

# The LuaTeX-ja package

The LuaTeX-ja project team

October 4, 2011

# Contents

<b>I</b>	<b>User's manual</b>	<b>3</b>
<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Backgrounds . . . . .	3
1.2	Major Changes from p $\text{\TeX}$ . . . . .	3
1.3	Notations . . . . .	3
1.4	About the project . . . . .	4
<b>2</b>	<b>Getting Started</b>	<b>5</b>
2.1	Installation . . . . .	5
2.2	Cautions . . . . .	5
2.3	Using in plain $\text{\TeX}$ . . . . .	5
2.4	Using in $\text{\LaTeX}$ . . . . .	6
2.5	Changing Fonts . . . . .	6
<b>3</b>	<b>Changing Parameters</b>	<b>7</b>
3.1	Editing the range of <b>J</b> Achars . . . . .	8
3.2	kanjiskip and xkanjiskip . . . . .	10
3.3	Insertion Setting of xkanjiskip . . . . .	10
3.4	Shifting Baseline . . . . .	10
3.5	Cropmark . . . . .	11
<b>II</b>	<b>Reference</b>	<b>11</b>
<b>4</b>	<b>Font Metric and Japanese Font</b>	<b>11</b>
4.1	$\backslash$ font primitive . . . . .	11
4.2	Structure of JFM file . . . . .	12
4.3	Math Font Family . . . . .	13
<b>5</b>	<b>Parameters</b>	<b>13</b>
5.1	$\backslash$ tjsetparameter primitive . . . . .	13
5.2	List of Parameters . . . . .	14
<b>6</b>	<b>Other Primitives</b>	<b>15</b>
6.1	Compatibility with p $\text{\TeX}$ . . . . .	15
<b>7</b>	<b>Control Sequences for <math>\text{\LaTeX 2}_\epsilon</math></b>	<b>15</b>
7.1	Patch for NFSS2 . . . . .	15
7.2	Cropmark/'tombow' . . . . .	15
<b>8</b>	<b>Extensions</b>	<b>15</b>
8.1	luatexja-fontspec.sty . . . . .	15
8.2	luatexja-otf.sty . . . . .	15
<b>III</b>	<b>Implementations</b>	<b>16</b>
<b>9</b>	<b>Storing Parameters</b>	<b>16</b>
9.1	Used Dimensions, Attributes and whatsit nodes . . . . .	16
9.2	Stack System of Lua $\text{\TeX}$ -ja . . . . .	17

<b>10 Linebreak after Japanese Character</b>	<b>18</b>
10.1 Reference: Behavior in p $\text{T}_{\text{E}}\text{X}$ . . . . .	18
10.2 Behavior in Lua $\text{T}_{\text{E}}\text{X}$ -ja . . . . .	18
<b>11 Insertion of JFM glues, <code>kanjiskip</code> and <code>xkanjiskip</code></b>	<b>19</b>

**This documentation is far from complete. It may have many grammatical (and contextual) errors.**

# Part I

## User's manual

### 1 Introduction

The Lua $\TeX$ -ja package is a macro package for typesetting high-quality Japanese documents when using Lua $\TeX$ .

#### 1.1 Backgrounds

Traditionally, ASCII p $\TeX$ , an extension of  $\TeX$ , and its derivatives are used to typeset Japanese documents in  $\TeX$ . p $\TeX$  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: p $\TeX$  is left behind from other extensions of  $\TeX$ , especially  $\varepsilon$ - $\TeX$  and pdf $\TeX$ , and from changes about Japanese processing in computers (*e.g.*, the UTF-8 encoding).

Recently extensions of p $\TeX$ , namely p $\TeX$  (Unicode-implementation of p $\TeX$ ) and  $\varepsilon$ -p $\TeX$  (merging of p $\TeX$  and  $\varepsilon$ - $\TeX$  extension), have developed to fill those gaps to some extent, but gaps still exist.

