjiskip{\z@ plus .1\zw minus .01\zw}
\@tempskipa=\ltjgetparameter{xkanjiskip}
\ifdim\@tempskipa>\z@
  \if@slide
    \ltjsetxkanjiskip.1em
  \else
    \ltjsetxkanjiskip.25em plus .15em minus .06em
  \fi
\fi
```

Note that using `\ltjsetkanjiskip` or `\ltjsetxkanjiskip` alone, that is, without executing `\ltj@setpar@global` in advance, is *not* supported.

10 Other Commands for plain TeX and LaTeX 2$_\varepsilon$

10.1 Commands for compatibility with pTeX

The following commands are implemented for compatibility with pTeX. Note that the former five commands don't support JIS X 0213, but only JIS X 0208. The last `\kansuji` converts an integer into its Chinese numerals.

`\kuten`, `\jis`, `\euc`, `\sjis`, `\ucs`, `\kansuji`

These six commands takes an internal integer, and returns a *string*.

```
\newcount\hoge
\hoge="2423 %"
\the\hoge, \kansuji\hoge\\
\jis\hoge, \char\jis\hoge\\
\kansuji1701
```

9251, 九二五一
12355, い
一七〇一

To change characters of Chinese numerals for each digit, set kansujichar parameter:

```
\ltjsetparameter{kansujichar={1,`壹}}
\ltjsetparameter{kansujichar={7,`漆}}
\ltjsetparameter{kansujichar={0,`零}}
\kansuji1701
```

壹漆零壹

10.2 \inhibitglue, \disinhibitglue

`\inhibitglue` suppresses the insertion of a glue/kern soecified in JFM at the place. The following is an example, using a special JFM that there will be a glue between the beginning of a box and “あ”, and also between “あ” and “ウ”.

```
\jfont\g=HaranoAjiMincho-Regular:jfm=test \g
\fbox{\hbox{あウあ\inhibitglue ウ}}
\inhibitglue\par\noindent あ1
\par\inhibitglue\noindent あ2
\par\noindent\inhibitglue あ3
\par ）4）\inhibitglue 5
\par\hrule\noindent あoff\inhibitglue ice
```

あ　ウあウ
あ　1
あ　2
あ　3
）4）5
あ　office

With the help of this example, we remark the specification of `\inhibitglue`:

- The call of `\inhibitglue` in the (internal) vertical mode is simply ignored.
- `\inhibitglue` does *not* suppress kanjiskip or `\xkanjiskip`.
- The call of `\inhibitglue` in the (restricted) horizontal mode is only effective on the spot; does not get over boundary of paragraphs. Moreover, `\inhibitglue` cancels ligatures and kernings, as shown in the last line of above example.
- The call of `\inhibitglue` in math mode is just ignored.

`\disinhibitglue` suppresses the effect of `\inhbitglue`. In other words, `\disinhibitglue` allows the insertion of a glue/kern specified by JFM. If `\inhibitglue` and `\disinhibitglue` both specified at the same time, the latest one is effective. This commans is added in the version 20201224.0.

Note that `\disinhibitglue` also cancels ligatures and kernings.

10.3 \ltjfakeboxbdd, \ltjfakeparbegin

Sometimes `'parbdd'` and `'boxbdd'` specifications look like "fail", especially in paragraphs inside list environments. This is because `\everypar` inserts some nodes such as boxes and kerns, so the "first letter" in a paragraph is in fact not the first letter.

```
\parindent1\zw
\noindent ああああああああ\par % for comparison
「ああああああ \par            % normal paragraph

\everypar{\null}
「ああああああ \par            % ???
```

ああああああああ
「ああああああ
「ああああああ

`\ltjfakeboxbdd` and `\ltjfakeparbegin` primitives resolve this situation.

- `\ltjfakeparbegin` creates a node which indicates "beginning of an indented paragraph" to the insertion process of **JAglue**.
- `\ltjfakeboxbdd` creates a node which indicates "beginning/ending of a box" to the insertion process of **JAglue**.

As an example, the example above can be improved as follows:

```
\parindent1\zw
\noindent ああああああああ\par % for comparison
「ああああああ \par            % normal paragraph

\everypar{\null\ltjfakeparbegin}
「ああああああ \par
```

ああああああああ
「ああああああ
「ああああああ

10.4 \insertxkanjiskip, \insertkanjiskip

There are some situations which one wants to insert xkanjiskip manually. A simple approach is to use `\hskip\ltjgetparameter{xkanjiskip}`, but this approach has several weak points. To cope with these weak points, LuaTeX-ja defines a command `\insertxkanjiskip` which inserts xkanjiskip glue manually, from the version 20201224.0.

- "\insertxkanjiskip" (without any keyword) uses the value of xkanjiskip at the place.
- "\insertxkanjiskip late" (with "late" keyword) uses the value of xkanjiskip at the end of a paragraph/hbox.

See the example below.

```
\ltjsetparameter{xkanjiskip=0.25\zw}
あ（%                    0.5\zw  (from JFM)
あ\insertxkanjiskip （%   0.25\zw (xkanjiskip at here)
あ\insertxkanjiskip late （% 0.25\zw (xkanjiskip at EOP)
あa%                     1.25\zw (xkanjiskip at EOP)
\\%
\ltjsetparameter{xkanjiskip=1.25\zw}
あ\insertxkanjiskip （%   1.25\zw (xkanjiskip at here)
あa%                     1.25\zw (xkanjiskip at EOP)
%% At the end of the paragraph (EOP), xkanjiskip is 1.25\zw.
```

あ（あ（あ　（あ　a
あ　（あ　a

There is a similar command \insertkanjiskip (kanjiskip instead of xkanjiskip) is also defined. Note that any shorthand form of \insert[x]kanjiskip are not defined by LuaTeX-ja.

10.5 \ltjdeclarealtfont

Using \ltjdeclarealtfont, one can "compose" more than one Japanese fonts. This \ltjdeclarealtfont uses in the following form:

> \ltjdeclarealtfont⟨*base_font_cs*⟩⟨*alt_font_cs*⟩{⟨*range*⟩}

where ⟨*base_font_cs*⟩ and ⟨*alt_font_cs*⟩ are defined by \jfont. Its meaning is

> If the current Japanese font is ⟨*base_font_cs*⟩, characters which belong to ⟨*range*⟩ is typeset by another Japanese font ⟨*alt_font_cs*⟩, instead of ⟨*base_font_cs*⟩.

Here ⟨*range*⟩ is a comma-separated list of character codes, but also accepts negative integers: $-n$ ($n \geq 1$) means that all characters of character classes n, with respect to JFM used by ⟨*base_font_cs*⟩. Note that characters which do not exist in ⟨*alt_font_cs*⟩ are ignored.

For example, if \hoge uses jfm-ujis.lua, the standard JFM of LuaTeX-ja, then

```
\ltjdeclarealtfont\hoge\piyo{"3000-"30FF, {-1}-{-1}}
```

does

> If the current Japanese font is \hoge, U+3000–U+30FF and characters in class 1 (ideographic opening brackets) are typeset by \piyo.

Note that specifying negative numbers needs specification like {-1}-{-1}, because simple "-1" is treated as the range between 0 and 1.

```
\gtfamily\large
¶,\char`¶,\ltjalchar`¶,\ltjjachar`¶\\      % default: ALchar
α,\char`α,\ltjalchar`α,\ltjjachar`α\\      % default: JAchar
g,\char`g,\ltjalchar`g,\ltjjachar`g        % ALchar unless \ltjjachar
```

¶,¶,¶, ¶
α, α,α, α
g,g,g, g

11 Commands for LaTeX 2ε

11.1 Loading Japanese fonts in LaTeX 2ε

From version 20190107, *LuaTeX-ja does not load Japanese fonts for horizontal direction and that for vertical direction at same time*, to reduce the number of loaded fonts. This will save time for typesetting and memory consumption of Lua side ([11]).

- `\selectfont` loads (and chooses) only the Japanese font for the current direction, and does not load the Japanese font for other direction (LuaTeX-ja only detects its size and JFM, to calculate the amount of shifting the baseline).
- Direction changing commands (`\yoko`, `\tate`, `\dtou`, `\utod`) are patched to include the following process:

  If the Japanese font for new direction is not loaded, LuaTeX-ja loads it automatically.

  Original commands are saved as `\ltj@@orig@yoko` etc.
- Specifying Japanese font command which is defined by `\jfont`, `\tfont`, or `\DeclareFixedFont` directly actually loads (and selects) the Japanese font. For example, **JAchar**s in `\box0` will be typeset in `\HOGE`, in the following code:

```
% in horizontal direction (\yoko)
\DeclareFixedFont\HOGE{JT3}{gt}{m}{n}{12} % JT3: for vertical direction
\HOGE
\setbox0=\hbox{\tate あいう}
```

11.2 Patch for NFSS2

Japanese patch for NFSS2 in LuaTeX-ja is based on `plfonts.dtx` which plays the same role in pLaTeX 2_ε. We will describe commands which are not described in Subsection 3.1.

additonal dimensions
Like pLaTeX 2_ε, LuaTeX-ja defines the following dimensions for information of current Japanese font:

`\cht` (height), `\cdp` (depth), `\cHT` (sum of former two),
`\cwd` (width), `\cvs` (lineskip), `\chs` (equals to `\cwd`)

and its `\normalsize` version:

`\Cht` (height), `\Cdp` (depth), `\Cwd` (width),
`\Cvs` (equals to `\baselineskip`), `\Chs` (equals to `\cwd`).

Note that `\cwd` and `\cHT` may differ from `\zw` and `\zh` respectively. On the one hand the former dimensions are determined from a character whose character class is zero, but on the other hand `\zw` and `\zh` are specified by JFM.

`\DeclareYokoKanjiEncoding{`⟨*encoding*⟩`}{`⟨*text-settings*⟩`}{`⟨*math-settings*⟩`}`
`\DeclareTateKanjiEncoding{`⟨*encoding*⟩`}{`⟨*text-settings*⟩`}{`⟨*math-settings*⟩`}`
In NFSS2 under LuaTeX-ja, distinction between alphabetic fonts and Japanese fonts are only made by their encodings. For example, encodings OT1 and T1 are encodings for alphabetic fonts, and Japanese fonts cannot have these encodings. These command define a new encoding scheme for Japanese font families.

`\DeclareKanjiEncodingDefaults{`⟨*text-settings*⟩`}{`⟨*math-settings*⟩`}`
`\DeclareKanjiSubstitution{`⟨*encoding*⟩`}{`⟨*family*⟩`}{`⟨*series*⟩`}{`⟨*shape*⟩`}`
`\DeclareErrorKanjiFont{`⟨*encoding*⟩`}{`⟨*family*⟩`}{`⟨*series*⟩`}{`⟨*shape*⟩`}{`⟨*size*⟩`}`
The above 3 commands are just the counterparts for `\DeclareFontEncodingDefaults` and others.

`\reDeclareMathAlphabet{`⟨*unified-cmd*⟩`}{`⟨*al-cmd*⟩`}{`⟨*ja-cmd*⟩`}`

`\DeclareRelationFont{`⟨*ja-encoding*⟩`}{`⟨*ja-family*⟩`}{`⟨*ja-series*⟩`}{`⟨*ja-shape*⟩`}`
`{`⟨*al-encoding*⟩`}{`⟨*al-family*⟩`}{`⟨*al-series*⟩`}{`⟨*al-shape*⟩`}`
This command sets the "accompanied" alphabetic font (given by the latter 4 arguments) with respect to a Japanese font given by the former 4 arguments.

`\SetRelationFont`
This command is almost same as `\DeclareRelationFont`, except that this command does a local assignment, where `\DeclareRelationFont` does a global assignment.

`\userelfont`

(Only) at the next call of `\selectfont`, change current alphabetic font encoding/family/… to the 'accompanied' alphabetic font family with respect to current Japanese font family, which was set by `\DeclareRelationFont` or `\SetRelationFont`.

The following is an example of `\SetRelationFont` and `\userelfont`:

