The LuaTEX-ja package

The LuaTEX-ja project team

April 3, 2015

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\ltjgetparemeter{direction}=4 ... \fi
```

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

■**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 13.

■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 this version, LuaTEX-ja supports vertical writing; but this feature must *not* be confused with Ω-style direction support of LuaTEX itself. ...

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

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 (like prebreakpenalty) means an internal parameter for Japanese typesetting, and it is used as a key in `\ltjsetparameter` command.
- A word in typewriter font with 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.

1.4 About the Project

■Project Wiki Project Wiki is under construction.

- http://sourceforge.jp/projects/luatex-ja/wiki/FrontPage%28en%29 (English)
- http://sourceforge.jp/projects/luatex-ja/wiki/FrontPage (Japanese)
- http://sourceforge.jp/projects/luatex-ja/wiki/FrontPage%28zh%29 (Chinese)

This project is hosted by SourceForge.JP.

■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 beta-0.80.0 (or later)
- luaotfload v2.5 (or later)
- adobemapping (Adobe cmap and pdfmapping files)
- everysel (if you want to use LuaTeX-ja with LaTeX 2_ε)
- *IPAex fonts*(http://ipafont.ipa.go.jp/)

This version of LuaTeX-ja no longer supports TeX Live 2013 (or older version). If you want to use this version with TeX Live 2014, you have to build a development version of LuaTeX from its source.

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

- MiKTeX (in `luatexja.tar.lzma`); see the next subsection
- TeX Live (in `texmf-dist/tex/luatex/luatexja`)
- W32TeX (in `luatexja.tar.xz`)

IPAex fonts are also available in these distributions.

■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:

     ```
     $ git clone git://git.sourceforge.jp/gitroot/luatex-ja/luatexja.git
     ```

   - Download the `tar.gz` archive of HEAD in the `master` branch from

     http://git.sourceforge.jp/view?p=luatex-ja/luatexja.git;a=snapshot;h=HEAD;sf=tgz.

   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 and `ltj-kinsoku.lua` (the file which stores default "*kinsoku*" parameters):

   ```
   $ cd src
   $ lualatex ltjclasses.ins
   $ lualatex ltjsclasses.ins
   $ lualatex ltjltxdoc.ins
   $ luatex   ltj-kinsoku_make.tex
   ```

   Note that `*.{dtx,ins}` and `ltj-kinsoku_make.tex` are not needed in regular use.

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. No other encodings, such as EUC-JP or Shift-JIS, are not supported.
- LuaTEX-ja is very slower than pTEX. Generally speaking, LuaJITTEX processes LuaTEX-ja about 30% faster than LuaTEX, but not always.
- **Note for MiKTEX users** LuaTEX-ja requires that several CMap files[1] must be found from LuaTEX. Strictly speaking, those CMaps are needed only in the first run of LuaTEX-ja after installing or updating. But it seems that MiKTEX does not satisfy this condition, so you will encounter an error like the following:

  ```
  ! LuaTeX error ...iles (x86)/MiKTeX 2.9/tex/luatex/luatexja/ltj-rmlgbm.lua
  bad argument #1 to 'open' (string expected, got nil)
  ```

  If so, please execute a batch file which is written on the Project Wiki (English). This batch file creates a temporary directory, copy CMaps in it, run LuaTEX-ja in this directory, and finally delete the temporary directory.

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*	IPAex Mincho	`\tenmin`	`\sevenmin`	`\fivemin`
	gothic	IPAex Gothic	`\tengt`	`\sevengt`	`\fivegt`
tate (vertical)	*mincho*	IPAex Mincho	`\tentmin`	`\seventmin`	`\fivetmin`
	gothic	IPAex Gothic	`\tentgt`	`\seventgt`	`\fivetgt`

  - With `luatexja.cfg`, one can use other fonts as "default" Japanese fonts (Subsection 3.5).
  - 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 is `JY3` (for horizontal direction) and `JT3` (for vertical direction).
- Traditionally, Japanese documents use two typeface categories: *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.

[1] `UniJIS2004-UTF32-{H,V}` and `Adobe-Japan1-UCS2`.

classification			family name
mincho (明朝体)	`\textmc{...}`	`{\mcfamily ...}`	`\mcdefault`
gothic (ゴシック体)	`\textgt{...}`	`{\gtfamily ...}`	`\gtdefault`

- By default, the following fonts are used for *mincho* and *gothic*:

classification	family	`\mdseries`	`\bfseries`	scale
mincho (明朝体)	`mc`	IPAex Mincho	IPAex Gothic	0.962216
gothic (ゴシック体)	`gt`	IPAex Gothic	IPAex Gothic	0.962216

  Note that the bold series in both family are same as the medium series of *gothic* family. There is no italic nor slanted shape for these `mc` and `gt`.

- 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 *gothic* family. The following line changes the default Japanese font family to *gothic*:

```
\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, we have the counterparts of `jclasses` (standard classes in pLaTeX) and `jsclasses` (classes by Haruhiko Okumura), namely, `ltjclasses`[2] and `ltjsclasses`[3].

■`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 2ε

■plain TeX To change Japanese fonts in plain TeX, you must use the command `\jfont` and `\tfont`. So please see Subsection 7.1.

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

- Commands `\fontfamily`, `\fontseries`, `\fontshape`, and `\selectfont` can be used to change attributes of Japanese fonts.

	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`

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

[3] `ltjsarticle.cls`, `ltjsbook.cls`, `ltjskiyou.cls`.

`\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 10.1.