However, the appearance of Lua $\TeX$  changed the whole situation. With using Lua ‘callbacks’, users can customize the internal processing of Lua $\TeX$ . 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 p $\TeX$

The Lua $\TeX$ -ja package is under much influence of p $\TeX$  engine. The initial target of development was to implement features of p $\TeX$ . However, *Lua $\TeX$ -ja is not a just porting of p $\TeX$ ; unnatural specifications/behaviors of p $\TeX$  were not adopted.*

The followings are major changes from p $\TeX$ :

- A Japanese font is a tuple of a ‘real’ font, a Japanese font metric (**JFM**, for short), and an optional string called ‘variation’.
- In p $\TeX$ , a linebreak after Japanese character is ignored (and doesn’t yield a space), since linebreaks (in source files) are permitted almost everywhere in Japanese texts. However, Lua $\TeX$ -ja doesn’t have this function completely, because of a specification of Lua $\TeX$ .
- The insertion process of glues/kerns between two Japanese characters and between a Japanese character and other characters (we refer these glues/kerns as **JAg glue**) is rewritten from scratch.
  - As Lua $\TeX$ ’s internal character handling is ‘node-based’ (*e.g.*, `of{f}ice` doesn’t prevent ligatures), the insertion process of **JAg glue** is now ‘node-based’.
  - Furthermore, nodes between two characters which have no effects in linebreak (*e.g.*, `\special` node) are ignored in the insertion process.
  - In the process, two Japanese fonts which differ in their ‘real’ fonts only are identified.
- At the present, vertical typesetting (*tategaki*), is not supported in Lua $\TeX$ -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 Japanese characters such as Hiragana, Katakana, Kanji and other punctuation marks for Japanese.
  - **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`) represents an internal parameter for Japanese typesetting, and it is used as a key in `\ltjsetparameter` command.
- The word ‘primitive’ is used not only for primitives in Lua $\TeX$ , but also for control sequences that defined in the core module of Lua $\TeX$ -ja.
- 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)

This project is hosted by SourceForge.JP.

## Members

## 2 Getting Started

### 2.1 Installation

To install the Lua<sub>TEX</sub>-ja package, you will need:

- Lua<sub>TEX</sub> (version 0.65.0-beta or later) and its supporting packages.  
If you are using <sub>TEX</sub> Live 2011 or current W32<sub>TEX</sub>, you don't have to worry.
- The source archive of Lua<sub>TEX</sub>-ja, of course:)

The installation methods are as follows:

1. Download the source archive.

At the present, Lua<sub>TEX</sub>-ja has no official release, so you have to retrieve the archive from the repository. You can retrieve the Git repository via

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

or download the archive of HEAD in master branch from

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

Note that the forefront of development may not be in master branch.

2. Extract the archive. You will see `src/` and several other sub-directories.
3. Copy all the contents of `src/` into one of your `TEXMF` tree.
4. If `mktexlsr` is needed to update the filename 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.
- May be conflict with other packages.

For example, the default setting of **J<sub>A</sub>char** in the present version does not coexist with `unicode-math` package. Putting the following line in preamble makes that mathematical symbols will be typeset correctly, but several Japanese characters will be treated as an **A<sub>L</sub>char** as side-effect:

```
\ltjsetparameter{jacharrange={-3, -8}}
```

### 2.3 Using in plain <sub>TEX</sub>

To use Lua<sub>TEX</sub>-ja in plain <sub>TEX</sub>, 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>\sevient</code>	<code>\fivegt</code>

- The 'Q' is a unit used in Japanese phototypesetting, and 1 Q = 0.25 mm. This length is stored in a dimension `\jq`.
- It is widely accepted that the font 'Ryumin-Light' and 'GothicBBB-Medium' aren't embedded into PDF files, and PDF reader substitute them by some external Japanese fonts (*e.g.*, Kozuka Mincho is used for Ryumin-Light 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

$$(0.25 \cdot 13.5 \text{ Q})_{-1 \text{ pt}}^{+1 \text{ pt}} = \frac{27}{32} \text{ mm}_{-1 \text{ pt}}^{+1 \text{ pt}}.$$