```
\makeatletter
\SetRelationFont{JY3}{\k@family}{m}{n}{TU}{lmss}{m}{n}
  % \k@family: current Japanese font family
\userelfont\selectfont あいうabc
```

あいう abc

`\adjustbaseline`

In pLaTeX 2_ε, `\adjustbaseline` sets `\tbaselineshift` to match the vertical center of "M" and that of "漢" in vertical typesetting:

$$\texttt{\textbackslash tbaselineshift} \leftarrow \frac{(h_{\mathrm{M}} + d_{\mathrm{M}}) - (h_{\text{漢}} + d_{\text{漢}})}{2} + d_{\text{漢}} - d_{\mathrm{M}},$$

where h_a and d_a denote the height of "a" and the depth, respectively. In LuaTeX-ja, this `\adjustbaseline` does similar task, namely setting the talbaselineshift parameter (a Japanese character whose character class is zero is used, instead of '漢').

`\fontfamily{⟨family⟩}`

As in LaTeX 2_ε, this command changes current font family (alphabetic, Japanese, *or both*) to ⟨*family*⟩. See Subsection 11.3 for detail.

`\fontshape{⟨shape⟩}`, `\fontshapeforce{⟨shape⟩}`

As in LaTeX 2_ε, this command changes current alphabetic font shape according to shape change rules.

Traditionally, `\fontshape` changes also current Japanese font shape always. However, this leads a lot of LaTeX font warning like

```
Font shape `JY3/mc/m/it' undefined
using `JY3/mc/m/n' instead on ....
```

when `\itshape` is called, because almost all Japanese fonts only have shape "n", and `\itshape` calls `\fontshape`.

LuaTeX-ja 20200323.0 change the behavior. Namely, `\fontshape{⟨shape⟩}` and `\fontshapeforce{⟨shape⟩}` change current Japanese font shape, only if the required shape (according to shape changing rules) or ⟨*shape*⟩ is avaliable in current Japanese font family/series. When this is not the case, an info such as

```
Kanji font shape JY3/mc/m/it' undefined
No change on ...
```

is issued instead of a warning.

`\kanjishape{⟨shape⟩}`, `\kanjishapeforce{⟨shape⟩}`

`\kanjishape{⟨shape⟩}` changes current Japanese font shape according to shape change rules, and `\kanjishapeforce{⟨shape⟩}` changes current Japanese font shape to ⟨*shape*⟩, regardless of the rules. Hence `\kanjishape{it}` produces a warning

```
Font shape `JY3/mc/m/it' undefined
using `JY3/mc/m/n' instead on ....
```

which is not produced by `\fontshape{it}`.

`\DeclareAlternateKanjiFont`

`{⟨base-encoding⟩}{⟨base-family⟩}{⟨base-series⟩}{⟨base-shape⟩}`
`{⟨alt-encoding⟩}{⟨alt-family⟩}{⟨alt-series⟩}{⟨alt-shape⟩}{⟨range⟩}`

As `\ltjdeclarealtfont` (Subsection 10.5), characters in ⟨*range*⟩ of the Japanese font (we say the *base font*) which specified by first 4 arguments are typeset by the Japanese font which specified by fifth to eighth arguments (we say the *alternate font*). An example is shown in Figure 11.

```
\DeclareKanjiFamily{JY3}{edm}{}
\DeclareFontShape{JY3}{edm}{m}{n}   {<-> s*HaranoAjiMincho-Regular:jfm=ujis}{}
\DeclareFontShape{JY3}{edm}{m}{fb}  {<-> s*HaranoAjiGothic-Regular:jfm=ujis;color=003FFF}{}
\DeclareFontShape{JY3}{edm}{m}{fb2} {<-> s*HaranoAjiGothic-Regular:jfm=ujis;color=FF1900}{}
\DeclareAlternateKanjiFont{JY3}{edm}{m}{n}{JY3}{edm}{m}{fb}{ "4E00-"67FF,{-2}-{-2}}
\DeclareAlternateKanjiFont{JY3}{edm}{m}{n}{JY3}{edm}{m}{fb2}{"6800-"9FFF}
{\kanjifamily{edm}\selectfont
日本国民は、正当に選挙された国会における代表者を通じて行動し、……}
```

日本国民は、正当に選挙された国会における代表者を通じて行動し、……

Figure 11. An example of `\DeclareAlternateKanjiFont`

- In `\ltjdeclarealtfont`, the base font and the alternate font must be already defined. But this `\DeclareAlternateKanjiFont` is not so. In other words, `\DeclareAlternateKanjiFont` is effective only after current Japanese font is changed, or only after `\selectfont` is executed.
- …

Furthermore, LuaTEX-ja applies patches which enables NFSS2 commands, such as `\DeclareSymbolFont` and `\SetSymbolFont`, to specify Japanee fonts as math fonts.

Specifying `disablejfam` option in `\usepackage` prevents applying these patches. Hence one cannot write Japanese Characters in math mode directly if `disablejfam` option is specified. The code below does not work either:

