

The LuaT_EX-ja package

The LuaT_EX-ja project team

November 2, 2013

Contents

I	User's manual	3
1	Introduction	3
1.1	Backgrounds	3
1.2	Major Changes from p \TeX	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 \LaTeX	6
3	Changing Fonts	7
3.1	plain \TeX and $\LaTeX 2_{\epsilon}$	7
3.2	fontspec	8
3.3	Preset	8
3.4	$\backslash CID$, $\backslash UTF$ and macros in <code>otf</code> package	11
4	Changing Parameters	11
4.1	Editing the range of <code>JAchars</code>	11
4.2	<code>kanjiskip</code> and <code>xkanjiskip</code>	13
4.3	Insertion Setting of <code>xkanjiskip</code>	13
4.4	Shifting Baseline	13
II	Reference	14
5	Font Metric and Japanese Font	14
5.1	<code>\jfont</code>	14
5.2	Prefix <code>psft</code>	15
5.3	Structure of JFM file	16
5.4	Math Font Family	18
5.5	Callbacks	18
6	Parameters	19
6.1	<code>\tjsetparameter</code>	19
6.2	List of Parameters	20
7	Other Control Sequences	21
7.1	Control Sequences for Compatibility	21
7.2	<code>\inhibitglue</code>	21
8	Control Sequences for $\LaTeX 2_{\epsilon}$	22
8.1	Patch for NFSS2	22

9 Extensions	23
9.1 luatexja-fontspec.sty	23
9.2 luatexja-otf.sty	23
9.3 luatexja-adjust.sty	23
III Implementations	23
10 Storing Parameters	24
10.1 Used Dimensions, Attributes and whatsit nodes	24
10.2 Stack System of LuaTeX-ja	25
11 Linebreak after Japanese Character	26
11.1 Reference: Behavior in pTeX	26
11.2 Behavior in LuaTeX-ja	26
12 Patch for the <u>listings</u> package	27
References	28
A Package versions used in this document	29

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 character 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:

```
\hskip2\zw ちよ{}つと\hskip2\zw ちよ\/つと
```

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

```
\hskip2\zw 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 divided into two types:
 - **JAchar**: standing for characters which used in Japanese typesetting, such as Hiragana, Katakana, Kanji and other Japanese punctuation marks.
 - **ALchar**: standing for all other characters like alphabets.

We say ‘alphabetic fonts’ for fonts used in **ALchar**, and ‘Japanese fonts’ for fonts used in **JAchar**.

- A word in a sans-serif font (like `prebreakpenalty`) means an internal parameter for Japanese typesetting, and it is used as a key in `\ltjsetParameter` command.
- A word in typewriter font with underline (like `fontspec`) means a package or a class of L^AT_EX.
- In this document, natural numbers start from 0.

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
- [luatexbase](#) v0.6 (2013/05/04)
- [xunicode](#) v0.981 (2011/09/09)
- [adobemapping](#) (Adobe cmap and pdfmapping files)

From this version of LuaTeX-ja, T_EX Live 2012 (or older version) is no longer supported, since LuaTeX binary and [luaotfload](#) is updated in T_EX Live 2013. And conversely, older versions of LuaTeX-ja (20130318.1 or earlier) don't work in T_EX 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`)
- T_EX Live (in `texmf-dist/tex/luatex/luatexja`)
- W32T_EX (in `luatexja.tar.xz`)

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

```
$ tlmgr install luatexja
```

If you want to install manually, do the following instructions:

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. Using LuaJITTeX slightly improve the situation.
- *Note for MiKTeX users:* LuaTeX-ja requires that two CMap files, UniJIS2004-UTF32-H and Adobe-Japan1-UCS2, must be found by Kpathsearch. You can check this by kpsewhich command (the output may be changed):

