

The LuaT_EX-ja package

The LuaT_EX-ja project team

March 18, 2013

Contents

I	User's manual	3
1	Introduction	3
1.1	Backgrounds	3
1.2	Major Changes from p \TeX	3
1.3	Notations	4
1.4	About the project	4
2	Getting Started	5
2.1	Installation	5
2.2	Cautions	6
2.3	Using in plain \TeX	6
2.4	Using in \LaTeX	6
3	Changing Fonts	7
3.1	plain \TeX and $\LaTeX 2_{\epsilon}$	7
3.2	fontspec	8
3.3	Preset	8
3.4	$\backslash CID$, $\backslash UTF$ and macros in <code>otf</code> package	10
4	Changing Parameters	10
4.1	Editing the range of <code>JAchars</code>	10
4.2	<code>kanjiskip</code> and <code>xkanjiskip</code>	12
4.3	Insertion Setting of <code>xkanjiskip</code>	12
4.4	Shifting Baseline	13
II	Reference	13
5	Font Metric and Japanese Font	13
5.1	<code>\jfont</code>	13
5.2	Prefix <code>psft</code>	14
5.3	Structure of JFM file	15
5.4	Math Font Family	17
5.5	Callbacks	17
6	Parameters	18
6.1	<code>\tjsetparameter</code>	18
6.2	List of Parameters	19
7	Other Control Sequences	20
7.1	Control Sequences for Compatibility	20
7.2	<code>\inhibitglue</code>	20
8	Control Sequences for $\LaTeX 2_{\epsilon}$	21
8.1	Patch for NFSS2	21

9 Extensions	22
9.1 luatexja-fontspec.sty	22
9.2 luatexja-otf.sty	22
9.3 luatexja-adjust.sty	22
III Implementations	22
10 Storing Parameters	23
10.1 Used Dimensions, Attributes and whatsit nodes	23
10.2 Stack System of LuaTeX-ja	24
11 Linebreak after Japanese Character	25
11.1 Reference: Behavior in pTeX	25
11.2 Behavior in LuaTeX-ja	25
12 Insertion of JFM glues, kanjiskip and xkanjiskip	26
12.1 Overview	26
12.2 definition of a ‘cluster’	26
12.3 段落／水平ボックスの先頭や末尾	28
12.4 概観と典型例：2つの「和文 A」の場合	29
12.5 その他の場合	31
13 psft	34
14 Patch for the listings package	35
15 Advanced line-adjustment for Japanese characters	36
References	36
A The category code of non-kanji characters defined in JIS X 0213	37
B Package versions used in this document	40

This documentation is far from complete. It may have many grammatical (and contextual) errors. Also, several parts (especially, Section 12) are written in Japanese only.

Part I

User's manual

1 Introduction

The LuaTeX-ja package is a macro package for typesetting high-quality Japanese documents when using LuaTeX.

1.1 Backgrounds

Traditionally, ASCII pTeX, an extension of TeX, and its derivatives are used to typeset Japanese documents in TeX. pTeX is an engine extension of TeX: so it can produce high-quality Japanese documents without using very complicated macros. But this point is a mixed blessing: pTeX is left behind from other extensions of TeX, especially ϵ -TeX and pdfTeX, and from changes about Japanese processing in computers (*e.g.*, the UTF-8 encoding).

Recently extensions of pTeX, namely upTeX (Unicode-implementation of pTeX) and ϵ -pTeX (merging of pTeX and ϵ -TeX extension), have developed to fill those gaps to some extent, but gaps still exist.

However, the appearance of LuaTeX changed the whole situation. With using Lua ‘callbacks’, users can customize the internal processing of LuaTeX. So there is no need to modify sources of engines to support Japanese typesetting: to do this, we only have to write Lua scripts for appropriate callbacks.

1.2 Major Changes from pTeX

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

The followings are major changes from pTeX:

- A Japanese font is a tuple of a ‘real’ font, a Japanese font metric (**JFM**, for short), and an optional string called ‘variation’.
- In pTeX, a line break after Japanese character is ignored (and doesn’t yield a space), since line breaks (in source files) are permitted almost everywhere in Japanese texts. However, LuaTeX-ja doesn’t have this function completely, because of a specification of LuaTeX.
- The insertion process of glues/kerns between two Japanese characters and between a Japanese character and other characters (we refer glues/kerns of both kinds as **JAg glue**) is rewritten from scratch.
 - As LuaTeX’s internal character handling is ‘node-based’ (*e.g.*, of `{ }fice` doesn’t prevent ligatures), the insertion process of **JAg glue** is now ‘node-based’.
 - Furthermore, nodes between two characters which have no effects in line break (*e.g.*, `\special` node) and kerns from italic correction are ignored in the insertion process.
 - *Caution: due to above two points, many methods which did for the dividing the process of the insertion of **JAg glue** in pTeX are not effective anymore.* In concrete terms, the following two methods are not effective anymore:

ちよ{}つと ちよ\つと

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

ちよ\hbox{}つと

- In the process, two Japanese fonts which only differ in their ‘real’ fonts are identified.
- At the present, vertical typesetting (*tategaki*), is not supported in LuaTeX-ja.

For detailed information, see Part III.

1.3 Notations

In this document, the following terms and notations are used:

- Characters are divided into two types:
 - **J_Achar**: 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 **J_Achar**.

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

1.4 About the project

Project Wiki Project Wiki is under construction.

- <http://sourceforge.jp/projects/luatex-ja/wiki/FrontPage%28en%29> (English)
- <http://sourceforge.jp/projects/luatex-ja/wiki/FrontPage> (Japanese)
- <http://sourceforge.jp/projects/luatex-ja/wiki/FrontPage%28zh%29> (Chinese)

This project is hosted by SourceForge.JP.

Members

- Hironori KITAGAWA
- Kazuki MAEDA
- Takayuki YATO
- Yusuke KUROKI
- Noriyuki ABE
- Munehiro YAMAMOTO
- Tomoaki HONDA
- Shuzaburo SAITO
- MA Qiyuan

2 Getting Started

2.1 Installation

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

- LuaTeX (version 0.65.0-beta or later) and its supporting packages.

You might need to add the following lines to `lualatex.ini` (just before `\dump` in the last line), if you are using recent LuaTeX whose Lua is 5.2.

```
{\catcode`\#=12\catcode`\~ =12%
\global\everyjob\expandafter{\the\everyjob%
  \directlua{%
    if not table.maxn then
      table.maxn = function(t)
        local r = 0
        for i,_ in pairs(t) do
          if type(i)=='number' then
            if i>r then r=i end
          end
        end
        return r
      end
    end
    if not package.loaders then package.loaders=package.searchers end
    if not string.explode then
      string.explode = function (str, separator)
        if not separator then separator=" +" end
        local t, nexti, pos = { }, 1, 1
        while true do
          local st, sp = str:find (separator, pos)
          if not st then break end
          if pos ~= st then
            t [ nexti ] = str:sub ( pos , st - 1 )
            nexti = nexti + 1
          end
          pos = sp + 1
        end
        t [ nexti ] = str:sub ( pos )
        return t
      end
    end
  }%
}%
```

- The source archive of LuaTeX-ja, of course :)
- The `xunicode` package, which version is *just v0.981 (2011/09/09)*. If you have the `fontspec` package, this `xunicode` package must be exist. But be careful about the version; other versions may not work correctly with LuaTeX-ja.

The installation methods are as follows:

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

- Copy the Git repository:

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

- Download the `tar.gz` archive of HEAD in the master branch from

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

- Now LuaTeX-ja is available from the following archive and distributions:
 - CTAN (in the `macros/luatex/generic/luatexja` directory)
 - MiKTeX (in `luatexja.tar.lzma`)
 - TeX Live (in `texmf-dist/tex/luatex/luatexja`)
 - W32TeX (in `luatexja.tar.xz`)

These are based on the master branch.

Note that the master branch, and hence the archive in CTAN, are not updated frequently; the forefront of development is not the master branch.

2. Extract the archive. You will see `src/` and several other sub-directories. But only the contents in `src/` are needed to work LuaTeX-ja.
3. Copy all the contents of `src/` into one of your TEXMF tree. `TEXMF/tex/luatex/luatexja/` is an example location. If you cloned entire Git repository, making a symbolic link of `src/` instead copying is also good.
4. If `mktexlsr` is needed to update the file name database, make it so.

2.2 Cautions

- The encoding of your source file must be UTF-8. No other encodings, such as EUC-JP or Shift-JIS, are not supported.
- LuaTeX-ja is very slower than pTeX. Using LuaJITTeX slightly improve the situation.

2.3 Using in plain TeX

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

```
\input luatexja.sty
```

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

- The following 6 Japanese fonts are preloaded:

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

- It is widely accepted that the font 'Ryumin-Light' and 'GothicBBB-Medium' aren't embedded into PDF files, and a 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 `xkanjskip`) is set to

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

2.4 Using in L^AT_εX

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

```
\usepackage{luatexja}
```

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

- JY3 is the font encoding for Japanese fonts (in horizontal direction).
When vertical typesetting is supported by LuaTeX-ja in the future, JT3 will be used for vertical fonts.
- 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^AT_EX. This is a trace that there were only 2 fonts (these are Ryumin-Light and GothicBBB-Medium) in early years of DTP.

- 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^AT_EX) and `jsclasses` (classes by Haruhiko Okumura), namely, `ltjclasses` and `ltjsclasses`.

3 Changing Fonts

3.1 plain T_EX and L^AT_EX 2_ε

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

L^AT_EX 2_ε (NFSS2) For L^AT_EX 2_ε, LuaTeX-ja adopted most of the font selection system of pL^AT_EX 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	<code>\romanencoding</code>	<code>\romanfamily</code>	<code>\romanseries</code>	<code>\romanshape</code>	<code>\useroman</code>
Japanese fonts	<code>\kanjiencoding</code>	<code>\kanjifamily</code>	<code>\kanjiserie</code>	<code>\kanjishape</code>	<code>\usekanji</code>
both	—	—	<code>\fontseries</code>	<code>\fontshape</code>	—
auto select	<code>\fontencoding</code>	<code>\fontfamily</code>	—	—	<code>\usefont</code>

`\fontencoding{<encoding>}` changes the encoding of alphabetic fonts or Japanese fonts depending on the argument. For example, `\fontencoding{JY3}` changes the encoding of Japanese fonts to JY3 and `\fontencoding{T1}` changes the encoding of alphabetic fonts to T1. `\fontfamily` also changes the family of Japanese fonts, alphabetic fonts, or both. For detail, see Subsection 8.1.