## 2.4 Using in L<sup>A</sup>T<sub>E</sub>X

**L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>** Using in L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> 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<sup>A</sup>T<sub>E</sub>X are `plfonts.dtx` and `pldefs.ltx`):

- JY3 is the font encoding for Japanese fonts (in horizontal direction).  
When vertical typesetting is supported by LuaT<sub>E</sub>X-ja in the future, JT3 will be used for vertical fonts.
- Two font families `mc` and `gt` are defined:

classification	family	\mdseries	\bfseries	scale
<i>mincho</i>	<code>mc</code>	Ryumin-Light	GothicBBB-Medium	0.962216
<i>gothic</i>	<code>gt</code>	GothicBBB-Medium	GothicBBB-Medium	0.962216

Remark that the bold series in both family are same as the medium series of *gothic* family. This is a convention in pL<sup>A</sup>T<sub>E</sub>X.

- 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 pL<sup>A</sup>T<sub>E</sub>X) and `jsclasses` (classes by Haruhiko Okumura), namely, `ltjclasses` and `ltjsclasses`.

**\CID, \UTF and macros in OTF package** Under pT<sub>E</sub>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, LuaT<sub>E</sub>X-ja supports some of functions in OTF package.

```

1 森
2 \UTF{9DD7}外と内田百\UTF{9592}とが\UTF{9AD9}鳥
   屋に行く。
3
4 \CID{7652}飾区の\CID{13706}野家 ,
5 葛飾区の吉野家

```

森鷗外と内田百間とが高島屋に行く。  
葛飾区の吉野家，葛飾区の吉野家

## 2.5 Changing Fonts

**Remark: Japanese Characters in Math Mode** Since pT<sub>E</sub>X supports Japanese characters in math mode, there are sources like the following:

```

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

```

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

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

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

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

```

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

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

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

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

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 Part II.

**plain T<sub>E</sub>X** To change Japanese fonts in plain T<sub>E</sub>X, you must use the primitive `\jfont`. So please see Part II.

**NFSS2** For L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>, LuaT<sub>E</sub>X-j<sub>a</sub> simply adopted the font selection system from that of pL<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> (in `plfonts.dtx`).

- Two control sequences `\mcdefault` and `\gtdefault` are used to specify the default font families for *mincho* and *gothic*, respectively. Default values: `mc` for `\mcdefault` and `gt` for `\gtdefault`.
- Commands `\fontfamily`, `\fontseries`, `\fontshape` and `\selectfont` can be used to change attributes of Japanese fonts.

	encoding	family	series	shape
alphabetic fonts	<code>\romanencoding</code>	<code>\romanfamily</code>	<code>\romanseries</code>	<code>\romanshape</code>
Japanese fonts	<code>\kanjiencoding</code>	<code>\kanjifamily</code>	<code>\kanjiseriess</code>	<code>\kanjishape</code>
both	—	—	<code>\fontseries</code>	<code>\fontshape</code>
auto select	<code>\fontencoding</code>	<code>\fontfamily</code>	—	—

- For defining a Japanese font family, use `\DeclareKanjiFamily` instead of `\DeclareFontFamily`.

**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 `fontspec`:

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>	

## 使用例

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 **JAg**lue will clash (see 4.1).

## 3 Changing Parameters

There are many parameters in LuaT<sub>E</sub>X-j<sub>a</sub>. And due to the behavior of LuaT<sub>E</sub>X, most of them are not stored as internal register of T<sub>E</sub>X, but as an original storage system in LuaT<sub>E</sub>X-j<sub>a</sub>. Hence, to assign or acquire those parameters, you have to use commands `\ltjsetparameter` and `\ltjgetparameter`.



### 3.1 Editing the range of JAchars

To edit the range of **JAchars**, 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` primitive. For example, the next line assigns whole characters in Supplementary Multilingual Plane and the character ‘漢’ to the range number 100.