```
$ kpsewhich -format=cmap UniJIS2004-UTF32-H
/opt/texlive/2013/texmf-dist/fonts/cmap/adobemapping/aj16/CMap/UniJIS2004-UTF32-H
$ kpsewhich -format=cmap Adobe-Japan1-UCS2
/opt/texlive/2013/texmf-dist/fonts/cmap/adobemapping/ToUnicode/Adobe-Japan1-UCS2
```

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	\tenmin	\sevenmin	\fivemin
<i>gothic</i>	GothicBBB-Medium	\tengt	\seventgt	\fivegt

- It is widely accepted that fonts 'Ryumin-Light' and 'GothicBBB-Medium' aren't embedded into PDF files, and a PDF reader substitute them by some external Japanese fonts (*e.g.*, Ryumin-Light is substituted with Kozuka Mincho in Adobe Reader). We adopt this custom to the default setting.
- A character in an alphabetic font is generally smaller than a Japanese font in the same size. So actual size specification of these Japanese fonts is in fact smaller than that of alphabetic fonts, namely scaled by 0.962216.

- The amount of glue that are inserted between a **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.
- 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)	\textmc{...}	{\mcfamily ...}	\mcdefault
<i>gothic</i> (ゴシック体)	\textgt{...}	{\gtfamily ...}	\gtdefault

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

classification	family name	\mdseries	\bfseries	scale
<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 p \LaTeX . 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 p \LaTeX) and `\jsclasses` (classes by Haruhiko Okumura), namely, `\ltjclasses` and `\ltjsclasses`.

3 Changing Fonts

3.1 plain \TeX and $\LaTeX 2_{\epsilon}$

■ **plain \TeX** To change Japanese fonts in plain \TeX , you must use the control sequence `\jfont`. So please see Subsection 5.1.

■ **$\LaTeX 2_{\epsilon}$ (NFSS2)** For $\LaTeX 2_{\epsilon}$, Lua \TeX -ja adopted most of the font selection system of p $\LaTeX 2_{\epsilon}$ (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>\kanjiserries</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 family of Japanese fonts, alphabetic fonts, *or both*. For detail, see Subsection 8.1.

- For defining a Japanese font family, use `\DeclareKanjiFamily` instead of `\DeclareFontFamily`. However, in the present implementation, using `\DeclareFontFamily` doesn't cause any problem.

■ **Remark: Japanese Characters in Math Mode** Since p \TeX supports Japanese characters in math mode, there are sources like the following:

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

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

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 5.4.

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 7 commands are defined as counterparts of original commands in the `fontspec` package:

Japanese fonts	<code>\jfontspec</code>	<code>\setmainjfont</code>	<code>\setsansjfont</code>	<code>\newfontfamily</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{IPAexMincho}
3 JIS-X-0213:2004→辻                               JIS X 0213:2004 →辻
4                                                     JIS X 0208:1990 →辻
5 \jfontspec[CJKShape=JIS1990]{IPAexMincho}
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 the kerning feature is set off by default in these 7 commands, since this feature and **JAgglue** will clash (see 5.1).

3.3 Preset

To use standard Japanese font settings easily, one can load `luatexja-preset` package with several options. This package provides functions in a part of `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*¹. 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`).

¹ Provided by `\mgfamily` and `\textmg{...}`, because *rounded gothic* is called *maru gothic* (丸ゴシック) in Japanese.

`expert` Use horizontal kana alternates, and define a control sequence `\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 settings** 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* KozGoPro-Medium	KozGoProVI-Regular* KozGoProVI-Medium	KozGoPr6N-Regular* 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).

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 Pro W6	Hiragino Kaku Gothic ProN W3* Hiragino Kaku Gothic ProN W6
	bold	Hiragino Kaku Gothic Pro W6	Hiragino Kaku Gothic ProN W6
	heavy	Hiragino Kaku Gothic Std W8	Hiragino Kaku Gothic StdN W8
<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-Jun101Pr6N-Light.otf

yu-win Yu fonts bundled with Windows 8.1.

yu-osx Yu fonts bundled with OSX Mavericks. They cover Adobe-Japan1-6 character collection.

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	YuGothic-Bold	YuGothic Bold
	heavy	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 `otf` package

Under $\text{p}\text{L}\text{T}\text{E}\text{X}$, `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 `otf` package. If you want to use these functions, load `luatexja-otf` package.