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

Remark: Japanese Characters in Math Mode Since pL^AT_EX 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 $5\in\text{素}:=\{\,p\in\mathbb{N}:\text{the }p\text{ is a prime}\,\}$.
```

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

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

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

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

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

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

```

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

```

We also believe that using Japanese characters as identifiers is rare, hence we don’t describe how to change Japanese fonts in math mode in this chapter. For the method, please see Subsection 5.4.

3.2 fontspec

To coexist with the `fontspec` package, it is needed to load `luatexja-fontspec` package in the preamble. This additional package automatically loads `luatexja` and `fontspec` package, if needed.

In `luatexja-fontspec` package, the following 7 commands are defined as counterparts of original commands in the `fontspec` package:

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

```

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

```

Note that there is no command named `\setmonojfont`, since it is popular for Japanese fonts that nearly all Japanese glyphs have same widths. Also note that the kerning feature is set off by default in these 7 commands, since this feature and **JAg** will clash (see 5.1).

3.3 Preset

To use standard Japanese font settings easily, one can load `luatexja-preset` package with several options. This package provides functions in a part of `otf` package and a part of `PXchfon` package by Takayuki Yato, and loads `luatexja-fontspec` internally.

General options

deluxe Specifying this option enables us to use *mincho* with two weights (medium and bold), *gothic* with three weights (medium, bold and heavy), and *rounded gothic*¹. The heavy weight of *gothic* can be used by “changing the family” `\gtebfamily`. This is because `fontspec` package can handle only medium (`\mdseries`) and bold (`\bfseries`).

expert Use horizontal kana alternates, and define a control sequence `\rubyfamily` to use kana characters designed for ruby.

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

`bold` Use bold gothic as bold mincho.

`90jis` Use fonts with 90JIS glyphs if possible.

`jis2004` Use fonts with JIS2004 glyphs if possible.

`jis` Use the JFM `jfm-jis.lua`, instead of `jfm-ujis.lua`, which is the default JFM of LuaTeX-ja.

Kozuka fonts When using single weight, we adopt Kozuka Gothic M as *gothic*, because we think that Kozuka Gothic R looks thin. There is not ‘Kozuka Maru Gothic’, therefore Kozuka Gothic H is used as a substitute for *rounded gothic*.

	<code>kozuka4</code>	<code>kozuka6</code>	<code>kozuka6n</code>
mincho medium	Kozuka Mincho Pro R	Kozuka Mincho ProVI R	Kozuka Mincho Pr6N R
mincho bold	Kozuka Mincho Pro B	Kozuka Mincho ProVI B	Kozuka Mincho Pr6N B
gothic medium without <code>deluxe</code> multiple weights	Kozuka Gothic Pro M Kozuka Gothic Pro R	Kozuka Gothic ProVI M Kozuka Gothic ProVI R	Kozuka Gothic Pr6N M Kozuka Gothic Pr6N R
gothic bold	Kozuka Gothic Pro B	Kozuka Gothic ProVI B	Kozuka Gothic Pr6N B
gothic heavy (rounded gothic)	Kozuka Gothic Pro H Kozuka Gothic Pro H	Kozuka Gothic ProVI H Kozuka Gothic ProVI H	Kozuka Gothic Pr6N H Kozuka Gothic Pr6N H

Hiragino and Morisawa Settings for Hiragino fonts:

	<code>hiragino</code>	<code>hiraginon</code>
mincho medium	Hiragino Mincho Pro W3	Hiragino Mincho Pr6N W3
mincho bold	Hiragino Mincho Pro W6	Hiragino Mincho Pr6N W6
gothic medium	Hiragino Kaku Gothic Pro W3	Hiragino Kaku Gothic ProN W3
gothic bold	Hiragino Kaku Gothic Pro W6	Hiragino Kaku Gothic ProN W6
gothic heavy	Hiragino Kaku Gothic Std W8	Hiragino Kaku Gothic StdN W8
rounded gothic	Hiragino Maru Gothic Pro W4	Hiragino Maru Gothic ProN W4

Settings for Morisawa fonts:

	<code>morisawa4</code>	<code>morisawa6n</code>
mincho medium	Ryumin Pro L-KL	Ryumin Pr6N L-KL
mincho bold	Futo Min A101 Pro Bold	Futo Min A101 Pr6N Bold
gothic medium	Chu Gothic BBB Pro Med	Chu Gothic BBB Pr6N Med
gothic bold	Futo Go B101 Pro Bold	Futo Go B101 Pr6N Bold
gothic heavy	Midashi Go Pro MB31	Midashi Go Pr6N MB31
rounded gothic	Jun Pro 101	Jun Pr6N 101

Settings for single weight Next, we describe settings for using only single weight. In four settings below, we use same fonts for medium and bold (and heavy) weights. (Hence `\mcfamily\bfseries` and `\mcfamily\mdseries` yields same Japanese fonts, if `deluxe` option is also specified).

	<code>noembed</code>	<code>ipa</code>	<code>ipaex</code>	<code>ms</code>
mincho	Ryumin-Light (non-embedded)	IPAMincho	IPAexMincho	MS Mincho
gothic	GothicBBB-Medium (non-embedded)	IPAGothic	IPAexGothic	MS Gothic

Using HG fonts We can use HG fonts bundled with Microsoft Office for realizing multiple weights in Japanese fonts.

	ipa-dx	ipaex-dx	ms-dx
mincho medium	IPAMincho	IPAexMincho	MS Mincho
mincho bold	HG Mincho E		
Gothic medium			
without deluxe	IPAGothic	IPAexGothic	MS Gothic
with jis2004	IPAGothic	IPAexGothic	MS Gothic
otherwise	HG Gothic M		
gothic bold	HG Gothic E		
gothic heavy	HG Soei Kaku Gothic UB		
rounded gothic	HG Maru Gothic PRO		

Note that HG Mincho E, HG Gothic E, HG Soei Kaku Gothic UB and HG Maru Gothic PRO are internally specified by:

default by font name (HGMinchoE, etc.).

90jis by filename (hgrme.ttc, hgrge.ttc, hgrsgu.ttc, hgrsmp.ttf).

jis2004 by filename (hgrme04.ttc, hgrge04.ttc, hgrsgu04.ttc, hgrsmp04.ttf).

3.4 \CID, \UTF and macros in `otf` package

Under $\text{p}\text{L}\text{A}\text{T}\text{E}\text{X}$, `otf` package (developed by Shuzaburo Saito) is used for typesetting characters which is in Adobe-Japan1-6 CID but not in JIS X 0208. Since this package is widely used, $\text{L}\text{u}\text{a}\text{T}\text{E}\text{X}$ -ja supports some of functions in `otf` package. If you want to use these functions, load `luatexja-otf` package.

```

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

```

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

4 Changing Parameters

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

4.1 Editing the range of `J`Achars

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

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

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

If some character has been belonged to some non-zero numbered range, this will be overwritten by the new setting. For example, whole SIP belong to the range 4 in the default setting of $\text{L}\text{u}\text{a}\text{T}\text{E}\text{X}$ -ja, and if you specify the above line, then SIP will belong to the range 100 and be removed from the range 4.

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

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		

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

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

Default Setting Lua \TeX -ja predefines eight character ranges for convenience. They are determined from the following data:

- Blocks in Unicode 6.0.
- The Adobe–Japan1–UCS2 mapping between a CID Adobe-Japan1-6 and Unicode.
- The `PXbase` bundle for up \TeX by Takayuki Yato.

Now we describe these eight ranges. The 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 the `prefercjk` settings defined in `PXbase` bundle.

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

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

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

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

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

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

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

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

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

```
\ltjdefcharrange{4}{%
  "500-"10FF, "1200-"1DFF, "2440-"245F, "27C0-"28FF, "2A00-"2AFF,
  "2C00-"2E7F, "4DC0-"4DFF, "A4D0-"A82F, "A840-"ABFF, "FB50-"FE0F,
  "FE20-"FE2F, "FE70-"FEFF, "FB00-"FB4F, "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.

4.2 kanjiskip and xkanjiskip

JAglue is divided into the following three categories:

- Glues/kerns specified in JFM. If `\inhibitglue` is issued around a Japanese character, this glue will not be inserted at the place.
- The default glue which inserted between two **J**Achars (kanjiskip).
- The default glue which inserted between a **J**Achar and an **A**Lchar (**x**kanjiskip).

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

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

It may occur that JFM contains the data of ‘ideal width of **kanjiskip**’ and/or ‘ideal width of **xkanjiskip**’. To use these data from JFM, set the value of **kanjiskip** or **xkanjiskip** to `\maxdimen`.

4.3 Insertion Setting of xkanjiskip

It is not desirable that **xkanjiskip** is inserted into every boundary between **J**Achars and **A**Lchars. For example, **xkanjiskip** should not be inserted after opening parenthesis (e.g., compare ‘(あ’ and ‘(あ’). Lua_{TeX}-ja can control whether **xkanjiskip** can be inserted before/after a character, by changing `jaxspmode` for **J**Achars and `alxspmode` parameters **A**Lchars respectively.

```

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

```

The second argument `preonly` means ‘the insertion of `xkanjiskip` is allowed before this character, but not after’. the other possible values are `postonly`, `allow` and `inhibit`.

`jaxspmode` and `alxspmode` use a same table to store the parameters on the current version. Therefore, line 1 in the code above can be rewritten as follows:

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

One can use also numbers to specify these two parameters (see Subsection 6.2).

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

4.4 Shifting Baseline

To make a match between a Japanese font and an alphabetic font, sometimes shifting of the baseline of one of the pair is needed. In $\text{p}\text{T}\text{E}\text{X}$, 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, $\text{Lua}\text{T}\text{E}\text{X}\text{-ja}$ 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):

```

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

```

Part II

Reference

5 Font Metric and Japanese Font

5.1 \jfont

To load a font as a Japanese font, you must use the `\jfont` instead of `\font`, while `\jfont` admits the same syntax used in `\font`. $\text{Lua}\text{T}\text{E}\text{X}\text{-ja}$ 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           當／體／醫／區
3 \tradgt{}当／体／医／区

```

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

Table 4. Differences between JFMs shipped with LuaTeX-ja

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

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

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

The following JFMs are shipped with LuaTeX-ja:

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

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

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

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

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 **JAgglue**, and this causes incorrect spacing between two characters when both a glue/kern from the data in the font and it from JFM are present.

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

5.2 Prefix `psft`

Besides ‘`file:`’ and ‘`name:`’ prefixes, one can use ‘`psft:`’ prefix in `\jfont` (and `\font`), to specify a ‘name-only’ Japanese font which will not be embedded to PDF. Typical use of this prefix is to specify the ‘standard’ Japanese fonts, namely, ‘Ryumin-Light’ and ‘GothicBBB-Medium’. *You should not specify any font features, such as ‘+jp90’, in the definition of ‘name-only’ fonts using this ‘psft:’ prefix.*

cid key The default font defined by using `psft:` prefix is for Japanese typesetting; it is Adobe-Japan1-6 CID-keyed font. One can specify `cid` key to use other CID-keyed non-embedded fonts for Chinese or Korean typesetting.

```

1 \jfont\testJ={psft:Ryumin-Light:cid=Adobe-Japan1-6;jfm=jis} % Japanese
2 \jfont\testD={psft:Ryumin-Light:jfm=jis} % default value is Adobe-Japan1-6
3 \jfont\testC={psft:AdobeMingStd-Light:cid=Adobe-CNS1-5;jfm=jis} % Traditional Chinese
4 \jfont\testG={psft:SimSun:cid=Adobe-GB1-5;jfm=jis} % Simplified Chinese
5 \jfont\testK={psft:Batang:cid=Adobe-Korea1-2;jfm=jis} % Korean

```

Note that the code above specifies `jfm-jis.lua`, which is for Japanese fonts, as JFM for Chinese and Korean fonts.

At present, LuaTeX-ja supports only 4 values written in the sample code above. Specifying other values, e.g.,

```
\jfont\test={psft:Ryumin-Light:cid=Adobe-Japan2;jfm=jis}
```

occurs the following error:

