The LuaTEX-ja package

The LuaTEX-ja project team

March 22, 2014

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, *LuaTEX-ja is not a just porting of pTEX; unnatural specifications/behaviors of pTEX were not adopted*.

The followings are major changes from pTEX:

- A Japanese font is a tuple of a "real" font, a Japanese font metric (*JFM*, for short).
- 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 function completely, because of a specification of LuaTEX.
- 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.
- At the present, vertical typesetting (*tategaki*), is not supported in LuaTEX-ja.

For detailed information, see Part III.

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 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

To install the LuaTeX-ja package, you will need:

- LuaTeX beta-0.74.0 (or later)
- luaotfload v2.2 (or later)
- luatexbase v0.6
- xunicode v0.981 (2011/09/09)
- adobemapping (Adobe cmap and pdfmapping files)

This version of LuaTeX-ja no longer supports TeX Live 2012 (or older version), since LuaTeX binary and luaotfload is updated in TeX Live 2013.

Now LuaTeX-ja is available from the following archive and distributions:

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

If you are using TeX Live 2013, you can install LuaTeX-ja from TeX Live manager (`tlmgr`):

```
$ tlmgr install luatexja
```

■Manual installation

1. Download the source archive, by one of the following method. At the present, LuaTeX-ja has no *stable* release.

   - Copy 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

- 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 temporaly directory, copy CMaps in it, run LuaTeX-ja in this directory, and finally delete the temporaly 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 6 Japanese fonts are preloaded:

classification	font name	'10 pt'	'7 pt'	'5 pt'
mincho	Ryumin-Light	`\tenmin`	`\sevenmin`	`\fivemin`
gothic	GothicBBB-Medium	`\tengt`	`\sevengt`	`\fivegt`

  - It is widely accepted that fonts "Ryumin-Light" and "GothicBBB-Medium" aren't embedded into PDF files, and a PDF reader substitute them by some external Japanese fonts (*e.g.*, Ryumin-Light is substituted with Kozuka Mincho in Adobe Reader). We adopt this custom to the default setting.
  - 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

■LaTeX 2ε Using in LaTeX 2ε 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`):

- `JY3` is the font encoding for Japanese fonts (in horizontal direction).
  When vertical typesetting is supported by LuaTeX-ja in the future, `JT3` will be used for vertical fonts.

[1] `UniJIS2004-UTF32-H` and `Adobe-Japan1-UCS2`.

- Traditionally, Japanese documents use two typeface category: *mincho* (明朝体) and *gothic* (ゴシック体). *mincho* is used in the main text, while *gothic* is used in the headings or for emphasis.

classification			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 name	`\mdseries`	`\bfseries`	scale
mincho (明朝体)	`mc`	Ryumin-Light	GothicBBB-Medium	0.962216
gothic (ゴシック体)	`gt`	GothicBBB-Medium	GothicBBB-Medium	0.962216

  Note that the bold series in both family are same as the medium series of *gothic* family. This is a convention in pLaTeX. This is trace that there were only 2 fonts (these are Ryumin-Light and GothicBBB-Medium) in early years of DTP. There is no italic nor slanted shape for these `mc` and `gt`.

- Japanese characters in math mode are typeset by the font family `mc`.

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` and `ltjsclasses`.

3 Changing Fonts

3.1 plain TeX and LaTeX 2$_\varepsilon$

■plain TeX To change Japanese fonts in plain TeX, you must use the command `\jfont`. So please see Subsection 6.1.

■LaTeX 2$_\varepsilon$ (NFSS2) For LaTeX 2$_\varepsilon$, LuaTeX-ja adopted most of the font selection system of pLaTeX 2$_\varepsilon$ (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`

  `\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 9.1.

- 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 6.4.

3.2 fontspec

To coexist with the fontspec package, it is needed to load luatexja-fontspec package in the preamble. This additional 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	\newjfontfamily \newfontfamily
Japanese fonts alphabetic fonts	\newjfontface \newfontface	\defaultjfontfeatures \defaultfontfeatures	\addjfontfeatures \addfontfeatures	

```
1 \fontspec[Numbers=OldStyle]{LMSans10-Regular}
2 \jfontspec[CJKShape=NLC]{KozMinPr6N-Regular}
3 JIS~X~0213:2004→辻
4
5 \jfontspec[CJKShape=JIS1990]{KozMinPr6N-Regular}
6 JIS~X~0208:1990→辻
```

JIS X 0213:2004 →辻
JIS X 0208:1990 →辻

Note that there is no command named \setmonojfont, since 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 commands. This is because of the compatibility with previous versions of LuaTEX-ja (see 6.1).