```
1 \fontspec{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,      森鷗外と内田百間とが高島屋に行く。
5 \CID{1481}城市, 葛西駅,              葛飾区の吉野家, 葛城市, 葛西駅, 高崎と高崎
6 高崎と\CID{8705}\UTF{FA11}          はんかくかか
7
8 \aj半角{はんかくカタカナ}
```

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 **J**Achars

To edit the range of **J**Achars, you have to assign a non-zero natural number which is less than 217 to the character range first. This can be done by using `\ltjdefcharrange`. For example, the next line assigns whole characters in Supplementary Ideographic Plane and the character ‘漢’ to the range number 100.

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

This assignment of numbers to ranges are always global, so you should not do this in the middle of a document.

If some character has been belonged to some non-zero numbered range, this will be overwritten by the new setting. 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 specify the above line, then SIP will belong to the range 100 and be removed from the range 4.

After assigning numbers to ranges, the `jacharrange` parameter can be used to customize which character range will be treated as ranges of **J**Achars, as the following line (this is just the default setting of $\text{L}\text{u}\text{a}\text{T}\text{E}\text{X}$ -ja):

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

The argument to `jacharrange` parameter is a list of integer. Negative integer $-n$ in the list means that ‘the characters that belong to range n are treated as **AL**char’, and positive integer $+n$ means that ‘the characters that belong to range n are treated as **J**Achar’.

■**Default Setting** $\text{L}\text{u}\text{a}\text{T}\text{E}\text{X}$ -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 $\text{u}\text{p}\text{T}\text{E}\text{X}$ by Takayuki Yato.

Now we describe these eight ranges. The alphabet ‘J’ or ‘A’ after the number shows whether characters in the range is treated as **J**Achars or not by default. These settings are similar to the `preferCJK` settings defined in `PXbase` bundle.

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

Table 1. Unicode blocks in predefined character range 3.

U+2000–U+206F	General Punctuation	U+2070–U+209F	Superscripts and Subscripts
U+20A0–U+20CF	Currency Symbols	U+20D0–U+20FF	Comb. Diacritical Marks for Symbols
U+2100–U+214F	Letterlike Symbols	U+2150–U+218F	Number Forms
U+2190–U+21FF	Arrows	U+2200–U+22FF	Mathematical Operators
U+2300–U+23FF	Miscellaneous Technical	U+2400–U+243F	Control Pictures
U+2500–U+257F	Box Drawing	U+2580–U+259F	Block Elements
U+25A0–U+25FF	Geometric Shapes	U+2600–U+26FF	Miscellaneous Symbols
U+2700–U+27BF	Dingbats	U+2900–U+297F	Supplemental Arrows-B
U+2980–U+29FF	Misc. Mathematical Symbols-B	U+2B00–U+2BFF	Miscellaneous Symbols and Arrows

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+FA90–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+00A7, Section Sign)
- ¨ (U+00A8, Diaeresis)
- ° (U+00B0, Degree sign)
- ± (U+00B1, Plus-minus sign)
- ´ (U+00B4, Spacing acute)
- ¶ (U+00B6, Paragraph sign)
- × (U+00D7, Multiplication sign)
- ÷ (U+00F7, Division Sign)

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

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

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

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

Range 3^J Punctuations and Miscellaneous symbols. The block list is indicated in Table 1.

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

```
\ltjdefcharrange{4}{%
"500-"10FF, "1200-"1DFF, "2440-"245F, "27C0-"28FF, "2A00-"2AFF,
"2C00-"2E7F, "4DC0-"4DFF, "A4D0-"A82F, "A840-"ABFF, "FB00-"FE0F,
"FE20-"FE2F, "FE70-"FEFF, "10000-"1FFFF, "E000-"F8FF} % non-Japanese
```

Range 5^A Surrogates and Supplementary Private Use Areas.

Range 6^J Characters used in Japanese. The block list is indicated in Table 2.

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

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

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 **A**Lchar (`xkanjiskip`).

The value (a skip) of `kanjiskip` or `xkanjiskip` can be changed as the following.

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

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 **A**Lchars. 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 **A**Lchars respectively.

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

The second argument `preonly` means ‘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 6.2).

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

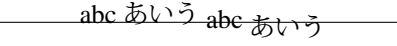
4.4 Shifting Baseline

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 alphabetic fonts 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

```



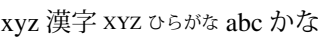
Here the horizontal line in above is the baseline of a line.

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}
5 XYZひらがな
6 }abcかな

```



Part II

Reference

5 Font Metric and Japanese Font

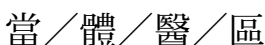
5.1 \jfont

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

```



Note that the defined control sequence (`\tradgt` in the example above) using `\jfont` is not a `font_def` token, hence the input like `\fontname\tradgt` causes an error. We denote control sequences which are defined in `\jfont` by `<jfont_cs>`.

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

`jfm=<name>` Specify the name of JFM. If specified JFM has not been loaded, LuaTeX-ja search and load a file named `jfm-<name>.lua`.

The following JFMs are shipped with LuaTeX-ja:

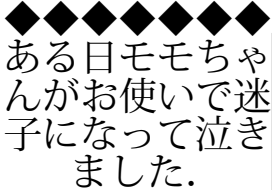
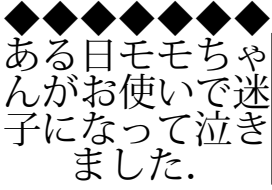
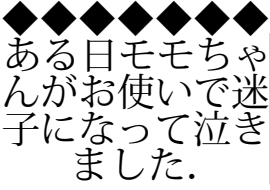



`jfm-ujis.lua` A standard JFM in LuaTeX-ja. This JFM is based on `upnmlminr-h.tfm`, a metric for UTF/OTF package that is used in pTeX. When you use the `luatexja-otf` package, you should use this JFM.

