

The LuaT_EX-ja package

The LuaT_EX-ja project team

March 23, 2014

Contents

I	User's manual	3
1	Introduction	3
1.1	Backgrounds	3
1.2	Major Changes from pTeX	3
1.3	Notations	4
1.4	About the Project	4
2	Getting Started	5
2.1	Installation	5
2.2	Cautions	6
2.3	Using in plain TeX	6
2.4	Using in L ^A T _E X	6
3	Changing Fonts	7
3.1	plain TeX and L ^A T _E X 2 _ε	7
3.2	fontspec	8
3.3	Presets	8
3.4	\CID, \UTF, and macros in japanese-otf package	11
4	Changing Parameters	11
4.1	Editing the Range of JAchars	11
4.2	kanjiskip and xkanjiskip	13
4.3	Insertion Setting of xkanjiskip	13
4.4	Shifting the baseline	14
II	Reference	14
5	\catcode in LuaTeX-ja	14
5.1	Preliminaries: \kcatcode in pTeX and upTeX	14
5.2	Case of LuaTeX-ja	14
5.3	Non-kanji Characters in a Control Word	15
6	Font Metric and Japanese Font	15
6.1	\jfont	15
6.2	Prefix psft	18
6.3	Structure of a JFM File	18
6.4	Math Font Family	21
6.5	Callbacks	21
7	Parameters	23
7.1	\ltjsetparameter	23
7.2	\ltjgetparameter	25
8	Other Commands for plain TeX and L^AT_EX 2_ε	25
8.1	Commands for Compatibility with pTeX	25
8.2	\inhibitglue	26
8.3	\ltjdeclarealtfont	26

9	Commands for L^AT_EX 2_ε	27
9.1	Patch for NFSS2	27
10	Addons	28
10.1	luatexja-fontspec.sty	29
10.2	luatexja-otf.sty	29
10.3	luatexja-adjust.sty	30
III	Implementations	30
11	Storing Parameters	30
11.1	Used Dimensions, Attributes and Whatsit Nodes	30
11.2	Stack System of LuaT _E X-ja	31
11.3	Lua Functions of the Stack System	32
11.4	Extending Parameters	33
12	Linebreak after a Japanese Character	33
12.1	Reference: Behavior in p _T E _X	33
12.2	Behavior in LuaT _E X-ja	34
13	Patch for the <u>listings</u> Package	35
13.1	Notes	35
13.2	Class of Characters	35
14	Cache Management of LuaT_EX-ja	36
14.1	Use of Cache	37
14.2	Internal	37
	References	38
A	Package versions used in this document	39

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 **JAg glue**) 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 **JAg glue** 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 JAg glue in pTeX are not effective anymore.* In concrete terms, the following two methods are not effective anymore:
ちよ{}つと ちよ\つと
If you want to do so, please put an empty horizontal box (hbox) between it instead:
ちよ\hbox{}つと
- 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).
 - **J**Achar: standing for characters which used in Japanese typesetting, such as Hiragana, Katakana, Kanji, and other Japanese punctuation marks.
 - **AL**char: standing for all other characters like latin alphabets.

We say *alphabetic fonts* for fonts used in **AL**char, and *Japanese fonts* for fonts used in **J**Achar.

- 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 L^AT_EX.
- 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 | • Kazuki MAEDA | • Takayuki YATO |
| • Yusuke KUROKI | • Noriyuki ABE | • Munehiro YAMAMOTO |
| • Tomoaki HONDA | • Shuzaburo SAITO | • 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 ltjclasses.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 files¹ must be found from LuaTeX. Strictly speaking, those CMaps are needed only in the first run of LuaTeX-ja after installing or updating. But it seems that MiKTeX does not satisfy this condition, so you will encounter an error like the following:

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

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

2.3 Using in plain TeX

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

```
\input luatexja.sty
```

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

- The following 6 Japanese fonts are preloaded:

classification	font name	'10 pt'	'7 pt'	'5 pt'
<i>mincho</i>	Ryumin-Light	<code>\tenmin</code>	<code>\sevenmin</code>	<code>\fivemin</code>
<i>gothic</i>	GothicBBB-Medium	<code>\tengt</code>	<code>\sevengt</code>	<code>\fivegt</code>

- 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 **J**Achar and an **A**Lchar (the parameter `xkanjiskip`) is set to

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

2.4 Using in L^AT_εX

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

```
\usepackage{luatexja}
```

It also does minimal settings (counterparts in pL^AT_εX 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.

¹UniJIS2004-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		
<i>mincho</i> (明朝体)	<code>\textmc{...}</code>	<code>{\mcfamily ...}</code>	<code>\mcdefault</code>	
<i>gothic</i> (ゴシック体)	<code>\textgt{...}</code>	<code>{\gtfamily ...}</code>	<code>\gtdefault</code>	

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

classification	family name	\mdseries	\bfseries	scale
<i>mincho</i> (明朝体)	mc	Ryumin-Light	GothicBBB-Medium	0.962216
<i>gothic</i> (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 $\text{p}\text{L}\text{A}\text{T}\text{E}\text{X}$. 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 $\text{p}\text{L}\text{A}\text{T}\text{E}\text{X}$) and `jsclasses` (classes by Haruhiko Okumura), namely, `ltjclasses` and `ltjsclasses`.

