The LuaTEX-ja package

The LuaTEX-ja project team

October 12, 2011

Contents

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

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 pTEX (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), and an optional string called 'variation'.
- In pTEX, 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, 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 these glues/kerns as **JAglue**) 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 **JAglue** 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 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 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 LuaTEX, but also for control sequences that defined in the core module of LuaTEX-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

- Hironori KITAGAWA
- Kazuki MAEDA
- Takayuki YATO
- Yusuke KUROKI
- Noriyuki ABE
- Munehiro YAMAMOTO
- Tomoaki HONDA

2 Getting Started

2.1 Installation

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

- LuaTEX (version 0.65.0-beta or later) and its supporting packages.
  If you are using TEX Live 2011 or current W32TEX, you don't have to worry.
- The source archive of LuaTEX-ja, of course:)

The installation methods are as follows:

1. Download the source archive.

   At the present, LuaTEX-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 **JAchar** 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 **ALchar** as side-effect:

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

2.3 Using in plain TEX

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

```
\input luatexja.sty
```

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

- The following 6 Japanese fonts are preloaded:

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

  - 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 **JAchar** and an **ALchar** (the parameter xkanjiskip) is set to

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

2.4 Using in LaTeX

LaTeX 2ε Using in LaTeX 2ε is basically same. To set up the minimal environment for Japanese, you only have to load `luatexja.sty`:

```
\usepackage{luatexja}
```

It also does minimal settings (counterparts in pLaTeX are `plfonts.dtx` and `pldefs.ltx`):

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

classification	family	\mdseries	\bfseries	scale
mincho	mc	Ryumin-Light	GothicBBB-Medium	0.962216
gothic	gt	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 pLaTeX.
- Japanese characters in math mode are typeset by the font family `mc`.

However, above settings are not sufficient for Japanese-based documents. To typeset Japanese-based documents, You are better to use class files other than `article.cls`, `book.cls`, and so on. At the present, we have the counterparts of `jclasses` (standard classes in pLaTeX) and `jsclasses` (classes by Haruhiko Okumura), namely, `ltjclasses` and `ltjsclasses`.

`\CID`, `\UTF` and macros in OTF package Under pTeX, `OTF` package (developed by Shuzaburo Saito) is used for typesetting characters which is in Adobe-japan1-6 CID but not in JIS X 0208. Since this package is widely used, LuaTeX-ja supports some of functions in `OTF` package.

```
森
\UTF{9DD7}外と内田百\UTF{9592}とが\UTF{9AD9}島屋に行く。

\CID{7652}飾区の\CID{13706}野家，
葛飾区の吉野家
```

森鷗外と内田百閒とが髙島屋に行く。
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 pTeX supports Japanese characters in math mode, there are sources like the following:

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

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

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

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

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

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

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

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

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

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

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

We also believe that using Japanese characters as identifiers is rare, hence we don't describe how to change Japanese fonts in math mode in this chapter. For the method, please see Part II.

plain TEX To change Japanese fonts in plain TEX, you must use the primitive `\jfont`. So please see Part II.

NFSS2 For LATEX 2_ε, LuaTEX-ja simply adopted the font selection system from that of pLATEX 2_ε (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**	**selection**
alphabetic fonts	`\romanencoding`	`\romanfamily`	`\romanseries`	`\romanshape`	`\useroman`
Japanese fonts	`\kanjiencoding`	`\kanjifamily`	`\kanjiseries`	`\kanjishape`	`\usekanji`
both	—	–	`\fontseries`	`\fontshape`	—
auto select	`\fontencoding`	`\fontfamily`	—	—	`\usefont`

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

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 **JAglue** will clash (see 4.1).

3 Changing Parameters

There are many parameters in LuaTEX-ja. And due to the behavior of LuaTEX, most of them are not stored as internal register of TEX, but as an original storage system in LuaTEX-ja. Hence, to assign or acquire those parameters, you have to use commands `\ltjsetparameter` and `\ltjgetparameter`.

3.1 Editing the range of JAchars

To edit the range of **JAchar**s, 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 **JAchar**s, as the following line (this is just the default setting of LuaTEX-ja):

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

The argument to jacharrange parameter is a list of integer. Negative interger $-n$ in the list means that 'the character range n is ...'.

Default Setting LuaTEX-ja predefines eight character ranges for convinience. 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 **JAchar**s or not by default. These settings are similar to `prefercjk` ...

Range 8$^{\mathbf{J}}$ 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 charatcers:

- § (`U+00A7`, Section Sign)
- ¨ (`U+00A8`, Umlaut or 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$^{\mathbf{A}}$ 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$^{\mathbf{J}}$ 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$^{\mathbf{J}}$ Punctuations and Miscellaneous symbols. The block list is indicated in Table 1.

