The Lua\TeX\-ja package

The Lua\TeX\-ja project team

20220311.0 (March 11, 2022)
Contents

I User’s manual 4

1 Introduction 4
  1.1 Backgrounds .......................................................... 4
  1.2 Major changes from p\TeX{} ........................................ 4
  1.3 Notations .............................................................. 5
  1.4 About the project .................................................... 6

2 Getting Started 7
  2.1 Installation ............................................................ 7
  2.2 Cautions ............................................................... 8
  2.3 Using in plain \TeX{} ................................................. 8
  2.4 Using in E\TeX{} .................................................... 9

3 Changing Fonts 10
  3.1 plain \TeX{} and E\TeX{} 2\greek{2} ................................. 10
  3.2 luatexja-fontspec package ......................................... 11
  3.3 Presets of Japanese fonts ........................................ 12
  3.4 \CID, \UTF, and macros in japanese-otf package ................ 12

4 Changing Internal Parameters 12
  4.1 Range of JAchars .................................................. 13
  4.2 \kanjiskip and x\kanjiskip ....................................... 15
  4.3 Insertion setting of x\kanjiskip ................................ 16
  4.4 Shifting the baseline ............................................ 16
  4.5 kinsoku parameters and OpenType features ..................... 17

II Reference 18

5 \catcode in Lua\TeX{}-ja 18
  5.1 Preliminaries: \kcatcode in p\TeX{} and up\TeX{} ................ 18
  5.2 Case of Lua\TeX{}-ja ............................................ 18
  5.3 Non-kanji characters in a control word ......................... 18

6 Directions 18
  6.1 Boxes in different direction .................................... 19
  6.2 Getting current direction ....................................... 20

7 Redefined primitives by Lua\TeX{}-ja 21
  7.1 Suppressing redefinitions ....................................... 22

8 Font Metric and Japanese Font 22
  8.1 \jfont .............................................................. 22
  8.2 \tfont ............................................................. 26
  8.3 Default Japanese fonts and JFMs ................................ 26
  8.4 Prefix \psft ........................................................ 27
  8.5 Structure of a JFM file .......................................... 28
  8.6 Math font family ................................................ 31
  8.7 Callbacks .......................................................... 32
9 Parameters 33
  9.1 \ltjsetparameter
  9.2 \ltjgetparameter
  9.3 Alternative Commands to \ltjsetparameter 36

10 Other Commands for plain \TeX and \LaTeX\textastisk 37
  10.1 Commands for compatibility with p\TeX 37
  10.2 \inhibitglue, \disinhibitglue 37
  10.3 \ltjfakeboxleft, \ltjfakeparbegin 38
  10.4 \insertxkanjiskip, \insertkanjiskip 38
  10.5 \ltjdeclarealtfont 39

11 Commands for \LaTeX\textastisk 39
  11.1 Loading Japanese fonts in \LaTeX\textastisk 39
  11.2 Patch for NFSS2 40
  11.3 Detail of \fontfamily command 42
  11.4 Notes on \DeclareTextSymbol 43
  11.5 \strutbox 43

12 expl3 interface 43

13 Addon packages 44
  13.1 luatexja-fontspec 44
  13.2 luatexja-tf 46
  13.3 luatexja-adjust 46
  13.4 luatexja-ruby 47
  13.5 \ltjtext.sty 47
  13.6 luatexja-preset 48
    13.6.1 General Options 48
    13.6.2 Presets which support multi weights 50
    13.6.3 Presets which do not support multi weights 52
    13.6.4 Presets which use HG fonts 53
    13.6.5 Define/Use Custom Presets 53

III Implementations 55

14 Storing Parameters 55
  14.1 Used dimensions, attributes and whatsit nodes 55
  14.2 Stack system of Lua\TeX\text-l 56
  14.3 Lua functions of the stack system 57
  14.4 Extending Parameters 57

15 Linebreak after a Japanese Character 58
  15.1 Reference: behavior in p\TeX 58
  15.2 Behavior in Lua\TeX\text-l 59
  15.3 Composition of Kana from Combining Character Sequences 60

16 Patch for the listings Package 60
  16.1 Notes and additional keys 61
  16.2 Class of characters 61
This documentation is far from complete. It may have many grammatical (and contextual) errors. Also, several parts are written in Japanese only.
Part I

User’s manual

1 Introduction

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

1.1 Backgrounds

Traditionally, ASCII pTEX, an extension of TEX, and its derivatives are used to typeset Japanese documents in TEX. pTEX is an engine extension of TEX: so it can produce high-quality Japanese documents without using very complicated macros. But this point is a mixed blessing: pTEX is left behind from other extensions of TEX, especially eTEX 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 e-pTEX (merging of pTEX and e-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, implementing all feature of pTEX is impossible, since all process of LuaTEX-ja must be implemented only by Lua and TEX macros. Hence LuaTEX-ja is not a just porting of pTEX; unnatural specifications/behaviors of pTEX were not adopted.

The followings are major changes from pTEX. For more detailed information, see Part III or other sections of this manual.

Command names pTEX adds several primitives, such as \kanjiskip, \prebreakpenalty, and \ifydir. They can be used as follows:

\kanjiskip=10pt \dimen0=kanjiskip
\tbaselineshift=0.1zw
\dimen0=tbaselineshift
\prebreakpenalty=`ぁ=100
\ifydir ... \fi

However, we cannot use them under LuaTEX-ja. Instead of them, we have to write as the following.

\ltjsetparameter{kanjiskip=10pt} \dimen0=\ltjgetparameter{kanjiskip}
\ltjsetparameter{tbaselineshift=0.1\zw}
\dimen0=\ltjgetparameter{tbaselineshift}
\ltjsetparameter{prebreakpenalty={`ぁ,100}}
\ifnum\ltjgetparameter{direction}=4 ... \fi

Note that pTEX adds new two useful units, namely zw and zh. As shown above, they are changed to \zw and \zh respectively in LuaTEX-ja.¹

Linebreak after a Japanese character 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 feature completely, because of a specification of LuaTEX. For the detail, see Section 15.

¹LuaTEX-ja 20200127.0 introduces \ltj@zw and \ltj@zh, which are copy of \zw and \zh.
Spaces related to Japanese characters  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 $\text{JAglue}$) is rewritten from scratch.

- As LuaTeX’s internal ligature handling is node-based (e.g., \texttt{office} doesn’t prevent ligatures), the insertion process of $\text{JAglue}$ is now node-based.

- Furthermore, nodes between two characters which have no effects in line break (e.g., \texttt{\special} node) and kerns from italic correction are ignored in the insertion process.

- 
  \textit{Caution: due to above two points, many methods which did for the dividing the process of the insertion of $\text{JAglue}$ in \TeX{} are not effective anymore.} In concrete terms, the following two methods are not effective anymore:

  \begin{itemize}
  \item ちょっと ちょっと
  \item ちょっと/ちょっと
  \end{itemize}

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

  \begin{itemize}
  \item ちょっと{\hbox{}} ちょっと
  \end{itemize}

- In the process, two Japanese fonts which only differ in their “real” fonts are identified.