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*[2]. 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 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.

■Presets for multi weight Besides `morisawa-pro` and `morisawa-pr6n` presets, fonts are specified by fontname, not by filename.

`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).

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

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)	IPAMincho	IPAexMincho	MS Mincho
gothic	GothicBBB-Medium (non-embedded)	IPAGothic	IPAexGothic	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	IPAMincho	IPAexMincho	MS Mincho
mincho bold	HG Mincho E		
Gothic medium			
without `deluxe`	IPAGothic	IPAexGothic	MS Gothic
with `jis2004`	IPAGothic	IPAexGothic	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 filename (`hgrme.ttc`, `hgrge.ttc`, `hgrsgu.ttc`, `hgrsmp.ttf`).

`jis2004` by filename (`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
ﾊﾝｶｸｶﾀｶﾅ

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 $+n$ meansthat "… is a **JAchar**".

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

■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^{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:

- § (`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
```

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

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.
- 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.

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`.

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 7.1).

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` 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 (yalbaselineshift parameter) and that of Japanese fonts (yjabaselineshift parameter).

```
1 \vrule width 150pt height 0.4pt depth 0pt \
    hskip-120pt
2 \ltjsetparameter{yjabaselineshift=0pt,
    yalbaselineshift=0pt}abcあいう
3 \ltjsetparameter{yjabaselineshift=5pt,
    yalbaselineshift=2pt}abcあいう
```

abc あいう abc あいう

Here the horizontal line in above is the baseline of a line.

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):

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

xyz 漢字 XYZ ひらがな abc かな

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[3] 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 caan be divided into the following four kinds, and LuaTEX-ja can control these four kinds separately:

[3] 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

- *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 “・”, “〴〵”, “〻”, “＝”, 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.

6 Font Metric and Japanese Font

6.1 \jfont

To load a font as a Japanese font, 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*⟩.

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+3000)	1	1	N	N	Y
・(U+30FB)	1	6	N	Y	N
゛(U+309B)	1	11	N	Y	N
゜(U+309C)	1	12	N	Y	N
´(U+FF40)	1	14	N	N	Y
¨(U+FF3E)	1	16	N	N	Y
￣(U+FFE3)	1	17	N	N	Y
＿(U+FF3F)	1	18	N	N	Y
ヽ(U+30FD)	1	19	N	Y	Y
ヾ(U+30FE)	1	20	N	Y	Y
ゝ(U+309D)	1	21	N	Y	Y
ゞ(U+309E)	1	22	N	Y	Y
〃(U+3003)	1	23	N	N	Y
仝(U+4EDD)	1	24	N	Y	Y
々(U+3005)	1	25	N	N	Y
〆(U+3006)	1	26	N	N	Y
〇(U+3007)	1	27	N	N	Y
ー(U+30FC)	1	28	N	Y	Y

	row	col.	pTEX	upTEX	LuaTEX-ja
／(U+FF0F)	1	31	N	N	Y
＼(U+FF3C)	1	32	N	N	Y
｜(U+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

Table 6. Differences between JFMs shipped with LuaTEX-ja

	jfm-ujis.lua	jfm-jis.lua	jfm-min.lua
Example 1[6]	◆◆◆◆◆◆◆ ある日モモちゃんがお使いで迷子になって泣きました. ちょっと！何	◆◆◆◆◆◆◆ ある日モモちゃんがお使いで迷子になって泣きました. ちょっと！何	◆◆◆◆◆◆◆ ある日モモちゃんがお使いで迷子になって泣きました. ちょっと！何
Example 2	漢	漢	漢
Bounding Box			

■JFM As noted in Introduction, 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 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. There are notable difference between this JFM and other 2 JFMs, as shown in Table 6.

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 1. Kerning information and kanjiskip

`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

\ltjsetparameter{differentjfm=paverage}
```

■**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 1 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=file:KozMinPr6N-Regular.otf:extend=1.5;jfm=ujis;-kern
\E あいうえお

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

あいうえお
*あいう*ABC

6.2 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. Typical use of this prefix is to specify standard, non-embedded Japanese fonts, namely, "Ryumin-Light" and "GothicBBB-Medium".