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

`jfm-min.lua` A counterpart for `min10.tfm`, which is one of the default Japanese font metric shipped with pTeX. There are notable difference between this JFM and other 2 JFMs, as shown in Table 4.

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

Table 4. Differences between JFM's shipped with LuaTeX-ja

	jfm-ujis.lua	jfm-jis.lua	jfm-min.lua
Example 1[4]			
Example 2	ちよつと！何	ちよつと！何	ちよつと！何
Bounding Box			

```

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
6 \F ) {\G 〔 } ( % halfwidth space
7   ) {\H 『 } ( % fullwidth space
8
9 \ltjsetparameter{differentjfm=paverage}

```

■**Note: kern feature** Some fonts have information for inter-glyph spacing. However, this information is not well-compatible with LuaTeX-ja. More concretely, this kerning space from this information are inserted *before* the insertion process of **JAGlue**, and this causes incorrect spacing between two characters when both a glue/kern from the data in the font and it from JFM are present.

- You should specify `-kern` in `jfont` when you want to use other font features, such as `script=...`
- If you want to use Japanese fonts in proportional width, and use information from this font, use `jfm-prop.lua` for its JFM, and TODO: `kanjiskip`?

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

`extend=<extend>` expand the font horizontally by *<extend>*.

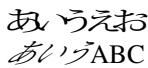
`slant=<slant>` slant the font.

Note that LuaTeX-ja doesn't adjust 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
2 \E あいうえお
3
4 \jfont\S=file:KozMinPr6N-Regular.otf:slant=1;jfm=ujis
5 \S あいう\ABC

```



5.2 Prefix `psft`

Besides `'file:'` and `'name:'` prefixes, one can use `'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 the `'standard'` Japanese fonts, namely, `'Ryumin-Light'` and `'GothicBBB-Medium'`.

OpenType font features, such as '+jp90', have no meaning in 'name-only' fonts using this 'psft:' prefix. This is 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 \par
8
9 1.78
10
11 ? h
12 I couldn't find any non-embedded font information for the CID
13 `Adobe-Japan2'. For now, I'll use `Adobe-Japan1-6'.
14 Please contact the LuaTeX-ja project team.
15 ?

```

5.3 Structure of 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)

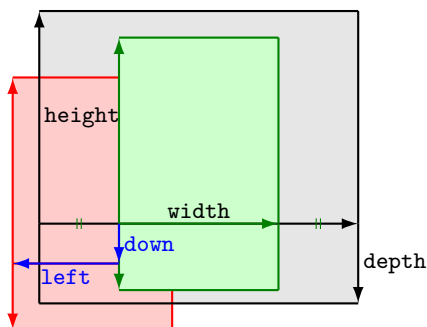
The amount of the 'full-height' (height + depth).

`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 0pt). 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](#).



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 1. The position of the 'real' glyph.

■ **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 width of characters in character class i , height, 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: if 'prop' is specified in width field, 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 1.

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*]}, ...}

`end_stretch`={*kern*}

`end_shrink`={*kern*}

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

'boxbdd' The beginning/ending of a horizontal box, and the beginning of a noindented paragraph.

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

Table 5. Control sequences 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>

'jcharbdd' A boundary between **J**Achar and anything else (such as **AL**char, kern, glue, ...).

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

■Porting JFM from pTeX ...

5.4 Math Font Family

TeX handles fonts in math formulas by 16 font families², and each family has three fonts: `\textfont`, `\scriptfont` and `\scriptscriptfont`.

LuaTeX-ja's handling of Japanese fonts in math formulas is similar; Table 5 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.

5.5 Callbacks

Like LuaTeX itself, LuaTeX-ja also has 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
```

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

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

You may assume that `jfont_info` has the following fields:

`size_cache` A table which contains the information of a JFM, and *this table must not be changed*. The contents of this table are similar to that which is written in the JFM file, but the following differ:

- There is a `chars` table, ...
- The value in `zw`, `zh`, `kanjiskip`, `xkanjiskip` fields are now scaled by real font size, and in scaled-pont unit.
- ...
- There is no `dir` field in this table.

²Omega, Aleph, LuaTeX and ε -u)TeX can handles 256 families, but an external package is needed to support this in plain TeX and L^ATeX.

var The value specified in `jfmvar=...` at a call of `\jfont`.

The returned table `new_jfont_info` also should include these two fields. 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
```

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

luatexja.set_width callback This callback is called when LuaTeX-ja is trying to encapsule a **J**Achar *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

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

6 Parameters

6.1 \ltjsetparameter

As noted before, `\ltjsetparameter` and `\ltjgetparameter` are control sequences for accessing most parameters of LuaTeX-ja. One of the main reason that LuaTeX-ja didn't adopted the syntax similar to that of pTeX (*e.g.*, `\prebreakpenalty`)=10000`) is the position of `hpack_filter` callback in the source of LuaTeX, see Section 10.