Directions  From version 20150420.0, LuaTeX-ja supports vertical writing. We implement this feature by using callbacks of LuaTeX; so it must \textit{not} be confused with $\Omega$-style direction support of LuaTeX itself. Due to implementation, the dimension returned by \texttt{\wd}, \texttt{\ht}, or \texttt{\dp} depends on the content of the register \textit{only}. This is major difference with \TeX{}.

\begin{itemize}
\item \texttt{\discretionary} Japanese characters in discretionary break (\texttt{\discretionary}) is not supported.
\end{itemize}

Greek and Cyrillic letters, and ISO 8859-1 symbols  By default, LuaTeX-ja uses Japanese fonts to typeset Greek and Cyrillic letters. To change this behavior, put \texttt{\ltjsetparameter{jacharrange={-2,-3}}} in the preamble. For the detailed description, see Subsection 4.1.

From version 20150906.0, characters which belongs both ISO 8859-1 and JIS X 0208, such as ¶ and §, are now typeset in alphabetic fonts.

1.3 Notations

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

- 
  \texttt{\discretionary} characters are classified into following two types. Note that the classification can be customized by a user (see Subsection 4.1).

  \begin{itemize}
  \item \texttt{JAchar}: standing for characters which is used in Japanese typesetting, such as Hiragana, Katakana, Kanji, and other Japanese punctuation marks.
  \item \texttt{ALchar}: standing for all other characters like latin alphabets.
  \end{itemize}

  We say \textit{alphabetic fonts} for fonts used in \texttt{ALchar}, and \textit{Japanese fonts} for fonts used in \texttt{JAchar}.

- A word in a sans-serif font with underline (like prebreakpenalty) means an internal parameter for Japanese typesetting, and it is used as a key in \texttt{\ltjsetparameter} command.

- A word in a sens-serif font without underline (like fontspec) means a package or a class of \TeX{}.

- In this document, natural numbers start from zero. $\omega$ denotes the set of all natural numbers which can be used in \TeX{}.
1.4 About the project

Project Wiki  Project Wiki is under construction.

- https://osdn.jp/projects/luatex-ja/wiki/FrontPage%28en%29 (English)

This project is hosted by OSDN.

Members

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

2.1 Installation

The following packages are needed for the LuaT\TeX-ja package.

- Lua\TeX 1.10.0 (or later) (DVI output \textbackslash outputmode=8 is not supported.)
- recent luaotfload (v3.1 or later recommended)
- adobemapping (Adobe cmap and pdfmapping files)
- \LaTeX\ 2\epsilon 2020-02-02 patch level 5 or later (if you want to use Lua\TeX-ja with \LaTeX\ 2\epsilon)
- etoolbox (if you want to use Lua\TeX-ja with \LaTeX\ 2\epsilon)
- everysec (only for \LaTeX\ 2\epsilon 2020-02-02 and 2020-10-01)
- filehook, atbegshi (only for \LaTeX\ 2\epsilon 2020-02-02)
- \texttt{ltcmds}, \texttt{pdftexcmds}
- \texttt{fontspec} v2.7c (or later)
- Harano Aji fonts (https://github.com/trueroad/HaranoAjiFonts)
  More specifically, HaranoAjiMincho-Regular and HaranoAjiGothic-Medium.

Now Lua\TeX-ja is available from CTAN (in the macros/luatex/generic/luatexja directory), and the following distributions:

- Ti\TeX Live (in texmf-dist/tex/luatex/luatexja)
- W32Ti\TeX (in luatexja.tar.xz)
- MiK\TeX (in luatexja.tar.xz)

Harano Aji fonts are also available in these distributions (haranoaji in Ti\TeX Live and MiK\TeX, and luatexja.tar.xz in W32\TeX).

\textbf{HarfBuzz and Lua\TeX-ja} Using Lua\TeX-ja with LuaHB\TeX (Lua\TeX integrated with HarfBuzz) is not well tested. Maybe documents can typeset without an error, but with unwanted results (especially, vertical typesetting and \texttt{CID}).

Especially, \textit{We don't recommend defining a Japanese font with HarfBuzz}, by specifying \texttt{Renderer=Harfbuzz} etc. (\texttt{fontspec}) or \texttt{mode=harf} (otherwise).

\textbf{Manual installation}

1. Download the source, by one of the following method. At the present, Lua\TeX-ja has no stable release.

- Clone the Git repository by
  
  $\texttt{git\ clone\ git@osdn.jp/gitroot/luatex-ja/luatexja.git}$

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

  \texttt{http://git.osdn.jp/view?p=luatex-ja/luatexja.git;a=snapshot;h=HEAD;sf=tgz}.

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

2. Extract the archive. You will see \texttt{src/} and several other sub-directories. But only the contents in \texttt{src/} are needed to work Lua\TeX-ja.
3. If you downloaded this package from CTAN, you have to run following commands to generate classes:

   $ cd src
   $ lualatex ltjclasses.ins
   $ lualatex ltjsclasses.ins
   $ lualatex ltjltxdoc.ins

4. Copy all the contents of src/ into one of your TEXMF tree. TEXMF/tex/luatex/luatexja/ is an example location. If you cloned entire Git repository, making a symbolic link of src/ instead copying is also good.

5. If `mktexlsr` is needed to update the file name database, make it so.

### 2.2 Cautions

For changes from \TeX, see Subsection 1.2.

- The encoding of your source file must be UTF-8. Other encodings, such as EUC-JP or Shift-JIS, are not supported.
- \LuaTeX-ja is very slower than \TeX, and uses a lot of memory.
- **(Outdated) note for MiKTeX users** \LuaTeX-ja requires that several CMap files\(^2\) must be found from Lua\TeX. Strictly speaking, those CMaps are needed only in the first run of Lua\TeX-ja after installing or updating. But it seems that MiKTeX does not satisfy this condition, so you will encounter an error like the following:

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

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

- Note that when Lua\TeX-ja is loaded in plain Lua\TeX, we cannot use color specification on font loading, such as

```latex
\font\hoge=lmroman10-regular.otf:color=FF0000 % \font primitive
```

This is because codes for shifting baseline in math mode (Lua\TeX-ja) collide with and prevents loading codes for font color (luaotfload) in these environments. We recommend to use \TeX 2020-02-02 (or later), since we can avoid this collision in there.

### 2.3 Using in plain \TeX

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

```latex
\input luatexja.sty
```

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

- The following 12 Japanese fonts are preloaded:

<table>
<thead>
<tr>
<th>direction</th>
<th>classification</th>
<th>font name</th>
<th>&quot;10 pt&quot;</th>
<th>&quot;7 pt&quot;</th>
<th>&quot;5 pt&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>yoko (horizontal)</td>
<td>mincho</td>
<td>HaranoAjiMincho-Regular</td>
<td>\tenmin</td>
<td>\sevenmin</td>
<td>\fivemin</td>
</tr>
<tr>
<td></td>
<td>gothic</td>
<td>HaranoAjiMincho-Medium</td>
<td>\tengt</td>
<td>\seventhgt</td>
<td>\fivegt</td>
</tr>
<tr>
<td>tate (vertical)</td>
<td>mincho</td>
<td>HaranoAjiMincho-Regular</td>
<td>\tentmin</td>
<td>\seventhmin</td>
<td>\fivemin</td>
</tr>
<tr>
<td></td>
<td>gothic</td>
<td>HaranoAjiGothic-Medium</td>
<td>\tengt</td>
<td>\seventhgt</td>
<td>\fivegt</td>
</tr>
</tbody>
</table>

The “default” Japanese fonts (and JFMs for them) can be modified by defining `\ltj@stdmcfont` etc. before one inputs `luatexja.sty` (Subsection 8.3).