- For defining a Japanese font family, use `\DeclareKanjiFamily` instead of `\DeclareFontFamily`. However, in the present implementation, using `\DeclareFontFamily` doesn't cause any problem.
- Defining a Japanese font shape can be done by usual `\DeclareFontShape`:

```
\DeclareFontShape{JY3}{mc}{bx}{n}{<-> s*KozMinPr6N-Bold:jfm=ujis;-kern}{}
    % Kozuka Mincho Pr6N Bold
```

■Remark: 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}\,\}$.
```

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

$$y=(x-1)^2+2 \quad よって \quad y>0$$

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

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:

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

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

$$y=(x-1)^2+2 \quad よって \quad y>0$$

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

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 7.5.

3.2 fontspec

To coexist with the fontspec package, it is needed to load luatexja-fontspec package in the preamble, as follows:

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

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

In luatexja-fontspec package, the following seven commands are defined as counterparts of original commands in the fontspec package:

Japanese fonts alphabetic fonts	`\jfontspec` `\fontspec`	`\setmainjfont` `\setmainfont`	`\setsansjfont` `\setsansfont`	`\setmonojfont`* `\setmonofont`
Japanese fonts alphabetic fonts	`\newjfontfamily` `\newfontfamily`	`\newjfontface` `\newfontface`	`\defaultjfontfeatures` `\defaultfontfeatures`	`\addjfontfeatures` `\addfontfeatures`

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.

  Note that `\setmonojfont` *is defined if and only if this* `match` *option is specified.*

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

The reason that `\setmonojfont` is not defined by default is that it is popular for Japanese fonts that nearly all Japanese glyphs have same widths. Also 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 7.1).

Below is an example of `\jfontspec`.

```
\jfontspec[CJKShape=NLC]{KozMinPr6N-Regular}
JIS-X-0213:2004→辻\par
\jfontspec[CJKShape=JIS1990]{KozMinPr6N-Regular}
JIS-X-0208:1990→辻
```

JIS X 0213:2004 →辻
JIS X 0208:1990 →辻

3.3 Presets

To use standard Japanese font settings easily, one can load luatexja-preset package with several options. This package provides functions in a part of japanese-otf package and a part of PXchfon package by Takayuki Yato, and loads luatexja-fontspec, hence fontspec internally.

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

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

■General options

`nodeluxe`
: Use one-weighted *mincho* and *gothic* font families. This means that `\mcfamily\bfseries`, `\gtfamily\bfseries` and `\gtfamily\mdseries` use the same font. *This option is enabled by default.*

`deluxe`
: Use *mincho* with two weights (medium and bold), *gothic* with three weights (medium, bold and heavy), and *rounded gothic*[4]. The heavy weight of *gothic* can be used by "changing the family" `\gtebfamily`, or `\textgteb{...}`. This is because fontspec package can handle only medium (`\mdseries`) and bold (`\bfseries`).

`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 bold series of *mincho*.

`90jis`
: Use 90JIS glyph variants if possible.

`jis2004`
: Use JIS2004 glyph variants if possible.

`jis`
: Use the JFM `jfm-jis.lua`, instead of `jfm-ujis.lua`, which is the default JFM of LuaTEX-ja.

Note that `90jis` and `jis2004` only affect with *mincho*, *gothic* (and possibly *rounded gothic*) defined by this package. We didn't taken account of when both `90jis` and `jis2004` are specified.

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

■**Presets for multi weight** Besides `morisawa-pro` and `morisawa-pr6n` presets, fonts are specified by font name, not by file name.

`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	medium	KozMinPro-Regular	KozMinProVI-Regular	KozMinPr6N-Regular
	bold	KozMinPro-Bold	KozMinProVI-Bold	KozMinPr6N-Bold
gothic	medium	KozGoPro-Regular*	KozGoProVI-Regular*	KozGoPr6N-Regular*
		KozGoPro-Medium	KozGoProVI-Medium	KozGoPr6N-Medium
	bold	KozGoPro-Bold	KozGoProVI-Bold	KozGoPr6N-Bold
	heavy	KozGoPro-Heavy	KozGoProVI-Heavy	KozGoPr6N-Heavy
rounded gothic		KozGoPro-Heavy	KozGoProVI-Heavy	KozGoPr6N-Heavy

In above table, starred fonts (KozGo…-Regular) are used for medium series of *gothic*, *if and only if* `deluxe` *option is specified.*

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

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

Hiragino fonts are bundled with Mac OS X 10.5 or later. Some editions of a Japanese word-processor "一太郎 2012" includes Hiragino ProN fonts. Note that the heavy weight of *gothic* family only supports Adobe-Japan1-3 character collection (Std/StdN).

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

`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`
	heavy	`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-osx` Yu fonts bundled with OSX Mavericks.

family	series	yu-win	yu-osx
mincho	medium	YuMincho-Regular	YuMincho Medium
	bold	YuMincho-Demibold	YuMincho Demibold
gothic	medium	YuGothic-Regular*	YuGothic Medium*
		YuGothic-Bold	YuGothic Bold
	bold	YuGothic-Bold	YuGothic Bold
	heavy	YuGothic-Bold	YuGothic Bold
rounded gothic		YuGothic-Bold	YuGothic Bold

■**Presets for single weight** Next, we describe settings for using only single weight. In four settings below, we use same fonts for medium and bold (and heavy) weights. (Hence `\mcfamily\bfseries` and `\mcfamily\mdseries` yields same Japanese fonts, even if `deluxe` option is also specified).

	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

■**Using HG fonts** We can use HG fonts bundled with Microsoft Office for realizing multiple weights.

	ipa-hg	ipaex-hg	ms-hg
mincho medium	IPA Mincho	IPAex Mincho	MS Mincho
mincho bold		HG Mincho E	
Gothic medium			
without `deluxe`	IPA Gothic	IPAex Gothic	MS Gothic
with `jis2004`	IPA Gothic	IPAex Gothic	MS Gothic
otherwise		HG Gothic M	
gothic bold		HG Gothic E	
gothic heavy		HG Soei Kaku Gothic UB	
rounded gothic		HG Maru Gothic 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.).

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

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

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 `japanese-otf` package. If you want to use these functions, load `luatexja-otf` package.