`\ltjsetparameter` and `\ltjglobalsetparameter` are control sequences for assigning parameters. These take one argument which is a `<key>=<value>` list. 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.

`\ltjgetparameter` is for acquiring parameters. It always takes a parameter name as first argument, and also takes the additional argument—a character code, for example—in some cases.

```

1 \ltjgetparameter{differentjfm},
2 \ltjgetparameter{autospacing},           paverage, 1, 10000.
3 \ltjgetparameter{prebreakpenalty}{^} }.

```

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).

6.2 List of Parameters

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

- No mark: values at the end of the paragraph or the hbox are adopted in the whole paragraph/hbox.
- ‘*’: local parameters, which can change everywhere inside a paragraph/hbox.
- ‘†’: assignments are always global.

`jcharwidowpenalty = <penalty> [\jcharwidowpenalty]` Penalty value for suppressing orphans. This penalty is inserted just after the last **J**Achar 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]

`postbreakpenalty = {<chr_code>, <penalty>}` [\postbreakpenalty]

`jatextfont = {<jfam>, <jfont_cs>}` [\textfont in T_EX]

`jascriptfont = {<jfam>, <jfont_cs>}` [\scriptfont in T_EX]

`jascriptscriptfont = {<jfam>, <jfont_cs>}` [\scriptscriptfont in T_EX]

`yjabaselineshift = <dimen>*`

`yalbaselineshift = <dimen>*` [\ybaselineshift]

`jaxspmode = {<chr_code>, <mode>}` Setting whether inserting `xkanjjskip` is allowed before/after a **J**Achar whose character code is `<chr_code>`. The followings are allowed for `<mode>`:

- 0, inhibit** Insertion of `xkanjjskip` is inhibited before the character, nor after the character.
- 1, preonly** Insertion of `xkanjjskip` is allowed before the character, but not after.
- 2, postonly** Insertion of `xkanjjskip` is allowed after the character, but not before.
- 3, allow** Insertion of `xkanjjskip` is allowed both before the character and after the character. This is the default value.

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

`alxspmode = {<chr_code>, <mode>}` [\xspcode]

Setting whether inserting `xkanjjskip` is allowed before/after a **A**Lchar whose character code is `<chr_code>`. The followings are allowed for `<mode>`:

- 0, inhibit** Insertion of `xkanjjskip` is inhibited before the character, nor after the character.
- 1, preonly** Insertion of `xkanjjskip` is allowed before the character, but not after.
- 2, postonly** Insertion of `xkanjjskip` is allowed after the character, but not before.
- 3, allow** Insertion of `xkanjjskip` 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.

```

autospaceing=<bool>* [\autospaceing]
autoxspaceing=<bool>* [\autoxspaceing]
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
jacharrange=<ranges>*
kansujichar={<digit>, <chr_code>} [\kansujichar]

```

7 Other Control Sequences

7.1 Control Sequences for Compatibility

The following control sequences are implemented for compatibility with p_TE_X. Note that these don't support JIS X 0213, but only JIS X 0208.

```

\kuten
\jis
\euc
\sjis
\ucs
\kansuji

```

7.2 \inhibitglue

\inhibitglue suppresses the insertion of **J**Aglue. The following is an example, using a special JFM that there will be a glue between the beginning of a box and ‘あ’, and also between ‘あ’ and ‘ウ’.

<pre> 1 \jfont\g=file:KozMinPr6N-Regular.otf:jfm=test \g 2 \fbox{\hbox{あウあ\inhibitglue ウ}} 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 </pre>	<table border="1" style="border-collapse: collapse; text-align: center;"> <tr> <td style="padding: 2px;">あ</td> <td style="padding: 2px;">ウあウ</td> </tr> </table> <pre> あ 1 あ 2 あ 3 <hr style="width: 100%;"/> あ office </pre>	あ	ウあウ
あ	ウあウ		

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 Control Sequences for $\LaTeX 2_{\epsilon}$

8.1 Patch for NFSS2

As described in Subsection 2.4, $\text{Lua}\TeX\text{-ja}$ simply adopted `plfonts.dtx` in $\text{p}\LaTeX 2_{\epsilon}$ for the Japanese patch for NFSS2. For an convenience, we will describe control sequences which are not described in Subsection 3.1.

```
\DeclareYokoKanjiEncoding{<encoding>}{<text-settings>}{<math-settings>}
```

In NFSS2 under $\text{Lua}\TeX\text{-ja}$, 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
```

...