A character in an alphabetic font is generally smaller than a Japanese font in the same size. So actual size specification of these Japanese fonts is in fact smaller than that of alphabetic fonts, namely scaled by 0.962216.

- The amount of glue that are inserted between a `JAchar` and an `ALchar` (the parameter `xkanjiskip`) is set to
  
  \[
  (0.25 \cdot 0.962216 \cdot 10 \text{ pt})_{+1 \text{ pt}}^{-1 \text{ pt}} = 2.40554 \text{ pt}^{+1 \text{ pt}}_{-1 \text{ pt}}.
  \]

### 2.4 Using in \LaTeX

Using in \LaTeX2e is basically same. To set up the minimal environment for Japanese, you only have to load `luatexja.sty`:

\begin{verbatim}
\usepackage{luatexja}
\end{verbatim}

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

- Font encodings for Japanese fonts are `JY3` (for horizontal direction) and `JT3` (for vertical direction).
- Traditionally, Japanese documents use only two families: `mincho` (明朝体) and `gothic` (ゴシック体). `mincho` is used in the main text, while `gothic` is used in the headings or for emphasis.

<table>
<thead>
<tr>
<th>classification (Japanese counterpart for typewriter font)</th>
<th>commands</th>
<th>family</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mincho</code> (明朝体)</td>
<td><code>\textmc{...}</code> <code>\mcdefault</code></td>
<td></td>
</tr>
<tr>
<td><code>gothic</code> (ゴシック体)</td>
<td><code>\textgt{...}</code> <code>\gtdefault</code></td>
<td></td>
</tr>
</tbody>
</table>

Here `\jttdefault` specifies the Japanese font family in `\verb` or `verbatim` environment, and its default value is `\mcdefault` (mincho family).\footnote{When \ltjclasses classes are used, or `luatexja-fontspec` (or `luatexja-preset`) is loaded with `match` option, `\ttfamily` changes the current Japanese font family to `\jttdefault`. These classes and packages also redefine `\jttdefault` to `\gtdefault` (gothic family).} Lua\TeX-ja does not define commands to only switch current Japanese font family to `\jttdefault`.

- By default, the following fonts are used for these two families.

<table>
<thead>
<tr>
<th>classification (Japanese counterpart for typewriter font)</th>
<th>family</th>
<th><code>\mdseries</code></th>
<th><code>\bfseries</code></th>
<th>scale</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mincho</code> (明朝体)</td>
<td><code>mc</code></td>
<td>HaranoAjiMincho-Regular</td>
<td>HaranoAjiGothic-Medium</td>
<td>0.962216</td>
</tr>
<tr>
<td><code>gothic</code> (ゴシック体)</td>
<td><code>gt</code></td>
<td>HaranoAjiGothic-Medium</td>
<td>HaranoAjiGothic-Medium</td>
<td>0.962216</td>
</tr>
</tbody>
</table>

- Note that the bold series (series `bx` or `b`) in both family are same as the medium series of gothic family. There is no italic nor slanted shape for these `mc` and `gt`.

- From version 20181102.0, one can specifies `disablejfam` option at loading Lua\TeX-ja. This option prevents loading a patch for \LaTeX, which are needed to support Japanese characters in math mode. Without `disablejfam` option, one can typeset Japanese characters in math mode as `\$\&\$` (see Page 11) as before. Japanese characters in math mode are typeset by the font family `mc`.

- If you use the beamer class with the default font theme (which uses sans serif fonts) and with Lua\TeX-ja, you might want to change default Japanese fonts to the gothic family. The following line changes the default Japanese font family to `\gtdefault`:

\begin{verbatim}
\renewcommand{\kanjifamilydefault}{\gtdefault}
\end{verbatim}
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, LuaTEX-ja has the counterparts of jclasses (standard classes in pLaTeX) and jsclasses (classes by Haruhiko Okumura), namely, `ltjclasses` and `ltjsclasses`.

Original jsclasses use \mag primitive to set the main document font size. However, LuaTEX does not support \mag in PDF output, so ltjsclasses uses the nomag* option by default to set the main font size. If this causes some unexpected behavior, specify nomag option in \documentclass.

### geometry package and classes for vertical writing

It is well-known that the geometry package produces the following error, when classes for vertical writing is used:

\begin{verbatim}
! Incompatible direction list can't be unboxed.\n@begindvi ->\unvbox \@begindvibox\n\global \let \@begindvi \@empty\n\end{verbatim}

Now, LuaTEX-ja automatically applies the patch `ltjp-geometry` to the geometry package, when the direction of the document is `tate` (vertical writing). This patch `ltjp-geometry` also can be used in pLaTeX; for the detail, please refer `ltjp-geometry.pdf` (Japanese).

### 3 Changing Fonts

#### 3.1 plain \TeX{} and \LaTeX{} 2\epsilon

- **plain \TeX{}** To change Japanese fonts in plain \TeX{}, you must use the command `\jfont` and `\tfont`. So please see Subsection 8.1.

- **\LaTeX{} 2\epsilon** (NFSS2) For \LaTeX{} 2\epsilon, LuaTEX-ja adopted most of the font selection system of pLaTeX{} 2\epsilon (in `plfonts.dtx`).

<table>
<thead>
<tr>
<th>encoding</th>
<th>family</th>
<th>series</th>
<th>shape</th>
<th>selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alphabetic fonts</td>
<td><code>\romanencoding</code></td>
<td><code>\romanfamily</code></td>
<td><code>\romanseries</code></td>
<td><code>\romanshape</code></td>
</tr>
<tr>
<td>Japanese fonts</td>
<td><code>\kanjiencoding</code></td>
<td><code>\kanjifamily</code></td>
<td><code>\kanjiseries</code></td>
<td><code>\kanjishape</code></td>
</tr>
<tr>
<td>both</td>
<td>--</td>
<td>--</td>
<td><code>\fontseries</code></td>
<td><code>\fontshape</code></td>
</tr>
<tr>
<td>auto select</td>
<td><code>\fontencoding</code></td>
<td><code>\fontfamily</code></td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

- `\fontfamily`, `\fontseries`, and `\fontshape` try to change attributes of Japanese fonts, as well as those of alphabetic fonts. Of course, `\selectfont` is needed to select current text fonts.

Note that `\fontshape` always changes current alphabetic font shape, but it does not change current Japanese font shape if the target shape is unavailable for current Japanese encoding/family/series. For the detail, see Subsection 11.2.