```

1 ! Package luatexja Error: bad cid key `Adobe-Japan2'.
2
3 See the luatexja package documentation for explanation.
4 Type H <return> for immediate help.
5 <to be read again>
6
7 \par
8
9 1.78
10
11 ? h
12 I couldn't find any non-embedded font information for the CID
13 `Adobe-Japan2'. For now, I'll use `Adobe-Japan1-6'.
14 Please contact the LuaTeX-ja project team.
15 ?

```

5.3 Structure of JFM file

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

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

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

`dir=<direction>` (required)

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

`zw=<length>` (required)

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

`zh=<length>` (required)

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

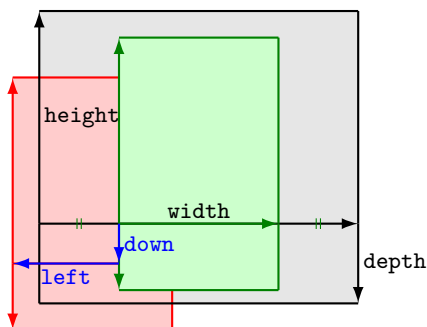
`kanjiskip={<natural>, <stretch>, <shrink>}` (optional)

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

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

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

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



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.

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

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

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

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

Specify width of characters in character class i , height, depth and the amount of italic correction. All characters in character class i are regarded that its width, height and depth are as values of these fields. But there is one exception: if 'prop' is specified in width field, width of a character becomes that of its 'real' glyph

`left=⟨length⟩, down=⟨length⟩, align=⟨align⟩`

These fields are for adjusting the position of the 'real' glyph. Legal values of `align` field are 'left', 'middle' and 'right'. If one of these 3 fields are omitted, `left` and `down` are treated as 0, and `align` field is treated as 'left'. The effects of these 3 fields are indicated in Figure 1.

In most cases, `left` and `down` fields are 0, while it is not uncommon that the `align` field is 'middle' or 'right'. For example, setting the `align` field to 'right' is practically needed when the current character class is the class for opening delimiters'.

`kern={ [j]=⟨kern⟩, [j']={⟨kern⟩, [⟨ratio⟩]}}, ...}`

`glue={ [j]={⟨width⟩, ⟨stretch⟩, ⟨shrink⟩, [⟨priority⟩], [⟨ratio⟩]}}, ...}`

`end_stretch=⟨kern⟩`

`end_shrink=⟨kern⟩`

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

- 'boxbdd' The beginning/ending of a horizontal box, and the beginning of a noindented paragraph.
- 'parbdd' The beginning of an (indented) paragraph.
- 'jcharbdd' A boundary between **J**Achar and anything else (such as **AL**char, kern, glue, ...).
- 1 The left/right boundary of an inline math formula.

Table 5. Control sequences for Japanese math fonts

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

Porting JFM from p \TeX ...

5.4 Math Font Family

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

Lua \TeX -ja's handling of Japanese fonts in math formulas is similar; Table 5 shows counterparts to \TeX 's primitives for math font families. There is no relation between the value of `\fam` and that of `\jfam`; with appropriate settings, you can set both `\fam` and `\jfam` to the same value.

5.5 Callbacks

Like Lua \TeX itself, Lua \TeX -ja also has callbacks. These callbacks can be accessed via `luatexbase.add_to_callback` function and so on, as other callbacks.

luatexja.load_jfm callback With this callback you can overwrite JFMs. This callback is called when a new JFM is loaded.

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

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

An example of this callback is the `ltjarticle` class, with forcefully assigning character class 0 to 'parbdd' in the JFM `jfm-min.lua`.

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

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

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

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

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

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

²Omega, Aleph, Lua \TeX and ϵ -(u)p \TeX can handles 256 families, but an external package is needed to support this in plain \TeX and L \TeX .

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

A good example of this and the next callbacks is the `luatexja-otf` package, supporting "AJ1-xxx" form for Adobe-Japan1 CID characters in a JFM. This callback doesn't replace any code of LuaTeX-ja.

luatexja.find_char_class callback This callback is called just when LuaTeX-ja is trying to determine which character class a character `chr_code` belongs. A function used in this callback should be in the following form:

```

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

```

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

luatexja.set_width callback This callback is called when LuaTeX-ja is trying to encapsule a **JAchar** *glyph_node*, to adjust its dimension and position.

```

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

```

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

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

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

Then, the position of glyphs is shifted up by

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

6 Parameters

6.1 \ltjsetparameter

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

`\ltjsetparameter` and `\ltjglobalsetparameter` are control sequences for assigning parameters. These take one argument which is a *(key)=(value)* list. Allowed keys are described in the next subsection. The difference between `\ltjsetparameter` and `\ltjglobalsetparameter` is only the scope of assignment; `\ltjsetparameter` does a local assignment and `\ltjglobalsetparameter` does a global one. They also obey the value of `\globaldefs`, like other assignment.

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

```

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

```

The return value of `\ltjgetparameter` is always a string. This is outputted by `tex.write()`, so any character other than space ‘ ’ (U+0020) has the category code 12 (other), while the space has 10 (space).

6.2 List of Parameters

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

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

`jcharwidowpenalty=<penalty>` [`\jcharwidowpenalty`] Penalty value for suppressing orphans. This penalty is inserted just after the last **J**Achar which is not regarded as a (Japanese) punctuation mark.

`kcatcode=<chr_code>,<natural number>` An additional attributes which each character whose character code is *<chr_code>* has. At the present version, the lowermost bit of *<natural number>* indicates whether the character is considered as a punctuation mark (see the description of `jcharwidowpenalty` above).

`prebreakpenalty=<chr_code>,<penalty>` [`\prebreakpenalty`]

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

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

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

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

`yjabaselineshift=<dimen>*`

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

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

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

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

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

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

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

Note that parameters `jaxspmode` and `alxspmode` use a common table, hence these two parameters are synonyms of each other.

`autospacing` = $\langle bool \rangle^*$ [`\autospacing`]
`autoxspacing` = $\langle bool \rangle^*$ [`\autoxspacing`]
`kanjiskip` = $\langle skip \rangle$ [`\kanjiskip`]
`xkanjiskip` = $\langle skip \rangle$ [`\xkanjiskip`]
`differentjfm` = $\langle mode \rangle^\dagger$ Specify how glues/kerns between two **J**Achars whose JFM (or size) are different. The allowed arguments are the followings:
 `average`
 `both`
 `large`
 `small`
 `pleft`
 `pright`
 `paverage`
`jacharrange` = $\langle ranges \rangle^*$
`kansujichar` = $\{ \langle digit \rangle, \langle chr_code \rangle \}$ [`\kansujichar`]

7 Other Control Sequences

7.1 Control Sequences for Compatibility

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

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

7.2 `\inhibitglue`

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

<pre> 1 \jfont\g=name:IPAMincho:jfm=test \g 2 \fbox{\hbox{あウあ\inhibitglue ウ}} 3 \inhibitglue\par\noindent あ1 4 \par\inhibitglue\noindent あ2 5 \par\noindent\inhibitglue あ3 6 \par\hrule\noindent あoff\inhibitglue ice </pre>	<table border="1" style="border-collapse: collapse; text-align: center;"> <tr> <td style="padding: 2px;">あ</td> <td style="padding: 2px;">ウ</td> <td style="padding: 2px;">あ</td> <td style="padding: 2px;">ウ</td> </tr> </table> <p> あ 1 あ 2 あ 3 <hr style="width: 100%;"/> あ office </p>	あ	ウ	あ	ウ
あ	ウ	あ	ウ		

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

- The call of `\inhibitglue` in the (internal) vertical mode is simply ignored.
- The call of `\inhibitglue` in the (restricted) horizontal mode is only effective on the spot; does not get over boundary of paragraphs. Moreover, `\inhibitglue` cancels ligatures and kernings, as shown in the last line of above example.
- The call of `\inhibitglue` in math mode is just ignored.

8 Control Sequences for L^AT_EX 2_ε

8.1 Patch for NFSS2

As described in Subsection 2.4, LuaT_EX-ja simply adopted `plfonts.dtx` in pL^AT_EX 2_ε for the Japanese patch for NFSS2. For an convenience, we will describe control sequences which are not described in Subsection 3.1.

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

In NFSS2 under LuaT_EX-ja, distinction between alphabetic font families and Japanese font families are only made by their encodings. For example, encodings OT1 and T1 are for alphabetic font families, and a Japanese font family cannot have these encodings. This command defines a new encoding scheme for Japanese font family (in horizontal direction).

```
\DeclareKanjiEncodingDefaults{<text-settings>}{<math-settings>}
```

```
\DeclareKanjiSubstitution{<encoding>}{<family>}{<series>}{<shape>}
```

```
\DeclareErrorKanjiFont{<encoding>}{<family>}{<series>}{<shape>}{<size>}
```

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

```
\reDeclareMathAlphabet{<unified-cmd>}{<al-cmd>}{<ja-cmd>}
```

```
\DeclareRelationFont{<ja-encoding>}{<ja-family>}{<ja-series>}{<ja-shape>}  
{<al-encoding>}{<al-family>}{<al-series>}{<al-shape>}
```

This command sets the ‘accompanied’ alphabetic font family (given by the latter 4 arguments) with respect to a Japanese font family given by the former 4 arguments.

```
\SetRelationFont
```

This command is almost same as `\DeclareRelationFont`, except that this command does a local assignment, where `\DeclareRelationFont` does a global assignment.

```
\userelfont
```

Change current alphabetic font encoding/family/... to the ‘accompanied’ alphabetic font family with respect to current Japanese font family, which was set by `\DeclareRelationFont` or `\SetRelationFont`. Like `\fontfamily`, `\selectfont` is required to take an effect.

```
\adjustbaseline
```

...

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

As in L^AT_EX 2_ε, this command changes current font family (alphabetic, Japanese, or both) to `<family>`. Which family will be changed is determined as follows:

- Let current encoding scheme for Japanese fonts be `<ja-enc>`. Current Japanese font family will be changed to `<family>`, if one of the following two conditions is met:
 - The family `<family>` under the encoding `<ja-enc>` has been already defined by `\DeclareKanjifamily`.
 - A font definition named `<ja-enc><family>.fd` (the file name is all lowercase) exists.
- Let current encoding scheme for alphabetic fonts be `<al-enc>`. For alphabetic font family, the criterion as above is used.
- There is a case which none of the above applies, that is, the font family named `<family>` doesn’t seem to be defined neither under the encoding `<ja-enc>`, nor under `<al-enc>`. In this case, the default family for font substitution is used for alphabetic and Japanese fonts. Note that current encoding will not be set to `<family>`, unlike the original implementation in L^AT_EX.

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

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

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

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

Figure 2. Line adjustment

9 Extensions

9.1 `luatexja-fontspec.sty`

As described in Subsection 3.2, this optional package provides the counterparts for several commands defined in the `fontspec` package. In addition to ‘font features’ in the original `fontspec`, the following ‘font features’ specifications are allowed for the commands of Japanese version:

`CID=<name>`

`JFM=<name>`

`JFM-var=<name>`

These 3 font features correspond to `cid`, `jfm` and `jfmvar` keys for `\jfont` respectively. `CID` is effective only when with `NoEmbed` described below. See Subsections 5.1 and 5.2 for details.

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

9.2 `luatexja-otf.sty`

This optional package supports typesetting characters in Adobe-Japan1. `luatexja-otf.sty` offers the following 2 low-level commands:

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

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

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

- Always treated as **J**Achars.
- Processing codes for supporting OpenType features (e.g., glyph replacement and kerning) by the `luaotfload` package is not performed to these characters.

Additional Syntax of JFM `luatexja-otf.sty` extends the syntax of `JFM`; the entries of `chars` table in `JFM` now allows a string in the form ‘AJ1-xxx’, which stands for the character whose CID number in Adobe-Japan1 is xxx.

9.3 `luatexja-adjust.sty`

...

Part III

Implementations

10 Storing Parameters

10.1 Used Dimensions, Attributes and whatsit nodes

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

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

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

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

`\ltj@zh` (dimension) A temporal register for the ‘full-height’ (usually the sum of height of imaginary body and its depth) of current Japanese font.

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

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

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

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

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

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

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

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

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

packed (2)

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

from_jfm (6) Glues/kerns from JFM.

kanji_skip (9) Glues for `kanjiskip`.

xkanji_skip (10) Glues for `xkanjiskip`.

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

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

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

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

Furthermore, LuaTeX-ja uses several ‘user-defined’ whatsit nodes for internal processing. All those nodes store a natural number (hence the node’s `type` is 100). The following `user_ids` are used:

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, and the character code is stored in the `value` field. Each node having this `user_id` is converted to a 'glyph_node' after the callback process of `luaotfload`. This `user_id` is only used by the `luatexja-otf` package.

30114 Nodes for indicating beginning of a paragraph. A paragraph which is started by `\item` in list-like environments has a horizontal box for its label before the actual contents. So ...

These whatsits will be removed during the process of inserting **JAg**lues.

10.2 Stack System of LuaTeX-ja

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

```

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

```

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