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.
?
```

6.3 Structure of a JFM File

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

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

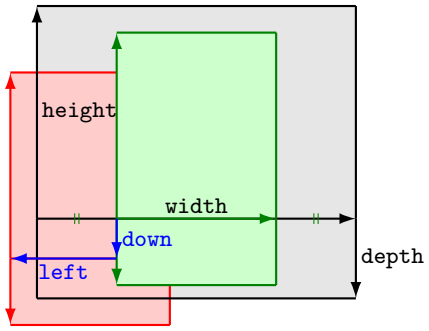
Real data are stored in the table which indicated above by `{ ... }`. So, the rest of this subsection are devoted to describe the structure of this table. Note that all lengths in a JFM file are floating-point numbers in design-size unit.

`dir=`⟨*direction*⟩ (required)

The direction of JFM. At the present, only `'yoko'` is supported.

`zw=`⟨*length*⟩ (required)

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



Consider a node containing Japanese character whose value of the `align` field is `'middle'`.

- The black rectangle is a frame 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).
- 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 2. The position of the real glyph.

`zh=`⟨*length*⟩ (required)

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

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

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

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

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

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

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

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

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

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

Specifythe 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 Figure 2.

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 10.3). 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, Thw 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', 'ヒ*' },
   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:

```
1 \jfont\a=file:KozMinPr6N-Regular.otf:jfm=test;+vert
2 \setbox0\hbox{\a 。\inhibitglue 漢}                    20.0pt
3 \the\wd0
```

Now we look why the above source outputs 20 pt, not 15 pt.

1. The ideographic full stop "。" is converted to its vertical form "﹒" (`U+FE12`), by `vert` feature.
2. The character class of "﹒" is zero, hence its width is fullwidth.
3. The character class of "漢", hence its width is fullwidth.
4. `\inhibitglue` makes that no glue will be inserted between "。" and "漢".
5. Hence the width of `\hbox` equals to 20 pt.

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

However, a starred specificaion like "`'、*'`" changes the rule. Consider the following input:

```
1 \jfont\a=file:KozMinPr6N-Regular.otf:jfm=test;+vert
2 \a 漢、\inhibitglue 漢                                 漢 漢
```

Here, the character class of the ideographic comma "、" (`U+3001`) is determined as following:

1. As the case of "。", the ideographic comma "、" is converted to its vertical form "﹑" (`U+FE11`).
2. The character class of "﹑" 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 comma "、" in above belongs the character class 2000.

Table 7. 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.

6.4 Math Font Family

TEX handles fonts in math formulas by 16 font families[4], and each family has three fonts: `\textfont`, `\scriptfont` and `\scriptscriptfont`.

LuaTEX-ja's handling of Japanese fonts in math formulas is similar; Table 7 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.

6.5 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.

```
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*:

[4] 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.

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

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

`jfm` The internal number of the JFM.

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

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

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

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

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

- 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[5]. 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.

```
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

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

- 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

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

7 Parameters

7.1 `\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 11.

`\ltjsetparameter` and `\ltjglobalsetparameter` are commands for assigning parameters. These take one argument which is a ⟨*key*⟩=⟨*value*⟩ list. The list of allowed keys are described in the next subsection. 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 assignment.

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.

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. 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.

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]

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]

7.2 \ltjgetparameter

`\ltjgetparameter` is a conteol 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*. This is outputted by `tex.write()`, so 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

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*⟩.

- 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
```

8 Other Commands for plain TeX and LaTeX 2ε

8.1 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, ゝ
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

8.2 \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.

8.3 \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.