- `\fontencoding{⟨encoding⟩}` changes the encoding of alphabetic fonts or Japanese fonts depending on the argument. For example, `\fontencoding{JY3}` changes the encoding of Japanese fonts to JY3, and `\fontencoding{T1}` changes the encoding of alphabetic fonts to T1. `\fontfamily` also changes the current Japanese font family, the current alphabetic font family, or both. For the detail, see Subsection 11.2.

- For defining a Japanese font family, use `\DeclareKanjiFamily` instead of `\DeclareFontFamily`. (In previous version of LuaTEX-ja, using `\DeclareFontFamily` didn’t cause any problem. But this no longer applies the current version.)

- Defining a Japanese font shape can be done by usual `\DeclareFontShape`:

  `\DeclareFontShape{JY3}{mc}{b}{n}{<-> s*HaranoAjiMincho--Bold:jfm=ujis;kern} {}`%

  Harano Aji Mincho Bold

---

4 `ltjarticle.cls`, `ltjbook.cls`, `ltjreport.cls`, `ltjarticle.cls`, `ltjbook.cls`, `ltjreport.cls`. The latter `ltjt*.cls` are for vertically written Japanese documents.

5 `ltjsarticle.cls`, `ltjsbook.cls`, `ltjsreport.cls`, `ltjskiyou.cls`, `ltjkiyou.cls`. Same effect as the BXjcs classes (by Takayuki Yato) and jsclasses. However, these classes uses only \TeX{} code, but ltjsclasses uses Lua code.
### Japanese characters in math mode

Since \LaTeX{} supports Japanese characters in math mode, there are sources like the following:

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

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

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

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

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

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 8.6.

When Lua\TeX-ja is loaded with \texttt{disablejfamily} option, one cannot write Japanese characters in math mode as $\text{素}$. At that case, one have to use \texttt{\mbox} (or \texttt{\text} in the amsmath package).

### 3.2 \texttt{luatexja-fontspec} package

To use the functionality of the fontspec package to Japanese fonts, it is needed to load the \texttt{luatexja-fontspec} package in the preamble, as follows:

\begin{verbatim}
\usepackage[[\texttt{options}]]{luatexja-fontspec}
\end{verbatim}

This \texttt{luatexja-fontspec} package automatically loads \texttt{luatexja} and \texttt{fontspec} packages, if needed.

In the \texttt{luatexja-fontspec} package, several commands are defined as counterparts of original commands in the fontspec package (see Table 1):

The package option of \texttt{luatexja-fontspec} are the followings:

**match**

If this option is specified, usual family-changing commands such as \texttt{\rmfamily}, \texttt{\textrm}, \texttt{\sffamily}, ... also change Japanese font family.

**pass=⟨\texttt{opts}⟩**

(Obsolete) Specify options \langle\texttt{opts}\rangle which will be passed to the fontspec package.

---

<table>
<thead>
<tr>
<th><strong>Japanese fonts</strong></th>
<th>\texttt{%fontspec}</th>
<th>\texttt{\setmainjfont}</th>
<th>\texttt{\setsansjfont}</th>
<th>\texttt{\setmonojfont}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alphabetic fonts</td>
<td>\texttt{\fontspec}</td>
<td>\texttt{\setmainfont}</td>
<td>\texttt{\setsansfont}</td>
<td>\texttt{\setmonofont}</td>
</tr>
<tr>
<td><strong>Japanese fonts</strong></td>
<td>\texttt{\newjfontfamily}</td>
<td>\texttt{\renewjfontfamily}</td>
<td>\texttt{\setjfontfamily}</td>
<td>\texttt{\newfontfamily}</td>
</tr>
<tr>
<td>Alphabetic fonts</td>
<td>\texttt{\newfontface}</td>
<td>\texttt{\defaultjfontfeatures}</td>
<td>\texttt{\addjfontfeatures}</td>
<td>\texttt{\newfontface}</td>
</tr>
</tbody>
</table>
Override the ratio of the font size of Japanese fonts to that of alphabetic fonts. The default value is determined as follows:

- The value of \Cjascale is used, if this control sequence is already defined.
- It is calculated automatically from the current Japanese font at the loading of the package, if \Cjascale is not defined.

\Cjascale is defined in ltjclasses and ltjsclasses.

All other options listed above are simply passed to the fontspec package. This means that two lines below are equivalent, for example.

\usepackage[no-math]{fontspec}
\usepackage{luatexja-fontspec}
\usepackage[no-math]{luatexja-fontspec}

Note that kerning information in a font is not used (that is, kern feature is set off) by default in these seven (or eight) commands. This is because of the compatibility with previous versions of LuaTeX-ja (see 8.1).

Below is an example of \jfontspec.

1 \jfontspec\{CJKShape=NLC\}{HaranoAjiMincho-Regular}
2 JIS~X~0213:2004 →辻鯵
3 \jfontspec\{CJKShape=JIS1990\}{HaranoAjiMincho-Regular}
4 JIS~X~0208-1990 →辻鯵
5 \jfontspec\{CJKShape=JIS1978\}{HaranoAjiMincho-Regular}
6 JIS~C~6226-1978 →辻鯵

3.3 Presets of Japanese fonts

With luatexja-preset package, one use one of "preset" to simplify Japanese font setting. For details of package options, and those of each presets, please see Subsection 13.6. The following presets are defined:

- haranoaji
- hiragino-pro
- hiragino-pron
- ipa
- ipa-hg
- ipaex
- ipaex-hg
- kozuka-pr
- kozuka-pron
- kozuka-pro
- moga-mobo
- moga-mobo-ex
- bizud
- morisawa-pron
- morisawa-pro
- ms
- ms-hg
- noembed
- noto-otc
- noto-otf
- sourcehan
- sourcehan-jp
- ume
- yu-osx
- yu-win
- yu-win10

For example, this document loads luatexja-preset package by
\usepackage\{haranoaji\}\{luatexja-preset\}
which means that Harano Aji fonts will be used in this document.

3.4 \CID, \UTF, and macros in japanese-otf package

Under p\TeX, japanese-otf package (developed by Shuzaburo Saito) is used for typesetting characters which is in Adobe-Japan1-6 CID but not in JIS X 0208. Since this package is widely used, Lua\TeX-ja supports some of functions in the japanese-otf package, as an external package luatexja-otf.

4 Changing Internal Parameters

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

LuaTeX-ja divides the Unicode codespace U+0080–U+10FFFF into character ranges, numbered 1 to 217. The grouping can be (globally) customized by \ltjdefcharrange. The next line adds whole characters in Supplementary Ideographic Plane and the character “漢” to the character range 100.

\ltjdefcharrange{100}{“20000-”2FFFF,`漢}.

A character can belong to only one character range. For example, whole SIP belong to the range 4 in the default setting of LuaTeX-ja, and if one executes the above line, then SIP will belong to the range 100 and be removed from the range 4.

The distinction between ALchar and JAchar is performed by character ranges. This can be edited by setting the jacharrange parameter. For example, the code below is just the default setting of LuaTeX-ja, and it sets

- a character which belongs character ranges 1, 4, 5, and 8 is ALchar,
- a character which belongs character ranges 2, 3, 6, 7, and 9 is JAchar.