Range 4$^{\mathbf{A}}$ 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:

Table 1. Unicode blocks in predefined character range 3.

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

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

```
\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
```

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.

3.2 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 be not inserted at the place.
- The default glue which inserted between two **JAchar**s (kanjiskip).
- The default glue which inserted between a **JAchar** and an **ALchar** (xkanjiskip).

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

```
\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 **JAchar**s and **ALchar**s. For example, xkanjiskip should not be inserted after opening parenthesis (*e.g.*, compare '(あ' and '(あ').

LuaTEX-ja can control whether xkanjiskip can be inserted before/after a character, by changing jaxspmode for **JAchar**s and alxspmode parameters **ALchar**s respectively.

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

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[1].

If you want to enable/disable all insertions of kanjiskip and xkanjiskip, set autospacing 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).

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

abc あいう abc あいう

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):

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

xyz 漢字 XYZ ひらがな abc かな

3.5 Cropmark

Cropmark is a mark for indicating 4 corners and horizontal/vertical center of the paper. In Japanese, we call cropmark as tombo(w). pLATEX and this LuaTEX-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)}%
   ```

[1] But we don't recommend this: since numbers 1 and 2 have opposite meanings in jaxspmode and alxspmode.

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`. LuaTEX-ja automatically loads `luaotfload` package, so TrueType/OpenType fonts with features can be used for Japanese fonts:

```
\jfont\tradgt={file:ipaexg.ttf:script=latn;%
  +trad;-kern;jfm=ujis} at 14pt
\tradgt{}当／体／医／区
```

當／體／醫／區

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

Prefix `psft` Besides `file:` and `name:` prefixes, `psft:` can be used 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 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 `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 JFMs, as shown in Table 4.

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

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` 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?

Table 4. Differences between JFMs shipped with LuaTEX-ja

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

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.

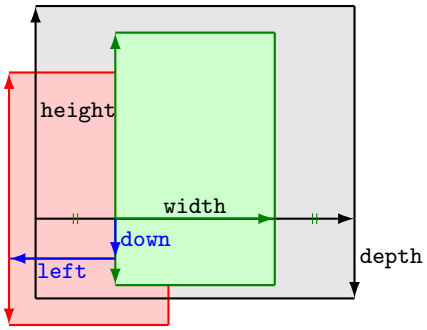
Besides from above fields, a JFM file have several sub-tables those indices are natural numbers. The table indexed by $i \in \omega$ stores informations of 'character class' i. At least, the character class 0 is always present, so each JFM file must have a sub-table whose index is `[0]`. Each sub-table (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 **JAchar** which are not in any character class other than 0 (hence, the character class 0 contains most of **JAchar**s). In the list, a character can be specified by its code number, or by the character itself (as a string of length 1). Moreover, there are 'imaginary characters' which specified in the list. 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



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.

Table 5. Primitives for Japanese math fonts.

	Japanese fonts	alphabetic fonts
font family	`\jfam` $\in [0, 256)$	`\fam`
text size	jatextfont ={⟨*jfam*⟩,⟨*jfont_cs*⟩}	`\textfont`⟨*fam*⟩=⟨*font_cs*⟩
script size	jascriptfont ={⟨*jfam*⟩,⟨*jfont_cs*⟩}	`\scriptfont`⟨*fam*⟩=⟨*font_cs*⟩
scriptscript size	jascriptscriptfont ={⟨*jfam*⟩,⟨*jfont_cs*⟩}	`\scriptscriptfont`⟨*fam*⟩=⟨*font_cs*⟩

`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*⟩`}, ...}`

`'lineend'` An ending of a line.

`'diffmet'` Used at a boundary between two **JAchar**s whose JFM or size is different.

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

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

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

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

4.3 Math Font Family

TEX handles fonts in math formulas by 16 font families[2], 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 appropreate settings, you can set both `\fam` and `\jfam` to the same value.

[2]Omega, Aleph, LuaTEX and ε-(u)pTEXcan handles 256 families, but an external package is needed to support this in plain TEX and LATEX.

4.4 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.

```
function (<table> jfm_info, <string> jfm_name)
  return <table> new_jfm_info
end
```

The argument `jfm_info` contains a table similar to the table in a JFM file, except this argument has `chars` field which contains character codes whose character class is not 0.

An example of this callback is the `ltjarticle` class, with forcefully assigning character class 0 to `'parbdd'` in the JFM `jfm-min.lua`. This callback doesn't replace any code of LuaTEX-ja.

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

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

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

`jfm` The index number of JFM.

`size` Font size in a scaled point ($= 2^{-16}$ pt).

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

The returned table `new_jfont_info` also should include these three fields. The `font_number` is a font number.