```

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

```

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

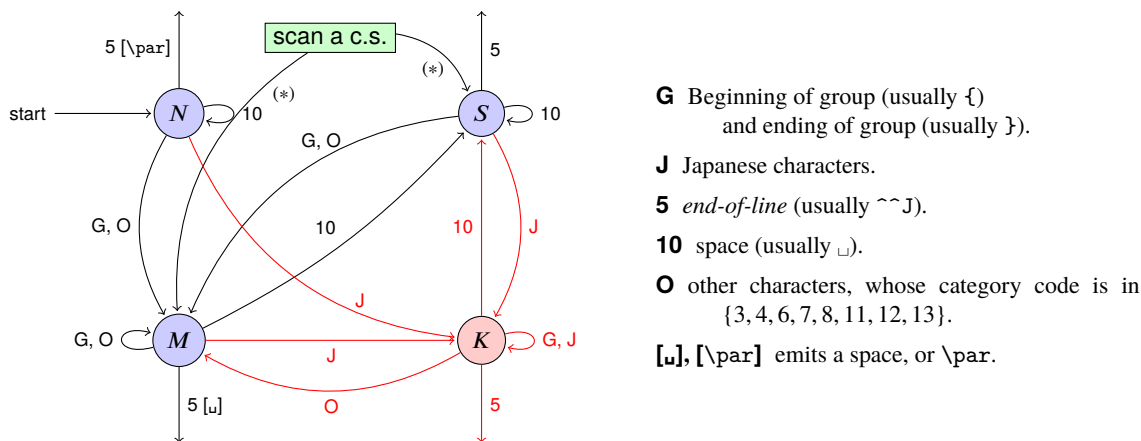
The method The code of stack system is based on that in a post of Dev-luatex mailing list³.

These are two TeX count registers for maintaining information: `\ltj@@stack` for the stack level, and `\ltj@@group@level` for the TeX's group level when the last assignment was done. Parameters are stored in one big table named `charprop_stack_table`, where `charprop_stack_table[i]` stores data of stack level *i*. If a new stack level is created by `\ltjsetparameter`, all data of the previous level is copied.

To resolve the problem mentioned in 'Background' above, LuaTeX-ja uses another thing: When a new stack level is about to be created, a whatsit node whose type, subtype and value are 44 (*user_defined*), 30112, and current group level respectively is appended to the current list (we refer this node by *stack_flag*). This enables us to know whether assignment is done just inside a hbox. Suppose that the stack level is *s* and the TeX's group level is *t* just after the hbox group, then:

- If there is no *stack_flag* node in the list of the hbox, then no assignment was occurred inside the hbox. Hence values of parameters at the end of the hbox are stored in the stack level *s*.
- If there is a *stack_flag* node whose value is *t* + 1, then an assignment was occurred just inside the hbox group. Hence values of parameters at the end of the hbox are stored in the stack level *s* + 1.

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



- We omitted about category codes 9 (*ignored*), 14 (*comment*) and 15 (*invalid*) from the above diagram. We also ignored the input like $\text{\^{\^}A}$ or $\text{\^{\^}df}$.
- When a character whose category code is 0 (*escape character*) is seen by \TeX , the input processor scans a control sequence (**scan a c.s.**). These paths are not shown in the above diagram. After that, the state is changed to State *S* (skipping blanks) in most cases, but to State *M* (middle of line) sometimes.

Figure 3. State transitions of $\text{p}\text{\TeX}$'s input processor.

- If there are *stack_flag* nodes but all of their values are more than $t + 1$, then an assignment was occurred in the box, but it is done is 'more internal' group. Hence values of parameters at the end of the hbox are stored in the stack level s .

Note that to work this trick correctly, assignments to \ltj@stack and \ltj@group@level have to be local always, regardless the value of \globaldefs . This problem is resolved by using $\text{\directlua{tex.globaldefs=0}}$ (this assignment is local).

11 Linebreak after Japanese Character

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

In $\text{p}\text{\TeX}$, a line break after a Japanese character doesn't emit a space, since words are not separated by spaces in Japanese writings. However, this feature isn't fully implemented in $\text{Lua}\text{\TeX-ja}$ due to the specification of callbacks in $\text{Lua}\text{\TeX}$. To clarify the difference between $\text{p}\text{\TeX}$ and $\text{Lua}\text{\TeX}$, We briefly describe the handling of a line break in $\text{p}\text{\TeX}$, in this subsection.

$\text{p}\text{\TeX}$'s input processor can be described in terms of a finite state automaton, as that of \TeX in Section 2.5 of [1]. The internal states are as follows:

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

The first three states—*N*, *S* and *M*—are as same as \TeX 's input processor. State *K* is similar to state *M*, and is entered after Japanese characters. The diagram of state transitions are indicated in Figure 3. Note that $\text{p}\text{\TeX}$ doesn't leave state *K* after 'beginning/ending of a group' characters.

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

States in the input processor of $\text{Lua}\text{\TeX}$ is the same as that of \TeX , and they can't be customized by any callbacks. Hence, we can only use `process_input_buffer` and `token_filter` callbacks for to suppress a space by a line break which is after Japanese characters.

However, `token_filter` callback cannot be used either, since a character in category code 5 (end-of-line) is converted into an space token *in the input processor*. So we can use only the `process_input_buffer` callback. This means that suppressing a space must be done *just before* an input line is read.

Considering these situations, handling of an end-of-line in LuaTeX-ja are as follows:

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

1. The category code of `\endlinechar`⁴ is 5 (end-of-line).
2. The category code of U+FFFFF itself is 14 (comment).
3. The input line matches the following ‘regular expression’:

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

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

```

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

```

xyzあ u

- There is no space between ‘x’ and ‘y’, since the line 2 ends with a **JA**char ‘あ’ (this ‘あ’ considered as an **JA**char 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 **AL**char (the letter ‘あ’ considered as an **AL**char at the ending of line 2).

12 Insertion of JFM glues, kanjiskip and xkanjiskip

12.1 Overview

LuaTeX-ja における **JA**glue の挿入方法は、pTeX のそれとは全く異なる。pTeX では次のような仕様であった：

- JFM グルーの挿入は、和文文字を表すトークンを元に水平リストに（文字を表す）`<char_node>` を追加する過程で行われる。
- **xkanjiskip** の挿入は、水平ボックスへのパッケージングや行分割前に行われる。
- **kanjiskip** はノードとしては挿入されない。パッケージングや行分割の計算時に「和文文字を表す 2 つの `<char_node>` の間には **kanjiskip** がある」ものとみなされる。

しかし、LuaTeX-ja では、水平ボックスへのパッケージングや行分割前に全ての **JA**glue、即ち JFM グルー・**xkanjiskip**・**kanjiskip** の 3 種類を一度に挿入することになっている。これは、LuaTeX において欧文の合字・カーニング処理がノードベースになったことに対応する変更である。

LuaTeX-ja における **JA**glue 挿入処理では、次節で定義する「クラスタ」を単位にして行われる。大雑把にいうと、「クラスタ」は文字とそれに付随するノード達（アクセント位置補正用のカーンや、イタリック補正）をまとめたものであり、2 つのクラスタの間には、ペナルティ、`\vadjust`、`whatsit` など、行組版には関係しないものがある。

12.2 definition of a ‘cluster’

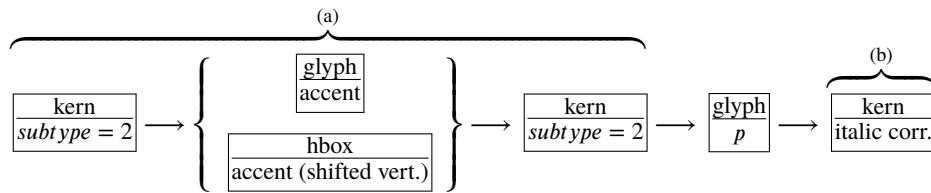
Definition 1. A *cluster* is a list of consecutive nodes in one of the following forms, with the *id* of it:

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_nodes* at the boundary of it. The *id* is `id_math`.

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

3. A *glyph_node p* with nodes which relate with it:

- (1) A kern for the italic correction of *p*.
- (2) An accent attached to *p* by `\accent`.



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*, a *vbox*, a *rule* (`\vrule`) and an *unset_node*. 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.

Let *Np*, *Nq* and *Nr* denote a cluster.

id の意味 *Np.id* の意味を述べるとともに、「先頭の文字」を表す *glyph_node Np.head* と、「最後の文字」を表す *glyph_node Np.tail* を次のように定義する。直感的に言う、*Np* は *Np.head* で始まり *Np.tail* で終わるような単語、と見做すことができる。これら *Np.head*, *Np.tail* は説明用に準備した概念であって、実際の Lua コード中にそのように書かれているわけではないことに注意。

id_jglyph 和文文字。

Np.head, *Np.tail* は、その和文文字を表している *glyph_node* そのものである。

id_glyph 和文文字を表していない *glyph_node p*。

多くの場合、*p* は欧文文字を格納しているが、「ffi」などの合字によって作られた *glyph_node* である可能性もある。前者の場合、*Np.head*, *Np.tail* = *p* である。一方、後者の場合、

- *Np.head* は、合字の構成要素の先頭→(その *glyph_node* における) 合字の構成要素の先頭→……と再帰的に検索していったどり着いた *glyph_node* である。
- *Np.last* は、同様に末尾→末尾→と検索してたどり着いた *glyph_node* である。

id_math インライン数式。

便宜的に、*Np.head*, *Np.tail* とともに「文字コード -1 の欧文文字」とおく。

id_hlist 縦方向にシフトされていない水平ボックス。

この場合、*Np.head*, *Np.tail* はそれぞれ *p* の内容を表すリストの、先頭・末尾のノードである。

- 状況によっては、 \TeX ソースで言うと