9 Commands for LaTeX 2ε

9.1 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:

```
\DeclareKanjiFamily{JY3}{edm}{}
\DeclareFontShape{JY3}{edm}{m}{n}    {<-> s*KozMinPr6N-Regular:jfm=ujis;-kern}{}
\DeclareFontShape{JY3}{edm}{m}{green}{<-> s*KozMinPr6N-Regular:jfm=ujis;-kern;color=007F00}{}
\DeclareFontShape{JY3}{edm}{m}{blue} {<-> s*KozMinPr6N-Regular:jfm=ujis;-kern;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 3. An example of `\DeclareAlternateKanjiFont`

`\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*⟩`}`

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. This command defines a new encoding scheme for Japanese font family (in horizontal direction).

`\DeclareKanjiEncodingDefaults{`⟨*text-settings*⟩`}{`⟨*math-settings*⟩`}`

`\DeclareKanjiSubstitution{`⟨*encoding*⟩`}{`⟨*family*⟩`}{`⟨*series*⟩`}{`⟨*shape*⟩`}`

`\DeclareErrorKanjiFont{`⟨*encoding*⟩`}{`⟨*family*⟩`}{`⟨*series*⟩`}{`⟨*shape*⟩`}{`⟨*size*⟩`}`

The above 3 commands are just the counterparts for `\DeclareFontEncodingDefaults` and others.

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

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

This command sets the "accompanied" alphabetic font 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.

Current LuaTEX-ja does not support vertical typesetting, so this `\adjustbaseline` has almost no effect.

`\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 8.3), 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 3.

- 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.
- …

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

10 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`.

10.1 `luatexja-fontspec.sty`

As described in Subsection 3.2, this optional package provides the counterparts for several commands defined in the `fontspec` package. 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` respectively. `CID` is effective only when with `NoEmbed` described below. See Subsections 6.1 and 6.2 for details.

`NoEmbed` By specifying this key, one can use "name-only" Japanese font which will not be embedded in the output PDF file. See Subsection 6.2.

10.2 luatexja-otf.sty

This optional package supports typesetting characters in Adobe-Japan1 character collection (or other CID character collecton, 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.

■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 extened 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&#xE0100;城市と，東京都葛&#xE0101;飾区．\\
こんにちは，渡
邉&#xE0100;邉&#xE0101;邉&#xE0102;邉&#xE0103;邉&#xE0104;
邉&#xE0105;邉&#xE0106;邉&#xE0107;邉&#xE0108;邉&#xE0109;
邉&#xE010A;邉&#xE010B;邉&#xE010C;邉&#xE010D;邉&#xE010E;
さん．
```

奈良県葛城市と，東京都葛飾区．
こんにちは，渡邉邉邉邉邉邉邉邉邉
邉邉邉邉邉邉さん．

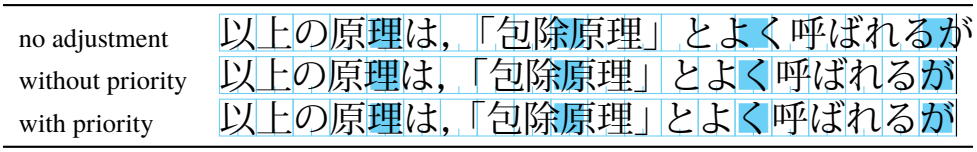
Specifying glyph variants by IVS precedes glyph replacement by font features. For example, only “葛” in “葛西” is changed by font features `jp78` or `jp90`, which does not followed by any variation selector.

```
\def\TEST#1{%
  {\jfontspec[#1]{KozMinPr6N-Regular}%
  葛&#xE0100;城市，葛&#xE0101;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，葛飾区，葛西
jp78：葛城市，葛飾区，葛西
jp90：葛城市，葛飾区，葛西

10.3 luatexja-adjust.sty

(see Japanese version of this manual)



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 4. Line adjustment

Part III
Implementations

11 Storing Parameters

11.1 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.

`\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@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)** Glues from an italic correction (`\/`). This distinction of origins of glues (from explicit `\kern`, or from `\/`) is needed in the insertion process of xkanjiskip.

***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.

Furthermore, LuaTEX-ja uses several user-defined whatsit nodes for inrernal processing. All those nodes store a natural number (hence the node's `type` is 100). 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 …

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

11.2 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 7.1, 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.

```
\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 5. Definiton of parameter setting commands

■**Implementation** The code of stack system is based on that in a post of Dev-luatex mailing list[6].

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.

11.3 Lua Functions of the Stack System

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

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

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

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

Stored data can be obtained as the return value of

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

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

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

11.4 Extending Parameters

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

■Setting parameters Figure 5 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`.

12 Linebreak after a Japanese Character

12.1 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 6. Note that pTEX doesn't leave state K after "beginning/ending of a group" characters.