```
\fontfamily{<family>}
```

As in $\LaTeX 2_{\epsilon}$, 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 `\DeclareKanjifamily`.
 - A font definition named `<ja-enc><family>.fd` (the file name is all lowercase) exists.
- Let current encoding scheme for alphabetic fonts be `<al-enc>`. For alphabetic font family, the criterion as above is used.
- There is a case which none of the above applies, that is, the font family named `<family>` doesn’t seem to be defined neither under the encoding `<ja-enc>`, nor under `<al-enc>`. In this case, the default family for font substitution is used for alphabetic and Japanese fonts. Note that current encoding will not be set to `<family>`, unlike the original implementation in \LaTeX .

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
```

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 2. Line adjustment

9 Extensions

9.1 `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 ‘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 font features correspond to `cid`, `jfm` and `jfmvar` keys for `\jfont` respectively. `CID` is effective only when with `NoEmbed` described below. See Subsections 5.1 and 5.2 for details.

`NoEmbed` By specifying this font feature, one can use ‘name-only’ Japanese font which will not be embedded in the output PDF file. See Subsection 5.2.

9.2 `luatexja-otf.sty`

This optional package supports typesetting characters in Adobe-Japan1. `luatexja-otf.sty` 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 **J**Achars.
- 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** `luatexja-otf.sty` 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`.

9.3 `luatexja-adjust.sty`

...

Part III

Implementations

10 Storing Parameters

10.1 Used Dimensions, Attributes and whatsit nodes

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

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

`\jH` (dimension) There is also a unit called ‘齒’ which equals to 0.25 mm and used in Japanese phototypesetting. This `\jH` is a synonym of `\jQ`.

`\ltj@zw` (dimension) A temporal register for the ‘full-width’ of current Japanese font.

`\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.

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

`\ltj@curjfnt` (attribute) The font index of current Japanese font.

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

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

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

`\ltj@autospc` (attribute) Whether the auto insertion of [kanjiskip](#) is allowed at the node.

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

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

italic (1) Glues from an italic correction (`\/`). This distinction of origins of glues (from explicit `\kern`, or from `\/`) is needed in the insertion process of [xkanjiskip](#).

packed (2)

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

from_jfm (6) Glues/kerns from JFM.

kanji_skip (9) Glues for [kanjiskip](#).

xkanji_skip (10) Glues for [xkanjiskip](#).

processed (11) Nodes which is already processed by ...

ic_processed (12) Glues from an italic correction, but also already processed.

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, LuaTeX-ja uses several ‘user-defined’ whatsit nodes for internal processing. All those nodes store a natural number (hence the node’s type is 100).

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

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

`char_by_cid` Nodes for Japanese Characters which the callback process of `luaotfload` won't be applied, and the character code is stored in the `value` field. Each node having this `user_id` is converted to a 'glyph_node' after the callback process of `luaotfload`. This `user_id` is only used by the `luatexja-otf` package.

`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 **JAg**lues.

10.2 Stack System of LuaTeX-ja

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

```
1 \ltjsetparameter{kanjiskip=0pt}ふかふか.%
2 \setbox0=\hbox{\ltjsetparameter{kanjiskip=5pt}{ほ   ふかふか.ほ げ ほ げ.ひよひよ
   げ{ほげ}}
3 \box0.ひよひよ\par
```

As described in Subsection 6.2, the only effective value of `kanjiskip` in an `hbox` is the latest value, so the value of `kanjiskip` which applied in the entire `hbox` should be 5 pt. However, by the implementation method of LuaTeX, this '5 pt' cannot be known from any callbacks. In the `tex/packaging.w` (which is a file in the source of LuaTeX), there are the following codes:

```
void package(int c)
{
    scaled h;          /* height of box */
    halfword p;       /* first node in a box */
    scaled d;         /* max depth */
    int grp;
    grp = cur_group;
    d = box_max_depth;
    unsave();
    save_ptr -= 4;
    if (cur_list.mode_field == -hmode) {
        cur_box = filtered_hpack(cur_list.head_field,
                                cur_list.tail_field, saved_value(1),
                                saved_level(1), grp, saved_level(2));
        subtype(cur_box) = HLIST_SUBTYPE_HBOX;
    }
}
```