```
\hbox{\hbox{abc}...\hbox{\lower1pt\hbox{xyz}}}
```

のように、*p* の内容が別の水平ボックスで開始・終了している可能性も十分あり得る。そのような場合、*Np.head*, *Np.tail* の算出は、垂直方向にシフトされていない水平ボックスの場合だけ内部を再帰的に探索する。例えば上の例では、*Np.head* は文字「a」を表すノードであり、一方 *Np.tail* は垂直方向にシフトされた水平ボックス、`\lower1pt\hbox{xyz}` に対応するノードである。

- また、先頭にアクセント付きの文字がきたり、末尾にイタリック補正用のカーンが来ることもあり得る。この場合は、クラスタの定義のところにもあったように、それらは無視して算出を行う。
- 最初・最後のノードが合字によって作られた *glyph_node* のときは、それぞれに対して *id_glyph* と同様に再帰的に構成要素をたどっていく。

id_pbox 「既に処理された」ノードのリストであり、これらのノードが二度処理を受けないためにまとめて 1 つのクラスタとして取り扱うだけである。*id_hlist* と同じ方法で *Np.head*, *Np.tail* を算出する、

id_disc discretionary break (`\discretionary{pre}{post}{nobreak}`).

id_hlist と同じ方法で *Np.head*, *Np.tail* を算出するが、第 3 引数の *nobreak* (行分割が行われない時の内容) を使う。言い換えれば、ここで行分割が発生した時の状況は全く考慮に入れない。

id_box_like *id_hlist* とならない box や、rule.

この場合は、*Np.head*, *Np.tail* のデータは利用されないの、2 つの算出は無意味である。敢えて明示するならば、*Np.head*, *Np.tail* は共に nil 値である。

他 以上がない *id* に対しても、*Np.head*, *Np.tail* の算出は無意味。

クラスタの別の分類 さらに、JFM グルー挿入処理の実際の説明により便利なように、*id* とは別のクラスタの分類を行っておく。挿入処理では 2 つの隣り合ったクラスタの間に空白等の実際の挿入を行うことは前に書いたが、ここでの説明では、問題にしているクラスタ *Np* は「後ろ側」のクラスタであるとする。「前側」のクラスタについては、以下の説明で *head* が *last* に置き換わることに注意すること。

和文 A リスト中に直接出現している和文文字。 *id* が *id_jglyph* であるか、*id* が *id_pbox* であって *Np.head* が **J**Achar であるとき。

和文 B リスト中の水平ボックスの中身の先頭として出現した和文文字。和文 A との違いは、これの前に JFM グルーの挿入が行われない (`xkanjiskip`, `kanjiskip` は入り得る) ことである。*id* が *id_hlist* か *id_disc* であって *Np.head* が **J**Achar であるとき。

欧文 リスト中に直接／水平ボックスの中身として出現している欧文文字。次の 3 つの場合が該当：

- *id* が *id_glyph* である。
- *id* が *id_math* である。
- *id* が *id_pbox* か *id_hlist* か *id_disc* であって、*Np.head* が **AL**char。

箱 box, またはそれに類似するもの。次の 2 つが該当：

- *id* が *id_pbox* か *id_hlist* か *id_disc* であって、*Np.head* が *glyph_node* でない。
- *id* が *id_box_like* である。

12.3 段落／水平ボックスの先頭や末尾

先頭部の処理 まず、段落／水平ボックスの一番最初にあるクラスタ *Np* を探索する。水平ボックスの場合は何の問題もないが、段落の場合では以下のノード達を事前に読み飛ばしておく：

`\parindent` 由来の水平ボックス (*subtype* = 3), 及び *subtype* が 44 (*user_defined*) でないような `whatsit`。

これは、`\parindent` 由来の水平ボックスがクラスタを構成しないようにするためである。

次に、*Np* の直前に空白 *g* を必要なら挿入する：

1. この処理が働くような *Np* は和文 A である。
2. 問題のリストが字下げありの段落 (`\parindent` 由来の水平ボックスあり) の場合は、この空白 *g* は「文字コード '`parbdd`' の文字」と *Np* の間に入るグルー／カーンである。
3. そうでないとき (`noindent` で開始された段落や水平ボックス) は、*g* は「文字コード '`boxbdd`' の文字」と *Np* の間に入るグルー／カーンである。

ただし、もし *g* が `glue` であった場合、この挿入によって *Np* による行分割が新たに可能になるべきではない。そこで、以下の場合には、*g* の直前に `\penalty10000` を挿入する：

- 問題にしているリストが段落であり、かつ
- *Np* の前には予めペナルティがなく、*g* は `glue`。

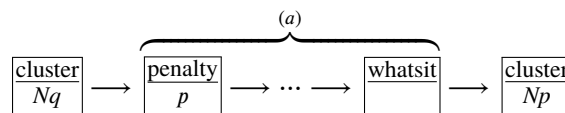
末尾の処理 末尾の処理は、問題のリストが段落のものか水平ボックスのものかによって異なる。後者の場合は容易い：最後のクラスタを Nq とおくと、 Nq と「文字コード 'boxb' の文字」の間に入るグルー／カーンを、 Nq の直後に挿入するのみである。

一方、前者（段落）の場合は、リストの末尾は常に `\penalty10000` と、`\parfillskip` 由来のグルーが存在する。よって、最後のクラスタ Np はこの `\parfillskip` 由来のグルーとなり、実質的な中身の最後はその 1 つ前のクラスタ Nq となる。

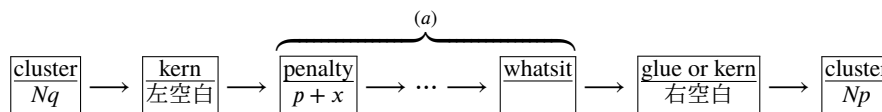
1. まず Nq の直後に（後に述べる）`line-end [E]` によって定まる空白を挿入する。
2. 次に、段落の最後の「通常の和文文字 + 句点」が独立した行となるのを防ぐために、`jcharwidowpenalty` の値の分だけ適切な場所のペナルティを増やす。
ペナルティ量を増やす場所は、`head` が **JAchar** であり、かつその文字の `kcatcode` が偶数であるような最後のクラスタの直前にあるものたちである⁵。

12.4 概観と典型例：2つの「和文 A」の場合

先に述べたように、2つの隣り合ったクラスタ、 Nq と Np の間には、ペナルティ、`\vadjust`、`whatsit` など、行組版には関係しないものがある。模式的に表すと、



のようになっている。間の (a) に相当する部分には、何のノードもない場合ももちろんあり得る。そうして、JFM グルー挿入後には、この 2 クラスタ間は次のようになる：



以後、典型的な例として、クラスタ Nq と Np が共に和文 A である場合を見ていこう、この場合が全ての場合の基本となる。

「右空白」の算出 まず、「右空白」にあたる量を算出する。通常はこれが、隣り合った 2 つの和文文字間に入る空白量となる。

JFM 由来 [M] JFM の文字クラス指定によって入る空白を以下によって求める。この段階で空白量が未定義（未指定）だった場合、デフォルト値 `kanjiskip` を採用することとなるので、次へ。

1. もし両クラスタの間で `\inhibitglue` が実行されていた場合（証として `whatsit` ノードが自動挿入される）、代わりに `kanjiskip` が挿入されることとなる。次へ。
2. Nq と Np が同じ JFM・同じ `jfmvar` キー・同じサイズの和文フォントであったならば、共通に使っている JFM 内で挿入される空白（グルーかカーン）が決まっているか調べ、決まっていればそれを採用。
3. 1. でも 2. でもない場合は、 Nq と Np が違う JFM/`jfmvar`/サイズである。この場合、まず

$$gb := (Nq \text{ と「使用フォントが } Nq \text{ のそれと同じで、} \\ \text{文字コードが } Np \text{ のその文字」との間に入るグルー／カーン})$$

$$ga := (\text{「使用フォントが } Np \text{ のそれと同じで、} \\ \text{文字コードが } Nq \text{ のその文字」と } Np \text{ との間に入るグルー／カーン})$$

として、前側の文字の JFM を使った時の空白（グルー／カーン）と、後側の文字の JFM を使った時のそれを求める。

gb, ga それぞれに対する $\langle ratio \rangle$ の値を d_b, d_a とする。

⁵大雑把に言えば、`kcatcode` が奇数であるような **JAchar** を約物として考えていることになる。`kcatcode` の最下位ビットはこの `jcharwidowpenalty` 用にも利用される。

- ga と gb の両方が未定義であるならば、JFM 由来のグルーは挿入されず、`kanjiskip` を採用することとなる。どちらか片方のみが未定義であるならば、次のステップでその未定義の方は長さ 0 の kern で、 $\langle ratio \rangle$ の値は 0 であるかのように扱われる。
- `differentjfm` の値が `pleft`, `pright`, `paverage` のとき、 $\langle ratio \rangle$ の指定に従って比例配分を行う。JFM 由来のグルー／カーン値は以下の値となる：

$$f\left(\frac{1-d_b}{2}gb + \frac{1+d_b}{2}ga, \frac{1-d_a}{2}gb + \frac{1+d_a}{2}ga\right)$$

ここで、 $f(x, y)$ は

$$f(x, y) = \begin{cases} x & \text{if } \text{differentjfm} = \text{pleft}; \\ y & \text{if } \text{differentjfm} = \text{pright}; \\ (x + y)/2 & \text{if } \text{differentjfm} = \text{paverage}; \end{cases}$$

- `differentmet` がそれ以外の値の時は、 $\langle ratio \rangle$ の値は無視され、JFM 由来のグルー／カーン値は以下の値となる：

$$f(gb, ga)$$

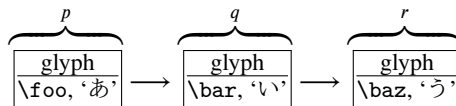
ここで、 $f(x, y)$ は

$$f(x, y) = \begin{cases} \min(x, y) & \text{if } \text{differentjfm} = \text{small}; \\ \max(x, y) & \text{if } \text{differentjfm} = \text{large}; \\ (x + y)/2 & \text{if } \text{differentjfm} = \text{average}; \\ x + y & \text{if } \text{differentjfm} = \text{both}; \end{cases}$$

例えば、