\ltjsetparameter{jacharrange={-1, +2, +3, -4, -5, +6, +7, -8, +9}}

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

Note that characters U+0000–U+007F are always treated as an ALchar (this cannot be customized).

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

- Blocks in Unicode 12.0.0.
- The PXbase bundle for upTeX by Takayuki Yato.

Now we describe these nine ranges. The superscript "J" or "A" after the number shows whether each character in the range is treated as JAchar or not by default. These settings are similar to the prefercjk settings defined in PXbase bundle. Any characters equal to or above U+0080 which does not belong to these eight ranges belongs to the character range 217.

**Range 8\(^J\)** The intersection of the upper half of ISO 8859-1 (Latin-1 Supplement) and JIS X 0208 (a basic character set for Japanese). The character list is indicated in Table 2.

**Range 1\(^A\)** Latin characters that some of them are included in Adobe-Japan1-7. This range consists of the Unicode ranges indicated in Table 3, except characters in the range 8 above.

**Range 2\(^J\)** Greek and Cyrillic letters. JIS X 0208 (hence most of Japanese fonts) has some of these characters.
Table 4. Unicode blocks in predefined character range 3.

<table>
<thead>
<tr>
<th>Range 3</th>
<th>Miscellaneous symbols. The block list is indicated in Table 4.</th>
</tr>
</thead>
<tbody>
<tr>
<td>U+2070–U+209F</td>
<td>Superscripts and Subscripts</td>
</tr>
<tr>
<td>U+20A0–U+20CF</td>
<td>Currency Symbols</td>
</tr>
<tr>
<td>U+2100–U+214F</td>
<td>Letterlike Symbols</td>
</tr>
<tr>
<td>U+2150–U+218F</td>
<td>Number Forms</td>
</tr>
<tr>
<td>U+2190–U+21FF</td>
<td>Arrows</td>
</tr>
<tr>
<td>U+2200–U+22FF</td>
<td>Control Pictures</td>
</tr>
<tr>
<td>U+22D0–U+22FF</td>
<td>Mathematical Operators</td>
</tr>
<tr>
<td>U+2300–U+23FF</td>
<td>Miscellaneous Technical</td>
</tr>
<tr>
<td>U+2400–U+243F</td>
<td>Control Pictures</td>
</tr>
<tr>
<td>U+2500–U+257F</td>
<td>Box Drawing</td>
</tr>
<tr>
<td>U+2580–U+259F</td>
<td>Block Elements</td>
</tr>
<tr>
<td>U+25A0–U+25FF</td>
<td>Geometric Shapes</td>
</tr>
<tr>
<td>U+2600–U+26FF</td>
<td>Miscellaneous Symbols</td>
</tr>
<tr>
<td>U+2700–U+27BF</td>
<td>Dingbats</td>
</tr>
<tr>
<td>U+2900–U+297F</td>
<td>Supplemental Arrows-B</td>
</tr>
<tr>
<td>U+2980–U+29FF</td>
<td>Misc. Math Symbols-B</td>
</tr>
<tr>
<td>U+2B00–U+2BFF</td>
<td>Misc. Symbols and Arrows</td>
</tr>
</tbody>
</table>

Table 5. Characters in predefined character range 9.

<table>
<thead>
<tr>
<th>Range 9</th>
<th>The intersection of the “General Punctuation” block (U+2000–U+206F) and Adobe-Japan1-7 character collection. This character range characters in Table 5.</th>
</tr>
</thead>
<tbody>
<tr>
<td>U+2002</td>
<td>En space</td>
</tr>
<tr>
<td>U+2010</td>
<td>Hyphen</td>
</tr>
<tr>
<td>U+2011</td>
<td>Non-breaking hyphen</td>
</tr>
<tr>
<td>U+2015</td>
<td>Em dash</td>
</tr>
<tr>
<td>U+2016</td>
<td>Double vertical line</td>
</tr>
<tr>
<td>U+2018</td>
<td>Left single quotation mark</td>
</tr>
<tr>
<td>U+2019</td>
<td>Right single quotation mark</td>
</tr>
<tr>
<td>U+201A</td>
<td>Single low-9 quotation mark</td>
</tr>
<tr>
<td>U+201C</td>
<td>Left double quotation mark</td>
</tr>
<tr>
<td>U+201D</td>
<td>Right double quotation mark</td>
</tr>
<tr>
<td>U+201E</td>
<td>Double low-9 quotation mark</td>
</tr>
<tr>
<td>U+2020</td>
<td>Dagger</td>
</tr>
<tr>
<td>U+2021</td>
<td>Double dagger</td>
</tr>
<tr>
<td>U+2022</td>
<td>Bullet</td>
</tr>
<tr>
<td>U+2025</td>
<td>Two dot leader</td>
</tr>
<tr>
<td>U+2026</td>
<td>Horizontal ellipsis</td>
</tr>
<tr>
<td>U+2027</td>
<td>Per mille sign</td>
</tr>
<tr>
<td>U+2028</td>
<td>Single left-pointing angle quot.</td>
</tr>
<tr>
<td>U+2029</td>
<td>Single right-pointing angle quot.</td>
</tr>
<tr>
<td>U+202A</td>
<td>Single right-pointing angle quot.</td>
</tr>
<tr>
<td>U+202B</td>
<td>Reference mark</td>
</tr>
<tr>
<td>U+202C</td>
<td>Underline</td>
</tr>
<tr>
<td>U+202D</td>
<td>Asterism</td>
</tr>
<tr>
<td>U+202E</td>
<td>Overline</td>
</tr>
<tr>
<td>U+202F</td>
<td>Bullet</td>
</tr>
<tr>
<td>U+2030</td>
<td>Double question mark</td>
</tr>
<tr>
<td>U+2031</td>
<td>Exclamation question mark</td>
</tr>
</tbody>
</table>

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

Range 3 | Miscellaneous symbols. The block list is indicated in Table 4.

Range 9A | The intersection of the “General Punctuation” block (U+2000–U+206F) and Adobe-Japan1-7 character collection. This character range characters in Table 5.

Range 4 | 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.

```
\ltdfcharrange{4}{%
    "580-"10FF, "1280-"1DFF, "2440-"245F, "27C0-"28FF, "2A00-"2AFF,
    "2C80-"2E7F, "4DC0-"4DFF, "A000-"A9FF, "F000-"F8FF,
    "F800-"FE0F, "FE20-"FE2F, "FE70-"FEFF, "10000-"1AFF, "1B170-"1F8FF,
    "1F880-"1FFFF, ... (and characters in U+2000–U+206F which are not in range 9)
} % non-Japanese
```

Range 5A | Surrogates and Supplementary Private Use Areas.

Range 6A | Characters used in Japanese. The block list is indicated in Table 6.

Range 7A | Characters used in CJK languages, but not included in Adobe-Japan1-7. The block list is indicated in Table 7.