```
\DeclareSymbolFont{mincho}{JY3}{mc}{m}{n}
\DeclareSymbolFontAlphabet{\mathmc}{mincho}
```

11.3 Detail of `\fontfamily` command

In this subsection, we describe when `\fontfamily`⟨*family*⟩ changes current Japanese/alphabetic font family. Basically, current Japanese fotn family is changed to ⟨*family*⟩ if it is recognized as a Japanese font family, and similar with alphabetic font family. There is a case that current Japanese/alphabetic font family are both changed to ⟨*family*⟩, and another case that ⟨*family*⟩ isn't recognized as a Japanese/alphabetic font family either.

■Recognition as Japanese font family First, Whether Japanese font family will be changed is determined in following order. This order is very similar to `\fontfamily` in pLATEX 2$_\varepsilon$, but we re-implemented in Lua. We use an auxiliary list N_J.

1. If the family ⟨*family*⟩ has been defined already by `\DeclareKanjiFamily`, ⟨*family*⟩ is recognized as a Japanese font family. Note that ⟨*family*⟩ need not be defined under *current* Japanese font encoding.
2. If the family ⟨*family*⟩ has been listed in a list N_J, this means that ⟨*family*⟩ is not a Japanese font family.
3. If the luatexja-fontspec package is loaded, we stop here, and ⟨*family*⟩ is not recognized as a Japanese font family.

   If the luatexja-fontspec package is *not* loaded, now LuaTEX-ja looks whether there exists a Japanese font encoding ⟨*enc*⟩ such that a font definition named ⟨*enc*⟩⟨*family*⟩`.fd` (the file name is all lowercase) exists. If so, ⟨*family*⟩ is recognized as a Japanese font family (the font definition file won't be loaded here). If not, ⟨*family*⟩ is not a Japanese font family, and ⟨*family*⟩ is appended to the list N_J.

■Recognition as alphabetic font family Next, whether alphabetic font family will be changed is determined in following order. We use auxiliary lists F_A and N_A,

1. If the family ⟨*family*⟩ has been listed in a list F_A, ⟨*family*⟩ is recognized as an alphabetic font family.

Table 15. strut

box	direction	width	height	depth	user command
`\ystrutbox`	yoko	0	`0.7\baselineskip`	`0.3\baselineskip`	`\ystrut`
`\tstrutbox`	tate, utod	0	`0.5\baselineskip`	`0.5\baselineskip`	`\tstrut`
`\dstrutbox`	dtou	0	`0.7\baselineskip`	`0.3\baselineskip`	`\dstrut`
`\zstrutbox`	—	0	`0.7\baselineskip`	`0.3\baselineskip`	`\zstrut`

2. If the family ⟨*family*⟩ has been listed in a list N_{A}, this means that ⟨*family*⟩ is not an alphabetic font family.
3. If there exists an alphabetic font encoding such that the family ⟨*family*⟩ has been defined under it, ⟨*family*⟩ is recognized as an alphabetic font family, and to memorize this, ⟨*family*⟩ is appended to the list F_{A}.
4. Now LuaTEX-ja looks whether there exists an alphabetic font encoding ⟨*enc*⟩ such that a font definition named ⟨*enc*⟩⟨*family*⟩`.fd` (the file name is all lowercase) exists. If so, current alphabetic font family will be changed to ⟨*family*⟩ (the font definition file won't be loaded here). If not, current alphabetic font family won't be changed, and ⟨*family*⟩ is appended to the list N_{A}.

Also, each call of `\DeclareFontFamily` *after* loading of LuaTEX-ja makes the second argument (family) is appended to the list F_{A}.

The above order is very similar to `\fontfamily` in pLATEX 2$_\varepsilon$, but more complicated (clause 3.). This is because pLATEX 2$_\varepsilon$ is a *format* however LuaTEX-ja is not, hence LuaTEX-ja does not know calls of `\DeclareFontFamily` *before* itself is loaded.

■**Remarks** Of course, there is a case that ⟨*family*⟩ is not recognized as a Japanese font family, nor an alphabetic font family. In this case, LuaTEX-ja treats "the argument ⟨*family*⟩ is wrong", so set both current alphabetic and Japanese font family to ⟨*family*⟩, to use the default family for font substitution.

11.4 Notes on `\DeclareTextSymbol`

From LATEX 2017/01/01, the standard encoding of LuaLATEX is changed to the `TU` encoding. This menas that symbols defined by `T1` and `TS1` encodings can be used without loading any package. To produces these symbols in alphabetic fonts in LuaTEX-ja, LuaTEX-ja patches `\DeclareTextSymbol`, and reloads `tuenc.def`.

Under original definition of `\DeclareTextSymbol`, internal commands which is defined by `\DeclareTextSymbol` (such as `\T1\textquotedblleft`) are *chardef* tokens. However, this no longer holds in LuaTEX-ja; for example, the meaning of `\TU\textquotedblleft` is `\ltjalchar8220␣`.

11.5 `\strutbox`

As pLATEX (2017/04/08 or later), `\strutbox` is *a macro* which is expanded to one of `\ystrutbox`, `\tstrutbox`, and `\dstrutbox` (all of them are shown in Table 15), according to the current direction. Similarly, `\strut` now uses one of these boxes.

12 expl3 interface

This section describes expl3 interfaces provided by LuaTEX-ja. All of them belong to te `platex` module, since they are provided for compatibility with Japanese pLATEX. Note that commands which are marked with dagger (" † ") are additions by LuaTEX-ja.

`\platex_direction_yoko:`, `\platex_direction_tate:`, `\platex_direction_dtou:`
Synonyms for `\yoko`, `\tate` and `\dtou`, respectively.

`\platex_if_direction_yoko_p:`

`\platex_if_direction_yoko:TF` {⟨*true code*⟩}{⟨*false code*⟩}
Tests if the current direction is *yoko* (horizontal writing).

`\platex_if_direction_tate_nomath_p:`†
`\platex_if_direction_tate_nomath:TF`† {⟨*true code*⟩}{⟨*false code*⟩}
Tests if the current direction is *tate* (vertical writing).

`\platex_if_direction_tate_math_p:`†
`\platex_if_direction_tate_math:TF`† {⟨*true code*⟩}{⟨*false code*⟩}
Tests if the current direction is *utod*.

`\platex_if_direction_tate_p:`
`\platex_if_direction_tate:TF` {⟨*true code*⟩}{⟨*false code*⟩}
Tests if the current direction is *tate* or *utod*.

`\platex_if_direction_dtou_p:`
`\platex_if_direction_dtou:TF` {⟨*true code*⟩}{⟨*false code*⟩}
Tests if the current direction is *dtou*.

`\platex_if_box_yoko_p:N` ⟨*box*⟩
`\platex_if_box_yoko:NTF` ⟨*box*⟩ {⟨*true code*⟩}{⟨*false code*⟩}
Tests if the direction of ⟨*box*⟩ is *yoko*.

`\platex_if_box_tate_nomath_p:N`† ⟨*box*⟩
`\platex_if_box_tate_nomath:NTF`† ⟨*box*⟩ {⟨*true code*⟩}{⟨*false code*⟩}
Tests if the direction of ⟨*box*⟩ is *tate*.

`\platex_if_box_tate_math_p:N`† ⟨*box*⟩
`\platex_if_box_tate_math:NTF`† ⟨*box*⟩ {⟨*true code*⟩}{⟨*false code*⟩}
Tests if the direction of ⟨*box*⟩ is *utod*.

`\platex_if_box_tate_p:N` ⟨*box*⟩
`\platex_if_box_tate:NTF` ⟨*box*⟩ {⟨*true code*⟩}{⟨*false code*⟩}
Tests if the direction of ⟨*box*⟩ is *tate* or *utod*.

`\platex_if_box_dtou_p:N` ⟨*box*⟩
`\platex_if_box_dtou:NTF` ⟨*box*⟩ {⟨*true code*⟩}{⟨*false code*⟩}
Tests if the direction of ⟨*box*⟩ is *dtou*.

13 Addon packages

LuaTEX-ja has several addon packages. These addons are written as LATEX packages, but luatexja-otf and luatexja-adjust can be loaded in plain LuaTEX by `\input`.

13.1 luatexja-fontspec

As described in Subsection 3.2, this optional package provides the counterparts for several commands defined in the fontspec package (requires fontspec v2.4). In addition to OpenType font features in the original fontspec, the following "font features" specifications are allowed for the commands of Japanese version:

`CID=`⟨*name*⟩, `JFM=`⟨*name*⟩, `JFM-var=`⟨*name*⟩
These 3 keys correspond to `cid`, `jfm` and `jfmvar` keys for `\jfont` and `\tfont` respectively. See Subsections 8.1 and 8.4 for details of `cid`, `jfm` and `jfmvar` keys.

The `CID` key is effective only when with `NoEmbed` described below. The same JFM cannot be used in both horizontal Japanese fonts and vertical Japanese fonts, hence the `JFM` key will be actually used in `YokoFeatures` and `TateFeatures` keys.

```
\jfontspec[
  YokoFeatures={Color=FF1900}, TateFeatures={Color=003FFF},
  TateFont=HaranoAjiGothic-Regular
]{HaranoAjiMincho-Regular}
\hbox{\yoko 横組のテスト}\hbox{\tate 縦組のテスト}
\addjfontfeatures{Color=00AF00}
\hbox{\yoko 横組}\hbox{\tate 縦組}
```

横組のテスト
縦組のテスト
横組
縦組

Figure 12. An example of `TateFeatures` etc.

```
\jfontspec[
  AltFont={
    {Range="4E00-"67FF, Font=HaranoAjiGothic-Regular, Color=003FFF},
    {Range="6800-"9EFF, Color=FF1900},
    {Range="3040-"306F, Font=HaranoAjiGothic-Regular, Color=35A16B},
  }
]{HaranoAjiMincho-Regular}
日本国民は、正当に選挙された国会における代表者を通じて行動し、われらとわれらの子孫のために、
諸国民との協和による成果と、わが国全土にわたつて自由のもたらす恵沢を確保し、……
```

日本国民は、正当に選挙された国会における代表者を通じて行動し、われらとわれらの子孫のために、諸国民との協和による成果と、わが国全土にわたつて自由のもたらす恵沢を確保し、……

Figure 13. An example of `AltFont`

`NoEmbed`
: By specifying this key, one can use "name-only" Japanese font which will not be embedded in the output PDF file. See Subsection 8.4.

`Kanjiskip=`⟨*bool*⟩

`TateFeatures={`⟨*features*⟩`}`, `TateFont=`⟨*font*⟩
: The `TateFeatures` key specifies font features which are only turned on in vertical writing, such as `Style=VerticalKana` (vkna feature). Similarly, the `TateFont` key specifies the Japanese font which will be used only in vertical writing. A demonstrarion is shown in Figure 12.

`YokoFeatures={`⟨*features*⟩`}`
: The `YokoFeatures` key specifies font features which are only turned on in horizontal writing,. A demonstrarion is shown in Figure 12.

`AltFont`
: As `\ltjdeclarealtfont` (Subsection 10.5) and `\DeclareAlternateKanjiFont` (Subsection 11.2), with this key, one can typeset some Japanese characters by a different font and/or using different features. The `AltFont` feature takes a comma-separated list of comma-separated lists, as the following:

```
AltFont = {
  ...
  { Range=⟨range⟩, ⟨features⟩},
  { Range=⟨range⟩, Font=⟨font name⟩, ⟨features⟩ },
  { Range=⟨range⟩, Font=⟨font name⟩> },
  ...
}
```

Each sublist should have the `Range` key (sublist which does not contain `Range` key is simply ignored). A demonstrarion is shown in Figure 13.

■Remark on `AltFont`, `YokoFeatures`, `TateFeatures` keys

In `AltFont`, `YokoFeatures`, `TateFeatures` keys, one cannot specify per-shape settings such as `BoldFeatures`. For example,

```
AltFont = {
  { Font=HogeraMin-Light, BoldFont=HogeraMin-Bold,
    Range="3000-"30FF, BoldFeatures={Color=FF1900} }
}
```

does *not* work. Instead, one have to write

```
UprightFeatures = {
  AltFont = { { Font=HogeraMin-Light, Range="3000-"30FF, } },
},
BoldFeatures = {
  AltFont =  { { Font=HogeraMin-Bold, Range="3000-"30FF, Color=FF1900 } },
}
```

On the other hand, `YokoFeatures`, `TateFeatures` and `TateFont` keys can be specified in each list in the `AltFont` key. Also, one can specify `AltFont` inside `YokoFeatures`, `TateFeatures`.

Note that features which are specified in `YokoFeatures` and `TateFeatures` are always interpreted *after* other "direction-independent" features. This explains why `\addjfontfeatures` at line 6 in Figure 12 has no effect, because a color specification is already done in `YokoFeatures` and `TateFeatures` keys.

13.2 luatexja-otf

This optional package supports typesetting glyphs by specifying a CID number. The package luatexja-otf offers the following 2 low-level commands:

`\CID{`⟨*number*⟩`}`
Typeset a glyph whose CID number is ⟨*number*⟩. If the Japanese font is neither Adobe-Japan1, Adobe-GB1, Adobe-CNS1, Adobe-Korea1, nor Adobe-KR CID-keyed font, LuaTEX-ja treats that ⟨*number*⟩ is a CID number of Adobe-Japan1 character collection, and tries to typeset a "most suitable glyph".

Note that if the Japanese font is loaded using the HarfBuzz library, this `\CID` command does not work.

`\UTF{`⟨*hex_number*⟩`}`
Typeset a character whose character code is ⟨*hex_number*⟩ (in hexadecimal). This command is similar to `\char"`⟨*hex_number*⟩, but please remind remarks below.

This package automatically loads `luatexja-ajmacros.sty`, which is slightly modified version of `ajmacros.sty`[11]. Hence one can use macros which sre defined in `ajmacros.sty`, such as `\aj半角`.

■Remarks Characters by `\CID` and `\UTF` commands are different from ordinary characters in the following points:

- Always treated as **JAchar**s.
- In vertical direction, `vert`/`vrt2` feature are automatically applied to characters by `\UTF`, regardless these feature are not activated in current Japanese font.
- Processes for supporting other OpenType features (for example, glyph replacement and kerning) by the luaotfload package is not performed to these characters.

■Additional syntax of JFM The package luatexja-otf extends the syntax of JFM; the entries of `chars` table in JFM now allows a string in the form `'AJ1-xxx'`, which stands for the character whose CID number in Adobe-Japan1 is `xxx`.

This extended notation is used in the standard JFM `jfm-ujis.lua` to typeset halfwidth Hiragana glyphs (CID 516–598) in halfwidth.

no adjustment	以上の原理は，「包除原理」とよく呼ばれるが
without priority	以上の原理は，「包除原理」とよく呼ばれるが
with priority	以上の原理は，「包除原理」とよく呼ばれるが

The value of kanjiskip is 0 pt$^{+1/5\,\mathrm{em}}_{-1/5\,\mathrm{em}}$ in this figure, for making the difference obvious.

Figure 14. Line adjustment

13.3 luatexja-adjust

(see Japanese version of this manual)

13.4 luatexja-ruby

This addon package provides functionality of "ruby" (*furigana*) annotations using callbacks of LuaTEX-ja. There is no detailed manual of luatexja-ruby.sty in English. (Japanese manual is another PDF file, `luatexja-ruby.pdf`.)

Group-ruby By default, ruby characters (the second argument of \ruby) are attached to base characters (the first argument), as one object. This type of ruby is called *group-ruby*.