```
\font\foo=psft:Ryumin-Light:jfm=ujis
\font\bar=psft:GothicBBB-Medium:jfm=ujis
\font\baz=psft:GothicBBB-Medium:jfm=ujis;jfmvar=piyo
```

という 3 フォントを考え、



という 3 ノードを考える（それぞれ単独でクラスタをなす）。この場合、 p と q の間は、実フォントが異なるにもかかわらず (2) の状況となる一方で、 q と r の間は（実フォントが同じなのに）`jfmvar` キーの内容が異なるので (3) の状況となる。

kanjiskip [K] 上の [M] において空白が定まらなかった場合、以下で定めた量「右空白」として採用する。この段階においては、`\inhibitglue` は効力を持たないため、結果として、2 つの和文文字間には常に何らかのグルー／カーンが挿入されることとなる。

1. 両クラスタ（厳密には $Nq.tail$, $Np.head$ ）の中身の文字コードに対する `autospacing` パラメータが両方とも `false` だった場合は、長さ 0 の `glue` とする。
2. ユーザ側から見た `kanjiskip` パラメータの自然長が $\backslashmaxdimen = (2^{30} - 1)sp$ でなければ、`kanjiskip` パラメータの値を持つ `glue` を採用する。
3. 2. でない場合は、 Nq , Np で使われている JFM に指定されている `kanjiskip` の値を用いる。どちらか片方のクラスタだけが和文文字（和文 A・和文 B）のときは、そちらのクラスタで使われている JFM 由来の値だけを用いる。もし両方で使われている JFM が異なった場合は、上の [M] 3. と同様の方法を用いて調整する。

「左空白」の算出とそれに伴う補正 「左空白」は過去のバージョンでは定義していたが、このバージョンでは挿入は一切行われず（機能自体削除している）。しかし、仕様は流動的であり、将来復活する可能性もあるため、マニュアル中の記述は今のところ極力変更しない。

Table 6. Summary of JFM glues.

$Np \downarrow$	和文 A	和文 B	欧文	箱	glue	kern
和文 A	$\frac{E \quad M \rightarrow K}{PN}$	$\frac{\quad O_A \rightarrow K}{PN}$	$\frac{\quad O_A \rightarrow X}{PN}$	$\frac{\quad O_A}{PA}$	$\frac{\quad O_A}{PN}$	$\frac{\quad O_A}{PS}$
和文 B	$\frac{E \quad O_B \rightarrow K}{PA}$	$\frac{\quad K}{PS}$	$\frac{\quad X}{PS}$			
欧文	$\frac{E \quad O_B \rightarrow X}{PA}$	$\frac{\quad X}{PS}$				
箱	$\frac{E \quad O_B}{PA}$					
glue	$\frac{E \quad O_B}{PN}$					
kern	$\frac{E \quad O_B}{PS}$					

Here $\frac{E \quad M \rightarrow K}{PN}$ means that

1. To determine the ‘right-space’, LuaTeX-ja first attempts by the method ‘JFM-origin [M]’. If this attempt fails, LuaTeX-ja use the method ‘kanjiskip [K]’.
2. The ‘left space’ between Nq and Np is determined by the method ‘line-end [E]’.
3. LuaTeX-ja adopts the method ‘P-normal [PN]’ to adjust the penalty between two clusters for *kinsoku shori*.

禁則用ペナルティの挿入 ます,

$$a := (Nq^6 \text{ の文字に対する } \text{postbreakpenalty} \text{ の値}) + (Np^7 \text{ の文字に対する } \text{prebreakpenalty} \text{ の値})$$

とおく。ペナルティは通常 $[-10000, 10000]$ の整数値をとり、また ± 10000 は正負の無限大を意味することになっているが、この a の算出では単純な整数の加減算を行う。

a は禁則処理用に Nq と Np の間に加えられるべきペナルティ量である。

P-normal [PN] Nq と Np の間の (a) 部分にペナルティ (*penalty_node*) があれば処理は簡単である：それらの各ノードにおいて、ペナルティ値を (± 10000 を無限大として扱いつつ) a だけ増加させればよい。また、 $10000 + (-10000) = 0$ としている。

少々困るのは、(a) 部分にペナルティが存在していない場合である。直感的に、補正すべき量 a が 0 でないとき、その値をもつ *penalty_node* を作って「右空白」の (もし未定義なら Np の) 直前に挿入……ということになるが、実際には僅かにこれより複雑である。

- 「右空白」がカーンであるとき、それは「 Nq と Np の間で改行は許されない」ことを意図している。そのため、この場合は $a \neq 0$ であってもペナルティの挿入はしない。
- 「左空白」がカーンとしてきっちり定義されている時 (このとき、「右空白」はカーンでない)、この「左空白」の直後での行分割を許容しないとイケないので、 $a = 0$ であっても *penalty_node* を作って挿入する。
- 以上のどれでもないときは、 $a \neq 0$ ならば *penalty_node* を作って挿入する。

12.5 その他の場合

本節の内容は表 6 にまとめてある。

⁷厳密にはそれぞれ $Nq.tail$, $Np.head$.

和文 A と欧文の間 Nq が和文 A で、 Np が欧文の場合、JFM グルー挿入処理は次のようにして行われる。

- 「右空白」については、まず以下に述べる Boundary-B [O_B] により空白を決定しようと試みる。それが失敗した場合は、`xkanjiskip [X]` によって定める。
- 「左空白」については、既に述べた line-end [E] をそのまま採用する。それに伴う「右空白」の補正も同じ。
- 禁則用ペナルティも、以前述べた P-normal [PN] と同じである。

Boundary-B [O_B] 和文文字と「和文でないもの」との間に入る空白を以下によって求め、未定義でなければそれを「右空白」として採用する。JFM-origin [M] の変種と考えて良い。これによって定まる空白の典型例は、和文の閉じ括弧と欧文文字の間に入る半角アキである。

1. もし両クラスタの間で `\inhibitglue` が実行されていた場合（証として `whatsit` ノードが自動挿入される）、次へ。
2. そうでなければ、 Nq と「文字コードが `'jcharbdd'` の文字」との間に入るグルー／カーンとして定まる。

xkanjiskip [X] この段階では、`kanjiskip [K]` のときと同じように、以下で定めた量を「右空白」として採用する。この段階で `\inhibitglue` は効力を持たないのも同じである。

1. 以下のいずれかの場合は、`xkanjiskip` の挿入は抑止される。しかし、実際には行分割を許容するために、長さ 0 の `glue` を採用する：
 - 両クラスタにおいて、それらの中身の文字コードに対する `autoxspacing` パラメタが共に `false` である。
 - Nq の中身の文字コードについて、「直後への `xkanjiskip` の挿入」が禁止されている（つまり、`jaxspmode` (or `alxspmode`) パラメタが 2 以上）。
 - Np の中身の文字コードについて、「直前への `xkanjiskip` の挿入」が禁止されている（つまり、`jaxspmode` (or `alxspmode`) パラメタが偶数）。
2. ユーザ側から見た `xkanjiskip` パラメタの自然長が `\maxdimen = (230 - 1) sp` でなければ、`xkanjiskip` パラメタの値を持つ `glue` を採用する。
3. 2. でない場合は、 Nq, Np （和文 A/和文 B なのは片方だけ）で使われている JFM に指定されている `xkanjiskip` の値を用いる。

欧文と和文 A の間 Nq が欧文で、 Np が和文 A の場合、JFM グルー挿入処理は上の場合とほぼ同じである。和文 A のクラスタが逆になるので、Boundary-A [O_A] の部分が変わるだけ。

- 「右空白」については、まず以下に述べる Boundary-A [O_A] により空白を決定しようと試みる。それが失敗した場合は、`xkanjiskip [X]` によって定める。
- Nq が和文でないので、「左空白」は算出されない。
- 禁則用ペナルティは、以前述べた P-normal [PN] と同じである。

Boundary-A [O_A] 「和文でないもの」と和文文字との間に入る空白を以下によって求め、未定義でなければそれを「右空白」として採用する。JFM-origin [M] の変種と考えて良い。これによって定まる空白の典型例は、欧文文字と和文の開き括弧との間に入る半角アキである。

1. もし両クラスタの間で `\inhibitglue` が実行されていた場合（証として `whatsit` ノードが自動挿入される）、次へ。
2. そうでなければ、「文字コードが `'jcharbdd'` の文字」と Np との間に入るグルー／カーンとして定まる。

和文 A と箱・グルー・カーンの間 N_q が和文 A で、 N_p が箱・グルー・カーンのいずれかであった場合、両者の間に挿入される JFM グルーについては同じ処理である。しかし、そこでの行分割に対する仕様が異なるので、ペナルティの挿入処理は若干異なったものとなっている。

- 「右空白」については、既に述べた **Boundary-B** [O_B] により空白を決定しようと試みる。それが失敗した場合は、「右空白」は挿入されない。
- 「左空白」については、既に述べた **line-end** [E] の算出方法をそのまま採用する。それに伴う「右空白」の補正も同じ。
- 禁則用ペナルティの処理は、後ろのクラスタ N_p の種類によって異なる。なお、 $N_p.head$ は無意味であるから、「 $N_p.head$ に対する **prebreakpenalty** の値」は 0 とみなされる。言い換えれば、

$$a := (N_q^8 \text{ の文字に対する } \mathbf{postbreakpenalty} \text{ の値}).$$

箱 N_p が箱であった場合は、両クラスタの間での行分割は（明示的に両クラスタの間に `\penalty10000` があった場合を除き）いつも許容される。そのため、ペナルティ処理は、後に述べる **P-allow** [PA] が **P-normal** [PN] の代わりに用いられる。

グルー N_p がグルーの場合、ペナルティ処理は **P-normal** [PN] を用いる。

カーン N_p がカーンであった場合は、両クラスタの間での行分割は（明示的に両クラスタの間にペナルティがあった場合を除き）許容されない。ペナルティ処理は、後に述べる **P-suppress** [PS] を使う。

これらの **P-normal** [PN]、**P-allow** [PA]、**P-suppress** [PS] の違いは、 N_q と N_p の間（以前の図だと (a) の部分）にペナルティが存在しない場合にのみ存在する。

P-allow [PA] N_q と N_p の間の (a) 部分にペナルティがあれば、**P-normal** [PN] と同様に、それらの各ノードにおいてペナルティ値を a だけ増加させる。

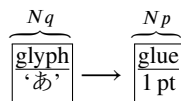
(a) 部分にペナルティが存在していない場合、**LuaTeX-já** は N_q と N_p の間の行分割を可能にしようとする。そのために、以下の場合に a をもつ *penalty_node* を作って「右空白」の（もし未定義なら N_p の）直前に挿入する：

- 「右空白」がグルーでない（カーンか未定義）であるとき。
- 「左空白」がカーンとしてきっちり定義されている時。

P-suppress [PS] N_q と N_p の間の (a) 部分にペナルティがあれば、**P-normal** [PN] と同様に、それらの各ノードにおいてペナルティ値を a だけ増加させる。

(a) 部分にペナルティが存在していない場合、 N_q と N_p の間の行分割は元々不可能のはずだったのであるが、**LuaTeX-já** はそれをわざわざ行分割可能にはしない。そのため、「右空白」が **glue** であれば、その直前に `\penalty10000` を挿入する。

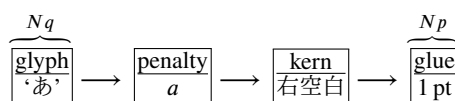
なお、「右空白」はカーン、「左空白」は未定義の



のような状況を考える。このとき、 a 、即ち「あ」の **postbreakpenalty** がいかなる値であっても、この 2 クラスタ間は最終的に



となり、 a 分のペナルティは挿入されないことに注意して欲しい。**postbreakpenalty** は (a は) 殆どの場合が非負の値と考えられ、そのような場合では (1) と



との間に差異は生じない⁹。

⁹`kern→glue` が 1 つの行分割可能点（行分割に伴うペナルティは 0）であるため、たとえ $a = 10000$ であっても、 N_q と N_p の間で行分割を禁止することはできない。

箱・グルー・カーンと和文 A の間 N_p が箱・グルー・カーンのいずれかで、 N_p が和文 A であった場合は、すぐ上の (N_q と N_p の順序が逆になっている) 場合とほぼ同じであるが、「左空白」がなくなることにのみ注意。

- 「右空白」については、既に述べた Boundary-A [O_A] により空白を決定しようと試みる。それが失敗した場合は、「右空白」は挿入されない。
- N_q が和文でないので、「左空白」は算出されない。
- 禁則用ペナルティの処理は、 N_q の種類によって異なる。 $N_q.tail$ は無意味なので、

$$a := (N_p^{10} \text{ の文字に対する } \text{prebreakpenalty} \text{ の値}).$$

箱 N_q が箱の場合は、P-allow [PA] を用いる。

グルー N_q がグルーの場合は、P-normal [PN] を用いる。

カーン N_q がカーンの場合は、P-suppress [PS] を用いる。

和文 A と和文 B の違い 先に述べたように、和文 B は水平ボックスの中身の先頭 (or 末尾) として出現している和文文字である。リスト内に直接ノードとして現れている和文文字 (和文 A) との違いは、

- 和文 B に対しては、JFM の文字クラス指定から定まる空白 JFM-origin [M], Boundary-A [O_A], Boundary-B [O_B] の挿入は行われない。「左空白」の算出も行われない。例えば、
 - 片方が和文 A, もう片方が和文 B のクラスタの場合、Boundary-A [O_A] または Boundary-B [O_B] の挿入を試み、それがダメなら kanjiskip [K] の挿入を行う。
 - 和文 B の 2 つのクラスタの間には、kanjiskip [K] が自動的に入る。
- 和文 B と箱・グルー・カーンが隣接したとき (どちらが前かは関係ない)、間に JFM グルー・ペナルティの挿入は一切しない。
- 和文 B と和文 B, また和文 B と欧文とが隣接した時は、禁則用ペナルティ挿入処理は P-suppress [PS] が用いられる。
- 和文 B の文字に対する prebreakpenalty, postbreakpenalty の値は使われず、0 として計算される。

次が具体例である：

1 あ. \inhibitglue A\\	あ. A
2 \hbox{あ. }A\\	あ. A
3 あ. A	あ. A

- 1 行目の \inhibitglue は Boundary-B [O_B] の処理のみを抑止するので、ピリオドと「A」の間には xkanjiskip (四分アキ) が入ることに注意。
- 2 行目のピリオドと「A」の間においては、前者が和文 B となる (水平ボックスの中身の末尾として登場しているから) ので、そもそも Boundary-B [O_B] の処理は行われない。よって、xkanjiskip が入ることとなる。
- 3 行目では、ピリオドの属するクラスタは和文 A である。これによって、ピリオドと「A」の間には Boundary-B [O_B] 由来の半角アキが入ることになる。

13 psft

...

14 Patch for the `listings` package

It is well-known that the `listings` package outputs weird results for Japanese input. The `listings` package makes most of letters active and assigns output command for each letter [2]. But Japanese characters are not included in these activated letters. For \TeX series, there is no method to make Japanese characters active; a patch `jlisting.sty` [3] resolves the problem forcibly.

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

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

Class of characters Roughly speaking, the `listings` package processes input as follows:

1. Collects *letters* and *digits*, which can be used for the name of identifiers.