Notes on U+0080–U+00FF You should treat characters in textttU+0080–U+00FF as Alchar, when you use traditional 8-bit fonts, such as the marvosym package.
xkanjiskip: \ltjgetparameter{xkanjiskip}
\\kanjiskip: \ltjgetparameter{kanjiskip}, \\\%
\n\begin{align*}
\alpha \alpha \alpha, \ltjalchar \alpha, \ltjjachar \alpha\%
\end{align*}
\\% default: ALchar

\textbf{4.2 \textit{kanjiskip} and \textit{xkanjiskip}}

JAGlue is divided into the following three categories:
\begin{itemize}
\item Glues/kerns specified in JFM. If \texttt{\inhibitglue} is issued around a \texttt{JAchar}, this glue will not be inserted at the place.
\item The default glue which inserted between two \texttt{JAchars} (\texttt{kanjiskip}).
\item The default glue which inserted between a \texttt{JAchar} and an \texttt{ALchar} (\texttt{xkanjiskip}).
\end{itemize}

The value (a skip) of \texttt{kanjiskip} or \texttt{xkanjiskip} can be changed as the following. Note that only their values at the end of a paragraph or a hbox are adopted in the whole paragraph or the whole hbox.
\begin{verbatim}
\ltjsetparameter{kanjiskip={0.25\zw plus 0.4pt minus 0.4pt},
\texttt{xkanjiskip}={0.0pt plus 0.4pt minus 0.5pt},\}
\end{verbatim}

Here \texttt{\zw} is a internal dimension which stores fullwidth of the current Japanese font. This \texttt{\zw} can be used as the unit \texttt{\zw} in \TeX.

The value of these parameter can be get by \texttt{\ltjgetparameter}. Note that the result by \texttt{\ltjgetparameter} is not the internal quantities, but a string (hence \texttt{\the} the cannot be prefixed).
\begin{verbatim}
kanjiskip: \ltjgetparameter[kanjiskip], \texttt{kanjiskip}: 0.0pt plus 0.4pt minus 0.5pt,
xkanjiskip: \ltjgetparameter[xkanjiskip] \texttt{xkanjiskip}: 2.40533pt plus 1.0pt minus 1.0pt
\end{verbatim}

It may occur that JFM contains the data of "ideal width of \texttt{kanjiskip}" and/or "ideal width of \texttt{xkanjiskip}". To use these data from JFM, set the value of \texttt{kanjiskip} or \texttt{xkanjiskip} to \texttt{\maxdimen} (these "ideal width" cannot be retrieved by \texttt{\ltjgetparameter}).

---

\begin{table}
\centering
\begin{tabular}{llll}
\hline
\rowcolor{gray!20} U+2460–U+246F & Enclosed Alphanumerics & U+2E80–U+2EFF & CJK Radicals Supplement \\
\rowcolor{gray!20} U+3000–U+303F & CJK Symbols and Punctuation & U+3840–U+38FF & Hiragana \\
\rowcolor{gray!20} U+30AB–U+30FF & Katakana & U+3190–U+31FF & Kanbun \\
\rowcolor{gray!20} U+3170–U+31FF & Katakana Phonetic Extensions & U+3200–U+32FF & Enclosed CJK Letters and Months \\
\rowcolor{gray!20} U+3200–U+32FF & CJK Compatibility Forms & U+3400–U+40FF & CJK Unified Ideographs Ext-A \\
\rowcolor{gray!20} U+4E00–U+8DBF & CJK Unified Ideographs & U+F900–U+F9FF & CJK Unified Ideographs Ext-A \\
\rowcolor{gray!20} U+31F0–U+31FF & Vertical Forms & U+FE30–U+FE4F & CJK Compatibility Ideographs \\
\rowcolor{gray!20} U+FE50–U+FE6F & Small Form Variants & U+FF00–U+FFEF & Halfwidth and Fullwidth Forms \\
\rowcolor{gray!20} U+1B100–U+1B2FF & Kana Supplement & U+1B100–U+1B2F & Kana Extended-A \\
\rowcolor{gray!20} U+1F100–U+1F2FF & Enclosed Alphanumeric Supp. & U+1F100–U+1F2F & Enclosed Ideographic Supp. \\
\rowcolor{gray!20} U+20000–U+200FF & \texttt{(Supp. Ideographic Plane)} & U+30000–U+30FFF & (Tert. Ideographic Plane) \\
\rowcolor{gray!20} U+Eric0–U+Eric1F & \texttt{Variation Selectors Supp.} & & \\
\hline
\end{tabular}
\caption{Unicode blocks in predefined character range 6.}
\end{table}

\begin{table}
\centering
\begin{tabular}{llll}
\hline
\rowcolor{gray!20} U+1100–U+11FF & Hangul Jamo & U+2F00–U+2FFF & Kangxi Radicals \\
\rowcolor{gray!20} U+2FF8–U+2FFF & Ideographic Description Characters & U+3100–U+31FF & Bopomofo \\
\rowcolor{gray!20} U+3130–U+318F & Hangul Compatibility Jamo & U+31AB–U+31BF & Bopomofo Extended \\
\rowcolor{gray!20} U+3190–U+31EF & CJK Strokes & U+3080–U+309F & Yi Syllables \\
\rowcolor{gray!20} U+4A90–U+4ACF & Yi Radicals & U+4900–U+49FF & Hangul Jamo Extended-A \\
\rowcolor{gray!20} U+6C80–U+6D7F & Hangul Syllables & U+6780–U+67FF & Hangul Jamo Extended-B \\
\hline
\end{tabular}
\caption{Unicode blocks in predefined character range 7.}
\end{table}
4.3 Insertion setting of \textit{xkanjiskip}

It is not desirable that \textit{xkanjiskip} is inserted into every boundary between \texttt{JAchar}s and \texttt{ALchar}s. For example, \textit{xkanjiskip} should not be inserted after opening parenthesis (e.g., compare “(あ)” and “(あ)”). \texttt{LuaTeX-ja} can control whether \textit{xkanjiskip} can be inserted before/after a character, by changing \texttt{jaxspmode} for \texttt{JAchar}s and \texttt{alxspmode} parameters \texttt{ALchar}s respectively.

\begin{verbatim}
\letjsetparameter{jaxspmode={`あ,preonly},
alxspmode=`\!,postonly}} pあqいう
\end{verbatim}

The second argument \texttt{preonly} means that the insertion of \textit{xkanjiskip} is allowed before this character, but not after. the other possible values are \texttt{postonly}, \texttt{allow}, and \texttt{inhibit}.

\texttt{jaxspmode} and \texttt{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:

\begin{verbatim}
\letjsetparameter{alxspmode={`あ,preonly}, jaxspmode=`\!,postonly}}
\end{verbatim}

One can use also numbers to specify these two parameters (see Subsection 9.1).

If you want to enable/disable all insertions of \textit{kanjiskip} and \textit{xkanjiskip}, set \texttt{autospacing} and \texttt{autoxspacing} parameters to \texttt{true}/\texttt{false}, respectively.

4.4 Shifting the baseline

To make a match between a Japanese font and an alphabetic font, sometimes shifting of the baseline of one of the pair is needed. In \texttt{p\TeX}, this is achieved by setting \texttt{\vbaselineshift} (or \texttt{\tbaselineshift}) to a non-zero length (the baseline of \texttt{ALchar} is shifted below). However, for documents whose main language is not Japanese, it is good to shift the baseline of Japanese fonts, but not that of alphabetic fonts. Because of this, \texttt{Lua\TeX-ja} can independently set the shifting amount of the baseline of alphabetic fonts and that of Japanese fonts.