3 Changing Fonts

3.1 plain TEX and $\text{L}\text{A}\text{T}\text{E}\text{X} 2_{\epsilon}$

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

■ **$\text{L}\text{A}\text{T}\text{E}\text{X} 2_{\epsilon}$ (NFSS2)** For $\text{L}\text{A}\text{T}\text{E}\text{X} 2_{\epsilon}$, `Lua TEX -ja` adopted most of the font selection system of $\text{p}\text{L}\text{A}\text{T}\text{E}\text{X} 2_{\epsilon}$ (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	<code>\romanencoding</code>	<code>\romanfamily</code>	<code>\romanseries</code>	<code>\romanshape</code>	<code>\useroman</code>
Japanese fonts	<code>\kanjiencoding</code>	<code>\kanjifamily</code>	<code>\kanjiserie</code>	<code>\kanjishape</code>	<code>\usekanji</code>
both	—	—	<code>\fontseries</code>	<code>\fontshape</code>	—
auto select	<code>\fontencoding</code>	<code>\fontfamily</code>	—	—	<code>\usefont</code>

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

```

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

```

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	<code>\jfontspec</code>	<code>\setmainjfont</code>	<code>\setsansjfont</code>	<code>\newjfontfamily</code>
alphabetic fonts	<code>\fontspec</code>	<code>\setmainfont</code>	<code>\setsansfont</code>	<code>\newfontfamily</code>
Japanese fonts	<code>\newjfontface</code>	<code>\defaultjfontfeatures</code>	<code>\addjfontfeatures</code>	
alphabetic fonts	<code>\newfontface</code>	<code>\defaultfontfeatures</code>	<code>\addfontfeatures</code>	

```

1 \fontspec[Numbers=OldStyle]{LMSans10-Regular}
2 \jfontspec[CJKShape=NLC]{KozMinPr6N-Regular}
3 JIS-X-0213:2004→辻                               JIS X 0213:2004 →辻
4                                                     JIS X 0208:1990 →辻
5 \jfontspec[CJKShape=JIS1990]{KozMinPr6N-Regular}
6 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
<i>mincho</i>	medium	KozMinPro-Regular	KozMinProVI-Regular	KozMinPr6N-Regular
	bold	KozMinPro-Bold	KozMinProVI-Bold	KozMinPr6N-Bold
<i>gothic</i>	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
<i>rounded gothic</i>		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).

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

family	series	hiragino-pro	hiragino-pron
<i>mincho</i>	medium	Hiragino Mincho Pro W3	Hiragino Mincho ProN W3
	bold	Hiragino Mincho Pro W6	Hiragino Mincho ProN W6
<i>gothic</i>	medium	Hiragino Kaku Gothic Pro W3*	Hiragino Kaku Gothic ProN W3*
		Hiragino Kaku Gothic Pro W6	Hiragino Kaku Gothic ProN W6
	bold heavy	Hiragino Kaku Gothic Pro W6 Hiragino Kaku Gothic Std W8	Hiragino Kaku Gothic ProN W6 Hiragino Kaku Gothic StdN W8
<i>rounded gothic</i>		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
<i>mincho</i>	medium	A-OTF-RyuminPro-Light.otf	A-OTF-RyuminPr6N-Light.otf
	bold	A-OTF-FutoMinA101Pro-Bold.otf	A-OTF-FutoMinA101Pr6N-Bold.otf
<i>gothic</i>	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
<i>rounded gothic</i>		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
<i>mincho</i>	medium	YuMincho-Regular	YuMincho Medium
	bold	YuMincho-Demibold	YuMincho Demibold
<i>gothic</i>	medium	YuGothic-Regular*	YuGothic Medium*
		YuGothic-Bold	YuGothic Bold
	bold heavy	YuGothic-Bold YuGothic-Bold	YuGothic Bold YuGothic Bold
<i>rounded gothic</i>		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
<i>mincho</i>	Ryumin-Light (non-embedded)	IPAMincho	IPAexMincho	MS Mincho
<i>gothic</i>	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 <code>deluxe</code>	IPAGothic	IPAexGothic	MS Gothic
with <code>jis2004</code>	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 $\text{pL}\text{T}\text{E}\text{X}$, 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, $\text{L}\text{u}\text{a}\text{T}\text{E}\text{X}$ -ja supports some of functions in japanese-otf package. If you want to use these functions, load luatexja-otf package.