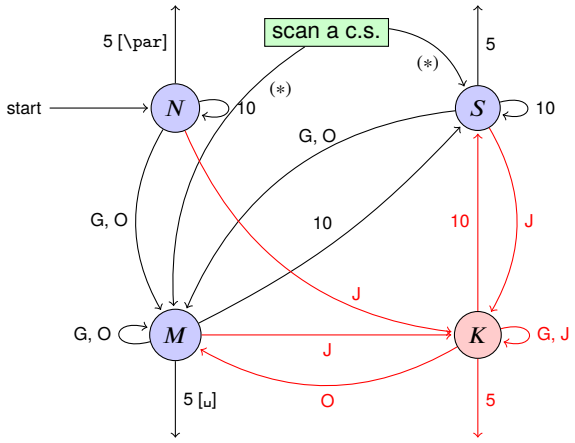
12.2 Behavior in LuaTEX-ja

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:



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 6. State transitions of pTEX's input processor

1. The category code of `\endlinechar`[7] 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]{TeX Gyre Termes}
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.

13 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

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

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}.

13.1 Notes

■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:

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

The line 2 is transformed by the callback to

#\OFFほOFFげ␣xOFFぴOFF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 "\OFF" will be executed, instead of "\ほげ".

13.2 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).

Charatcers above `U+0080` *except Variation Selectors* are classified into above 5 classes by the following rules:

- **ALchar**s above `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.

14 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.

14.1 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 8, if you specified `cid` key in `\jfont` to use other CID-keyed non-embedded fonts for Chinese or Korean, as in Page 18.

`ivs_***.lua` This file stores the table of Unicode variants in a font “`***`”. The structure of the table is the following:

Table 8. `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`

```
return {
 {
  [10955]={    -- U+2ACB "Subset Of Above Not Equal To"
   [65024]=983879, -- <2ACB FE00>
  },
  [37001]={    -- U+9089 "邉"
   [0]=37001,  --   <9089 E0100>
   991049,     --   <9089 E0101>
   ...
  },
  ...
 },
 ["chksum"]="FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF", -- checksum of the fontfile
 ["version"]=4, -- 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.

14.2 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 filename *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.

A Package versions used in this document

This document was typeset using the following packages:

```
geometry.sty             2010/09/12 v5.6 Page Geometry
keyval.sty               1999/03/16 v1.13 key=value parser (DPC)
ifpdf.sty                2011/01/30 v2.3 Provides the ifpdf switch (HO)
ifvtex.sty               2010/03/01 v1.5 Detect VTeX and its facilities (HO)
ifxetex.sty              2010/09/12 v0.6 Provides ifxetex conditional
luatexja-adjust.sty      2013/05/14
luatexja.sty             2013/05/14 Japanese Typesetting with LuaTeX
luatexja-core.sty        2014/02/01 Core of LuaTeX-ja
luaotfload.sty           2014/02/05 v2.4-3 OpenType layout system
luatexbase.sty           2013/05/11 v0.6 Resource management for the LuaTeX macro programmer
ifluatex.sty             2010/03/01 v1.3 Provides the ifluatex switch (HO)
luatex.sty               2010/03/09 v0.4 LuaTeX basic definition package (HO)
infwarerr.sty            2010/04/08 v1.3 Providing info/warning/error messages (HO)
etex.sty                 1998/03/26 v2.0 eTeX basic definition package (PEB)
luatex-loader.sty        2010/03/09 v0.4 Lua module loader (HO)
luatexbase-compat.sty    2011/05/24 v0.4 Compatibility tools for LuaTeX
luatexbase-modutils.sty  2013/05/11 v0.6 Module utilities for LuaTeX
luatexbase-loader.sty    2013/05/11 v0.6 Lua module loader for LuaTeX
luatexbase-regs.sty      2011/05/24 v0.4 Registers allocation for LuaTeX
luatexbase-attr.sty      2013/05/11 v0.6 Attributes allocation for LuaTeX
luatexbase-cctb.sty      2013/05/11 v0.6 Catcodetable allocation for LuaTeX
luatexbase-mcb.sty       2013/05/11 v0.6 Callback management for LuaTeX
ltxcmds.sty              2011/11/09 v1.22 LaTeX kernel commands for general use (HO)
pdftexcmds.sty           2011/11/29 v0.20 Utility functions of pdfTeX for LuaTeX (HO)
xkeyval.sty              2012/10/14 v2.6b package option processing (HA)
ltj-base.sty             2013/05/14
ltj-latex.sty            2013/05/14 LaTeX support of LuaTeX-ja
lltjfont.sty             2014/01/23 Patch to NFSS2 for LuaTeX-ja
lltjdefs.sty             2013/06/12 Default font settings of LuaTeX-ja
lltjcore.sty             2013/05/14 Patch to LaTeX2e Kernel for LuaTeX-ja
luatexja-compat.sty      2013/12/22 Compatibility with pTeX
expl3.sty                2014/01/07 v4646 L3 Experimental code bundle wrapper
l3names.sty              2014/01/04 v4640 L3 Namespace for primitives
l3bootstrap.sty          2014/01/04 v4640 L3 Experimental bootstrap code
l3basics.sty             2014/01/04 v4642 L3 Basic definitions
l3expan.sty              2014/01/04 v4642 L3 Argument expansion
l3tl.sty                 2013/12/27 v4625 L3 Token lists
l3seq.sty                2013/12/14 v4623 L3 Sequences and stacks
l3int.sty                2013/08/02 v4583 L3 Integers
l3quark.sty              2013/12/14 v4623 L3 Quarks
l3prg.sty                2014/01/04 v4642 L3 Control structures
l3clist.sty              2013/07/28 v4581 L3 Comma separated lists
l3token.sty              2013/08/25 v4587 L3 Experimental token manipulation
l3prop.sty               2013/12/14 v4623 L3 Property lists
l3msg.sty                2013/07/28 v4581 L3 Messages
l3file.sty               2013/10/13 v4596 L3 File and I/O operations
l3skip.sty               2013/07/28 v4581 L3 Dimensions and skips
l3keys.sty               2013/12/08 v4614 L3 Experimental key-value interfaces
l3fp.sty                 2014/01/04 v4642 L3 Floating points
l3box.sty                2013/07/28 v4581 L3 Experimental boxes
l3coffins.sty            2013/12/14 v4624 L3 Coffin code layer
l3color.sty              2012/08/29 v4156 L3 Experimental color support
l3luatex.sty             2013/07/28 v4581 L3 Experimental LuaTeX-specific functions
```