```
\jfontspec{KozMinPr6N-Regular.otf}
森\UTF{9DD7}外と内田百\UTF{9592}とが\UTF{9AD9
    }島屋に行く。

\CID{7652}飾区の\CID{13706}野家，
\CID{1481}城市，葛西駅，
高崎と\CID{8705}\UTF{FA11}

\aj半角{はんかくカタカナ}
```

森鷗外と内田百閒とが髙島屋に行く。
葛飾区の吉野家，葛城市，葛西駅，高崎と髙﨑
はんかくカタカナ

3.5 Changing default Japanese fonts

If `luatexja.cfg` can be seen from LuaTeX, LuaTeX-ja automatically reads it. The main use of `luatexja.cfg` is for changing default Japanese fonts, when IPAex fonts cannot be installed in TeX system. One should not overuse this `luatexja.cfg`; fonts which will be used in a document should be specified in its source.

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 IPAex Mincho and IPAex Gothic.

For another example, the following two lines makes that non-embedded fonts Ryumin-Light and GothicBBB-Medium as default Japanese fonts (as the earlier version of LuaTeX-ja):

```
\def\ltj@stdmcfont{psft:Ryumin-Light}
\def\ltj@stdgtfont{psft:GothicBBB-Medium}
```

4 Changing Parameters

There are many 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 Editing the 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 you execute 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 done for character ranges. This can be edited by setting the jacharrange parameter. For example, this is just the default setting of LuaTeX-ja, and it sets

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

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

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**".

■**Default setting** LuaTeX-ja predefines eight character ranges for convenience. They are determined from the following data:

- Blocks in Unicode 6.0.
- The `Adobe-Japan1-UCS2` mapping between a CID Adobe-Japan1-6 and Unicode.
- The `PXbase` bundle for upTeX by Takayuki Yato.

Now we describe these eight 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 above `U+0080` which does not belong to these eight ranges belongs to the character range 217.

Range 8$^{\mathrm{J}}$ The intersection of the upper half of ISO 8859-1 (Latin-1 Supplement) and JIS X 0208 (a basic character set for Japanese). This character range consists of the following characters:

Table 1. Unicode blocks in predefined character range 3.

U+2000–U+206F	General Punctuation	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. Mathematical Symbols-B	U+2B00–U+2BFF	Miscellaneous Symbols and Arrows

- § (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)

Range 1^{A} Latin characters that some of them are included in Adobe-Japan1-6. This range consists of the following Unicode ranges, *except characters in the range 8 above*:

- 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

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} Punctuations and Miscellaneous symbols. The block list is indicated in Table 1.

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-"A82F, "A840-"ABFF, "FB00-"FE0F,
  "FE20-"FE2F, "FE70-"FEFF, "10000-"1FFFF, "E000-"F8FF} % 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 2.

Range 7^{J} Characters used in CJK languages, but not included in Adobe-Japan1-6. The block list is indicated in Table 3.

4.2 kanjiskip and xkanjiskip

JAglue is divided into the following three categories:

- Glues/kerns specified in JFM. If `\inhibitglue` is issued around a Japanese character, this glue will not be inserted at the place.

Table 2. 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 Extension 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+20000–U+2FFFF	(Supplementary Ideographic Plane)
U+E0100–U+E01EF	Variation Selectors Supplement		

Table 3. 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+A830–U+A83F	Common Indic Number Forms
U+AC00–U+D7AF	Hangul Syllables	U+D7B0–U+D7FF	Hangul Jamo Extended-B

- 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).

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

kanjiskip: 0.0pt plus 0.4pt minus 0.4pt,
xkanjiskip: 2.40555pt 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 8.1).

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 0.4pt depth 0pt \
    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):

```
xyz漢字
{\scriptsize
  \ltjsetparameter{yjabaselineshift=-1pt,
    yalbaselineshift=-1pt}
  XYZひらがな
}abcかな
```

xyz 漢字 XYZ ひらがな abc かな

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 ...

This problem does not apply for yjabaselineshift nor tjabaselineshift.

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. ...

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 4.

\kcatcode can be set by a row of JIS X 0208 in pTEX, and generally by a Unicode block[5] 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 using 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.
- *Ignoring linebreak after a **JAchar*** is always ignored.

Default setting of \catcode of LuaTEX can be found in luatex-unicode-letters.tex, which is based on unicode-letters.tex (for XETEX). However, the default setting of \catcode differs between XETEX and LuaTEX, by the following reasons:

- luatex-unicode-letters.tex is based on old unicode-letters.tex.
- The latter half of unicode-letters.tex sets \catcode of Kanji and kana characters to 11, via setting \XeTeXcharclass.

  However, this latter half is simply omitted in luatex-unicode-letters.tex, hence \catcode of Kanji and kana characters remains 12 in LuaTEX.

In other words, Kanji nor kana characters cannot be used in a control word, in the default setting of LuaTEX.

This would be inconvenient for pTEX users to shifting to LuaTEX-ja, since several control words containing Kanji, such as \西暦, are used in pTEX. Hence, LuaTEX-ja have a counterpart of unicode-letters.tex for LuaTEX, *to match the* \catcode *setting with that of XETEX.*

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 5 shows the difference. Except for four characters “・”, “゛”, “゜”, “＝”, LuaTEX-ja admits more characters in a control word than upTEX. *Note that the ideographic space U+3000 can be used in a control word in LuaTEX-ja.*

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

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

Table 4. \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 5. 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
〃 (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
／ (U+FF0F)	1	31	N	N	Y
＼ (U+FF3C)	1	32	N	N	Y

	row	col.	pTEX	upTEX	LuaTEX-ja
｜ (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 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 6. 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. Also, the direction of a math formula is changed to *utod*, when the direction outside the math formula is *tate* (vertical writing).

6.1 Boxes in different direction

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

```
1 ここは横組%      yoko
2 \hbox{\tate %    tate
3   \hbox{縦組}%   tate
4   の中に
5   \hbox{\yoko 横組の内容}% yoko
6   を挿入する
7 }
8 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 6. Directions supported by LuaTEX-ja

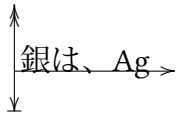
	horizontal (*yoko* direction)	vertical (*tate* direction)	*dtou* direction	*utod* direction
Commands	`\yoko`	`\tate`	`\dtou`	`\utod`
Beginning of the page	Top	Right	Left	Right
Beginning of the line	Left	Top	Bottom	Top
Used Japanese font	horizontal (`\jfont`)	vertical (`\tfont`)	horizontal (90˚ rotated)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

Table 7 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.

```
\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.}
```

YOKO
38.48877pt,
8.46753pt,
1.15466pt.