```

1 \jfontspec{KozMinPr6N-Regular.otf}
2 森\UTF{9DD7}外と内田百\UTF{9592}とが\UTF{9AD9
   }島屋に行く。
3
4 \CID{7652}飾区の\CID{13706}野家,
5 \CID{1481}城市, 葛西駅,
6 高崎と\CID{8705}\UTF{FA11}
7
8 \aj半角{はんかくカタカナ}

```

森鷗外と内田百間とが高島屋に行く。
葛飾区の吉野家, 葛城市, 葛西駅, 高崎と高崎
はんかくカタ

4 Changing Parameters

There are many parameters in $\text{L}\text{u}\text{a}\text{T}\text{E}\text{X}$ -ja. And due to the behavior of $\text{L}\text{u}\text{a}\text{T}\text{E}\text{X}$, most of them are not stored as internal register of TEX , but as an original storage system in $\text{L}\text{u}\text{a}\text{T}\text{E}\text{X}$ -ja. Hence, to assign or acquire those parameters, you have to use commands `\ltjsetparameter` and `\ltjgetparameter`.

4.1 Editing the Range of JAchars

$\text{L}\text{u}\text{a}\text{T}\text{E}\text{X}$ -ja divides the Unicode codespace $\text{U}+0080\text{--}\text{U}+10\text{FFFF}$ 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 $\text{L}\text{u}\text{a}\text{T}\text{E}\text{X}$ -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 $\text{L}\text{u}\text{a}\text{T}\text{E}\text{X}$ -ja, and it sets

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

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

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

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** Lua \TeX -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 up \TeX 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 **J**Achars 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) | • ˆ (U+00B4, Spacing acute) |
| • ¨ (U+00A8, Diaeresis) | • ¶ (U+00B6, Paragraph sign) |
| • ° (U+00B0, Degree sign) | • × (U+00D7, Multiplication sign) |
| • ± (U+00B1, Plus-minus 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+0300–U+036F:
Combining Diacritical Marks |
| • U+0100–U+017F: Latin Extended-A | |
| • U+0180–U+024F: Latin Extended-B | • U+1E00–U+1EFF:
Latin Extended Additional |
| • U+0250–U+02AF: IPA Extensions | |
| • U+02B0–U+02FF: Spacing Modifier Letters | |

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+1F00–U+1FFF: Greek Extended |
| • U+0400–U+04FF: Cyrillic | |

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 **J**Achars (kanjiskip).
- The default glue which inserted between a **J**Achar and an **AL**char (*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 **J**Achars and **AL**chars. 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 **J**Achars and `alxspmode` parameters **AL**chars respectively.

```
1 \ltjsetparameter{jaxspmode={`あ,preonly},
                 alxspmode={`\!,postonly}}
2 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}}
```

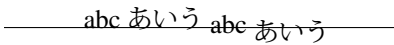
One can use also numbers to specify these two parameters (see Subsection 7.1).

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
```



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} xyz 漢字 XYZ ひらがな abc かな
5 XYZひらがな
6 }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 block³ in `upTeX`. So characters which can be used in a control word slightly differ between `pTeX` and `upTeX`.

5.2 Case of LuaTeX-ja

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

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`

<code>\kcatcode</code>	meaning	control word	widow penalty*	linebreak
15	non-cjk		(treated as usual <code>L^AT_EX</code>)	
16	kanji	Y	Y	ignored
17	kana	Y	Y	ignored
18	other	N	N	ignored
19	hangul	Y	Y	space

- *Distinction between **J**Achar or **A**Lchar* 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 **J**Achar* 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 `XYTeX`). However, the default setting of `\catcode` differs between `XYTeX` 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 `\西曆`, are used in `pTeX`. Hence, `LuaTeX-ja` have a counterpart of `unicode-letters.tex` for `LuaTeX`, to match the `\catcode` setting with that of `XYTeX`.

5.3 Non-kanji Characters in a Control Word

Because the engine differ, so non-kanji JIS X 0208 characters which can be used in a control word differ in `pTeX`, in `upTeX`, and in `LuaTeX-ja`. Table 5 shows the difference. Except for four characters “•”, “°”, “°”, “=”, `LuaTeX-ja` admits more characters in a control word than `upTeX`. *Note that the ideographic space U+3000 can be used in a control word in `LuaTeX-ja`.*

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

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:

```

1 \jfont\tradgt={file:KozMinPr6N-Regular.otf:script=latn;%
2 +trad;-kern;jfm=ujis} at 14pt
3 \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


































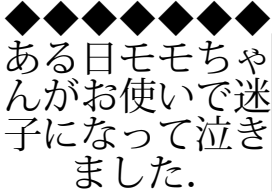
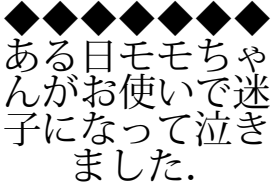
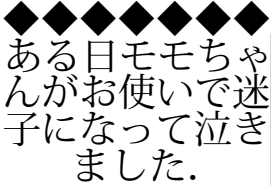



	row	col.	p \TeX	up \TeX	Lua \TeX -ja		row	col.	p \TeX	up \TeX	Lua \TeX -ja
 (U+3000)	1	1	N	N	Y	 (U+FF0F)	1	31	N	N	Y
 (U+30FB)	1	6	N	Y	N	 (U+FF3C)	1	32	N	N	Y
 (U+309B)	1	11	N	Y	N	 (U+FF5C)	1	35	N	N	Y
 (U+309C)	1	12	N	Y	N	 (U+FF0B)	1	60	N	N	Y
 (U+FF40)	1	14	N	N	Y	 (U+FF1D)	1	65	N	N	Y
 (U+FF3E)	1	16	N	N	Y	 (U+FF1C)	1	67	N	N	Y
 (U+FFE3)	1	17	N	N	Y	 (U+FF1E)	1	68	N	N	Y
 (U+FF3F)	1	18	N	N	Y	 (U+FF03)	1	84	N	N	Y
 (U+30FD)	1	19	N	Y	Y	 (U+FF06)	1	85	N	N	Y
 (U+30FE)	1	20	N	Y	Y	 (U+FF0A)	1	86	N	N	Y
 (U+309D)	1	21	N	Y	Y	 (U+FF20)	1	87	N	N	Y
 (U+309E)	1	22	N	Y	Y	 (U+3012)	2	9	N	N	Y
 (U+3003)	1	23	N	N	Y	 (U+3013)	2	14	N	N	Y
 (U+4EDD)	1	24	N	Y	Y	 (U+FFE2)	2	44	N	N	Y
 (U+3005)	1	25	N	N	Y	 (U+212B)	2	82	N	N	Y
 (U+3006)	1	26	N	N	Y	Greek letters (row 6)			Y	N	Y
 (U+3007)	1	27	N	N	Y	Cyrillic letters (row 7)			N	N	Y
 (U+30FC)	1	28	N	Y	Y						

Table 6. Differences between JFM's shipped with Lua \TeX -ja

	jfm-ujis.lua	jfm-jis.lua	jfm-min.lua
Example 1[6]			
Example 2	ちよつと！何	ちよつと！何	ちよつと！何
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, Lua \TeX -ja search and load a file named `jfm-<name>.lua`.

The following JFM's are shipped with Lua \TeX -ja:

`jfm-ujis.lua` A standard JFM in Lua \TeX -ja. This JFM is based on `upnmlminr-h.tfm`, a metric for UTF/OTF package that is used in up \TeX . 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 p \TeX . 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 p \TeX .

The difference among these three JFM's is shown in Table 6.