```
東西線\ruby{妙典}{みようでん}駅は……\\
東西線の\ruby{妙典}{みようでん}駅は……\\
東西線の\ruby{妙典}{みようでん}という駅……\\
東西線\ruby{葛西}{かさい}駅は……
```

みようでん
東西線妙典駅は……
みようでん
東西線の妙典駅は……
みようでん
東西線の妙典という駅……
かさい
東西線葛西駅は……

As the above example, ruby hangover is allowed on the Hiragana before/after its base characters.

Mono-ruby To attach ruby characters to each base characters (*mono-ruby*), one should use \ruby multiple times:

```
東西線の\ruby{妙}{みよう}\ruby{典}{でん}駅は……
```

みようでん
東西線の妙典駅は……

Jukugo-ruby Vertical bar | denotes a boundary of *groups*.

```
\ruby{妙|典}{みよう|でん}\
\ruby{葛|西}{か|さい}\
\ruby{神楽|坂}{かぐら|ざか}
```

みようでん かさい かぐらざか
妙典 葛西 神楽坂

If there are multiple groups in one \ruby call, A linebreak between two groups is allowed.

```
\vbox{\hsize=6\zw\noindent
  \hbox to 2.5\zw{}\ruby{京|急|蒲|田}{けい|きゆう|かま|た}
  \hbox to 2.5\zw{}\ruby{京|急|蒲|田}{けい|きゆう|かま|た}
  \hbox to   3\zw{}\ruby{京|急|蒲|田}{けい|きゆう|かま|た}
}
```

けいきゆうかま
京急蒲
た　けいきゆう
田　京急
かまた　けい
蒲田　京
きゆうかまた
急蒲田

If the width of ruby characters are longer than that of base characters, \ruby automatically selects the appropriate form among the line-head form, the line-middle form, and the line-end form.

```
\vbox{\hsize=8\zw\noindent
\null\kern3\zw ……を\ruby{承}{うけたまわ}る
      \kern1\zw ……を\ruby{承}{うけたまわ}る\\
\null\kern5\zw ……を\ruby{承}{うけたまわ}る
}
```

うけたまわ
……を承
うけたまわ
る　……を承る
……を
うけたまわ
承る

[11]Useful macros by iNOUE Koichi!, for the japanese-otf package.

13.5 `lltjext.sty`

pLaTeX supplies additional macros for vertical writing in the plext package. The lltjext package which we want to describe here is the LuaTeX-ja counterpart of the plext package.

`tabular`, `array`, `minipage` environments
: These environments are extended by `<dir>`, which specifies the direction, as follows:

```
\begin{tabular}<dir>[pos]{table spec} ... \end{tabular}
\begin{array}<dir>[pos]{table spec}   ... \end{array}
\begin{minipage}<dir>[pos]{width}     ... \end{minipage}
```

This option permits one of the following five values. If none of them is specified, the direction inside the environment is same as that outside the enviromnent.

y *yoko* direction (horizontal writing)

t *tate* direction (vertical writing)

z *utod* direction if direction outside the env. is *tate*.

d *dtou* direction

u *utod* direction

`\parbox<`⟨*dir*⟩`>[`⟨*pos*⟩`]{`⟨*width*⟩`}{`⟨*contents*⟩`}`
: `\parbox` command is also extended by `<`⟨*dir*⟩`>`.

`\pbox<`⟨*dir*⟩`>[`⟨*width*⟩`][`⟨*pos*⟩`]{`⟨*contents*⟩`}`
: This commands typeset ⟨*contents*⟩ in LR-mode, in ⟨*dir*⟩ direction. If ⟨*width*⟩ is positive, the width of the box becomes this ⟨*width*⟩. In this case, ⟨*contents*⟩ will be aligned to left (when ⟨*pos*⟩ is `l`), center (`c`), or right (`r`).

`picture` environment
: `picture` environment also extended by `<`⟨*dir*⟩`>`, as follows:

```
\begin{picture}<dir>(x_size, y_size)(x_offset,y_offset)
  ...
\end{picture}
```

`\rensuji[`⟨*pos*⟩`]{`⟨*contents*⟩`}`, `\rensujiskip`

`\Kanji{`⟨*counter_name*⟩`}`

`\kasen{`⟨*contents*⟩`}`, `\bou{`⟨*contents*⟩`}`, `\boutenchar`

参照番号

13.6 luatexja-preset

As described in Subsection 3.3, One can load the luatexja-preset package to use several “presets” of Japanese fonts. This package provides functions in a part of japanese-otf package (changing fonts) and a part of PXchfon package (presets) by Takayuki Yato.

Options which are given in `\usepackage` but not described in this subsection are simply passed to the luatexja-fontspec[12]. For example, the line 5 in below example is equivalent to lines 1–3.

```
\usepackage[no-math]{fontspec}
\usepackage[match]{luatexja-fontspec}
\usepackage[kozuka-pr6n]{luatexja-preset}
%%--------
\usepackage[no-math,match,kozuka-pr6n]{luatexja-preset}
```

[12] if `nfssonly` option is *not* specified; in this case these options are simply ignored.

13.6.1 General Options

`fontspec` (enabled by default)
With this option, Japanese fonts are selected using functionality of the luatexja-fontspec package. This means that the fontspec package is automatically loaded by this package.

If you need to pass some options to fontspec, you can load fontspec manually before luatexja-preset:

```
\usepackage[no-math]{fontspec}
\usepackage[...]{luatexja-preset}
```

`nfssonly`
With this option, selecting Japanese fonts won't be performed using the functionality of the fontspec package, but only standard NFSS2 (hence without `\addjfontfeatures` etc.). This option is ignored when luatexja-fontspec package is loaded.

When this option is specified, fontspec and luatexja-fontspec are *not* loaded by default. Nevertheless, the packagefontspec can coexist with the option, as the following:

```
\usepackage{fontspec}
\usepackage[hiragino-pron,nfssonly]{luatexja-preset}
```

In this case, one can use `\setmainfont` etc. to select *alphabetic* fonts.

`match`
If this option is specified, usual family-changing commands such as `\rmfamily`, `\textrm`, `\sffamily`, … also change Japanese font family. This option is passed to luatexja-fontspec, if `fontspec` option is specifed.

`nodeluxe` (enabled by default)
The nagation of `deluxe` option. Use one-weighted *mincho* and *gothic* font families. This means that `\mcfamily\bfseries`, `\gtfamily\bfseries` and `\gtfamily\mdseries` use the same font.

`deluxe`
Use the mincho family with three weights (light, medium, and bold), the gothic family with three weights (medium, bold, and extra bold), and *rounded gothic*[13]. Mincho light and gothic extra bold can be by `\mcfamily\ltseries` and `\gtfamily\ebseries`, respectively.

- Some presets do not have the light weight of mincho. In this case, we substitute the medium weight for the light weight.
- luatexja-preset does not produce an error (only produces a warning), even if (one of) fonts for `\mcfamily\ltseries`, `\gtfamily\ebseries`, `\mgfamily` do not exist.

`expert`
Use horizontal/vertical kana alternates, and define a command `\rubyfamily` to use kana characters designed for ruby.

`bold`
Substitute bold series of gothic for medium series of gothic and bold series of mincho. If `nodeluxe` option is enabled, medium series of gothic is also changed, since we use same font for both series of gothic.

`jis90`, `90jis`
Use JIS X 0208:1990 glyph variants if possible.

`jis2004`, `2004jis`
Use JIS X 0213:2004 glyph variants if possible.

`jfm_yoko=`⟨*jfm*⟩
Use the JFM `jfm-`⟨*jfm*⟩`.lua` for horizontal direction, instead of `jfm-ujis.lua` (default JFM).

`jfm_tate=`⟨*jfm*⟩
Use the JFM `jfm-`⟨*jfm*⟩`.lua` for vertical direction, instead of `jfm-ujisv.lua` (default JFM).

[13]Provided by `\mgfamily` and `\textmg`, because "rounded gothic" is called *maru gothic* (丸ゴシック) in Japanese.

`jis` Same as `jfm_yoko=jis`.

Note that `jis90`, `90jis`, `jis2004` and `2004jis` only affect with mincho, gothic (and, possibly rounded gothic) families defined by this package. We didn't taken account of when more than one options among them are specified.

13.6.2 Presets which support multi weights

Besides `bizud`, `haranoaji`, `morisawa-pro`, and `morisawa-pr6n` presets, fonts are specified by font name, not by file name. In following tables, starred fonts (e.g. KozGo…-Regular) are used for medium series of *gothic*, *if and only if* `deluxe` *option is specified.*

`kozuka-pro` Kozuka Pro (Adobe-Japan1-4) fonts.

`kozuka-pr6` Kozuka Pr6 (Adobe-Japan1-6) fonts.

`kozuka-pr6n` Kozuka Pr6N (Adobe-Japan1-6, JIS04-savvy) fonts.

Kozuka Pro/Pr6N fonts are bundled with Adobe's software, such as Adobe InDesign. There is not rounded gothic family in Kozuka fonts.

family	**series**	`kozuka-pro`	`kozuka-pr6`	`kozuka-pr6n`
mincho	light	KozMinPro-Light	KozMinProVI-Light	KozMinPr6N-Light
	medium	KozMinPro-Regular	KozMinProVI-Regular	KozMinPr6N-Regular
	bold	KozMinPro-Bold	KozMinProVI-Bold	KozMinPr6N-Bold
gothic	medium	KozGoPro-Regular* KozGoPro-Medium	KozGoProVI-Regular* KozGoProVI-Medium	KozGoPr6N-Regular* KozGoPr6N-Medium
	bold	KozGoPro-Bold	KozGoProVI-Bold	KozGoPr6N-Bold
	extra bold	KozGoPro-Heavy	KozGoProVI-Heavy	KozGoPr6N-Heavy
rounded gothic		KozGoPro-Heavy	KozGoProVI-Heavy	KozGoPr6N-Heavy

`hiragino-pro` Hiragino Pro (Adobe-Japan1-5) fonts.

`hiragino-pron` Hiragino ProN (Adobe-Japan1-5, JIS04-savvy) fonts.

Hiragino fonts (except Hiragino Mincho W2) are bundled with Mac OS X 10.5 or later. Note that fonts for gothic extra bold (HiraKakuStd[N]-W8) only contains characters in Adobe-Japan1-3 character collection, while others contains those in Adobe-Japan1-5 character collection.

family	**series**	`hiragino-pro`	`hiragino-pron`
mincho	light	Hiragino Mincho Pro W2	Hiragino Mincho ProN W2
	medium	Hiragino Mincho Pro W3	Hiragino Mincho ProN W3
	bold	Hiragino Mincho Pro W6	Hiragino Mincho ProN W6
gothic	medium	Hiragino Kaku Gothic Pro W3* Hiragino Kaku Gothic Pro W6	Hiragino Kaku Gothic ProN W3* Hiragino Kaku Gothic ProN W6
	bold	Hiragino Kaku Gothic Pro W6	Hiragino Kaku Gothic ProN W6
	extra bold	Hiragino Kaku Gothic Std W8	Hiragino Kaku Gothic StdN W8
rounded gothic		Hiragino Maru Gothic Pro W4	Hiragino Maru Gothic ProN W4

`bizud` BIZ UD fonts (by Morisawa Inc.) bundled with Windows 10 October 2018 Update.

family	**series**	
mincho		`BIZ-UDMinchoM.ttc`
gothic	medium	`BIZ-UDGothicR.ttc`
	bold	`BIZ-UDGothicB.ttc`
	extra bold	`BIZ-UDGothicB.ttc`
rounded gothic		`BIZ-UDGothicB.ttc`

`morisawa-pro` Morisawa Pro (Adobe-Japan1-4) fonts.
`morisawa-pr6n` Morisawa Pr6N (Adobe-Japan1-6, JIS04-savvy) fonts.

family	**series**	`morisawa-pro`	`morisawa-pr6n`
mincho	medium	`A-OTF-RyuminPro-Light.otf`	`A-OTF-RyuminPr6N-Light.otf`
	bold	`A-OTF-FutoMinA101Pro-Bold.otf`	`A-OTF-FutoMinA101Pr6N-Bold.otf`
gothic	medium	`A-OTF-GothicBBBPro-Medium.otf`	`A-OTF-GothicBBBPr6N-Medium.otf`
	bold	`A-OTF-FutoGoB101Pro-Bold.otf`	`A-OTF-FutoGoB101Pr6N-Bold.otf`
	extra bold	`A-OTF-MidashiGoPro-MB31.otf`	`A-OTF-MidashiGoPr6N-MB31.otf`
rounded gothic		`A-OTF-Jun101Pro-Light.otf`	`A-OTF-ShinMGoPr6N-Light.otf`

`yu-win` Yu fonts bundled with Windows 8.1.
`yu-win10` Yu fonts bundled with Windows 10.
`yu-osx` Yu fonts bundled with OSX Mavericks.

family	**series**	`yu-win`	`yu-win10`	`yu-osx`
mincho	light	YuMincho-Light	YuMincho-Light	(YuMincho Medium)
	medium	YuMincho-Regular	YuMincho-Regular	YuMincho Medium
	bold	YuMincho-Demibold	YuMincho-Demibold	YuMincho Demibold
gothic	medium	YuGothic-Regular*	YuGothic-Regular*	YuGothic Medium*
		YuGothic-Regular	YuGothic-Medium	YuGothic Medium
	bold	YuGothic-Bold	YuGothic-Bold	YuGothic Bold
	extra bold	YuGothic-Bold	YuGothic-Bold	YuGothic Bold
rounded gothic		YuGothic-Bold	YuGothic-Bold	YuGothic Bold

`moga-mobo` MogaMincho, MogaGothic, and MoboGothic.
`moga-mobo-ex` MogaExMincho, MogaExGothic, and MoboExGothic.

These fonts can be downloaded from `http://yozvox.web.fc2.com/`.

family	**series**	**default, `90jis` option**	**`jis2004` option**
mincho	medium	Moga90Mincho	MogaMincho
	bold	Moga90Mincho Bold	MogaMincho Bold
gothic	medium	Moga90Gothic	MogaGothic
	bold	Moga90Gothic Bold	MogaGothic Bold
	extra bold	Moga90Gothic Bold	MogaGothic Bold
rounded gothic		Mobo90Gothic	MoboGothic

When `moga-mobo-ex` is specified, the font "MogaEx90Mincho" etc. are used.

`ume` Ume Mincho and Ume Gothic.

These fonts can be downloaded from
`https://ja.osdn.net/projects/ume-font/wiki/FrontPage`.

family	**series**	**default**
mincho	medium	Ume Mincho
	bold	Ume Mincho
gothic	medium	Ume Gothic*
		Ume Gothic O5
	bold	Ume Gothic O5
	extra bold	Ume Gothic O5
rounded gothic		Ume Gothic O5

`sourcehan` Source Han Serif and Source Han Sans fonts (Language-specific OTF *or* OTC)

`sourcehan-jp` Source Han Serif JP and Source Han Sans JP fonts (Region-specific Subset OTF)

family	**series**	`sourcehan`	`sourcehan-jp`
mincho	light	Source Han Serif Light	Source Han Serif JP Light
	medium	Source Han Serif Regular	Source Han Serif JP Regular
	bold	Source Han Serif Bold	Source Han Serif JP Bold
gothic	medium	Source Han Sans Regular*	Source Han Sans JP Regular*
		Source Han Sans Medium	Source Han Sans JP Medium
	bold	Source Han Sans Bold	Source Han Sans JP Bold
	extra bold	Source Han Sans Heavy	Source Han Sans JP Heavy
rounded gothic		Source Han Sans Medium	Source Han Sans JP Medium

`noto-otc` Noto Serif CJK and Noto Sans CJK fonts (OTC)

`noto-otf`, `noto` Noto Serif CJK and Noto Sans CJK fonts (Language-specific OTF)

`noto-jp` Noto Serif CJK and Noto Sans CJK fonts (Region-specific subset OTF)

family	**series**	`noto-otc`	`noto-otf`, `noto`	`noto-jp`
mincho	light	Noto Serif CJK Light	Noto Serif CJK JP Light	Noto Serif JP Light
	medium	Noto Serif CJK Regular	Noto Serif CJK JP Regular	Noto Serif JP Regular
	bold	Noto Serif CJK Bold	Noto Serif CJK JP Bold	Noto Serif JP Bold
gothic	medium	Noto Sans CJK Regular*	Noto Sans CJK JP Regular*	Noto Sans JP Regular*
		Noto Sans CJK Medium	Noto Sans CJK JP Medium	Noto Sans JP Medium
	bold	Noto Sans CJK Bold	Noto Sans CJK JP Bold	Noto Sans JP Bold
	extra bold	Noto Sans CJK Black	Noto Sans CJK JP Black	Noto Sans JP Black
rounded gothic		Noto Sans CJK Medium	Noto Sans CJK JP Medium	Noto Sans JP Medium

`haranoaji` Harano Aji Fonts.

These fonts can be downloaded from `https://github.com/trueroad/HaranoAjiFonts`. There is not rounded gothic family in Harano Aji Fonts.

family	**series**	`haranoaji`
mincho	light	HaranoAjiMincho-Light.otf
	medium	HaranoAjiMincho-Regular.otf
	bold	HaranoAjiMincho-Bold.otf
gothic	medium	HaranoAjiGothic-Regular.otf*
		HaranoAjiGothic-Medium.otf
	bold	HaranoAjiGothic-Bold.otf
	extra bold	HaranoAjiGothic-Heavy.otf
rounded gothic		HaranoAjiGothic-Medium.otf

13.6.3 Presets which do not support multi weights

Next, we describe settings for using only single weight.

	`noembed`	`ipa`	`ipaex`	`ms`
mincho	Ryumin-Light (non-embedded)	IPA Mincho	IPAex Mincho	MS Mincho
gothic	GothicBBB-Medium (non-embedded)	IPA Gothic	IPAex Gothic	MS Gothic

13.6.4 Presets which use HG fonts

We can use HG fonts bundled with Microsoft Office for realizing multiple weights. In the table below, starred fonts (e.g., IPA Gothic*) are used only if `jis2004` or `nodeluxe` option is spefified.

family	**series**	`ipa-hg`	`ipaex-hg`	`ms-hg`
mincho	medium	IPA Mincho	IPAex Mincho	MS Mincho
	bold	HG Mincho E	HG Mincho E	HG Mincho E
gothic	medium	IPA Gothic* HG Gothic M	IPAex Gothic* HG Gothic M	MS Gothic* HG Gothic M
	bold	HG Gothic E	HG Gothic E	HG Gothic E
	extra bold	HG Soei Kaku Gothic UB	HG Soei Kaku Gothic UB	HG Soei Kaku Gothic UB
rounded gothic		HG MaruGothic M PRO	HG MaruGothic M PRO	HG MaruGothic M PRO

Note that HG Mincho E, HG Gothic E, HG Soei Kaku Gothic UB, and HG Maru Gothic PRO are internally specified by:

default by font name (`HGMinchoE`, etc.).

`jis90`, `90jis` by file name (`hgrme.ttc`, `hgrge.ttc`, `hgrsgu.ttc`, `hgrsmp.ttf`).

`jis2004`, `2004jis` by file name (`hgrme04.ttc`, `hgrge04.ttc`, `hgrsgu04.ttc`, `hgrsmp04.ttf`).

13.6.5 Define/Use Custom Presets

From version 20170904.0, one can define new presets using `\ltjnewpreset`, and use them by `\ltjapplypreset`. These two commands can only be used in the preamble.

`\ltjnewpreset{`⟨*name*⟩`}{`⟨*specification*⟩`}`

Define new preset ⟨*name*⟩. This <name> cannot be same as other presets, options described in Subsection 13.6.1, nor following 13 strings:

```
mc mc-l mc-m mc-b mc-bx gt gt-u gt-d gt-m gt-b gt-bx gt-eb mg-m
```

⟨*specification*⟩ is a comma-separated list which consists of other presets and/or the following keys:

`mc-l=`⟨*font*⟩ mincho light

`mc-m=`⟨*font*⟩ mincho medium

`mc-b=`⟨*font*⟩ mincho bold

`mc-bx=`⟨*font*⟩ synonym for `mc-b=`⟨*font*⟩

`gt-u=`⟨*font*⟩ gothic, when `deluxe` option is not specified.

`gt-d=`⟨*font*⟩ gothic medium, when `deluxe` option is specified.

`gt-m=`⟨*font*⟩ gothic medium. This key is equivalent to "`gt-u=`⟨*font*⟩`, gt-d`⟨*font*⟩".

`gt-b=`⟨*font*⟩ gothic bold
Note that this key also specifies mincho bold if `bold` option is specified.

`gt-bx=`⟨*font*⟩ synonym for `gt-b=`⟨*font*⟩

`gt-eb=`⟨*font*⟩ gothic extra bold

`mg-m=`⟨*font*⟩ rounded gothic

`mc=`⟨*font*⟩ Equivalent to

`mc-l=`⟨*font*⟩`, mc-m=`⟨*font*⟩`, mc-b=`⟨*font*⟩

`gt=`⟨*font*⟩ Equivalent to

`gt-u=`⟨*font*⟩`, gt-d=`⟨*font*⟩`, gt-b=`⟨*font*⟩`, gt-eb=`⟨*font*⟩

`\ltjnewpreset*{`⟨*name*⟩`}{`⟨*specification*⟩`}`

Almost same as `\ltjnewpreset`. However, if ⟨*name*⟩ matches a preset which already defined, this command simply overwrite it.

`\ltjapplypreset{⟨name⟩}`
: Set Japanese font families using preset ⟨*name*⟩.

Note that `\ltjnewpreset` does not "expand" the definition to define a preset. This means that one can write as the following:

```
\ltjnewpreset{hoge}{piyo,mc-b=HiraMinProN-W6}
\ltjnewpreset{piyo}{mg-m=HiraMaruProN-W4}
\ltjapplypreset{hoge}
```

■**Restrictions** Presets which are defined by `\ltjnewpreset` have following restrictions:

- One cannot specify non-embedded fonts (such as Ryumin-Light).
- Some presets, such as `ipa-hg`, have a feature that fonts are changed according to whether `90jis` or `jis2004` is speified. This feature is not usable in presets which are defined by `\ltjnewpreset`.

Part III

Implementations

14 Storing Parameters

14.1 Used dimensions, attributes and whatsit nodes

Here the following is the list of dimensions and attributes which are used in LuaTeX-ja.

`\jQ` (dimension) `\jQ` is equal to 1 Q = 0.25 mm, where "Q" (also called "級") is a unit used in Japanese phototypesetting. So one should not change the value of this dimension.

`\jH` (dimension) There is also a unit called "歯" which equals to 0.25 mm and used in Japanese phototypesetting. This `\jH` is the same `\dimen` register as `\jQ`.

`\ltj@dimen@zw` (dimension) A temporal register for the "full-width" of current Japanese font. The command `\zw` sets this register to the correct value, and "return" this register itself.

`\ltj@dimen@zh` (dimension) A temporal register for the "full-height" (usually the sum of height of imaginary body and its depth) of current Japanese font. The command `\zh` sets this register to the correct value, and "return" this register itself.

`\jfam` (attribute) Current number of Japanese font family for math formulas.

`\ltj@curjfnt` (attribute) If this attribute is a positive number, it stores the font number of current Japanese font for horizontal direction. If this attribute is negative, it means that the Japanese font for horizontal direction is not loaded—LuaTeX-ja only knows its size and JFM.

`\ltj@curtfnt` (attribute) Similar to `\ltj@curjfnt`, but with current Japanese font for vertical direction.

`\ltj@charclass` (attribute) The character class of a **JAchar**. This attribute is only set on a *glyph_node* which contains a **JAchar**.

`\ltj@yablshift` (attribute) The amount of shifting the baseline of alphabetic fonts in scaled point (2^{-16} pt). "unset" means zero.

`\ltj@ykblshift` (attribute) The amount of shifting the baseline of Japanese fonts in scaled point (2^{-16} pt).

`\ltj@tablshift` (attribute)

`\ltj@tkblshift` (attribute)

`\ltj@autospc` (attribute) Whether the auto insertion of kanjiskip is allowed at the node. 0 means "not allowed", and the other value (including "unset") means "allowed".

`\ltj@autoxspc` (attribute) Whether the auto insertion of xkanjiskip is allowed at the node. 0 means "not allowed", and the other value (including "unset") means "allowed".

`\ltj@icflag` (attribute) An attribute for distinguishing "kinds" of a node. One of the following value is assigned to this attribute:

***italic* (1)** Kerns from italic correction (`\/`), or from kerning information of a Japanese font. These kerns are "ignored" in the insertion process of **JAglue**, unlike explicit `\kern`.

***packed* (2)**

***kinsoku* (3)** Penalties inserted for the word-wrapping process (*kinsoku shori*) of Japanese characters.

***from_jfm*–(*from_jfm* + 63) (4–67)** Glues/kerns from JFM.

***kanji_skip* (68), *kanji_skip_jfm* (69)** Glues from kanjiskip.

xkanji_skip **(70),** ***xkanji_skip_jfm*** **(71)** Glues from xkanjiskip.

processed **(73)** Nodes which is already processed by ….

ic_processed **(74)** Glues from an italic correction, but already processed in the insertion process of **JAglue**s.

boxbdd **(75)** Glues/kerns that inserted just the beginning or the ending of an hbox or a paragraph.

special_jaglue **(76)** Glues from `\insert[x]kanjiskip`.

`\ltj@kcat`*i* (attribute) Where *i* is a natural number which is less than 7. These 7 attributes store bit vectors indicating which character block is regarded as a block of **JAchar**s.

`\ltj@dir` (attribute) ***dir_node_auto*** **(128)**

dir_node_manual **(256)**

`\ltjlineendcomment` (counter)

Furthermore, LuaTEX-ja uses several user-defined whatsit nodes for internal processing. All those nodes except *direction* whatsits store a natural number (hence its `type` is 100). *direction* whatsits store a node list, hence its `type` is 110. Their `user_id` (used for distinguish user-defined whatsits) are allocated by `luatexbase.newuserwhatsitid`.

inhibitglue Nodes for indicating that `\inhibitglue` is specified. The `value` field of these nodes doesn't matter.

stack_marker Nodes for LuaTEX-ja's stack system (see the next subsection). The `value` field of these nodes is current group level.

char_by_cid Nodes for **JAchar** which processes by luaotfload won't be applied, and the character code is stored in the `value` field. Each node of this type are converted to a *glyph_node after* processes by luaotfload. Nodes of this type is used in `\CID` and `\UTF`.

replace_vs Similar to *char_by_cid* whatsits above. These nodes are for **ALchar** which the callback process of luaotfload won't be applied.

begin_par Nodes for indicating beginning of a paragraph. A paragraph which is started by `\item` in list-like environments has a horizontal box for its label before the actual contents. So …

direction

These whatsits will be removed during the process of inserting **JAglue**s.

14.2 Stack system of LuaTEX-ja

■Background LuaTEX-ja has its own stack system, and most parameters of LuaTEX-ja are stored in it. To clarify the reason, imagine the parameter kanjiskip is stored by a skip, and consider the following source:

```
\ltjsetparameter{kanjiskip=0pt}ふがふが.%
\setbox0=\hbox{%
  \ltjsetparameter{kanjiskip=5pt}ほげほげ}
\box0.ぴよぴよ\par
```

ふがふが. ほ げ ほ げ. ぴよぴよ

As described in Subsection 9.1, the only effective value of kanjiskip in an hbox is the latest value, so the value of kanjiskip which applied in the entire hbox should be 5 pt. However, by the implementation method of LuaTEX, this “5 pt” cannot be known from any callbacks. In the `tex/packaging.w`, which is a file in the source of LuaTEX, there are the following codes:

```
void package(int c)
{
    scaled h;                /* height of box */
    halfword p;              /* first node in a box */
    scaled d;                /* max depth */
```

```
    int grp;
    grp = cur_group;
    d = box_max_depth;
    unsave();
    save_ptr -= 4;
    if (cur_list.mode_field == -hmode) {
        cur_box = filtered_hpack(cur_list.head_field,
                                 cur_list.tail_field, saved_value(1),
                                 saved_level(1), grp, saved_level(2));
        subtype(cur_box) = HLIST_SUBTYPE_HBOX;
```

Notice that `unsave()` is executed *before* `filtered_hpack()`, where `hpack_filter` callback is executed) here. So "5 pt" in the above source is orphaned at `unsave()`, and hence it can't be accessed from `hpack_filter` callback.

■Implementation The code of stack system is based on that in a post of Dev-luatex mailing list[14].

These are two TeX count registers for maintaining information: `\ltj@@stack` for the stack level, and `\ltj@@group@level` for the TeX's group level when the last assignment was done. Parameters are stored in one big table named `charprop_stack_table`, where `charprop_stack_table[`*i*`]` stores data of stack level *i*. If a new stack level is created by `\ltjsetparameter`, all data of the previous level is copied.

To resolve the problem mentioned in above paragraph "Background", LuaTeX-ja uses another trick. When the stack level is about to be increased, a whatsit node whose type, subtype and value are 44 (*user_defined*), *stack_marker* and the current group level respectively is appended to the current list (we refer this node by *stack_flag*). This enables us to know whether assignment is done just inside a hbox. Suppose that the stack level is s and the TeX's group level is t just after the hbox group, then:

- If there is no *stack_flag* node in the list of the contents of the hbox, then no assignment was occurred inside the hbox. Hence values of parameters at the end of the hbox are stored in the stack level s.
- If there is a *stack_flag* node whose value is $t + 1$, then an assignment was occurred just inside the hbox group. Hence values of parameters at the end of the hbox are stored in the stack level $s + 1$.
- If there are *stack_flag* nodes but all of their values are more than $t + 1$, then an assignment was occurred in the box, but it is done in more internal group. Hence values of parameters at the end of the hbox are stored in the stack level s.

Note that to work this trick correctly, assignments to `\ltj@@stack` and `\ltj@@group@level` have to be local always, regardless the value of `\globaldefs`. To solve this problem, we use another trick: the assignment `\directlua{tex.globaldefs=0}` is always local.

14.3 Lua functions of the stack system

In this subsection, we will see how a user use LuaTeX-ja's stack system to store some data which obeys the grouping of TeX.

The following function can be used to store data into a stack:

```
luatexja.stack.set_stack_table(index, <any> data)
```

Any values which except `nil` and NaN are usable as *index*. However, a user should use only negative integers or strings as *index*, since natural numbers are used by LuaTeX-ja itself. Also, whether *data* is stored locally or globally is determined by `luatexja.isglobal` (stored globally if and only if `luatexja.isglobal == 'global'`).

Stored data can be obtained as the return value of

```
luatexja.stack.get_stack_table(index, <any> default, <number> level)
```

where *level* is the stack level, which is usually the value of `\ltj@@stack`, and *default* is the default value which will be returned if no values are stored in the stack table whose level is *level*.

[14] `[Dev-luatex] tex.currentgrouplevel`, a post at 2008/8/19 by Jonathan Sauer.