A good example of this and the next callbacks is `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 inready 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.

This callback doesn't replace any code of LuaTEX-ja.

5 Parameters

5.1 `\ltjsetparameter` primitive

As noted before, `\ltjsetparameter` and `\ltjgetparameter` are primitives 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 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.

```
\ltjgetparameter{differentjfm},
\ltjgetparameter{autospacing},                  average, 1, 10000.
\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).

5.2 List of Parameters

The following is the list of parameters which can be specificated by the `\ltjsetparameter` command. [`\cs`] indicates the counterpart in pTEX, 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 supressing orphans. This penalty is inserted just after the last **JAchar** which is not regarded as a (Japanese) punctuation mark.

kcatcode ={⟨*chr_code*⟩,⟨*natural number*⟩}

An additional attributes having each character whose character code is ⟨*chr_code*⟩. 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 TEX]

jascriptfont ={⟨*jfam*⟩,⟨*jfont_cs*⟩} [`\scriptfont` in TEX]

jascriptscriptfont ={⟨*jfam*⟩,⟨*jfont_cs*⟩} [`\scriptscriptfont` in TEX]

yjabaselineshift =⟨*dimen*⟩*

yalbaselineshift =⟨*dimen*⟩* [`\ybaselineshift`]

jaxspmode ={⟨*chr_code*⟩,⟨*mode*⟩} [`\inhibitxspcode`]

Setting whether inserting xkanjiskip is allowed before/after a **JAchar** whose character code is ⟨*chr_code*⟩. The followings are allowed for ⟨*mode*⟩:

0, `inhibit` Insertion of xkanjiskip is inhibited before the 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 ={⟨*chr_code*⟩,⟨*mode*⟩} [`\xspcode`]

Setting whether inserting xkanjiskip is allowed before/after a **ALchar** whose character code is ⟨*chr_code*⟩. The followings are allowed for ⟨*mode*⟩:

0, `inhibit` Insertion of xkanjiskip is inhibited before the 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 both before the charater and after the charater. This is the default value.

Note that parameters jaxspmode and alxspmode use a common table.

autospacing =⟨*bool*⟩* [`\autospacing`]

autoxspacing =⟨*bool*⟩* [`\autoxspacing`]

kanjiskip =⟨*skip*⟩ [`\kanjiskip`]

xkanjiskip =⟨*skip*⟩ [`\xkanjiskip`]

differentjfm =⟨*mode*⟩† Specify how glues/kerns between two **JAchar**s whose JFM (or size) are different. The allowed arguments are the followings:

`average`

`both`

`large`

`small`

jacharrange =⟨*ranges*⟩*

kansujichar ={⟨*digit*⟩, ⟨*chr_code*⟩} [`\kansujichar`]

6 Other Primitives

6.1 Primitives for Compatibility

The following primtives are implemented for compatibility with pTEX:

`\kuten`

`\jis`

`\euc`

`\sjis`

`\ucs`

`\kansuji`

6.2 `\inhibitglue`

The primitive `\inhibitglue` suppresses the insertion of **JAglue**. The following is an example, using a special JFM that there will be a glue between the beginning of a box and 'あ', and also between 'あ' and 'ウ'.

```
1 \jfont\g=psft:Ryumin-Light:jfm=test \g
2 あウあ\inhibitglue{}ウ\inhibitglue\par
3 あ\par\inhibitglue{}あ
4 \par\inhibitglue\hrule{}あoff\inhibitglue ice
```

あ　ウあウ
あ
あ
あ office

With the help of this example, we remark the specification of `\inhibitglue`:

- The call of `\inhibitglue` in the (internal) vertical mode is effective at the beginning of the next paragraph. This is realized by hacking `\everypar`.
- 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 line 4 of above example.
- The call of `\inhibitglue` in math mode is just ignored.

7 Control Sequences for LaTeX 2ε

7.1 Patch for NFSS2

As described in Subsection 2.4, LuaTeX-ja simply adopted `plfonts.dtx` in pLaTeX 2ε for the Japanese patch for NFSS2. For an convinience, we will describe commands which are not described in Subsection 2.5.

`\DeclareYokoKanjiEncoding{`⟨*encoding*⟩`}{`⟨*text-settings*⟩`}{`⟨*math-settings*⟩`}`

In NFSS2 under LuaTeX-ja, distinction between alphabetic font families and Japanese font families is only made by its encoding. 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*⟩`}`

和文・欧文の数式用フォントファミリを一度に変更する命令を作成する．具体的には，欧文数式用フォントファミリ変更の命令 ⟨*al-cmd*⟩ と，和文数式用フォントファミリ変更の命令 ⟨*ja-cmd*⟩ の 2 つを同時に行う命令として ⟨*unified-cmd*⟩ を（再）定義する．実際の使用では ⟨*unified-cmd*⟩ と ⟨*al-cmd*⟩ に同じものを指定する，すなわち，⟨*al-cmd*⟩ に和文側も変更させるようにするのが一般的と思われる．

本コマンドの使用については，pLaTeX 配布中の `plfonts.dtx` に詳しく注意点が述べられているので，そちらを参照されたい．

`\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`

...

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