```

1 \ltjsetparameter{differentjfm=both}
2 \jfont\F=file:KozMinPr6N-Regular.otf:jfm=ujis
3 \jfont\G=file:KozGoPr6N-Medium.otf:jfm=ujis
4 \jfont\H=file:KozGoPr6N-Medium.otf:jfm=ujis;jfmvar=hoge
5 \F ) {\G 【 } ( % halfwidth space
6   ) {\H 『 } ( % fullwidth space
7
8 ほげ, {\G 「ほげ」 } (ほげ) \par
9 ほげ, {\H 「ほげ」 } (ほげ) % pTeX-like
10
11 \ltjsetparameter{differentjfm=paverage}

```

) 【 () 『 ()
ほげ, 「ほげ」 (ほげ)
ほげ, 「ほげ」 (ほげ)

Figure 1. Example of jfmvar key

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

```

1 \newcommand\test{\vrule ダイナミックダイクマ\vrule\}
2 \jfont\KMF = KozMinPr6N-Regular:jfm=prop;-kern at 17pt
3 \jfont\KMFk = KozMinPr6N-Regular:jfm=prop at 17pt % kern is activated
4 \jfont\KMPW = KozMinPr6N-Regular:jfm=prop;script=dflt;+pwid;-kern at 17pt
5 \jfont\KMPK = KozMinPr6N-Regular:jfm=prop;script=dflt;+pwid;+kern at 17pt
6 \begin{multicols}{2}
7 \ltjsetparameter{kanjiskip=0pt}
8 {\KMF\test \KMFk\test \KMPW\test \KMPK\test}
9
10 \ltjsetparameter{kanjiskip=3pt}
11 {\KMF\test \KMFk\test \KMPW\test \KMPK\test}
12 \end{multicols}

```

Figure 2. Kerning information and kanjiskip

`jfmvar=<string>` Sometimes there is a need that

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

Note that in `\setmainjfont` etc. which are provided by `luatex-ja-fontspec` package, kerning option is set *off* (`Kerning=Off`) by default, because of the compatibility with previous versions of Lua_T_EX-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 Lua_T_EX-ja doesn't adjust JFM's by these `extend` and `slant` settings; you have to write new JFM's 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:

```

1 \jfont\E=file:KozMinPr6N-Regular.otf:extend=1.5;jfm=ujis;-kern
2 \E あいうえお
3
4 \jfont\S=file:KozMinPr6N-Regular.otf:slant=1;jfm=ujis;-kern
5 \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.

```

1 \jfont\testJ={psft:Ryumin-Light:cid=Adobe-Japan1-6;jfm=jis} % Japanese
2 \jfont\testD={psft:Ryumin-Light:jfm=jis} % default value is Adobe-
   Japan1-6
3 \jfont\testC={psft:AdobeMingStd-Light:cid=Adobe-CNS1-6;jfm=jis} % Traditional Chinese
4 \jfont\testG={psft:SimSun:cid=Adobe-GB1-5;jfm=jis} % Simplified Chinese
5 \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:

```

1 ! Package luatexja Error: bad cid key `Adobe-Japan2'.
2
3 See the luatexja package documentation for explanation.
4 Type H <return> for immediate help.
5 <to be read again>
6
7 1.78
8
9 ? h
10 I couldn't find any non-embedded font information for the CID
11 `Adobe-Japan2'. For now, I'll use `Adobe-Japan1-6'.
12 Please contact the LuaTeX-ja project team.
13 ?

```

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”.

`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 **J**Achar which do not belong any character classes other than 0 are in the character class 0 (hence, the character class 0 contains most of **J**Achars). In the list, character(s) can be specified in the following form:

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

`width=<length>, height=<length>, depth=<length>, italic=<length>` (required)

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

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 3.

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’.

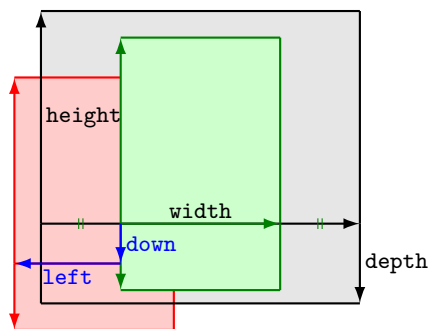
`kern={ [j]=<kern>, [j']={<kern>, [<ratio>]}, ...}`

`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 stretch, 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



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 3. The position of the real glyph

halfwidth from the ideographic full stop, and quarter-width from the fullwidth middle dot. In this case, we specify $\langle ratio \rangle$ to

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

end_stretch= $\langle kern \rangle$

end_shrink= $\langle kern \rangle$

■ **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', '匕*' },
  align = 'left', left = 0.0, down = 0.0,
  width = 1.0, height = 0.88, depth = 0.12, italic=0.0,
},
[2000] = {
  chars = { '。', '\ ', '* ', '匕' },
  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 specifaion 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
```