よこぐみ

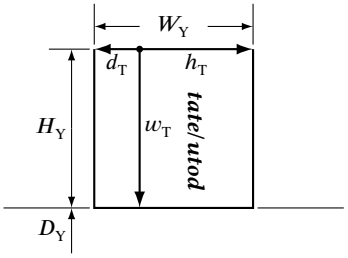
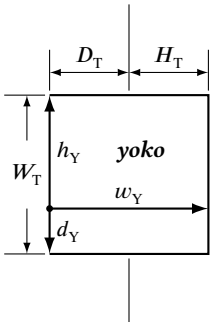
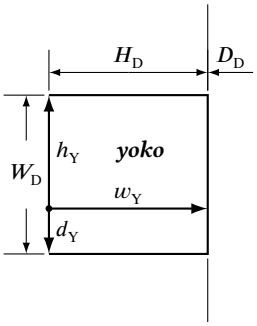
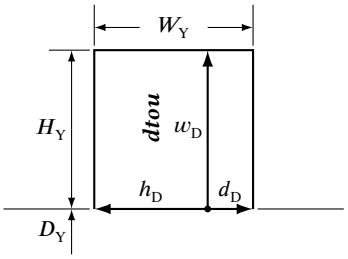
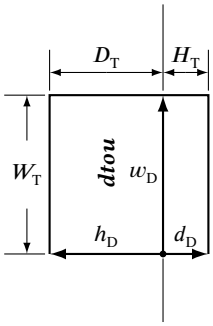
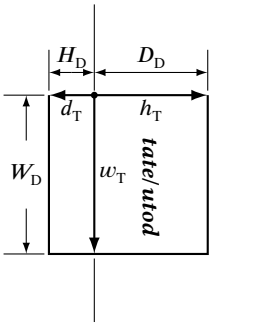
TATE
9.6222pt,
19.24438pt,
19.24438pt.

DTOU
9.6222pt,
38.48877pt,
0.0pt.

`\ltjsetwd`⟨*num*⟩=⟨*dimen*⟩, `\ltjsetht`⟨*num*⟩=⟨*dimen*⟩, `\ltjsetdp`⟨*num*⟩=⟨*dimen*⟩
: These commands set the dimension of `\box`⟨*num*⟩. Unlike `\ltjgetwd`, 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
```

Table 7. Boxes in different direction

typeset in *yoko* direction	typeset in *tate* or *utod* direction	typeset in *dtou* direction
		
$W_Y = h_T + d_T$, $H_Y = w_T$, $D_Y = 0$ pt	$W_T = h_Y + d_Y$, $H_T = w_Y/2$, $D_T = w_Y/2$	$W_D = h_Y + d_Y$, $H_D = w_Y$, $D_D = 0$ pt
		
$W_Y = h_D + d_D$, $H_Y = w_D$, $D_Y = 0$ *pt*	$W_T = h_D + d_D$, $H_T = d_D$, $D_T = h_D$	$W_D = w_T$, $H_D = d_T$, $D_D = h_T$

6.2 Getting current direction

The direction parameter returns the current direction, and the boxdir parameter (with the argumant ⟨*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

```
\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

6.3 Overridden box primitives

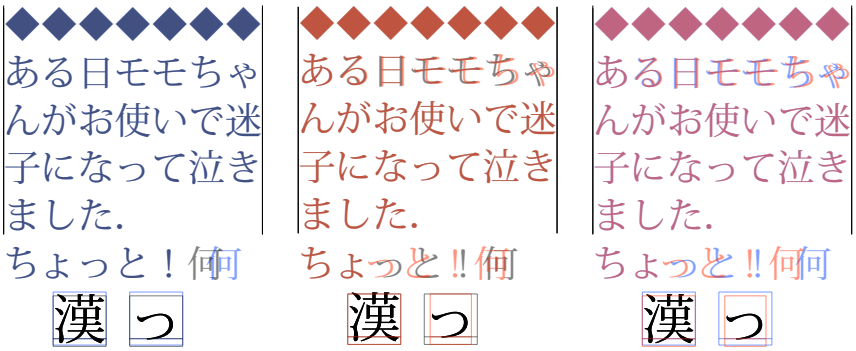
To cope with multiple directions, the following primitives are overridden by LuaTEX-ja, using \protected\def.

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

\vadjust{⟨*material*⟩}

\insert⟨*number*⟩{⟨*material*⟩}

Table 8. Differences between horizontal JFMs shipped with LuaTEX-ja



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

`\lastbox`

`\raise`⟨*dimen*⟩⟨*box*⟩, `\lower`⟨*dimen*⟩⟨*box*⟩ etc., `\vcenter`

`\vcenter`

7 Font Metric and Japanese Font

7.1 `\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\tradgt={file:KozMinPr6N-Regular.otf:script=latn;%
  +trad;-kern;jfm=ujis} at 14pt
\tradgt 当／体／医／区
```

當／體／醫／區

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

■JFM a JFM has measurements of characters and glues/kerns that are automatically inserted for Japanese typesetting. The structure of JFM will be described in the next subsection. At the calling of `\jfont`, you must specify which JFM will be used for this font by the following keys:

`jfm=`⟨*name*⟩
: Specify the name of (horizontal) JFM. If specified JFM has not been loaded, LuaTEX-ja search and load a file named `jfm-`⟨*name*⟩`.lua`.

  The following JFMs are shipped with LuaTEX-ja:

  `jfm-ujis.lua` A standard JFM in LuaTEX-ja. This JFM is based on `upnmlminr-h.tfm`, a metric for UTF/OTF package that is used in upTEX. When you 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.

  The difference among these three JFMs is shown in Table 8.

`jfmvar=`⟨*string*⟩
: Sometimes there is a need that ....