```
\ltjdefcharrange{100}{"10000-"1FFFF,‘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 SMP belong the range 4 in the default setting of LuaTeX-ja, and if you specify the above line, then SMP will belong 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 **JAchars**, as the following line (this is just the default setting of LuaTeX-ja):

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

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

- Blocks in Unicode 6.0.
- The Adobe-Japan1-UCS2 mapping between a CID Adobe-Japan1-6 and Unicode.
- The PXbase bundle for pTeX 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 **JAchars** or not by default. These settings are similar to `prefercjk ...`

**Range 8<sup>J</sup>** 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:

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

**Range 1<sup>A</sup>** 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+02B0–U+02FF: Spacing Modifier Letters
- U+0100–U+017F: Latin Extended-A
- U+0300–U+036F: Combining Diacritical Marks
- U+0180–U+024F: Latin Extended-B
- U+1E00–U+1EFF: Latin Extended Additional
- U+0250–U+02AF: IPA Extensions

**Range 2<sup>J</sup>** 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<sup>J</sup>** Punctuations and Miscellaneous symbols. The block list is indicated in Table 1.

**Range 4<sup>A</sup>** 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, "FB50-"FE0F,
"FE20-"FE2F, "FE70-"FEFF, "10000-"1FFFF} % non-Japanese
```

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	Combining 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	Miscellaneous Mathematical Symbols-B
U+2B00–U+2BFF	Miscellaneous Symbols and Arrows
U+E000–U+F8FF	Private Use Area
U+FB00–U+FB4F	Alphabetic Presentation Forms

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)

**Range 5<sup>A</sup>** Surrogates and Supplementary Private Use Areas.

**Range 6<sup>J</sup>** Characters used in Japanese. The block list is indicated in Table 2.

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

## 3.2 kanjiskip and xkanjiskip

**JAg**lue is divided into the following three categories:

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

```
\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`.

## 3.3 Insertion Setting of xkanjiskip

It is not desirable that xkanjiskip is inserted between 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}}           p あq!う
2 p あ q !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`. For the compatibility with pTeX, natural numbers between 0 and 3 are also allowed as the second argument<sup>1</sup>.

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

## 3.4 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           _____ abc あいう 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):

<sup>1</sup>But we don’t recommend this: since numbers 1 and 2 have opposite meanings in `jaxspmode` and `alxspmode`.

```

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

```

### 3.5 Cropmark

Cropmark is a mark for indicating 4 corners and horizontal/vertical center of the paper. In Japanese, we call cropmark as tomo(w). p<sup>A</sup>T<sub>E</sub>X and this Lua<sub>T</sub>E<sub>X</sub>-ja support ‘tombow’ by their kernel. The following steps are needed to typeset cropmark:

1. First, define the banner which will be printed at the upper left of the paper. This is done by assigning a token list to `\@bannertoken`.

For example, the following sets banner as ‘filename (2012-01-01 17:01)’:

```

\makeatletter

\hour\time \divide\hour by 60 \@tempcnta\hour \multiply\@tempcnta 60\relax
\minute\time \advance\minute-\@tempcnta
\@bannertoken{%
  \jobname\space(\number\year-\two@digits\month-\two@digits\day
  \space\two@digits\hour:\two@digits\minute)}%

```

2. ...