Table 7. Commands for Japanese math fonts

Japanese fonts	alphabetic fonts
<code>\jfam ∈ [0, 256)</code>	<code>\fam</code>
<code>jatextfont={⟨jfam⟩,⟨jfont_cs⟩}</code>	<code>\textfont⟨fam⟩=⟨font_cs⟩</code>
<code>jascriptfont={⟨jfam⟩,⟨jfont_cs⟩}</code>	<code>\scriptfont⟨fam⟩=⟨font_cs⟩</code>
<code>jascriptscriptfont={⟨jfam⟩,⟨jfont_cs⟩}</code>	<code>\scriptscriptfont⟨fam⟩=⟨font_cs⟩</code>

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, Lua \TeX -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.

■ **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 $\mu\TeX$. As a result, Lua \TeX -ja can control typesetting finer than $\mu\TeX$. 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 **J**char and anything else (such as **A**Lchar, kern, glue, ...).
- 1 The left/right boundary of an inline math formula.

■ **Porting JFM from $\mu\TeX$** See Japanese version of this manual.

6.4 Math Font Family

\TeX handles fonts in math formulas by 16 font families⁴, and each family has three fonts: `\textfont`, `\scriptfont` and `\scriptscriptfont`.

Lua \TeX -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

Lua \TeX -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.

```

1 function (⟨table⟩ jfm_info, ⟨string⟩ jfm_name)
2   return ⟨table⟩ new_jfm_info
3 end

```

⁴Omega, Aleph, Lua \TeX and ϵ - $\mu\TeX$ can handles 256 families, but an external package is needed to support this in plain \TeX and \LaTeX .

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_jfont` callback is called just when new Japanese font is loaded.

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

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

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

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:

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

- 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 node⁵. 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:

```
1 function (<number> char_class, <table> jfont_info, <number> chr_code)
2   if char_class~=0 then return char_class
3   else
4     ....
5     return (<number> new_char_class or 0)
6   end
7 end
```

⁵This version of `LuaTeX-ja` uses "direct access model" for accessing nodes, if possible.

The argument `char_class` is the result of LuaTeX-jā'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-jā's default routine.

luatexja.set_width callback This callback is called when LuaTeX-jā is trying to encapsule a **J***A*char *glyph_node*, to adjust its dimension and position.

```
1 function (<table> shift_info, <table> jfont_info, <number> char_class)
2   return <table> new_shift_info
3 end
```

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

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

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

Then, the position of glyphs is shifted up by

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

7 Parameters

7.1 \ltjsetparameter

As described before, `\ltjsetparameter` and `\ltjgetparameter` are commands for accessing most parameters of LuaTeX-jā. One of the main reason that LuaTeX-jā 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 **J***A*char 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-jā 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 **A**Lchar 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 JAchars 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 control sequence for acquiring parameters. It always takes a parameter name as first argument.

```
1 \ltjgetparameter{differentjfm},
2 \ltjgetparameter{autospacing},
3 \ltjgetparameter{kanjiskip},
4 \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 **J**Achar”, 1 if “...are **A**Lchar”. 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 **J**Achar or not by the following:

```
\ltjgetparameter{jacharrange}{\ltjgetparameter{chartorange}{<chr_code>}}
% 0 if JAchar, 1 if ALchar
```

8 Other Commands for plain T_EX and L^AT_EX 2_ε

8.1 Commands for Compatibility with pT_EX

The following commands are implemented for compatibility with pT_EX. 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*.

```

1 \newcount\hoge
2 \hoge="2423 %"
3 \the\hoge, \kansuji\hoge\
4 \jis\hoge, \char\jis\hoge\
5 \kansuji1701

```

9251, 九二五一
12355, い
一七〇一

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

```

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

```

壹漆零壹

8.2 \inhibitglue

`\inhibitglue` suppresses the insertion of **JAg**lue. 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 “う”.

```

1 \jfont\g=file:KozMinPr6N-Regular.otf:jfm=test \g
2 \fbox{\hbox{あうあ\inhibitglue う}}
3 \inhibitglue\par\noindent あ1
4 \par\inhibitglue\noindent あ2
5 \par\noindent\inhibitglue あ3
6 \par\hrule\noindent あoff\inhibitglue ice

```

あ	うあう
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`.