```
\ltjsetparameter{differentjfm=both}
\jfont\F=file:KozMinPr6N-Regular.otf:jfm=ujis
\jfont\G=file:KozGoPr6N-Medium.otf:jfm=ujis
\jfont\H=file:KozGoPr6N-Medium.otf: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 1. Example of `jfmvar` key

|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|

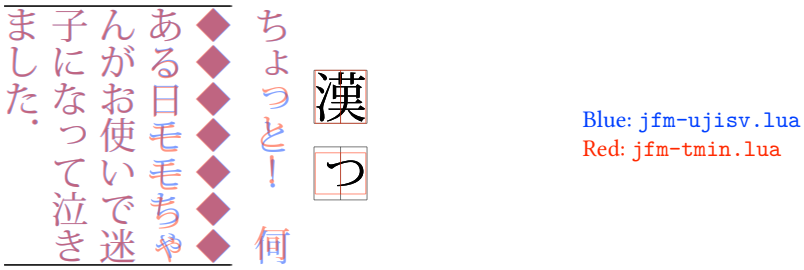
|ダ イ ナ ミ ッ ク ダ イ ク マ|
|ダ イ ナ ミ ッ ク ダ イ ク マ|
|ダ イ ナ ミ ッ ク ダ イ ク マ|
|ダ イ ナ ミ ッ ク ダ イ ク マ|

```
\newcommand\test{\vrule ダイナミックダイクマ\vrule\\}
\jfont\KMFW = KozMinPr6N-Regular:jfm=prop;-kern at 17pt
\jfont\KMFK = KozMinPr6N-Regular:jfm=prop at 17pt % kern is activated
\jfont\KMPW = KozMinPr6N-Regular:jfm=prop;script=dflt;+pwid;-kern at 17pt
\jfont\KMPK = KozMinPr6N-Regular:jfm=prop;script=dflt;+pwid;+kern at 17pt
\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 2. Kerning information and kanjiskip

Table 9. Differences between vertical JFMs shipped with LuaTEX-ja



■Using kerning information in a font Some fonts have information for inter-glyph spacing. This version of LuaTEX-ja treats kerning spaces like an italic correction; any glue and/or kern from the JFM and a kerning space can coexist. See Figure 2 for detail.

Note that in `\setmainjfont` etc. which are provided by `luatexja-fontspec` package, kerning option is set *off* (`Kerning=Off`) by default, because of the compatibility with previous versions of LuaTEX-ja.

■`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.

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

```
\jfont\E=KozMinPr6N-Regular:extend=1.5;jfm=ujis;-kern
\E あいうえお

\jfont\S=KozMinPr6N-Regular:slant=1;jfm=ujis;-kern
\S あいう\/ABC
```

あいうえお
あいうABC

7.2 `\tfont`

...

7.3 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-Japan1-6 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-Japan1-6;jfm=jis}      % Japanese
\jfont\testD={psft:Ryumin-Light:jfm=jis}                        % default value is Adobe-Japan1
    -6
\jfont\testC={psft:AdobeMingStd-Light:cid=Adobe-CNS1-6;jfm=jis} % Traditional Chinese
\jfont\testG={psft:SimSun:cid=Adobe-GB1-5;jfm=jis}              % Simplified Chinese
\jfont\testK={psft:Batang:cid=Adobe-Korea1-2;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 4 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
l.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.
?
```

7.4 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.

`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.

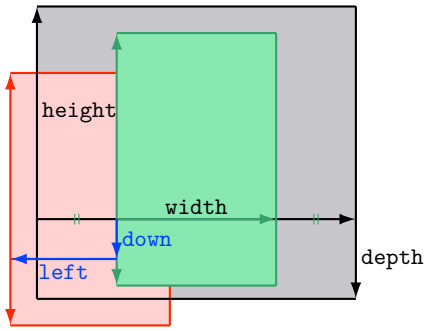
`xkanjiskip={`⟨*natural*⟩`,` ⟨*stretch*⟩`,` ⟨*shrink*⟩`}` (optional)

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)

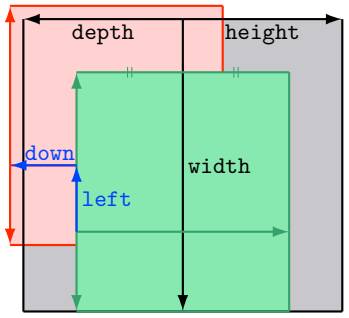
This field is a list of characters which are in this character type i. This field is optional if $i = 0$, since all **JAchar** which do not belong any character classes other than 0 are in the character class 0 (hence, the character class 0 contains most of **JAchar**s). In the list, character(s) can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



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



(...)

Figure 4. The position of the real glyph (vertical Japanese fonts)

- 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.

But there is one exception: `width` field can be `'prop'`. This means that width of a character becomes that of its "real" glyph.

`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 Figures 3 and 4.

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*⟩`]}, ...}`

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

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

⟨*priority*⟩ is an integer in [−2, 2] (treated as 0 if omitted), and this is used only in line adjustment with priority by `luatexja-adjust` (see Subsection 11.3). Higher value means the glue is easy to shretch, and is also easy to shrink.

⟨*ratio*⟩ is also an optional value between −1 and 1. 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

$$-1 \cdot \frac{0.5}{0.5+0.25} + 1 \cdot \frac{0.25}{0.5+0.25} = -\frac{1}{3}.$$

`end_stretch=`⟨*kern*⟩`, end_shrink=`⟨*kern*⟩

■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=file:KozMinPr6N-Regular.otf:jfm=test;+hwid
\setbox0\hbox{\a ヒ漢}                                    15.0pt
\the\wd0
```

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=file:KozMinPr6N-Regular.otf: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.

Table 10. 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*⟩

■**Imaginary characters** As described before, you 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.

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

`'jcharbdd'`
: A boundary between **JAchar** and anything else (such as **ALchar**, kern, glue, ...).

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

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

7.5 Math Font Family

TEX handles fonts in math formulas by 16 font families[6], and each family has three fonts: `\textfont`, `\scriptfont` and `\scriptscriptfont`.

LuaTEX-ja's handling of Japanese fonts in math formulas is similar; Table 10 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, you 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.

7.6 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 you 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.

[6] 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.

```
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 ($1\,\mathrm{sp} = 2^{-16}\,\mathrm{pt}$).