## Part II

# Reference

## 4 Font Metric and Japanese Font

### 4.1 \jfont primitive

To load a font as a Japanese font, you must use the `\jfont` primitive instead of `\font`, while `\jfont` admits the same syntax used in `\font`. Lua<sub>T</sub>E<sub>X</sub>-ja automatically loads `luaotfload` package, so TrueType/OpenType fonts with features can be used for Japanese fonts:

```

1 \jfont\tradgt={file:ipaexg.ttf: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, hence the input like `\fontname\tradgt` causes an error. We denote control sequences which are defined in `\jfont` by `<jfont_cs>`.

**Prefix psft** Besides `file:` and `name:` prefixes, `psft:` can be used as a prefix in `\jfont` (and `\font`) primitive. Using this prefix, you can specify a ‘name-only’ Japanese font which will be not embedded to PDF. Typical use of this prefix is to specify the ‘standard’ Japanese fonts, namely, ‘Ryumin-Light’ and ‘GothicBBB-Medium’. For kerning or other informations, that of Kozuka Mincho Pr6N Regular (this is a font by Adobe Inc., and included in Japanese Font Packs for Adore Reader) will be used.

**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` primitive, you must specify which JFM will be used for this font by the following keys:

`jfm=<name>` Specify the name of JFM. A file named `jfm-<name>.lua` will be searched and/or loaded.

The followings are JFMs shipped with Lua<sub>T</sub>E<sub>X</sub>-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 `luatexja-otf.sty`, please 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 haracters 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 JFM, as showed below:

何かいい例 . 単純に「min10 にはバグあり」ではなく , プロポーショナルな側面も見せたいよね ( 乙部さんの min10.pdf の例を使う? )

`jfmvar=<string> ...`

**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 **JAg**lue, 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` primitive, when you want to use other font features, such as `script=...`
- If you want to use Japanese fonts in proportinal width, and use information from this font, use `jfm-prop.lua` for its JFM, and ...

TODO: kanjiskip?

## 4.2 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)

`kanjiskip={<natural>, <stretch>, <shrink>}` (optional)

This field specifies the ‘ideal’ amount of `kanjiskip`. As noted in Subsection 3.2, 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)

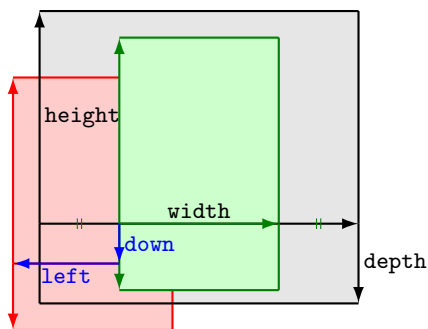
Like the `kanjiskip` field, this field specifies the ‘ideal’ amount of `xkanjiskip`.

Besides from above fields, a JFM file have several sub-tables those indices are natural numbers. The table indexed by  $i \in \omega$  stores informations 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 not required if  $i = 0$ , since all **J**Achar which are not in any character class other than 0 (hence, the character class 0 contains most of **J**Achars). In the list, a character can be specified by its code number, or by the character itself (as a string of length 1).

In addition to those ‘real’ characters, the following ‘imaginary characters’ can be specified in the list:



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.

`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>, ... }`

`glue={ [j]={<width>, <stretch>, <shrink>}, ... }`

### 4.3 Math Font Family

$\TeX$  handles fonts in math formulas by 16 font families<sup>2</sup>, and each family has three fonts: `\textfont`, `\scriptfont` and `\scriptscriptfont`.

Lua $\TeX$ -ja's handling of Japanese fonts in math formulas is similar; Table 4.3 shows counterparts to  $\TeX$ 's primitives for math font families.

## 5 Parameters

### 5.1 \ltjsetparameter primitive

As noted before, `\ltjsetparameter` and `\ltjgetparameter` are primitives for accessing most parameters of Lua $\TeX$ -ja. One of the main reason that Lua $\TeX$ -ja didn't adopted the syntax similar to that of  $\TeX$  (e.g., `\prebreakpenalty=10000`) is the position of `hpack_filter` callback in the source of Lua $\TeX$ , see Section 9.

`\ltjsetparameter` and `\ltjglobalsetparameter` are primitives 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 the primitive 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},           average, 1, 10000.
3 \ltjgetparameter{prebreakpenalty}{' }.
```

<sup>2</sup>Omega, Aleph, Lua $\TeX$  and  $\epsilon$ - $\TeX$  can handles 256 families, but an external package is needed to support this in plain  $\TeX$  and  $\LaTeX$ .

The return value of `\ljgetparameter` 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).

## 5.2 List of Parameters

In the following list of parameters, `[\cs]` indicates the counterpart in pTeX, and each symbol 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= $\langle penalt\y\rangle$  [\jcharwidowpenalty]`

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

`kcatcode= $\{\langle chr\_code\rangle, \langle natural\ number\rangle\}$`

An additional attributes having each character whose character code is  $\langle chr\_code\rangle$ . At the present version, the lowermost bit of  $\langle natural\ number\rangle$  indicates whether the character is considered as a punctuation mark (see the description of `jcharwidowpenalty` above).