```

1 \DeclareKanjiFamily{JY3}{edm}{}
2 \DeclareFontShape{JY3}{edm}{m}{n}    {<-> s*KozMinPr6N-Regular:jfm=ujis;-kern}{}
3 \DeclareFontShape{JY3}{edm}{m}{green}{<-> s*KozMinPr6N-Regular:jfm=ujis;-kern;color=007F00}{}
4 \DeclareFontShape{JY3}{edm}{m}{blue} {<-> s*KozMinPr6N-Regular:jfm=ujis;-kern;color=0000FF}{}
5 \DeclareAlternateKanjiFont{JY3}{edm}{m}{n}{JY3}{edm}{m}{green}{"4E00-"67FF,{-2}{-2}}
6 \DeclareAlternateKanjiFont{JY3}{edm}{m}{n}{JY3}{edm}{m}{blue}{ "6800-"9FFF}
7 {\kanjifamily{edm}\selectfont
8 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 4. An example of `\DeclareAlternateKanjiFont`

9 Commands for $\text{\LaTeX} 2_{\epsilon}$

9.1 Patch for NFSS2

Japanese patch for NFSS2 in \LaTeX -ja is based on `plfonts.dtx` which plays the same role in $\text{\LaTeX} 2_{\epsilon}$. We will describe commands which are not described in Subsection 3.1.

additional dimensions

Like $\text{\LaTeX} 2_{\epsilon}$, \LaTeX -ja defines the following dimensions for information of current Japanese font:

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

and its `\normalsize` version:

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

Note that `\c wd` and `\c HT` 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 \LaTeX -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 $\text{p}\text{L}\text{A}\text{T}\text{E}\text{X} 2_{\epsilon}$, `\adjustbaseline` sets `\tbaselineshift` to match the vertical center of “M” and that of “あ” in vertical typesetting:

$$\tbaselineshift \leftarrow \frac{(h_M + d_M) - (h_{\text{あ}} + d_{\text{あ}})}{2} + d_{\text{あ}} - d_M,$$

where h_a and d_a denote the height of “a” and the depth, respectively.

Current $\text{L}\text{u}\text{a}\text{T}\text{E}\text{X}-\text{ja}$ does not support vertical typesetting, so this `\adjustbaseline` has almost no effect.

`\fontfamily{⟨family⟩}`

As in $\text{L}\text{A}\text{T}\text{E}\text{X} 2_{\epsilon}$, 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 $\text{L}\text{A}\text{T}\text{E}\text{X}$.

`\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 4.

- 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`:

```

1 \makeatletter
2 \SetRelationFont{JY3}{\k@family}{m}{n}{OT1}{pag}{m}{n}          あいう abc
3 % \k@family: current Japanese font family
4 \userelfont\selectfont あいう abc

```

10 Addons

$\text{L}\text{u}\text{a}\text{T}\text{E}\text{X}-\text{ja}$ has several addon packages. These addons are written as $\text{L}\text{A}\text{T}\text{E}\text{X}$ packages, but `luatexja-otf` and `luatexja-adjust` can be loaded in plain $\text{L}\text{u}\text{a}\text{T}\text{E}\text{X}$ 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 collection, if the font is supported). The package `luatexja-otf` offers the following 2 low-level commands:

`\CID{<number>}` Typeset a character whose CID number is *<number>*.

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

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

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

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

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

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

```
1 \Large
2 \jfontspec{KozMinPr6N-Regular}
3 奈良県葛0E00城市と，東京都葛0E00飾区。 \
4 こんにちは，渡
5 邊0E00邊0E01邊0E02邊0E03邊0E04
6 邊0E05邊0E06邊0E07邊0E08邊0E09
7 邊0E0A邊0E0B邊0E0C邊0E0D邊0E0E
8 さん。
```

奈良県葛城市と，東京都葛飾区。
こんにちは，渡邊邊邊邊邊邊邊邊邊邊邊
邊邊邊邊邊邊さん。

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

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

Figure 5. Line adjustment

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.

```

1 \def\TEST#1{%
2   {\jfontspec[#1]{KozMinPr6N-Regular}}%   指定なし：葛城市，葛飾区，葛西
3   葛78城市，葛90飾区，葛西\}         jp78：葛城市，葛飾区，葛西
4 指定なし：\TEST{ }                       jp90：葛城市，葛飾区，葛西
5 \texttt{jp78}：\TEST{CJKShape=JIS1978}
6 \texttt{jp90}：\TEST{CJKShape=JIS1990}

```

10.3 luatexja-adjust.sty

(see Japanese version of this manual)

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 Lua_T_EX-ja.

- `\jQ` (dimension) `\jQ` is equal to $1\text{Q} = 0.25\text{mm}$, where “Q” (also called “綴”) 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 “齒” which equals to 0.25mm 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@curjfont` (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@ykbdshift` (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) Kerns from italic correction ($\backslash/$), or from kerning information of a Japanese font. These kerns are “ignored” in the insertion process of **JAg glue**, unlike explicit `\kern`.

packed (2)

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

(from_jfm - 2)–(from_jfm + 2) (4–8) Glues/kerns from JFM.

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

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

processed (13) Nodes which is already processed by

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

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

`\ltj@kcati` (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 **J Achars**.

Furthermore, Lua \TeX -ja uses several user-defined whatsit nodes for internal 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 Lua \TeX -ja’s stack system (see the next subsection). The `value` field of these nodes is current group level.

char_by_cid Nodes for **J Achar** 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 **J Ag lues**.

11.2 Stack System of Lua \TeX -ja

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

```
1 \ltjsetparameter{kanjiskip=0pt}ふがふが.%
2 \setbox0=\hbox{%
3   \ltjsetparameter{kanjiskip=5pt}ほげほげ}   ふがふが.ほげほげ.ひよひよ
4 \box0.ひよひよ\par
```

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 Lua \TeX , this “5 pt” cannot be known from any callbacks. In the `tex/packaging.w`, which is a file in the source of Lua \TeX , there are the following codes:


```

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

```

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

■ **Implementation** The code of stack system is based on that in a post of Dev-luatex mailing list⁶.