```
\protected\def\ltj@setpar@global{%
  \relax\ifnum\globaldefs>0\directlua{luatexja.isglobal='global'}%
    \else\directlua{luatexja.isglobal=''}\fi
}
\protected\def\ltjsetparameter#1{%
  \ltj@setpar@global\setkeys[ltj]{japaram}{#1}\ignorespaces}
\protected\def\ltjglobalsetparameter#1{%
  \relax\ifnum\globaldefs<0\directlua{luatexja.isglobal=''}%
    \else\directlua{luatexja.isglobal='global'}\fi%
  \setkeys[ltj]{japaram}{#1}\ignorespaces}
```

Figure 15. Definiton of parameter setting commands

14.4 Extending Parameters

Keys for `\ltjsetparameter` and `\ltjgetparameter` can be extended, as in luatexja-adjust.

■Setting parameters Figure 15 shows the *most outer* definition of two commands, `\ltjsetparameter` and `\ltjglobalsetparameter`. Most important part is the last `\setkeys`, which is offered by the xkeyval package.

Hence, to add a key in `\ltjsetparameter`, one only have to add a key whose prefix is `ltj` and whose family is `japaram`, as the following.

```
\define@key[ltj]{japaram}{...}{...}
```

`\ltjsetparameter` and `\ltjglobalsetparameter` automatically sets `luatexja.isglobal`. Its meaning is the following.

$$\texttt{luatexja.isglobal} = \begin{cases} \texttt{'global'} & \text{(global assignment),} \\ \texttt{''} & \text{(local assignment).} \end{cases} \tag{1}$$

This is determined not only by command name (`\ltjsetparameter` or `\ltjglobalsetparameter`), but also by the value of `\globaldefs`.

■Getting parameters `\ltjgetparameter` is implemented by a Lua script.

For parameters that do not need additional arguments, one only have to define a function in the table `luatexja.unary_pars`. For example, with the following function, `\ltjgetparameter{hoge}` returns a *string* `42`.

```
function luatexja.unary_pars.hoge (t)
  return 42
end
```

Here the argument of `luatexja.unary_pars.hoge` is the stack level of LuaTeX-ja's stack system (see Subsection 14.2).

On the other hand, for parameters that need an additional argument (this must be an integer), one have to define a function in `luatexja.binary_pars` first. For example,

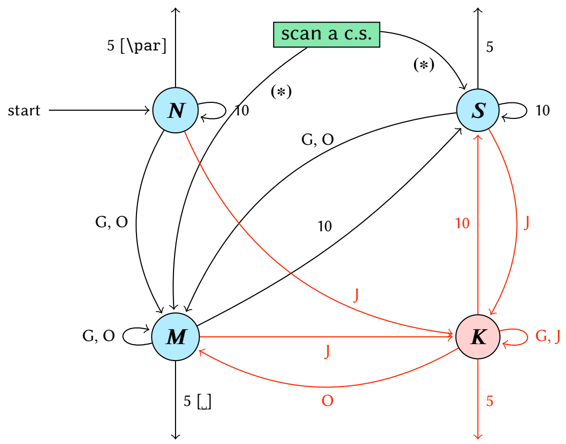
```
function luatexja.binary_pars.fuga (c, t)
  return tostring(c) .. ', ' .. tostring(42)
end
```

Here the first argument t is the stack level, as before. The second argument c is just the second argument of `\ltjgetparameter`.

For parameters that need an additional argument, one also have to execute the TeX code like

```
\ltj@@decl@array@param{fuga}
```

to indicate that "the parameter `fuga` needs an additional argument".



G Beginning of group (usually {)
and ending of group (usually }).

J Japanese characters.

5 *end-of-line* (usually `^^J`).

10 space (usually ␣).

O other characters, whose category code is in $\{3, 4, 6, 7, 8, 11, 12, 13\}$.

[␣], [\par] emits a space, or `\par`.

- We omitted about category codes 9 (*ignored*), 14 (*comment*), and 15 (*invalid*) from the above diagram. We also ignored the input like "`^^A`" or "`^^df`".
- When a character whose category code is 0 (*escape character*) is seen by TeX, the input processor scans a control sequence (scan a c.s.). These paths are not shown in the above diagram.

  After that, the state is changed to State S (skipping blanks) in most cases, but to State M (middle of line) sometimes.

Figure 16. State transitions of pTeX's input processor

15 Linebreak after a Japanese Character

15.1 Reference: behavior in pTeX

In pTeX, a line break after a Japanese character doesn't emit a space, since words are not separated by spaces in Japanese writings. However, this feature isn't fully implemented in LuaTeX-ja due to the specification of callbacks in LuaTeX. To clarify the difference between pTeX and LuaTeX, We briefly describe the handling of a line break in pTeX, in this subsection.

pTeX's input processor can be described in terms of a finite state automaton, as that of TeX in Section 2.5 of [1]. The internal states are as follows:

- State N: new line
- State S: skipping spaces
- State M: middle of line
- State K: after a Japanese character

The first three states—N, S, and M—are as same as TeX's input processor. State K is similar to state M, and is entered after Japanese characters. The diagram of state transitions are indicated in Figure 16. Note that pTeX doesn't leave state K after "beginning/ending of a group" characters.

15.2 Behavior in LuaTeX-ja

States in the input processor of LuaTeX is the same as that of TeX, and they can't be customized by any callbacks. Hence, we can only use `process_input_buffer` and `token_filter` callbacks for to suppress a space by a line break which is after Japanese characters.

However, `token_filter` callback cannot be used either, since a character in category code 5 (*end-of-line*) is converted into an space token *in the input processor*. So we can use only the `process_input_buffer` callback. This means that suppressing a space must be done *just before* an input line is read.

Considering these situations, handling of an end-of-line in LuaTeX-ja are as follows:

A character whose character code is `\ltjlineendcomment`[15] is appended to an input line, *before LuaTeX actually process it*, if and only if the following three conditions are satisfied:

1. The category code of `\endlinechar`[16] is 5 (*end-of-line*).
2. The category code of `\ltjlineendcomment` itself is 14 (*comment*).
3. The input line matches the following "regular expression":

$$(\text{any char})^*(\textbf{JAchar})\big(\{\text{catcode} = 1\} \cup \{\text{catcode} = 2\}\big)^*$$

■**Remark** The following example shows the major difference from the behavior of pTeX.

```
1 \fontspec[Ligatures=TeX]{Linux Libertine O}
2 \ltjsetparameter{autoxspacing=false}
3 \ltjsetparameter{jacharrange={-6}}xあ
4 y\ltjsetparameter{jacharrange={+6}}zい
5 u
```

x⊠yzい u

It is not strange that "あ" does not printed in the above output. This is because TeX Gyre Termes does not contain "あ", and because "あ" in line 3 is considered as an **ALchar**.

Note that there is no space before "y" in the output, but there is a space before "u". This follows from following reasons:

- When line 3 is processed by `process_input_buffer` callback, "あ" is considered as an **JAchar**. Since line 3 ends with an **JAchar**, the comment character (whose character code is `\ltjlineendcomment`) is appended to this line, and hence the linebreak immediately after this line is ignored.
- When line 4 is processed by `process_input_buffer` callback, "い" is considered as an **ALchar**. Since line 4 ends with an **ALchar**, the linebreak immediately after this line emits a space.

16 Patch for the listings Package

It is well-known that the listings package outputs weird results for Japanese input. The listings package makes most of letters active and assigns output command for each letter ([2]). But Japanese characters are not included in these activated letters. For pTeX series, there is no method to make Japanese characters active; a patch jlisting.sty ([4]) resolves the problem forcibly.

In LuaTeX-ja, the problem is resolved by using the `process_input_buffer` callback. The callback function inserts the output command (active character `\ltjlineendcomment`) before each letter above `U+0080`. This method can omits the process to make all Japanese characters active (most of the activated characters are not used in many cases).

If the listings package and LuaTeX-ja were loaded, then the patch lltjp-listings is loaded automatically at `\begin{document}`.

16.1 Notes and additional keys

■**Variation selectors** lltjp-listings add two keys, namely `vsraw` and `vscmd`, which specify how variation selectors are treated in `lstlisting` or other enviroments. Note that these additional keys are not usable in the preamble, since lltjp-listings is loaded at `\begin{document}`.

`vsraw` is a key which takes a boolean value, and its default value is false.

- If the `vsraw` key is true, then variation selectors are "combined" with the previous character.

```
\begin{lstlisting}[vsraw=true]
葛󠄀城市，葛󠄁飾区，葛西
\end{lstlisting}
```

1 葛城市，葛飾区，葛西

[15] Its default value is `"FFFFF`, so `U+FFFFF` is used. The category code of `U+FFFFF` is set to 14 (*comment*) by LuaTeX-ja.
[16] Usually, it is ⟨*return*⟩ (whose character code is 13).

- If the vsraw key is false, then variation selectors are typeset by an appropriate command, which is specified by the vscmd key. The default setting of the vscmd key produces the following.