<table>
<thead>
<tr>
<th></th>
<th>Horizontal writing (yoko direction) etc.</th>
<th>Vertical writing (tate direction)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alphabetic fonts</td>
<td>\texttt{yalbaselineshift} parameter</td>
<td>\texttt{talbaselineshift} parameter</td>
</tr>
<tr>
<td>Japanese fonts</td>
<td>\texttt{yjabaselineshift} parameter</td>
<td>\texttt{tjabaselineshift} parameter</td>
</tr>
</tbody>
</table>

Here the horizontal line in the below example is the baseline of a line.

\begin{verbatim}
\vrule width 150pt height 0.2pt depth 0.2pt \hskip-120pt
\letjsetparameter{yjabaselineshift=0pt, yalbaselineshift=0pt}abcあいう
\letjsetparameter{yjabaselineshift=5pt, yalbaselineshift=2pt}abcあいう
\end{verbatim}

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):

\begin{verbatim}
\vrule width 150pt height 4.417pt depth -4.217pt\%\hskip-150pt
\large\large{xyz漢字\
{\scriptsize\
\letjsetparameter{yjabaselineshift=-1.757pt, yalbaselineshift=-1.757pt}}\漢字\漢字\漢字\漢字
}\large{abc}\large{abc}\large{abc\large{abc}}
\end{verbatim}

Note that setting positive \texttt{yalbaselineshift} or \texttt{talbaselineshift} parameters does not increase the depth of one-letter syllable \texttt{p} of \texttt{Alchar}, if its left-protrusion (\texttt{\lpcode}) and right-protrusion (\texttt{\rpcode}) are both non-zero. This is because

- These two parameters are implemented by setting \texttt{yoffset} field of a glyph node, and this does not increase the depth of the glyph.
• To cope with the above situation, LuaTeX-ja automatically supplies a rule in every syllable.

• However, we cannot use this “supplying a rule” method if a syllable comprises just one letter whose \ Lance{1pcode} and \ Lance{rtpcode} are both non-zero.

This problem does not apply for \texttt{\v{y}jabaselineshift} nor \texttt{\tjabaselineshift}, because a \texttt{\JaChar} is encapsulated by a horizontal box if needed.

### 4.5 kinsoku parameters and OpenType features

Among parameters which related to Japanese word-wrapping process (kinsoku shori), \texttt{\jaspmodet}, \texttt{\alxspmode}, \texttt{\prebreakpenalty}, \texttt{\postbreakpenalty} and \texttt{\kcatcode} are stored by each character codes.

OpenType font features are ignored in these parameters. For example, a fullwidth katakana “ア” on line 10 in the below input is replaced to its halfwidth variant “ア”, by \texttt{\hwid} feature. However, the penalty inserted after it is 10 which is the \texttt{\postbreakpenalty} of “ア”, not 20.

```latex
\begin{verbatim}
\ltjsetparameter{postbreakpenalty={\ア, 10}}
\ltjsetparameter{postbreakpenalty={\ア, 20}}
\newcommand\showpostpena[1][1]{\leavevmode\setbox0=\hbox{#1\hbox{}}\unhbox0\setbox0=\lastbox\the\lastpenalty}
\showpostpena{ア}, \showpostpena{ア}, \newcommand{\addjfontfeatures{CharacterWidth=Half}}\showpostpena{ア}
\end{verbatim}
```

\begin{tabular}{l}
ア 10, ア 20, ア 10
\end{tabular}
Part II
Reference

5 \texttt{\textbackslash catcode} in \texttt{Lua\TeX-ja}

5.1 Preliminaries: \texttt{\textbackslash catcode} in \texttt{p\TeX} and \texttt{up\TeX}

In \texttt{p\TeX} and \texttt{up\TeX}, the value of \texttt{\textbackslash catcode} determines whether a Japanese character can be used in a control word. For the detail, see Table 8. \texttt{\textbackslash catcode} can be set by a row of JIS X 0208 in \texttt{p\TeX}, and generally by a Unicode block\footnote{\texttt{up\TeX} divides U+FF00–U+FFE0 (Halfwidth and Fullwidth Forms) into three subblocks, and \texttt{\textbackslash catcode} can be set by a subblock.} in \texttt{up\TeX}. So characters which can be used in a control word slightly differ between \texttt{p\TeX} and \texttt{up\TeX}.

5.2 Case of \texttt{Lua\TeX-ja}

The role of \texttt{\textbackslash catcode} in \texttt{p\TeX} and \texttt{up\TeX} can be divided into the following four kinds, and \texttt{Lua\TeX-ja} can control these four kinds separately:

- Distinction between \texttt{JAchar} or \texttt{ALchar} is controlled by the character range, see Subsection 4.1.

- Whether the character can be used in a control word is controlled by setting \texttt{\textbackslash catcode} to 11 (enabled) or 12 (disabled), as usual.

- Whether \texttt{\textbackslash charwidowpenalty} can be inserted before the character is controlled by the lowermost bit of the \texttt{\textbackslash catcode} parameter.

- Linebreak after a \texttt{JAchar} does not produce a space.

Default setting of \texttt{\textbackslash catcode} of Unicode characters with \texttt{Lua\TeX} is slightly inconvenient for \texttt{p\TeX} users to shifting to \texttt{Lua\TeX-ja}, because several fullwidth characters which can be used in a control word with \texttt{p\TeX}, such as “1” (FULLWIDTH DIGIT ONE), cannot be used in a control word with \texttt{Lua\TeX}. Hence, \texttt{Lua\TeX-ja} changes the \texttt{\textbackslash catcode} of some characters—whose line breaking class is “ID” (Ideographic) in UAX #14—, to allow these characters in the control word.

5.3 Non-kanji characters in a control word

Because the engine differ, so non-kanji JIS X 0208 characters which can be used in a control word differ in \texttt{p\TeX}, in \texttt{up\TeX}, and in \texttt{Lua\TeX-ja}. Table 9 shows the difference. Except for three characters \texttt{・}, \texttt{゛}, and \texttt{゜}, \texttt{Lua\TeX-ja} admits more characters in a control word than \texttt{up\TeX}.

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

6 Directions

\texttt{Lua\TeX} supports four \texttt{\Omega}-style directions: TLT, TRT, RTT and LTL. However, neither directions are not well-suited for typesetting Japanese vertically, hence we implemented vertical writing by rotating TLT-box by 90 degrees.

\texttt{Lua\TeX-ja} supports four directions, as shown in Table 10. The second column (\texttt{yoko} direction) is just horizontal writing, and the third column (\texttt{tate} direction) is vertical writing. The fourth column (\texttt{dtou} direction) is actually a hidden feature of \texttt{p\TeX}. We implemented this for debugging purpose. The fifth column (\texttt{utod} direction) corresponds the “\texttt{tate} (math) direction” of \texttt{p\TeX}.