```
l3candidates.sty         2014/01/06 v4643 L3 Experimental additions to l3kernel
amsmath.sty              2013/01/14 v2.14 AMS math features
amstext.sty              2000/06/29 v2.01
amsgen.sty               1999/11/30 v2.0
amsbsy.sty               1999/11/29 v1.2d
amsopn.sty               1999/12/14 v2.01 operator names
array.sty                2008/09/09 v2.4c Tabular extension package (FMi)
tikz.sty                 2013/12/13 v3.0.0 (rcs-revision 1.142)
pgf.sty                  2013/12/18 v3.0.0 (rcs-revision 1.14)
pgfrcs.sty               2013/12/20 v3.0.0 (rcs-revision 1.28)
everyshi.sty             2001/05/15 v3.00 EveryShipout Package (MS)
pgfcore.sty              2010/04/11 v3.0.0 (rcs-revision 1.7)
graphicx.sty             1999/02/16 v1.0f Enhanced LaTeX Graphics (DPC,SPQR)
graphics.sty             2009/02/05 v1.0o Standard LaTeX Graphics (DPC,SPQR)
trig.sty                 1999/03/16 v1.09 sin cos tan (DPC)
pgfsys.sty               2013/11/30 v3.0.0 (rcs-revision 1.47)
xcolor.sty               2007/01/21 v2.11 LaTeX color extensions (UK)
pgfcomp-version-0-65.sty 2007/07/03 v3.0.0 (rcs-revision 1.7)
pgfcomp-version-1-18.sty 2007/07/23 v3.0.0 (rcs-revision 1.1)
pgffor.sty               2013/12/13 v3.0.0 (rcs-revision 1.25)
pgfkeys.sty
pgfmath.sty
pict2e.sty               2014/01/12 v0.2z Improved picture commands (HjG,RN,JT)
multienum.sty
float.sty                2001/11/08 v1.3d Float enhancements (AL)
booktabs.sty             2005/04/14 v1.61803 publication quality tables
multicol.sty             2011/06/27 v1.7a multicolumn formatting (FMi)
listings.sty             2014/03/04 1.5c (Carsten Heinz)
lstmisc.sty              2014/03/04 1.5c (Carsten Heinz)
showexpl.sty             2014/01/19 v0.3l Typesetting example code (RN)
calc.sty                 2007/08/22 v4.3 Infix arithmetic (KKT,FJ)
ifthen.sty               2001/05/26 v1.1c Standard LaTeX ifthen package (DPC)
varwidth.sty             2009/03/30 ver 0.92;  Variable-width minipages
hyperref.sty             2012/11/06 v6.83m Hypertext links for LaTeX
hobsub-hyperref.sty      2012/05/28 v1.13 Bundle oberdiek, subset hyperref (HO)
hobsub-generic.sty       2012/05/28 v1.13 Bundle oberdiek, subset generic (HO)
hobsub.sty               2012/05/28 v1.13 Construct package bundles (HO)
intcalc.sty              2007/09/27 v1.1 Expandable calculations with integers (HO)
etexcmds.sty             2011/02/16 v1.5 Avoid name clashes with e-TeX commands (HO)
kvsetkeys.sty            2012/04/25 v1.16 Key value parser (HO)
kvdefinekeys.sty         2011/04/07 v1.3 Define keys (HO)
pdfescape.sty            2011/11/25 v1.13 Implements pdfTeX's escape features (HO)
bigintcalc.sty           2012/04/08 v1.3 Expandable calculations on big integers (HO)
bitset.sty               2011/01/30 v1.1 Handle bit-vector datatype (HO)
uniquecounter.sty        2011/01/30 v1.2 Provide unlimited unique counter (HO)
letltxmacro.sty          2010/09/02 v1.4 Let assignment for LaTeX macros (HO)
hopatch.sty              2012/05/28 v1.2 Wrapper for package hooks (HO)
xcolor-patch.sty         2011/01/30 xcolor patch
atveryend.sty            2011/06/30 v1.8 Hooks at the very end of document (HO)
atbegshi.sty             2011/10/05 v1.16 At begin shipout hook (HO)
refcount.sty             2011/10/16 v3.4 Data extraction from label references (HO)
hycolor.sty              2011/01/30 v1.7 Color options for hyperref/bookmark (HO)
auxhook.sty              2011/03/04 v1.3 Hooks for auxiliary files (HO)
kvoptions.sty            2011/06/30 v3.11 Key value format for package options (HO)
url.sty                  2013/09/16  ver 3.4  Verb mode for urls, etc.
rerunfilecheck.sty       2011/04/15 v1.7 Rerun checks for auxiliary files (HO)
bookmark.sty             2011/12/02 v1.24 PDF bookmarks (HO)
```