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

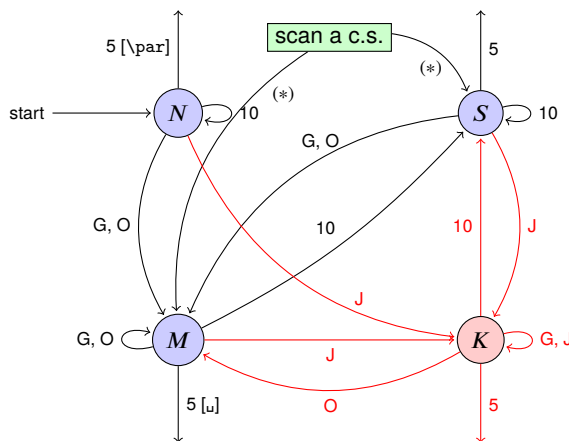
■ **The method** The code of stack system is based on that in a post of Dev-luatex mailing list³.

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 'Background' above, LuaTeX-ja uses another thing: When a new stack level is about to be created, a `whatsit` node whose type, subtype and value are 44 (*user_defined*), 30112, and 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 `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.

³ [Dev-luatex] `tex.currentgrouplevel`, a post at 2008/8/19 by Jonathan Sauer.



- G** Beginning of group (usually {) and ending of group (usually }).
- J** Japanese characters.
- 5** *end-of-line* (usually $\sim\sim$ J).
- 10** space (usually $_$).
- O** other characters, whose category code is in {3, 4, 6, 7, 8, 11, 12, 13}.
- [\u], [\par]** emits a space, or $\backslash\par$.

- We omitted about category codes 9 (*ignored*), 14 (*comment*) and 15 (*invalid*) from the above diagram. We also ignored the input like $\sim\sim$ A' or $\sim\sim$ 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 3. State transitions of $\text{p}\TeX$'s input processor.

- 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 is '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 $\backslash\text{lj}@stack$ and $\backslash\text{lj}@group@level$ have to be local always, regardless the value of $\backslash\text{globaldefs}$. This problem is resolved by using $\backslash\text{directlua}\{\text{tex.globaldefs}=0\}$ (this assignment is local).

11 Linebreak after Japanese Character

11.1 Reference: Behavior in $\text{p}\TeX$

In $\text{p}\TeX$, 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 $\text{Lua}\TeX\text{-ja}$ due to the specification of callbacks in $\text{Lua}\TeX$. To clarify the difference between $\text{p}\TeX$ and $\text{Lua}\TeX$, We briefly describe the handling of a line break in $\text{p}\TeX$, in this subsection.

$\text{p}\TeX$'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 3. Note that $\text{p}\TeX$ doesn't leave state *K* after 'beginning/ending of a group' characters.

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

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 LuaTeX-ja are as follows:

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

1. The category code of `\endlinechar`⁴ 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’:

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

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

```

1 \ltjsetparameter{autoxspacing=false}
2 \ltjsetparameter{jacharrange={-6}}xあ
3 y\ltjsetparameter{jacharrange={+6}}zあ
4 u

```

xyzあ u

- There is no space between ‘x’ and ‘y’, since the line 2 ends with a **JAchar** ‘あ’ (this ‘あ’ considered as an **JAchar** at the ending of line 1).
- There is no space between ‘あ’ (in the line 3) and ‘u’, since the line 3 ends with an **ALchar** (the letter ‘あ’ considered as an **ALchar** at the ending of line 2).

12 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 [3] resolves the problem forcibly.

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

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

■**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 parentheses, dashes, etc. To process Japanese characters, The pacth lltjp-listings.sty introduces a new flag `\lst@ifkanji`, which indicates whether the previous character is Japanese character or not. For illustration, we introduce the following classes of character:

⁴Usually, it is `\return` (whose character code is 13).

	Letter	Other	Kanji	Open	Close
<code>\lst@ifletter</code>	T	F	T	F	T
<code>\lst@ifkanji</code>	F	F	T	T	F
Meaning	identifier char	other alphabet	most of Japanese char	open paren	close paren

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 parenthesis, 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 _____	_____ outputs _____	_____ outputs _____	collects
	Other	outputs	collects	_____ outputs _____	_____ outputs _____	collects
	Kanji	_____ outputs _____	_____ outputs _____	_____ outputs _____	_____ outputs _____	collects
	Open	_____ outputs _____	_____ outputs _____	_____ outputs _____	collects	_____ outputs _____
	Close	_____ outputs _____	_____ outputs _____	_____ outputs _____	_____ 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).