`prebreakpenalty= $\{\langle chr\_code\rangle, \langle penalt\y\rangle\}$  [\prebreakpenalty]`

`postbreakpenalty= $\{\langle chr\_code\rangle, \langle penalt\y\rangle\}$  [\postbreakpenalty]`

`jatextfont= $\{\langle jfam\rangle, \langle jfont\_cs\rangle\}$  [\textfont in TEX]`

`jascriptfont= $\{\langle jfam\rangle, \langle jfont\_cs\rangle\}$  [\scriptfont in TEX]`

`jascriptscriptfont= $\{\langle jfam\rangle, \langle jfont\_cs\rangle\}$  [\scriptscriptfont in TEX]`

`yjabaselineshift= $\langle dimen\rangle^*$`

`yalbaselineshift= $\langle dimen\rangle^*$  [\ybaselineshift]`

`jaxspmode= $\{\langle chr\_code\rangle, \langle mode\rangle\}$  [\inhibitxspcode]`

Setting whether inserting `xkanjiskip` is allowed before/after a **JA**char whose character code is  $\langle chr\_code\rangle$ . The followings are allowed for  $\langle mode\rangle$ :

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

`alxspmode= $\{\langle chr\_code\rangle, \langle mode\rangle\}$  [\xspcode]`

Setting whether inserting `xkanjiskip` is allowed before/after a **AL**char whose character code is  $\langle chr\_code\rangle$ . The followings are allowed for  $\langle mode\rangle$ :

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

Note that parameters `jaxspmode` and `alxspmode` use a common table.

`autospacing= $\langle bool\rangle^*$  [\autospacing]`

`autoxspacing= $\langle bool\rangle^*$  [\autoxspacing]`

`kanjiskip= $\langle skip\rangle$  [\kanjiskip]`

`xkanjiskip=<skip>` [`\xkanjiskip`]

`differentjfm=<mode>`<sup>†</sup> Specify how glues/kerns between two **J**Achars whose JFM (or size) are different. The allowed arguments are the followings:

average

both

large

small

`jacharrange=<ranges>*`

`kansujichar=<{digit}>, <chr_code>` [`\kansujichar`]

## 6 Other Primitives

### 6.1 Compatibility with pT<sub>E</sub>X

`\kuten`

`\jis`

`\euc`

`\sjis`

`\ucs`

`\kansuji`

## 7 Control Sequences for L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>

### 7.1 Patch for NFSS2

As described in Subsection 2.4, LuaT<sub>E</sub>X-ja simply adopted `plfonts.dtx` in pL<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> for the Japanese patch for NFSS2.

### 7.2 Cropmark/‘tombow’

## 8 Extensions

### 8.1 `luatexja-fontspec.sty`

### 8.2 `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.



**Additionally 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`.

## Part III

# Implementations

## 9 Storing Parameters

### 9.1 Used Dimensions, Attributes and whatsit nodes

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

`\jQ` (dimension) As explained in Subsection 2.3, `\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  $0.25\text{ mm}$  and used in Japanese phototypesetting. The dimension `\jH` stores this length, similar to `\jQ`.

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

`\tj@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.

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

`\tj@charclass` (attribute) The character class of Japanese *glyph\_node*.

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

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

`\tj@autospc` (attribute) Whether the auto insertion of `kanjiskip` is allowed at the node.

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

`\tj@icflag` (attribute) For distinguishing ‘kinds’ of the node. To this attribute, one of the following value is assigned:

**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 (4)** Glues/kerns from JFM.

**LINE\_END (5)** Kerns for ...

**KANJL\_SKIP (6)** Glues for `kanjiskip`.

**XKANJI\_SKIP (7)** Glues for `xkanjiskip`.

**PROCESSED (8)** Nodes which is already processed by ...

**IC\_PROCESSED (9)** 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.

`\tj@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 typesetting. All those nodes store a natural number (hence the node’s `type` is 100).