```
1 \gtfamily{}あいうabc
2 \SetRelationFont{JY3}{gt}{m}{n}{OT1}{pag}{m}{n}  あいう abc  あいう abc
3 \userelfont\selectfont{}あいうabc
```

7.2 Cropmark/'tombow'

8 Extensions

8.1 `luatexja-fontspec.sty`

8.2 `luatexja-otf.sty`

This optional package supports typesetting charaters 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 **JAchar**s.
- 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 $1\,\mathrm{Q} = 0.25\,\mathrm{mm}$, where 'Q' (also called '級') is a unit used in Japanese phototypesetting. So one should not change the value of this dimension.

`\jH` (dimension) There is also a unit called '歯' which equals to 0.25 mm and used in Japanese phototypesetting. The dimension `\jH` stores this length, similar to `\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 itaric 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 ...

***kanji_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 itaric correction, but also already processed.

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

`\ltj@kcat`*i* (attribute) Where *i* is a natural number which is less than 7. These 7 attributes store bit vectors indicating which character block is regarded as a block of **JAchar**s.

Furthermore, LuaTEX-ja uses several 'user-defined' whatsit nodes for 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 doesn't matter.

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, andd 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.

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

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}
    ほげほげ}
3 \box0.ぴよぴよ\par
```

ふがふが. ほ げ ほ げ. ぴよぴよ

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 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;
```

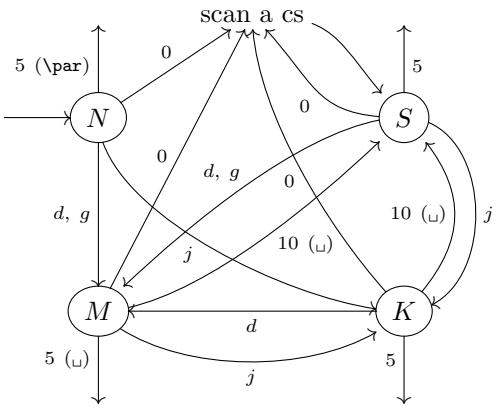
Notice that `unsave` is executed *before* `filtered_hpack` (this is where `hpack_filter` callback is executed): so '5 pt' in the above source is orphaned at +unsave+, and hence it can't be accessed from `hpack_filter` callback.

The method The code of stack system is based on that in a post of Dev-luatex mailing list[3].

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:

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



$d := \{3, 4, 6, 7, 8, 11, 12, 13\}, \quad g := \{1, 2\}, \quad j :=$ (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.

- 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 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 `\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).

10 Linebreak after Japanese Character

10.1 Reference: Behavior in pTEX

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 2. 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 processoe 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 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 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':
>
> $$(\text{any char})^*(\textbf{JAchar})\big(\{\text{catcode} = 1\} \cup \{\text{catcode} = 2\}\big)^*$$

Remark The following example shows the major difference from the behavior of pTEX:

```
1 \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).

11 Insertion of JFM glues, kanjiskip and xkanjiskip

11.1 Overview

NOT COMPLETED

11.2 Definition of a 'cluster'

Definition 1. A *cluster* is a list of nodes in one of the following forms, with the *id* of it:

1. Nodes whose value of `\ltj@icflag` is in $[3, 15)$. These nodes come from a hbox which is already packaged, by unpackaging (`\unhbox`). The *id* is *id_pbox*.
2. A inline math formula, including two *math_node*s at the boundary of it: HOGE The *id* is *id_math*.
3. A *glyph_node* with nodes which relate with it: HOGE The *id* is *id_jglyph* or *id_glyph*, according to whether the *glyph_node* represents a Japanese character or not.
4. An box-like node, that is, an hbox, an vbox and an rule (`\vrule`). The *id* is *id_hlist* if the node is an hbox which is not shifted vertically, or *id_box_like* otherwise.
5. A glue, a kern whose subtype is not 2 (*accent*), and a discretionary break. The *id* is *id_glue*, *id_kern* and *id_disc*, respectively.

We denote a cluster by *Np*, *Nq* and *Nr*.

Internally, a cluster is represented by a table *Np* with the following fields.

first, last The first/last node of the cluster.

id The *id* in above definition.

nuc

auto_kspc, auto_xspc

xspc_before, xspc_after

pre, post

char

class

lend

met, var