`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:

$$\begin{cases} 0 & \texttt{'left'} \text{ and the default value} \\ 0.5 & \texttt{'middle'} \\ 1 & \texttt{'right'} \end{cases}$$

- For $j \in \omega$, `[`j`]` stores a kern or a glue which will be inserted between character class i and class j.
  If a kern will be inserted, the value of this field is `[`j`]={false, `⟨*kern_node*⟩`, `⟨*ratio*⟩`}`, where ⟨*kern_node*⟩ is a node[7]. If a glue will be inserted, we have `[`j`]={false, `⟨*spec_node*⟩`,` ⟨*ratio*⟩`, `⟨*icflag*⟩`}`, where ⟨*spec_node*⟩ is also a node, and ⟨*icflag*⟩ = *from_jfm* + ⟨*priority*⟩.

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> chr_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.

[7] This version of LuaTEX-ja uses "direct access model" for accessing nodes, if possible.

```
function (<table> shift_info, <table> jfont_info, <number> char_class)
  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

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

8 Parameters

8.1 `\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 12.

`\ltjsetparameter` and `\ltjglobalsetparameter` are commands for assigning parameters. These take one argument which is a ⟨*key*⟩=⟨*value*⟩ list. The difference between `\ltjsetparameter` and `\ltjglobalsetparameter` is only the scope of assignment; `\ltjsetparameter` does a local assignment and `\ltjglobalsetparameter` does a global one. 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 ={⟨*chr_code*⟩,⟨*natural number*⟩}*

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

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

Set a penalty which is inserted automatically before the character ⟨*chr_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 ={⟨*chr_code*⟩,⟨*penalty*⟩}* [`\postbreakpenalty`]

Set a penalty which is inserted automatically after the character ⟨*chr_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 ={⟨*chr_code*⟩,⟨*mode*⟩}*

Setting whether inserting xkanjiskip is allowed before/after a **JAchar** whose character code is ⟨*chr_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 ={⟨*chr_code*⟩,⟨*mode*⟩}* [`\xspcode`]

Setting whether inserting xkanjiskip is allowed before/after a **ALchar** whose character code is ⟨*chr_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`]

xkanjiskip =⟨*skip*⟩* [`\xkanjiskip`]

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*⟩, ⟨*chr_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.

8.2 \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 0.4pt, 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 *down-to-up* direction, 3 means *tate* direction (vertical typesetting), and 4 means *yoko* direction (horizontal typesetting).

- For an integer ⟨*reg_num*⟩ between 0 and 65535, \ltjgetparameter{boxdim}{⟨*reg_num*⟩} returns the direction of \box⟨*reg_num*⟩. 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}{⟨*chr_code*⟩} returns the range number which ⟨*chr_code*⟩ belongs to (although there is no parameter named "chartorange").

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

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

  ```
  \ltjgetparameter{jacharrange}{\ltjgetparameter{chartorange}{⟨chr_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 Other Commands for plain TEX and LATEX 2ε

9.1 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`, `\jis`, `\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

9.2 `\inhibitglue`

`\inhibitglue` suppresses the insertion of **JAglue**. 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=file:KozMinPr6N-Regular.otf: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\hrule\noindent あoff\inhibitglue ice
```

あ　ウあウ
あ　1
あ　2
あ　3
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.
- 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.

9.3 \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*⟩.

⟨*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.

10 Commands for LaTeX 2ε

10.1 Patch for NFSS2

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

additonal dimensions

Like pLaTeX 2ε, 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 the character "あ", 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 font families and Japanese font families are only made by their encodings. For example, encodings OT1 and T1 are for alphabetic font families, and a Japanese font family 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*⟩}

```
\DeclareKanjiFamily{JY3}{edm}{}
\DeclareFontShape{JY3}{edm}{m}{n}    {<-> s*KozMinPr6N-Regular:jfm=ujis;}{}
\DeclareFontShape{JY3}{edm}{m}{green}{<-> s*KozMinPr6N-Regular:jfm=ujis;color=007F00}{}
\DeclareFontShape{JY3}{edm}{m}{blue} {<-> s*KozMinPr6N-Regular:jfm=ujis;color=0000FF}{}
\DeclareAlternateKanjiFont{JY3}{edm}{m}{n}{JY3}{edm}{m}{green}{"4E00-"67FF,{-2}-{-2}}
\DeclareAlternateKanjiFont{JY3}{edm}{m}{n}{JY3}{edm}{m}{blue}{ "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 5. An example of `\DeclareAlternateKanjiFont`

`\DeclareRelationFont{`⟨*ja-encoding*⟩`}{`⟨*ja-family*⟩`}{`⟨*ja-series*⟩`}{`⟨*ja-shape*⟩`}`
`{`⟨*al-encoding*⟩`}{`⟨*al-family*⟩`}{`⟨*al-series*⟩`}{`⟨*al-shape*⟩`}`

This command sets the "accompanied" alphabetic font family (given by the latter 4 arguments) with respect to a Japanese font family 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`

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`. Like `\fontfamily`, `\selectfont` is required to take an effect.

`\adjustbaseline`

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

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

where h_a and d_a denote the height of "a" and the depth, respectively. In LuaTeX-ja, this `\adjustbaseline` does same task.

`\fontfamily{`⟨*family*⟩`}`

As in LaTeX 2_ε, this command changes current font family (alphabetic, Japanese, *or both*) to ⟨*family*⟩. Which family will be changed is determined as follows:

- Let current encoding scheme for Japanese fonts be ⟨*ja-enc*⟩. Current Japanese font family will be changed to ⟨*family*⟩, if one of the following two conditions is met:
  - The family ⟨*family*⟩ under the encoding ⟨*ja-enc*⟩ has been already defined by `\DeclareKanijFamily`.
  - A font definition named ⟨*ja-enc*⟩⟨*family*⟩`.fd` (the file name is all lowercase) exists.
- Let current encoding scheme for alphabetic fonts be ⟨*al-enc*⟩. For alphabetic font family, the criterion as above is used.
- There is a case which none of the above applies, that is, the font family named ⟨*family*⟩ doesn't seem to be defined neither under the encoding ⟨*ja-enc*⟩, nor under ⟨*al-enc*⟩. In this case, the default family for font substitution is used for alphabetic and Japanese fonts. Note that current encoding will not be set to ⟨*family*⟩, unlike the original implementation in LaTeX.

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

As `\ltjdeclarealtfont` (Subsection 9.3), 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 5.