■ **Classification of characters** Characters are classified according to `jacharrange` parameter (see Section 4.1):

- **ALchars** above U+0080 are Letter.
- **JChars** are classified in the order as follows:
 1. Characters whose `prebreakpenalty` is greater than or equal to 0 are Open.
 2. Characters whose `postbreakpenalty` is greater than or equal to 0 are Close.
 3. Characters that don’t satisfy the above two conditions are 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**.

The classification process is executed every time a character appears in listing environments.

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] Thor Watanabe. Listings - MyTeXpert. <http://mytexpert.sourceforge.jp/index.php?Listings>
- [4] 乙部巖己, min10 フォントについて. <http://argent.shinshu-u.ac.jp/~otobe/tex/files/min10.pdf>
- [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] 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>	2013/05/14 Core of LuaTeX-ja
<code>luaotfload.sty</code>	2013/07/23 v2.3b 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>	2013/05/14 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>	2013/05/14 Compatibility with pTeX
<code>expl3.sty</code>	2013/10/13 v4597 L3 Experimental code bundle wrapper
<code>l3names.sty</code>	2012/12/07 v4346 L3 Namespace for primitives
<code>l3bootstrap.sty</code>	2013/07/28 v4581 L3 Experimental bootstrap code
<code>l3basics.sty</code>	2013/07/28 v4581 L3 Basic definitions
<code>l3expan.sty</code>	2013/08/17 v4584 L3 Argument expansion
<code>l3tl.sty</code>	2013/09/16 v4592 L3 Token lists
<code>l3seq.sty</code>	2013/07/28 v4581 L3 Sequences and stacks
<code>l3int.sty</code>	2013/08/02 v4583 L3 Integers
<code>l3quark.sty</code>	2013/07/21 v4564 L3 Quarks
<code>l3prg.sty</code>	2013/08/25 v4587 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>	2013/07/28 v4581 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>	2013/07/28 v4581 L3 Experimental key-value interfaces
<code>l3fp.sty</code>	2013/07/09 v4521 L3 Floating points
<code>l3box.sty</code>	2013/07/28 v4581 L3 Experimental boxes
<code>l3coffins.sty</code>	2012/09/09 v4212 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	2013/07/24 v4576 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	2010/10/13 v2.10 (rcs-revision 1.76)
pgf.sty	2008/01/15 v2.10 (rcs-revision 1.12)
pgfrcs.sty	2010/10/25 v2.10 (rcs-revision 1.24)
everyshi.sty	2001/05/15 v3.00 EveryShipout Package (MS)
pgfcore.sty	2010/04/11 v2.10 (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	2010/06/30 v2.10 (rcs-revision 1.37)
xcolor.sty	2007/01/21 v2.11 LaTeX color extensions (UK)
pgfcomp-version-0-65.sty	2007/07/03 v2.10 (rcs-revision 1.7)
pgfcomp-version-1-18.sty	2007/07/23 v2.10 (rcs-revision 1.1)
pgffor.sty	2010/03/23 v2.10 (rcs-revision 1.18)
pgfkeys.sty	
pict2e.sty	2011/04/05 v0.2y 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	2013/08/26 1.5b (Carsten Heinz)
lstmisc.sty	2013/08/26 1.5b (Carsten Heinz)
showexpl.sty	2013/03/21 v0.3k 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	2006/04/12 ver 3.3 Verb mode for urls, etc.
rerunfilecheck.sty	2011/04/15 v1.7 Rerun checks for auxiliary files (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	2013/08/17 fontspec support of LuaTeX-ja
fontspec.sty	2013/05/20 v2.3c Font selection for XeLaTeX and LuaLaTeX
xparse.sty	2013/10/13 v4597 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	2013/10/13 v4597 LaTeX2e option processing using LaTeX3 keys
catchfile.sty	2011/03/01 v1.6 Catch the contents of a file (HO)
fix-cm.sty	2006/09/13 v1.1m fixes to LaTeX
filehook.sty	2011/10/12 v0.5d Hooks for input files
unicode-math-luatex.sty	
lualatex-math.sty	2013/08/03 v1.3 Patches for mathematics typesetting with LuaLaTeX
etoolbox.sty	2011/01/03 v2.1 e-TeX tools for LaTeX
metalogo.sty	2010/05/29 v0.12 Extended TeX logo macros
lltjp-fontspec.sty	2013/05/14 Patch to fontspec for LuaTeX-ja
lltjp-xunicode.sty	2013/05/14 Patch to xunicode for LuaTeX-ja
lltjp-unicode-math.sty	2013/05/14 Patch to unicode-math for LuaTeX-ja
lltjp-listings.sty	2013/05/14 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)