2. When reading an *other*, outputs the collected character string (with modification, if needed).
3. Collects *others*.
4. When reading a *letter* or a *digit*, outputs the collected character string.
5. Turns back to 1.

By the above process, line breaks inside of an identifier are blocked. A flag `\lst@ifletter` indicates whether the previous character can be used for the name of identifiers or not.

For Japanese characters, line breaks are permitted on both sides except for parentheses, dashes, etc. To process Japanese characters, The patch `lltjp-listings.sty` introduces a new flag `\lst@ifkanji`, which indicates whether the previous character is Japanese character or not. For illustration, we introduce the following classes of character:

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

Note that *digits* in the `listings` package can be Letter or Other according to circumstances.

For example, let us consider the case an Open comes after a Letter. Since an Open represents Japanese open parenthesis, it is preferred to be permitted to insert line break after the Letter. Therefore, the collected character string is output in this case.

The following table summarizes $5 \times 5 = 25$ cases:

		Next				
		Letter	Other	Kanji	Open	Close
Prev	Letter	collects	_____	outputs	_____	collects
	Other	outputs	collects	_____	outputs	_____
	Kanji	_____	_____	outputs	_____	collects
	Open	_____	_____	collects	_____	_____
	Close	_____	_____	outputs	_____	collects

In the above table,

- “outputs” means to output the collected character string (i.e., line breaking is permitted there).
- “collects” means to append the next character to the collected character string (i.e., line breaking is prohibited there).

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

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

The width of halfwidth kana (U+FF61–U+FF9F) is same as the width of **ALchar**; the width of the other **JChars** is double the width of **ALchar**.

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

15 Advanced line-adjustment for Japanese characters

...

References

- [1] Victor Eijkhout, *T_EX by Topic, A T_EXnician's Reference*, Addison-Wesley, 1992.
- [2] C. Heinz, B. Moses. The Listings Package.
- [3] Thor Watanabe. Listings - MyTeXpert. <http://mytexpert.sourceforge.jp/index.php?Listings>
- [4] 乙部 巖己, min10 フォントについて. <http://argent.shinshu-u.ac.jp/~otobe/tex/files/min10.pdf>
- [5] W3C Japanese Layout Task Force (ed), Requirements for Japanese Text Layout (W3C Working Group Note), 2011, 2012. <http://www.w3.org/TR/jlreq/>
- [6] 日本工業規格 (Japanese Industrial Standard) JIS X 4051, 日本語文書の組版方法 (Formatting rules for Japanese documents), 1993, 1995, 2004.

A The category code of non-kanji characters defined in JIS X 0213

In these tables, the default catcode (Lua \TeX -ja) and kcatcode ((u)p \TeX) of non-kanji characters defined in JIS X 0213 from row 1 to row 13 is summarized. Each character is printed as follows:



The tables are generated by using `\jis` command for characters included in JIS X 0208. Each character in the tables means:

- The background of a character regarded as **ALchar** in Lua \TeX -ja is colored light blue.
- The first letter L means that the character is available for the name of a control sequence in X \TeX and Lua \TeX -ja (its catcode is 11).
- The second letter U means that the character is available for the name of a control sequence in up \TeX (its kcatcode is 16 or 17). up \TeX regards these characters as Japanese character.
- The third letter P means that the character is available for the name of a control sequence in p \TeX (its kcatcode is 16 or 17).
- If the third letter is - (or the character is printed in red), the character is not included in JIS X 0208. Therefore, you can consider the character is not available in p \TeX .
- The kana for bidakuon in row 4 and 5 are omitted.

Row 1

	"0	"1	"2	"3	"4	"5	"6	"7	"8	"9	"A	"B	"C	"D	"E	"F
"2x	\square	\square _L	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square
"3x	\square	\square _L	\square _L	\square _{LU}	\square _{LU}	\square _{LU}	\square _{LU}	\square _L	\square _{LU}	\square _L	\square _L	\square _L	\square _{LU}	\square	\square	\square _L
"4x	\square _L	\square	\square	\square _L	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square
"5x	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square
"6x	\square	\square _L	\square	\square _L	\square _L	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square
"7x	\square	\square	\square	\square	\square _L	\square _L	\square _L	\square _L	\square	\square	\square	\square	\square	\square	\square	\square

Row 2

	"0	"1	"2	"3	"4	"5	"6	"7	"8	"9	"A	"B	"C	"D	"E	"F
"2x	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square
"3x	\square _L	\square _L	\square _L	\square _L	\square _L	\square _L	\square _L	\square _{LU}	\square _{LU}	\square _{LU}	\square	\square	\square	\square	\square	\square
"4x	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square
"5x	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square
"6x	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square
"7x	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square	\square

Row 3

	"0	"1	"2	"3	"4	"5	"6	"7	"8	"9	"A	"B	"C	"D	"E	"F
"2x		▶	- ▶	◀	- ◀	↗	- ↘	↖	- ↙	↻	- ↻	↑	- ↓	↗	- ↘	
"3x	0	1	2	3	4	5	6	7	8	9	⊙	- ⊙	∞	- ∞	∞	- ∞
"4x	•	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
"5x	P	Q	R	S	T	U	V	W	X	Y	Z	〒	- 〒	℥	- ℥	℥
"6x	0	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
"7x	p	q	r	s	t	u	v	w	x	y	z	=	u-	-	#	-

Row 4

	"0	"1	"2	"3	"4	"5	"6	"7	"8	"9	"A	"B	"C	"D	"E	"F
"2x		あ	い	う	え	お	か	が	き	ぎ	く					
"3x	ぐ	け	げ	こ	ご	さ	ざ	し	じ	す	ず	せ	ぜ	そ	ぞ	た
"4x	だ	ち	ぢ	つ	づ	て	で	と	ど	な	に	ぬ	ね	の	は	
"5x	ば	ぱ	ひ	び	び	ふ	ぶ	ふ	へ	べ	べ	ほ	ぼ	ぼ	ま	み
"6x	む	め	も	や	や	ゆ	ゆ	よ	よ	ら	り	る	れ	ろ	わ	わ
"7x	ゐ	ゑ	を	ん	う	か	け									

Row 5

	"0	"1	"2	"3	"4	"5	"6	"7	"8	"9	"A	"B	"C	"D	"E	"F
"2x		ア	イ	ウ	エ	オ	カ	ガ	キ	ギ	ク					
"3x	グ	ケ	ゲ	コ	ゴ	サ	ザ	シ	ジ	ス	ズ	セ	ゼ	ソ	ゾ	タ
"4x	ダ	チ	ヂ	ツ	ヅ	テ	デ	ト	ド	ナ	ニ	ヌ	ネ	ノ	ハ	
"5x	バ	パ	ヒ	ビ	ビ	フ	ブ	フ	ヘ	ベ	ベ	ホ	ボ	ボ	マ	ミ
"6x	ム	メ	モ	ヤ	ヤ	ユ	ユ	ヨ	ヨ	ラ	リ	ル	レ	ロ	ワ	ワ
"7x	ヰ	ヱ	ヲ	ン	ウ	カ	ケ									

Row 6

	"0	"1	"2	"3	"4	"5	"6	"7	"8	"9	"A	"B	"C	"D	"E	"F
"2x		A	B	Γ	Δ	E	Z	H	Θ	I	K	Λ	M	N	Ξ	O
"3x	Π	P	Σ	T	Υ	Φ	X	Ψ	Ω	♠	- ♠	♦	- ♦	♥	- ♥	♣
"4x	♠	α	β	γ	δ	ε	ζ	η	θ	ι	κ	λ	μ	ν	ξ	ο
"5x	π	ρ	σ	τ	υ	φ	χ	ψ	ω	ς	①	- ②	③	- ④	⑤	- ⑥
"6x	⑦	⑧	⑨	⑩	⏏	⏏	☎	☀	☀	☂	☂	☂	☂	☂	☂	☂
"7x	ス	ト	ヌ	ハ	ヒ	フ	ヘ	ホ	フ	ム	ラ	リ	ル	レ	ロ	

Row 7

	"0	"1	"2	"3	"4	"5	"6	"7	"8	"9	"A	"B	"C	"D	"E	"F
"2x		A _L	B _L	B _L	Г _L	Д _L	Е _L	Ё _L	Ж _L	З _L	И _L	Й _L	К _L	Л _L	М _L	Н _L
"3x	О _L	П _L	Р _L	С _L	Т _L	У _L	Ф _L	Х _L	Ц _L	Ч _L	Ш _L	Щ _L	Ъ _L	Ы _L	Ь _L	Э _L
"4x	Ю _L	Я _L	Г _L	Г _L	Ф _L	Ф _L	Ф _L	Ф _L	Ф _L	Ф _L	Ф _L	Ф _L	Ф _L	Ф _L	Ф _L	Ф _L
"5x	Г _L	a _L	б _L	в _L	г _L	д _L	е _L	ё _L	ж _L	з _L	и _L	й _L	к _L	л _L	м _L	н _L
"6x	о _L	п _L	р _L	с _L	т _L	у _L	ф _L	х _L	ц _L	ч _L	ш _L	щ _L	ъ _L	ы _L	ь _L	э _L
"7x	ю _L	я _L	Г _L	Г _L	Г _L	Г _L	Г _L	Г _L	Г _L	Г _L	Г _L	Г _L	Г _L	Г _L	Г _L	Г _L

Row 8

	"0	"1	"2	"3	"4	"5	"6	"7	"8	"9	"A	"B	"C	"D	"E	"F
"2x		Г _L	Г _L	Г _L	Г _L	Г _L	Г _L	Г _L	Г _L	Г _L	Г _L	Г _L	Г _L	Г _L	Г _L	Г _L
"3x	Г _L	Г _L	Г _L	Г _L	Г _L	Г _L	Г _L	Г _L	Г _L	Г _L	Г _L	Г _L	Г _L	Г _L	Г _L	Г _L
"4x	Г _L	21 _L	22 _L	23 _L	24 _L	25 _L	26 _L	27 _L	28 _L	29 _L	30 _L	31 _L	32 _L	33 _L	34 _L	35 _L
"5x	36 _L	37 _L	38 _L	39 _L	40 _L	41 _L	42 _L	43 _L	44 _L	45 _L	46 _L	47 _L	48 _L	49 _L	50 _L	
"6x											!!	??	??	??	??	??
"7x	ă _L	ĭ _L	Ĭ _L	ĭ _L	Ń _L	ñ _L	Ō _L	ō _L	ŭ _L	ū _L	û _L	ü _L	Û _L	Û _L		

Row 9

	"0	"1	"2	"3	"4	"5	"6	"7	"8	"9	"A	"B	"C	"D	"E	"F
"2x	€		i	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı
"3x	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı
"4x	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı
"5x	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı
"6x	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı
"7x	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı

Row 10

	"0	"1	"2	"3	"4	"5	"6	"7	"8	"9	"A	"B	"C	"D	"E	"F
"2x	A _L	~	Ł _L	Ł _L	Ś _L	Ś _L	Ş _L	Ť _L	Ž _L	Ž _L	Ž _L	a _L	ı	ı	ı	ı
"3x	s _L	~	š _L	ş _L	ť _L	ž _L	~	ž _L	ž _L	ž _L	Ř _L	Ř _L	Ř _L	Ř _L	Ř _L	Ř _L
"4x	Đ _L	Ń _L	ň _L	ő _L	ř _L	ű _L	ű _L	ť _L	ř _L	ř _L	ř _L	ř _L	ř _L	ř _L	ř _L	ř _L
"5x	d _L	ň _L	ň _L	ő _L	ř _L	ű _L	ű _L	ť _L	ř _L	ř _L	ř _L	ř _L	ř _L	ř _L	ř _L	ř _L
"6x	g _L	h _L	j _L	s _L	u _L	u _L	r _L	j _L	z _L	t _L	b _L	j _L	t _L	d _L	n _L	
"7x	r _L	s _L	z _L	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı

Row 11

	"0	"1	"2	"3	"4	"5	"6	"7	"8	"9	"A	"B	"C	"D	"E	"F
"2x	?	Ħ	Ŧ	đ	đ	đ	đ	đ	đ	đ	đ	đ	đ	đ	đ	đ