- 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.

```
\jfontspec[
  YokoFeatures={Color=007F00},
  TateFeatures={Color=00007F},
  TateFont=KozGoPr6N-Regular
]{KozMinPr6N-Regular}
\hbox{\yoko 横組のテスト}
\hbox{\tate 縦組のテスト}
```

横組のテスト

縦組のテスト

Figure 6. An example of TateFeatures etc.

- ...

As closing this subsection, we shall introduce an example of \SetRelationFont and \userelfont:

```
\makeatletter
\SetRelationFont{JY3}{\k@family}{m}{n}{OT1}{pag}{m}{n}
  % \k@family: current Japanese font family
\userelfont\selectfont あいうabc
```

あいう abc

11 Addons

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.

11.1 luatexja-fontspec.sty

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 7.1 and 7.3 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.

NoEmbed
: By specifying this key, one can use "name-only" Japanese font which will not be embedded in the output PDF file. See Subsection 7.3.

TateFeatures={⟨*features*⟩}, TateFont=⟨*font*⟩
: ...

YokoFeatures={⟨*features*⟩}
: ...

AltFont
: As \ltjdeclarealtfont (Subsection 9.3) and \DeclareAlternateKanjiFont (Subsection 10.1), 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⟩ },
    ...
  }
  ```

```
\jfontspec[
  AltFont={
    {Range="4E00-"67FF, Color=007F00},
    {Range="6800-"9EFF, Color=0000FF},
    {Range="3040-"306F, Font=KozGoPr6N-Regular},
  }
]{KozMinPr6N-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 7. An example of `AltFont`

Each sublist should have the `Range` key (sublist which does not contain `Range` key is simply ignored). A demonstrarion is shown in Figure 7.

■**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=007F00} }
}
```

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=007F00 } },
}
```

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`.

11.2 `luatexja-otf.sty`

This optional package supports typesetting characters in Adobe-Japan1 character collection (or other CID character collection, if the font is supported). The package `luatexja-otf` offers the following 2 low-level commands:

`\CID{`⟨*number*⟩`}`
: Typeset a character whose CID number is ⟨*number*⟩.

`\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.

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

- Always treated as **JAchar**s.
- Processing codes for supporting OpenType features (*e.g.*, glyph replacement and kerning) by the `luaotfload` package is not performed to these characters.

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

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

Figure 8. Line adjustment

■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.

■IVS support Recent fonts support Ideographic Variation Selector (IVS). It seems that luaotfload and fontspec packages do not support IVS, so we implemented IVS support in luatexja-otf. *IVS support is experimental; if you want to enable this, load luatexja-otf and execute the following:*

```
\directlua{luatexja.otf.enable_ivs()}
```

After executing the command above, you can use IVS like the following:

```
\Large
\jfontspec{KozMinPr6N-Regular}
奈良県葛〓100城市と，東京都葛〓101飾区．\\
こんにちは，渡
邉〓100邉〓101邉〓102邉〓103邉〓104
邉〓105邉〓106邉〓107邉〓108邉〓109
邉〓10A邉〓10B邉〓10C邉〓10D邉〓10E
さん．
```

奈良県葛城市と，東京都茨飾区．
こんにちは，渡邉粮瀛煢爻猜瓊祀柀
柀齋禮邃簑爾さん．

Specifying glyph variants by IVS precedes glyph replacement by font features. For example, only “葛” in “葛西” is changed by font features jp78 or jp90, which does not followed by any variation selector.

```
\def\TEST#1{%
  {\jfontspec[#1]{KozMinPr6N-Regular}%
  葛〓100城市，葛〓101飾区，葛西}\\}
指定なし：\TEST{}
\texttt{jp78}：\TEST{CJKShape=JIS1978}
\texttt{jp90}：\TEST{CJKShape=JIS1990}
```

指定なし：葛城市，茨飾区，葛西
jp78：葛城市，茨飾区，葛西
jp90：葛城市，茨飾区，葛西

11.3 luatexja-adjust.sty

(see Japanese version of this manual)

11.4 luatexja-ruby.sty

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……
```

東西線妙典（みようでん）駅は……
東西線の妙典（みようでん）駅は……
東西線の妙典（みようでん）という駅……
東西線葛西（かさい）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……
```

東西線の妙(みよう)典(でん)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}
```

妙典(みようでん) 葛西(かさい) 神楽坂(かぐらざか)

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}
}
```

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

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
}
```

……を 承(うけたまわ)
る ……を承(うけたまわ)る
……を
承(うけたまわ) る

11.5 `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 ...

`picture`environment

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

`\Kanji{`⟨*counter_name*⟩`}`

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

参照番号

Part III
Implementations

12 Storing Parameters

12.1 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@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@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) The font index of current Japanese font for horizontal direction.

`\ltj@curtfnt` (attribute) The font index of current Japanese font for vertical direction.

`\ltj@charclass` (attribute) The character class of Japanese *glyph_node*.

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

`\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.

`\ltj@autoxspc` (attribute) Whether the auto insertion of xkanjiskip is allowed at the node.

`\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* − 2)–(*from_jfm* + 2) (4–8)** Glues/kerns from JFM.

***kanji_skip* (9), *kanji_skip_jfm* (10)** Glues from kanjiskip.

***xkanji_skip* (11), *xkanji_skip_jfm* (12)** Glues from xkanjiskip.

***processed* (13)** Nodes which is already processed by ....

***ic_processed* (14)** Glues from an italic correction, but already processed in the insertion process of **JAglue**s.

***boxbdd* (15)** Glues/kerns that inserted just the beginning or the ending of an hbox or a paragraph.

`\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)**

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 the callback process of `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* the callback process of luaotfload. Nodes of this type is used in `\CID`, `\UTF` and IVS support.

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.

12.2 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 8.1, 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:

```
1226 void package(int c)
1227 {
1228     scaled h;                    /* height of box */
1229     halfword p;                  /* first node in a box */
1230     scaled d;                    /* max depth */
1231     int grp;
1232     grp = cur_group;
1233     d = box_max_depth;
1234     unsave();
1235     save_ptr -= 4;
1236     if (cur_list.mode_field == -hmode) {
1237         cur_box = filtered_hpack(cur_list.head_field,
1238                                  cur_list.tail_field, saved_value(1),
1239                                  saved_level(1), grp, saved_level(2));
1240         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[8].