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”, Lua \TeX -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 Lua \TeX -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 Lua \TeX -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

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

```

380 \protected\def\ltj@setpar@global{%
381   \relax\ifnum\globaldefs>0\directlua{luatexja.isglobal='global'}%
382   \else\directlua{luatexja.isglobal=''}\fi
383 }
384 \protected\def\ltjsetparameter#1{%
385   \ltj@setpar@global\setkeys[ltj]{japaram}{#1}\ignorespaces}
386 \protected\def\ltjglobalsetparameter#1{%
387   \relax\ifnum\globaldefs<0\directlua{luatexja.isglobal=''}%
388   \else\directlua{luatexja.isglobal='global'}\fi%
389   \setkeys[ltj]{japaram}{#1}\ignorespaces}

```

Figure 6. Definition of parameter setting commands

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

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

11.4 Extending Parameters

Keys for `\ltjsetparameter` and `\ltjgetparameter` can be extended, as in [luatexja-adjust](#).

■ **Setting parameters** Figure 6 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.

$$\text{luatexja.isglobal} = \begin{cases} \text{'global'} & \text{global} \\ \text{''} & \text{local} \end{cases} \quad (1)$$

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

12 Linebreak after a Japanese Character

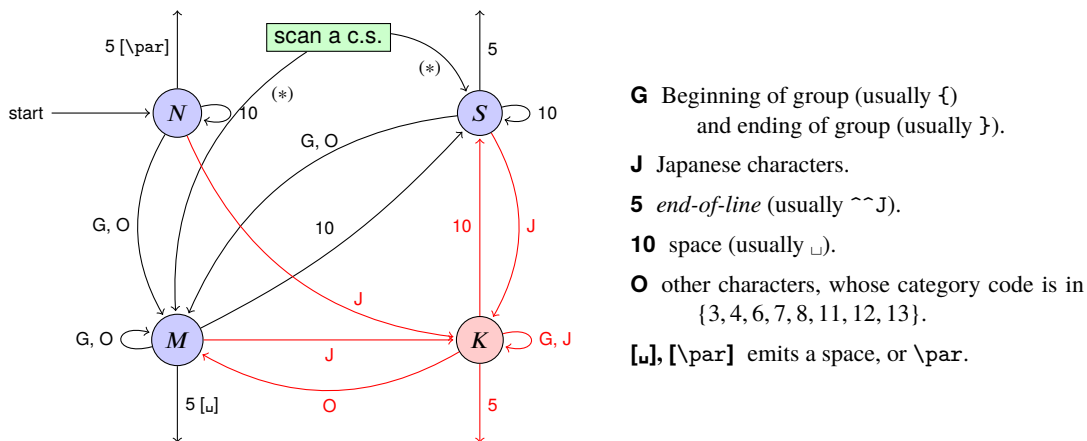
12.1 Reference: Behavior in pTeX

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 7. Note that pTeX doesn’t leave state *K* after “beginning/ending of a group” characters.



- 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 7. State transitions of $\text{p}\TeX$'s input processor

12.2 Behavior in $\text{Lua}\TeX$ -ja

States in the input processor of $\text{Lua}\TeX$ 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 $\text{Lua}\TeX$ -ja are as follows:

A character `U+FFFFF` (its category code is set to 14 (*comment*) by $\text{Lua}\TeX$ -ja) is appended to an input line, *before $\text{Lua}\TeX$ actually process it*, if and only if the following three conditions are satisfied:

1. The category code of `\endlinechar7` 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”:

$$(\text{any char})^*(\mathbf{JA}\text{char})({\text{catcode}} = 1) \cup {\text{catcode}} = 2)^*$$

■ **Remark** The following example shows the major difference from the behavior of $\text{p}\TeX$.

```

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

⁷Usually, it is `<return>` (whose character code is 13).

- When line 3 is processed by `process_input_buffer` callback, “あ” is considered as an **J**Achar. Since line 3 ends with an **J**Achar, 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 **A**Lchar. Since line 4 ends with an **A**Lchar, 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 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 omit 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 L^AT_EX** We used the `process_input_buffer` callback to output **J**Achars. But it has a drawback; any commands whose name contains a **J**Achar cannot be used in any “escape to L^AT_EX”.

Consider the following input:

```
\begin{lstlisting}[escapechar=`\#]
#\ほげ_x ひよ#
\end{lstlisting}
```

The line 2 is transformed by the callback to

```
#\☞☞☞ほ☞☞☞げ_x☞☞☞ひ☞☞☞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 “\☞☞☞” 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
<code>\lst@ifletter</code>	T	F	T	F	T
<code>\lst@ifkanji</code>	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	_____
	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:

- **ALchars** above U+0080 are classified as Letter.
- **JChars** 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 **JChars** 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.