"3x	ə	ə	ə	ə	ə	ə	ə	ə	ə	ə	ə	ə	ə	ə	ə	ə
"4x	z	z	z	z	z	z	z	z	z	z	z	z	z	z	z	z
"5x	é	é	é	é	é	é	é	é	é	é	é	é	é	é	é	é
"6x	l	l	l	l	l	l	l	l	l	l	l	l	l	l	l	l
"7x	l	l	l	l	l	l	l	l	l	l	l	l	l	l	l	l

Row 12

	"0	"1	"2	"3	"4	"5	"6	"7	"8	"9	"A	"B	"C	"D	"E	"F
"2x	①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩	⑪	⑫	⑬	⑭	⑮	
"3x	⑯	⑰	⑱	⑲	⑳	i	ii	iii	iv	v	vi	vii	viii	ix	x	xi
"4x	xii	Ⓐ	Ⓑ	Ⓒ	Ⓓ	Ⓔ	Ⓕ	Ⓖ	Ⓗ	Ⓘ	⓫	⓬	Ⓜ	Ⓨ	Ⓩ	ⓐ
"5x	ⓑ	ⓒ	ⓓ	ⓔ	ⓕ	ⓖ	ⓗ	ⓘ	ⓙ	ⓚ	ⓛ	ⓜ	ⓝ	ⓞ	ⓟ	ⓠ
"6x	ⓡ	ⓢ	ⓣ	ⓤ	ⓥ	ⓦ	ⓧ	ⓨ	ⓩ	⓪	⓫	⓬	⓭	⓮	⓯	⓰
"7x	⓱	⓲	⓳	⓴										*	**	

Row 13

	"0	"1	"2	"3	"4	"5	"6	"7	"8	"9	"A	"B	"C	"D	"E	"F
"2x	①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩	⑪	⑫	⑬	⑭	⑮	
"3x	⑯	⑰	⑱	⑲	⑳	I	II	III	IV	V	VI	VII	VIII	IX	X	XI
"4x	ミリ	キロ	キ	メ	グ	ト	ル	メ	メ	リ	フ	担	ル	フ	バ	認
"5x	mm	cm	km	mg	kg	cc	m	XII								職
"6x	シ	シ	No.	KK	TEL	⓪	⓫	⓬	⓭	⓮	⓯	⓰	⓱	⓲	⓳	⓴
"7x			f											◆	♣	

B Package versions used in this document

This document was typeset using the following packages:

geometry.sty	2010/09/12 v5.6 Page Geometry
keyval.sty	1999/03/16 v1.13 key=value parser (DPC)
ifpdf.sty	2011/01/30 v2.3 Provides the ifpdf switch (HO)
ifvtex.sty	2010/03/01 v1.5 Detect VTeX and its facilities (HO)
ifxetex.sty	2010/09/12 v0.6 Provides ifxetex conditional
luatexja-adjust.sty	2012/10/01 v0.1
luatexja.sty	2012/04/20 v0.2
luatexja-core.sty	2012/04/20 v0.2
luaotfload.sty	2012/05/28 v1.27 OpenType layout system
luatexbase.sty	2010/10/06 v0.3 Module utilities for LuaTeX
ifluatex.sty	2010/03/01 v1.3 Provides the ifluatex switch (HO)
luatexbase-compat.sty	2010/10/10 v0.3 Compatibility tools for LuaTeX
luatexbase-loader.sty	2010/10/10 v0.3 Lua module loader for LuaTeX

luatexbase-regs.sty	2010/10/10 v0.3 Registers allocation for LuaTeX
etex.sty	1998/03/26 v2.0 eTeX basic definition package (PEB)
luatexbase-attr.sty	2011/05/21 v0.31 Attributes allocation for LuaTeX
luatexbase-cctb.sty	2010/10/10 v0.3 Catcodetable allocation for LuaTeX
luatexbase-mcb.sty	2010/10/10 v0.3 Callback management for LuaTeX
luatexbase-modutils.sty	2010/10/10 v0.3 Module utilities for LuaTeX
infwarerr.sty	2010/04/08 v1.3 Providing info/warning/error messages (HO)
ltxcmds.sty	2011/11/09 v1.22 LaTeX kernel commands for general use (HO)
pdftexcmds.sty	2011/11/29 v0.20 Utility functions of pdfTeX for LuaTeX (HO)
luatex-loader.sty	2010/03/09 v0.4 Lua module loader (HO)
xkeyval.sty	2012/10/14 v2.6b package option processing (HA)
ltj-cctbreg.sty	2012/04/21 v0.2
luatex.sty	2010/03/09 v0.4 LuaTeX basic definition package (HO)
ltj-base.sty	2012/04/21 v0.2
ltj-latex.sty	2012/04/21 LuaLaTeX-ja
lltjfont.sty	2013/01/01 Patch to NFSS2 for LuaLaTeX-ja
lltjdefs.sty	2011/11/22 Default font settings for LuaLaTeX-ja
lltjcore.sty	2011/11/22 Patch to LaTeX2e Kernel for LuaLaTeX-ja
luatexja-compat.sty	2011/04/01 v0.1
expl3.sty	2012/12/21 v4390 L3 Experimental code bundle wrapper
l3names.sty	2012/12/07 v4346 L3 Namespace for primitives
l3bootstrap.sty	2012/07/16 v3991 L3 Experimental bootstrap code
l3basics.sty	2012/11/24 v4339 L3 Basic definitions
l3expan.sty	2012/08/28 v4149 L3 Argument expansion
l3tl.sty	2012/11/24 v4339 L3 Token lists
l3seq.sty	2012/11/24 v4339 L3 Sequences and stacks
l3int.sty	2012/09/26 v4237 L3 Integers
l3quark.sty	2012/11/04 v4268 L3 Quarks
l3prg.sty	2012/11/24 v4339 L3 Control structures
l3clist.sty	2012/11/24 v4339 L3 Comma separated lists
l3token.sty	2012/12/20 v4384 L3 Experimental token manipulation
l3prop.sty	2012/09/09 v4212 L3 Property lists
l3msg.sty	2012/09/09 v4212 L3 Messages
l3file.sty	2012/12/20 v4377 L3 File and I/O operations
l3skip.sty	2012/11/04 v4260 L3 Dimensions and skips
l3keys.sty	2012/11/02 v4256 L3 Experimental key-value interfaces
l3fp.sty	2012/11/10 v4305 L3 Floating points
l3box.sty	2012/12/08 v4347 L3 Experimental boxes
l3coffins.sty	2012/09/09 v4212 L3 Coffin code layer
l3color.sty	2012/08/29 v4156 L3 Experimental color support
l3luatex.sty	2012/08/03 v4049 L3 Experimental LuaTeX-specific functions
l3candidates.sty	2012/12/20 v4383 L3 Experimental additions to l3kernel
amsmath.sty	2000/07/18 v2.13 AMS math features
amstext.sty	2000/06/29 v2.01
amsgen.sty	1999/11/30 v2.0
amsbsy.sty	1999/11/29 v1.2d
amsopn.sty	1999/12/14 v2.01 operator names
tikz.sty	2010/10/13 v2.10 (rcs-revision 1.76)
pgf.sty	2008/01/15 v2.10 (rcs-revision 1.12)
pgfrcs.sty	2010/10/25 v2.10 (rcs-revision 1.24)
everyshi.sty	2001/05/15 v3.00 EveryShipout Package (MS)
pgfcore.sty	2010/04/11 v2.10 (rcs-revision 1.7)
graphicx.sty	1999/02/16 v1.0f Enhanced LaTeX Graphics (DPC,SPQR)
graphics.sty	2009/02/05 v1.0o Standard LaTeX Graphics (DPC,SPQR)
trig.sty	1999/03/16 v1.09 sin cos tan (DPC)
pgfsys.sty	2010/06/30 v2.10 (rcs-revision 1.37)
xcolor.sty	2007/01/21 v2.11 LaTeX color extensions (UK)

pgfcomp-version-0-65.sty	2007/07/03 v2.10 (rcs-revision 1.7)
pgfcomp-version-1-18.sty	2007/07/23 v2.10 (rcs-revision 1.1)
pgffor.sty	2010/03/23 v2.10 (rcs-revision 1.18)
pgfkeys.sty	
pict2e.sty	2011/04/05 v0.2y Improved picture commands (HjG,RN,JT)
multienum.sty	
float.sty	2001/11/08 v1.3d Float enhancements (AL)
booktabs.sty	2005/04/14 v1.61803 publication quality tables
multicol.sty	2011/06/27 v1.7a multicolumn formatting (FMi)
listings.sty	2007/02/22 1.4 (Carsten Heinz)
lstmisc.sty	2007/02/22 1.4 (Carsten Heinz)
showexpl.sty	2012/09/22 v0.3j Typesetting example code (RN)
calc.sty	2007/08/22 v4.3 Infix arithmetic (KKT,FJ)
ifthen.sty	2001/05/26 v1.1c Standard LaTeX ifthen package (DPC)
varwidth.sty	2009/03/30 ver 0.92; Variable-width minipages
hyperref.sty	2012/11/06 v6.83m Hypertext links for LaTeX
hobsub-hyperref.sty	2012/05/28 v1.13 Bundle oberdiek, subset hyperref (HO)
hobsub-generic.sty	2012/05/28 v1.13 Bundle oberdiek, subset generic (HO)
hobsub.sty	2012/05/28 v1.13 Construct package bundles (HO)
intcalc.sty	2007/09/27 v1.1 Expandable calculations with integers (HO)
etexcmds.sty	2011/02/16 v1.5 Avoid name clashes with e-TeX commands (HO)
kvsetkeys.sty	2012/04/25 v1.16 Key value parser (HO)
kvdefinekeys.sty	2011/04/07 v1.3 Define keys (HO)
pdfescape.sty	2011/11/25 v1.13 Implements pdfTeX's escape features (HO)
bigintcalc.sty	2012/04/08 v1.3 Expandable calculations on big integers (HO)
bitset.sty	2011/01/30 v1.1 Handle bit-vector datatype (HO)
uniquecounter.sty	2011/01/30 v1.2 Provide unlimited unique counter (HO)
letltxmacro.sty	2010/09/02 v1.4 Let assignment for LaTeX macros (HO)
hopatch.sty	2012/05/28 v1.2 Wrapper for package hooks (HO)
xcolor-patch.sty	2011/01/30 xcolor patch
atveryend.sty	2011/06/30 v1.8 Hooks at the very end of document (HO)
atbegshi.sty	2011/10/05 v1.16 At begin shipout hook (HO)
refcount.sty	2011/10/16 v3.4 Data extraction from label references (HO)
hycolor.sty	2011/01/30 v1.7 Color options for hyperref/bookmark (HO)
auxhook.sty	2011/03/04 v1.3 Hooks for auxiliary files (HO)
kvoptions.sty	2011/06/30 v3.11 Key value format for package options (HO)
url.sty	2006/04/12 ver 3.3 Verb mode for urls, etc.
rerunfilecheck.sty	2011/04/15 v1.7 Rerun checks for auxiliary files (HO)
amsthm.sty	2004/08/06 v2.20
luatexja-otf.sty	2012/04/20 v0.2
luatexja-ajmacros.sty	2012/05/08 v0.1a
luatexja-preset.sty	2012/09/17 v0.1
luatexja-fontspec.sty	2012/09/17 v0.2a
fontspec.sty	2013/02/25 v2.3 Font selection for XeLaTeX and LuaLaTeX
xparse.sty	2012/12/21 v4390 L3 Experimental document command parser
fontspec-patches.sty	2013/02/25 v2.3 Font selection for XeLaTeX and LuaLaTeX
fixltx2e.sty	2006/09/13 v1.1m fixes to LaTeX
fontspec-luatex.sty	2013/02/25 v2.3 Font selection for XeLaTeX and LuaLaTeX
fontenc.sty	
xunicode.sty	2011/09/09 v0.981 provides access to latin accents and many other characters in Unicode lower plane
unicode-math.sty	2013/02/25 v0.7c Unicode maths in XeLaTeX and LuaLaTeX
l3keys2e.sty	2012/12/21 v4390 LaTeX2e option processing using LaTeX3 keys
catchfile.sty	2011/03/01 v1.6 Catch the contents of a file (HO)
fix-cm.sty	2006/09/13 v1.1m fixes to LaTeX
filehook.sty	2011/10/12 v0.5d Hooks for input files
unicode-math-luatex.sty	

lualatex-math.sty	2012/10/13 v1.1 Patches for mathematics typesetting with LuaLaTeX
etoolbox.sty	2011/01/03 v2.1 e-TeX tools for LaTeX
metalogo.sty	2010/05/29 v0.12 Extended TeX logo macros
lltjp-xunicode.sty	2012/04/18 Patch to xunicode for LuaLaTeX-ja
lltjp-unicode-math.sty	2011/11/22 Patch to unicode-math for LuaLaTeX-ja
lltjp-listings.sty	2012/09/22 0.6
epstopdf-base.sty	2010/02/09 v2.5 Base part for package epstopdf
grfext.sty	2010/08/19 v1.1 Manage graphics extensions (HO)
nameref.sty	2012/10/27 v2.43 Cross-referencing by name of section
getttitlestring.sty	2010/12/03 v1.4 Cleanup title references (HO)