```
amsthm.sty              2004/08/06 v2.20
luatexja-otf.sty        2013/05/14
luatexja-ajmacros.sty   2013/05/14
luatexja-preset.sty     2013/10/28 Japanese font presets
luatexja-fontspec.sty   2014/01/23 fontspec support of LuaTeX-ja
fontspec.sty            2013/05/20 v2.3c Font selection for XeLaTeX and LuaLaTeX
xparse.sty              2013/12/31 v4634 L3 Experimental document command parser
fontspec-patches.sty    2013/05/20 v2.3c Font selection for XeLaTeX and LuaLaTeX
fixltx2e.sty            2006/09/13 v1.1m fixes to LaTeX
fontspec-luatex.sty     2013/05/20 v2.3c Font selection for XeLaTeX and LuaLaTeX
fontenc.sty
xunicode.sty            2011/09/09 v0.981 provides access to latin accents and many other characters
                        in Unicode lower plane
unicode-math.sty        2013/05/04 v0.7e Unicode maths in XeLaTeX and LuaLaTeX
l3keys2e.sty            2013/12/31 v4634 LaTeX2e option processing using LaTeX3 keys
catchfile.sty           2011/03/01 v1.6 Catch the contents of a file (HO)
fix-cm.sty              2006/09/13 v1.1m fixes to LaTeX
filehook.sty            2011/10/12 v0.5d Hooks for input files
unicode-math-luatex.sty
lualatex-math.sty       2013/08/03 v1.3 Patches for mathematics typesetting with LuaLaTeX
etoolbox.sty            2011/01/03 v2.1 e-TeX tools for LaTeX
metalogo.sty            2010/05/29 v0.12 Extended TeX logo macros
lltjp-fontspec.sty      2013/05/14 Patch to fontspec for LuaTeX-ja
lltjp-xunicode.sty      2013/05/14 Patch to xunicode for LuaTeX-ja
lltjp-unicode-math.sty  2013/05/14 Patch to unicode-math for LuaTeX-ja
lltjp-listings.sty      2014/01/09 Patch to listings for LuaTeX-ja
epstopdf-base.sty       2010/02/09 v2.5 Base part for package epstopdf
grfext.sty              2010/08/19 v1.1 Manage graphics extensions (HO)
nameref.sty             2012/10/27 v2.43 Cross-referencing by name of section
gettitlestring.sty      2010/12/03 v1.4 Cleanup title references (HO)
```