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.

12.3 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

[8] `[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 9. Definiton of parameter setting commands

```
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*.

12.4 Extending Parameters

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

■Setting parameters Figure 9 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.

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

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
```

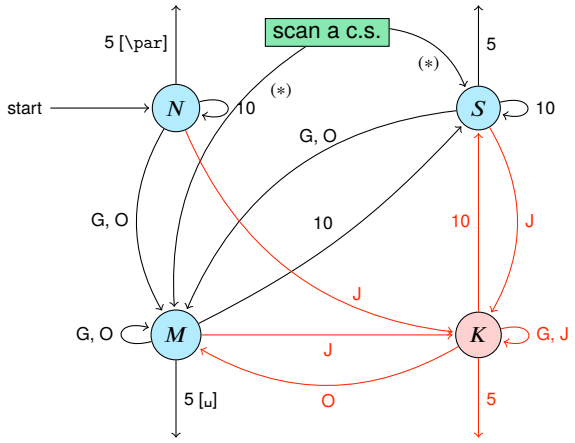
Here the argument of `luatexja.unary_pars.hoge` is the stack level of LuaTEX-ja's stack system (see Subsection 12.2).

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



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 10. State transitions of pTEX's input processor

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

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

13 Linebreak after a Japanese Character

13.1 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 10. Note that pTEX doesn't leave state *K* after "beginning/ending of a group" characters.

13.2 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 `U+FFFFF` (its category code is set to 14 (*comment*) by LuaTEX-ja) 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`[9] is 5 (*end-of-line*).
> 2. The category code of `U+FFFFF` 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
```

xyz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 `U+FFFFF` 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.

14 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 `U+FFFFF`) 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}`.

14.1 Notes and additional keys

■**Escaping to LATEX** We used the `process_input_buffer` callback to output **JAchar**s. But it has a drawback; any commands whose name contains a **JAchar** cannot be used in any "escape to LATEX".

Consider the following input:

[9] Usually, it is ⟨*return*⟩ (whose character code is 13).

```
\begin{lstlisting}[escapechar=\#]
#\ほげ ␣x ぴよ#
\end{lstlisting}
```

The line 2 is transformed by the callback to

#\U+FFFFFほU+FFFFFげ␣xU+FFFFFぴU+FFFFFよ#

before the line is actually processed. In the escape (between the character "#"), the category code of U+FFFFF is set to 9 (*ignored*). Hence the control symbol "\U+FFFFF" will be executed, instead of "\ほげ".

■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U+E0100城市，葛U+E0101飾区，葛西
\end{lstlisting}
```

```
葛城市，葛飾区，葛西
```

- 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U+E0100城市，葛U+E0101飾区，葛西
\end{lstlisting}
```

```
葛VS17城市，葛VS18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{%
  \textcolor{blue}{\raisebox{3.5pt}{\tt%
    \fboxsep=0.5pt\fbox{\tiny \oalign{0#1#2\crcr#3#4#5\crcr}}}}%
}
{\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   :
```

14.2 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	Kanji	Open	Close
`\lst@ifletter`	T	F	T	F	T
`\lst@ifkanji`	F	F	T	T	F
Meaning	char in an identifier	other alphabet	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:

		Next				
		Letter	Other	Kanji	Open	Close
Prev	Letter	collects	outputs			collects
	Other	outputs	collects	outputs		collects
	Kanji	outputs				collects
	Open	collects				
	Close	outputs				collects

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.

Table 11. cid key and corresponding files

cid key	name of the cache	used CMaps	
Adobe-Japan1-*	ltj-cid-auto-adobe-japan1.lua	UniJIS2004-UTF32-H	Adobe-Japan1-UCS2
Adobe-Korea1-*	ltj-cid-auto-adobe-korea1.lua	UniKS-UTF32-H	Adobe-Korea1-UCS2
Adobe-GB1-*	ltj-cid-auto-adobe-gb1.lua	UniGB-UTF32-H	Adobe-GB1-UCS2
Adobe-CNS1-*	ltj-cid-auto-adobe-cns1.lua	UniCNS-UTF32-H	Adobe-CNS1-UCS2

15 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"), caches of the *binary*, precompiled form are supported.
  - We cannot share same binary cache for LuaTEX and LuaJITTEX. Hence we distinguish them by their extension, ".luc" for LuaTEX and ".lub" for LuaJITTEX.
  - In loading a cache, the binary cache precedes the text form.
  - When LuaTEX-ja updates a cache hoge.lua, its binary version is also updated.

15.1 Use of Cache

LuaTEX-ja uses the following cache:

ltj-cid-auto-adobe-japan1.lua
: The font table of a CID-keyed non-embedded Japanese font. This is loaded in every run. It is created from two CMaps, UniJIS2004-UTF32-H 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 11, if you specified cid key in \jfont to use other CID-keyed non-embedded fonts for Chinese or Korean, as in Page 22.

extra_***.lua
: This file stores the table which stores the following.

  - unicode variants in a font "***"
  - vertical width of glyphs, if it is not equal to the sum of the height of ascender and the depth of descender
  - vertical variants

  The following is the structure of the that table.

```
return {
 {
  [10955]={    -- U+2ACB "Subset Of Above Not Equal To"
   [65024]=983879, -- <2ACB FE00>
   ["vwidth"]=0.98, -- vertical width
  },
  [37001]={    -- U+9089 "邉"
   [0]=37001,  --   <9089 E0100>
   991049,     --   <9089 E0101>
   ...
   ["vert"]=995025, -- vertical variant
  },
  ...
 },
 ["chksum"]="FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF", -- checksum of the fontfile
 ["version"]=2, -- version of the cache
}
```

`ltj-jisx0208.{luc|lub}`
: The binary 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.

15.2 Internal

Cache management system of LuaTEX-ja is stored in `luatexja.base` (`ltj-base.lua`). There are three 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 text form ⟨*filename*⟩`.lua` 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|lub}`. 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 text form ⟨*filename*⟩`.lua`. Hence, the return value of `load_cache` is non-nil, if and only if the updated cache is found.

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.sourceforge.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