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

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:

```
return {
  {
    [10955]={ -- U+2ACB "Subset Of Above Not Equal To"
      [65024]=983879, -- <2ACB FE00>
    },
    [37001]={ -- U+9089 "邊"
      [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. *T_EX by Topic, A T_EXnician'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 巖己, 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:

<code>geometry.sty</code>	2010/09/12 v5.6 Page Geometry
<code>keyval.sty</code>	1999/03/16 v1.13 key=value parser (DPC)
<code>ifpdf.sty</code>	2011/01/30 v2.3 Provides the ifpdf switch (HO)
<code>ifvtex.sty</code>	2010/03/01 v1.5 Detect VTeX and its facilities (HO)
<code>ifxetex.sty</code>	2010/09/12 v0.6 Provides ifxetex conditional
<code>luatexja-adjust.sty</code>	2013/05/14
<code>luatexja.sty</code>	2013/05/14 Japanese Typesetting with LuaTeX
<code>luatexja-core.sty</code>	2014/02/01 Core of LuaTeX-ja
<code>luaotfload.sty</code>	2014/02/05 v2.4-3 OpenType layout system
<code>luatexbase.sty</code>	2013/05/11 v0.6 Resource management for the LuaTeX macro programmer
<code>ifluatex.sty</code>	2010/03/01 v1.3 Provides the ifluatex switch (HO)
<code>luatex.sty</code>	2010/03/09 v0.4 LuaTeX basic definition package (HO)
<code>infwarerr.sty</code>	2010/04/08 v1.3 Providing info/warning/error messages (HO)
<code>etex.sty</code>	1998/03/26 v2.0 eTeX basic definition package (PEB)
<code>luatex-loader.sty</code>	2010/03/09 v0.4 Lua module loader (HO)
<code>luatexbase-compat.sty</code>	2011/05/24 v0.4 Compatibility tools for LuaTeX
<code>luatexbase-modutils.sty</code>	2013/05/11 v0.6 Module utilities for LuaTeX
<code>luatexbase-loader.sty</code>	2013/05/11 v0.6 Lua module loader for LuaTeX
<code>luatexbase-regs.sty</code>	2011/05/24 v0.4 Registers allocation for LuaTeX
<code>luatexbase-attr.sty</code>	2013/05/11 v0.6 Attributes allocation for LuaTeX
<code>luatexbase-cctb.sty</code>	2013/05/11 v0.6 Catcodetable allocation for LuaTeX
<code>luatexbase-mcb.sty</code>	2013/05/11 v0.6 Callback management for LuaTeX
<code>ltxcmds.sty</code>	2011/11/09 v1.22 LaTeX kernel commands for general use (HO)
<code>pdftexcmds.sty</code>	2011/11/29 v0.20 Utility functions of pdfTeX for LuaTeX (HO)
<code>xkeyval.sty</code>	2012/10/14 v2.6b package option processing (HA)
<code>ltj-base.sty</code>	2013/05/14
<code>ltj-latex.sty</code>	2013/05/14 LaTeX support of LuaTeX-ja
<code>lltjfont.sty</code>	2014/01/23 Patch to NFSS2 for LuaTeX-ja
<code>lltjdefs.sty</code>	2013/06/12 Default font settings of LuaTeX-ja
<code>lltjcore.sty</code>	2013/05/14 Patch to LaTeX2e Kernel for LuaTeX-ja
<code>luatexja-compat.sty</code>	2013/12/22 Compatibility with pTeX
<code>expl3.sty</code>	2014/01/07 v4646 L3 Experimental code bundle wrapper
<code>l3names.sty</code>	2014/01/04 v4640 L3 Namespace for primitives
<code>l3bootstrap.sty</code>	2014/01/04 v4640 L3 Experimental bootstrap code
<code>l3basics.sty</code>	2014/01/04 v4642 L3 Basic definitions
<code>l3expan.sty</code>	2014/01/04 v4642 L3 Argument expansion
<code>l3tl.sty</code>	2013/12/27 v4625 L3 Token lists
<code>l3seq.sty</code>	2013/12/14 v4623 L3 Sequences and stacks
<code>l3int.sty</code>	2013/08/02 v4583 L3 Integers
<code>l3quark.sty</code>	2013/12/14 v4623 L3 Quarks
<code>l3prg.sty</code>	2014/01/04 v4642 L3 Control structures
<code>l3clist.sty</code>	2013/07/28 v4581 L3 Comma separated lists
<code>l3token.sty</code>	2013/08/25 v4587 L3 Experimental token manipulation
<code>l3prop.sty</code>	2013/12/14 v4623 L3 Property lists
<code>l3msg.sty</code>	2013/07/28 v4581 L3 Messages
<code>l3file.sty</code>	2013/10/13 v4596 L3 File and I/O operations
<code>l3skip.sty</code>	2013/07/28 v4581 L3 Dimensions and skips
<code>l3keys.sty</code>	2013/12/08 v4614 L3 Experimental key-value interfaces
<code>l3fp.sty</code>	2014/01/04 v4642 L3 Floating points
<code>l3box.sty</code>	2013/07/28 v4581 L3 Experimental boxes
<code>l3coffins.sty</code>	2013/12/14 v4624 L3 Coffin code layer
<code>l3color.sty</code>	2012/08/29 v4156 L3 Experimental color support
<code>l3luatex.sty</code>	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
l1tjp-fontspec.sty	2013/05/14 Patch to fontspec for LuaTeX-ja
l1tjp-xunicode.sty	2013/05/14 Patch to xunicode for LuaTeX-ja
l1tjp-unicode-math.sty	2013/05/14 Patch to unicode-math for LuaTeX-ja
l1tjp-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
getttitlestring.sty	2010/12/03 v1.4 Cleanup title references (HO)