**30111** Nodes for indicating that `\inhibitglue` is specified. The `value` field of these nodes is not used.

**30112** Nodes for LuaTeX-ja’s stack system (see the next subsection). The `value` field of these nodes is current group.

**30113** 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.

## 9.2 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}ふがふが.%
2 \setbox0=\hbox{\ltjsetparameter{kanjiskip=5pt}   ふがふが. ほげほげ. びよびよ
   ほげほげ}
3 \box0. びよびよ\par

```

As described in Part II, 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 5pt. However, by the implementation method of LuaTeX, this ‘5pt’ 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 ‘5pt’ 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 list<sup>3</sup>.

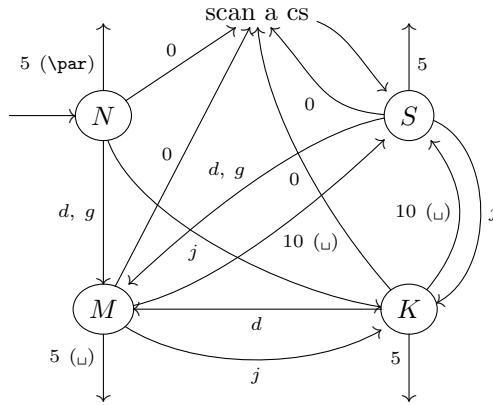
These are two TeX count registers for maintaining informations: `\ltj@@stack` for the stack level, and `\ltj@@group@level` for the TeX’s group level when the last assignment was done. Parameters are stored in one big table named `charprop_stack_table`, where `charprop_stack_table[i]` stores data of stack 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 `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`. This problem is resolved by using `\directlua{tex.globaldefs=0}` (this assignment is local).

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



$d := \{3, 4, 6, 7, 8, 11, 12, 13\}$ ,  $g := \{1, 2\}$ ,  $j := (\text{Japanese characters})$

- Numbers represent category codes.
- Category codes 9 (ignored), 14 (comment) and 15 (invalid) are omitted in above diagram.

Figure 2. State transitions of pTeX’s input processor.

## 10 Linebreak after Japanese Character

### 10.1 Reference: Behavior in pTeX

(NOT COMPLETED)

In pTeX, a linebreak 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 linebreak 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 [?]. The internal states are as follows:

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

The first three states—*N*, *S* and *M*—are as same as TeX’s input processor. State *K* is similar to state *M*, and is entered after Japanese characters. The diagram of state transitions are indicated in Figure 10.1. Note that pTeX doesn’t leave state *K* after ‘beginning/ending of a group’ characters.

### 10.2 Behavior in LuaTeX-ja

States in the input processor of LuaTeX is the same as that of TeX, and they can’t be customized by any callbacks. Hence, we can only use `process_input_buffer` and `token_filter` callbacks for to suppress a space by a linebreak 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 a 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 two conditions are satisfied:

1. The category code of the character `<return>` (whose character code is 13) is 5 (end-of-line).

2. The input line matches the following ‘regular expression’:

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

## 11 Insertion of JFM glues, **kanjiskip** and **xkanjiskip**

This is the longest section of the document.

jfmglue.tex の内容をここに入れる

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

Table 4. Primitives for Japanese math fonts.

	Japanese fonts	alphabetic fonts
font family	$\backslash\text{jfam} \in [0, 256)$	$\backslash\text{fam}$
text size	$\text{jatextfont}=\{\langle\text{jfam}\rangle, \langle\text{jfont\_cs}\rangle\}$	$\backslash\text{textfont}\langle\text{fam}\rangle=\langle\text{font\_cs}\rangle$
script size	$\text{jascriptfont}=\{\langle\text{jfam}\rangle, \langle\text{jfont\_cs}\rangle\}$	$\backslash\text{scriptfont}\langle\text{fam}\rangle=\langle\text{font\_cs}\rangle$
scriptscript size	$\text{jascriptscriptfont}=\{\langle\text{jfam}\rangle, \langle\text{jfont\_cs}\rangle\}$	$\backslash\text{scriptscriptfont}\langle\text{fam}\rangle=\langle\text{font\_cs}\rangle$