```
\begin{lstlisting}[vsraw=false,
  vscmd=\ltjlistingsvsstdcmd]
葛󠄀城市，葛󠄁飾区，葛西
\end{lstlisting}
```

```
葛󠄀城市，葛󠄁飾区，葛西
```

For example, the following code is the setting of the vscmd key in this document.

```
\def\IVSA#1#2#3#4#5{%
  \hbox to1em{\hss\textcolor{blue}{\raisebox{3.5pt}{\normalfont\ttfamily%
    \fboxsep=0.5pt\fbox{\hbox to0.75em{\hss\tiny \oalign{0#1#2\crcr#3#4#5\crcr}\hss}}}}\hss}
        %
}
{\catcode`\%=11
  \gdef\IVSB#1{\expandafter\IVSA\directlua{
    local cat_str = luatexbase.catcodetables['string']
    tex.sprint(cat_str, string.format('%X', 0xE00EF+#1))
}}}
\lstset{vscmd=\IVSB}
```

The default output command of variation selectors is stored in \ltjlistingsvsstdcmd.

■The doubleletterspace key Even the column format is [c]fixed, sometimes characters are not vertically aligned. The following example is typeset with basewidth=2em, and you'll see the leftmost "H" are not vertically aligned.

```
:   H   :
:    H   H   H   H    :
```

lltjp-listing adds the doubleletterspace key (not activated by default, for compatibility) to improve the situation, namely doubles inter-character space in each output unit. With this key, the above input now produces better output.

```
:   H   :
:   H   H   H   H   :
```

16.2 Class of characters

Roughly speaking, the listings package processes input as follows:

1. Collects *letters* and *digits*, which can be used for the name of identifiers.
2. When reading an *other*, outputs the collected character string (with modification, if needed).
3. Collects *others*.
4. When reading a *letter* or a *digit*, outputs the collected character string.
5. Turns back to 1.

By the above process, line breaks inside of an identifier are blocked. A flag \lst@ifletter indicates whether the previous character can be used for the name of identifiers or not.

For Japanese characters, line breaks are permitted on both sides except for brackets, dashes, etc. Hence the patch lltjp-listings introduces a new flag \lst@ifkanji, which indicates whether the previous character is a Japanese character or not. For illustration, we introduce following classes of characters:

	Letter	**Other**	
`\lst@ifletter`	T	F	
`\lst@ifkanji`	F	F	
Meaning	char in an identifier	other alphabet	
	Kanji	**Open**	**Close**
`\lst@ifletter`	T	F	T
`\lst@ifkanji`	T	T	F
Meaning	most of Japanese char	opening brackets	closing brackets

Note that *digits* in the listings package can be Letter or Other according to circumstances.

For example, let us consider the case an Open comes after a Letter. Since an Open represents Japanese open brackets, it is preferred to be permitted to insert line break after the Letter. Therefore, the collected character string is output in this case.

The following table summarizes $5 \times 5 = 25$ cases:

<table>
<tr><th></th><th></th><th colspan="5">Next</th></tr>
<tr><th></th><th></th><th>Letter</th><th>Other</th><th>Kanji</th><th>Open</th><th>Close</th></tr>
<tr><td rowspan="5">Prev</td><td>Letter</td><td>collects</td><td colspan="3">outputs</td><td>collects</td></tr>
<tr><td>Other</td><td>outputs</td><td>collects</td><td colspan="2">outputs</td><td>collects</td></tr>
<tr><td>Kanji</td><td colspan="4">outputs</td><td>collects</td></tr>
<tr><td>Open</td><td colspan="5">collects</td></tr>
<tr><td>Close</td><td colspan="4">outputs</td><td>collects</td></tr>
</table>

In the above table,

- "outputs" means to output the collected character string (i.e., line breaking is permitted there).
- "collects" means to append the next character to the collected character string (i.e., line breaking is prohibited there).

Characters above or equal to `U+0080` *except Variation Selectors* are classified into above 5 classes by the following rules:

- **ALchar**s above or equal to `U+0080` are classified as Letter.
- **JAchar**s are classified in the order as follows:
  1. Characters whose prebreakpenalty is greater than or equal to 0 are classified as Open.
  2. Characters whose postbreakpenalty is greater than or equal to 0 are classified as Close.
  3. Characters that don't satisfy the above two conditions are classified as Kanji.

The width of halfwidth kana (`U+FF61`–`U+FF9F`) is same as the width of **ALchar**; the width of the other **JAchar**s is double the width of **ALchar**.

This classification process is executed every time a character appears in the `lstlisting` environment or other environments/commands.

17 Cache Management of LuaTeX-ja

LuaTeX-ja creates some cache files to reduce the loading time. in a similar way to the luaotfload package:

- Cache files are usually stored in (and loaded from) `$TEXMFVAR/luatexja/`.
- In addition to caches of the text form (the extension is "`.lua.gz`", because they are compressed by gzip), caches of the *binary* (bytecode) form are supported.
  - In loading a cache, the binary cache precedes the text form.
  - When LuaTeX-ja updates a compressed text cache `hoge.lua.gz`, its binary version is also updated.

Table 16. `cid` key and corresponding files

`cid` key	name of the cache	used CMaps	
`Adobe-Japan1-*`	`ltj-cid-auto-adobe-japan1.{lua.gz,luc}`	`UniJIS2004-UTF32-*`	`Adobe-Japan1-UCS2`
`Adobe-Korea1-*`	`ltj-cid-auto-adobe-korea1.{lua.gz,luc}`	`UniKS-UTF32-*`	`Adobe-Korea1-UCS2`
`Adobe-KR-*`	`ltj-cid-auto-adobe-kr.{lua.gz,luc}`	`UniAKR-UTF32-*`	`Adobe-KR-UCS2`
`Adobe-GB1-*`	`ltj-cid-auto-adobe-gb1.{lua.gz,luc}`	`UniGB-UTF32-*`	`Adobe-GB1-UCS2`
`Adobe-CNS1-*`	`ltj-cid-auto-adobe-cns1.{lua.gz,luc}`	`UniCNS-UTF32-*`	`Adobe-CNS1-UCS2`

17.1 Use of cache

LuaTEX-ja uses the following cache:

`ltj-cid-auto-adobe-japan1.{lua.gz,luc}`
: The font table of a CID-keyed non-embedded Japanese font. This is loaded in every run. It is created from three CMaps, `UniJIS2004-UTF32-{H,V}` and `Adobe-Japan1-UCS2`, and this is why these two CMaps are needed in the first run of LuaTEX-ja.

  Similar caches are created as Table 16, if you specified `cid` key in `\jfont` to use other CID-keyed non-embedded fonts for Chinese or Korean, as in Page 27.

`ltj-kinsoku.luc`
: The bytecode cache which default *kinsoku* parameters are stored.

`ltj-jisx0208.luc`
: The bytecode version of `ltj-jisx0208.lua`. This is the conversion table between JIS X 0208 and Unicode which is used in Kanji-code conversion commands for compatibility with pTEX.

`ltj-ivd_aj1.luc`
: The bytecode version of `ltj-ivd_aj1.lua`.

`extra_***.{lua.gz,luc}`
: This file conains some information (especially for vertical typesetting) about the font '***'.

17.2 Internal

Cache management system of LuaTEX-ja is stored in `luatexja.base` (`ltj-base.lua`). There are four public functions for cache management in `luatexja.base`, where ⟨*filename*⟩ stands for the file name *without suffix*:

`save_cache(`⟨*filename*⟩`, `⟨*data*⟩`)`
: Save a non-nil table ⟨*data*⟩ into a cache ⟨*filename*⟩. Both the compressed text form ⟨*filename*⟩`.lua.gz` and its binary version are created or updated.

`save_cache_luc(`⟨*filename*⟩`, `⟨*data*⟩`[, `⟨*serialized_data*⟩`])`
: Same as `save_cache`, except that only the binary cache is updated. The third argument ⟨*serialized_data*⟩ is not usually given. But if this is given, it is treated as a string representation of ⟨*data*⟩.

`load_cache(`⟨*filename*⟩`, `⟨*outdate*⟩`)`
: Load the cache ⟨*filename*⟩. ⟨*outdate*⟩ is a function which takes one argument (the contents of the cache), and its return value is whether the cache is outdated.

  `load_cache` first tries to read the binary cache ⟨*filename*⟩`.luc`. If its contents is up-to-date, `load_cache` returns the contents. If the binary cache is not found or its contents is outdated, `load_cache` tries to read the compressed text form ⟨*filename*⟩`.lua.gz`. Hence, the return value of `load_cache` is non-nil, if and only if the updated cache is found.

`remove_cache(`⟨*filename*⟩`)`
: Remove the cahce ⟨*filename*⟩.

References

[1] Victor Eijkhout. *TEX by Topic, A TEXnician's Reference*, Addison-Wesley, 1992.

[2] C. Heinz, B. Moses. The Listings Package.

[3] Takuji Tanaka. upTeX—Unicode version of pTeX with CJK extensions, TUG 2013, October 2013. http://tug.org/tug2013/slides/TUG2013_upTeX.pdf

[4] Thor Watanabe. Listings - MyTeXpert.http://mytexpert.osdn.jp/index.php?Listings

[5] W3C Japanese Layout Task Force (ed). Requirements for Japanese Text Layout (W3C Working Group Note), 2011, 2012. http://www.w3.org/TR/jlreq/

[6] 乙部厳己．「min10 フォントについて」 http://argent.shinshu-u.ac.jp/~otobe/tex/files/min10.pdf

[7] 日本工業規格 (Japanese Industrial Standard).「JIS X 4051, 日本語文書の組版方法 (Formatting rules for Japanese documents)」, 1993, 1995, 2004.

[8] 濱野尚人，田村明史，倉沢良一．「TEX の出版への応用—縦組み機能の組み込み—」．.../texmf-dist/doc/ptex/base/ptexdoc.pdf

[9] Hisato Hamano. *Vertical Typesetting with TEX*, TUGBoat **11**(3), 346–352, 1990.

[10] International Organization for Standardization. ISO 32000-1:2008, *Document management – Portable document format – Part 1: PDF 1.7*, 2008. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=51502

[11] 北川弘典.「LuaTEX-ja の近況」, TEXConf 2018. https://raw.githubusercontent.com/h-kitagawa/presentations/main/tc18ltja.pdf

[12] Takuto ASAKURA. The BXghost Package. https://github.com/wtsnjp/BXghost