Directions can be changed by \texttt{\textbackslash yoko}, \texttt{\textbackslash tate}, \texttt{\textbackslash dtou}, \texttt{\textbackslash utod}, only when the current list is null. These commands cannot be executed in unrestricted horizontal modes, nor math modes. The direction of a math formula is changed to \texttt{utod}, when the direction outside the math formula is \texttt{tate} (vertical writing).
Table 8. $\verb|\kcatcode|$
in upTEX

| $\verb|\kcatcode|$ | meaning | control word | widow penalty | linebreak |
|----------------|---------|--------------|---------------|-----------|
| 15            | non-cjk | (treated as usual upTEX) | Y             | Y         |
| 16            | kanji   | Y            | Y             | ignored   |
| 17            | kana    | Y            | Y             | ignored   |
| 18            | other   | N            | N             | ignored   |
| 19            | hangul  | Y            | Y             | space     |

Table 9. Difference of the set of non-kanji JIS X 0208 characters which can be used in a control word

<table>
<thead>
<tr>
<th>row</th>
<th>col.</th>
<th>pTeX</th>
<th>upTeX</th>
<th>LuaTeX-ja</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>3</td>
<td>14</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>5</td>
<td>17</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>6</td>
<td>18</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>7</td>
<td>19</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>8</td>
<td>20</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>9</td>
<td>21</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>10</td>
<td>22</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>11</td>
<td>23</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>12</td>
<td>24</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>13</td>
<td>25</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>14</td>
<td>26</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>15</td>
<td>27</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>16</td>
<td>28</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

6.1 Boxes in different direction

As in pTeX, one can use boxes of different direction in one document. The below is an example.

```
\begin{tate}
\hbox{\yoko横組の内容}
\end{tate}
```

Table 11 shows how a box is arranged when the direction inside the box and that outside the box differ.

<table>
<thead>
<tr>
<th>row</th>
<th>col.</th>
<th>pTeX</th>
<th>upTeX</th>
<th>LuaTeX-ja</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>31</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>2</td>
<td>32</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>3</td>
<td>33</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>4</td>
<td>35</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>5</td>
<td>60</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>6</td>
<td>65</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>7</td>
<td>67</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>8</td>
<td>68</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>9</td>
<td>84</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>10</td>
<td>85</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>11</td>
<td>86</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>12</td>
<td>87</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>13</td>
<td>9</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>15</td>
<td>22</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>16</td>
<td>44</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>17</td>
<td>82</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>

Greek letters (row 6) Y N Y

Cyrillic letters (row 7) N N Y
Table 10. Directions supported by LuaTeX-ja

<table>
<thead>
<tr>
<th>Commands</th>
<th>horizontal (yoko direction)</th>
<th>vertical (tate direction)</th>
<th>dtou direction</th>
<th>utod direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beginning of the page</td>
<td>yoko</td>
<td>tate</td>
<td>dtou</td>
<td>utod</td>
</tr>
<tr>
<td>Beginning of the line</td>
<td>Top</td>
<td>Right</td>
<td>Left</td>
<td>Right</td>
</tr>
<tr>
<td>Used Japanese font</td>
<td>horizontal (\jfont)</td>
<td>vertical (\tfont)</td>
<td>horizontal (90° rotated)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>銀は、Ag</td>
</tr>
</tbody>
</table>

\textit{(Notation used in Ω)}

\begin{verbatim}
1 \setbox0=\hbox{foo}
2 \the\wd0, \hbox{tate\vrule\the\wd0}
3 \setbox0=100pt
4 \the\wd0, \hbox{tate\the\wd0}
\end{verbatim}

10.0pt, 20.0pt, 100.0pt, 100.0pt

To access box dimensions \textit{with respect to current direction}, one has to use the following commands instead of \texttt{\wd} wtc.

\texttt{\ltjgetwd(num), \ltjgetht(num), \ltjgetdp(num)}

These commands return \textit{an internal dimension of }\texttt{box(num)} with respect to the current direction. One can use these in \texttt{\dimexpr} primitive, as the followings.

\begin{verbatim}
\dimexpr 2\ltjgetwd42-3pt\relax, \the\ltjgetwd1701
\end{verbatim}

The following is an example.

\begin{verbatim}
1 \parindent0pt
2 \setbox0=\hbox{よこぐみ}
3 \fboxsep=0mm \fbox{\copy0}
4 \vbox{\hsize=20mm \raggedleft}
5 \yoko YOKO \the\ltjgetwd32767, \the\ltjgetht32767, \the\ltjgetdp32767.
6 \tate TATE \the\ltjgetwd32767, \the\ltjgetht32767, \the\ltjgetdp32767.
7 \dtou DTOU \the\ltjgetwd32767, \the\ltjgetht32767, \the\ltjgetdp32767.
\end{verbatim}

\texttt{\ltjsetwd(num)=(dimen), \ltjsetht(num)=(dimen), \ltjsetdp(num)=(dimen)}

These commands set the dimension of \texttt{box(num)}. One does not need to group the argument \texttt{(num)}; four calls of \texttt{\ltjsetwd} below have the same meaning.

\begin{verbatim}
\ltjsetwd42 20pt, \ltjsetwd42=20pt, \ltjsetwd=42 20pt, \ltjsetwd=42=20pt
\end{verbatim}

6.2 Getting current direction

The \texttt{direction} parameter returns the current direction, and the \texttt{boxdir} parameter (with the argument \texttt{(num)}) returns the direction of a box register \texttt{box(num)}. The returned value of these parameters are a \textit{string}:

<table>
<thead>
<tr>
<th>Direction</th>
<th>yoko</th>
<th>tate</th>
<th>dtou</th>
<th>utod</th>
<th>(empty)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returned value</td>
<td>4</td>
<td>3</td>
<td>11</td>
<td>11</td>
<td>0</td>
</tr>
</tbody>
</table>
Table 11. Boxes in different direction

<table>
<thead>
<tr>
<th>typeset in yoko direction</th>
<th>typeset in tate or utod direction</th>
<th>typeset in dtou direction</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="https://via.placeholder.com/150" alt="Diagram" /></td>
<td><img src="https://via.placeholder.com/150" alt="Diagram" /></td>
<td><img src="https://via.placeholder.com/150" alt="Diagram" /></td>
</tr>
</tbody>
</table>

\[
W_y = h_y + d_y, \\
H_y = w_y, \\
D_y = 0 \text{pt}
\]

\[
W_T = h_T + d_T, \\
H_T = w_T/2, \\
D_T = w_T/2
\]

\[
W_D = h_D + d_D, \\
H_D = d_D, \\
D_D = h_D
\]

\[
W_D = w_D, \\
H_D = d_D, \\
D_D = h_D
\]

\[
W_D = w_D, \\
H_D = d_D, \\
D_D = h_D
\]

7 Redefined primitives by LuaT\TeX-jā

The following primitives are redefined by LuaT\TeX-jā (using \protected\def), for supporting Japanese typesetting and multiple directions:

\[
/ \quad \\text{tate math: 11}
\]

1. \texttt{\leavevmode\def\DIR{\ltjgetparameter{direction}}}
2. \texttt{\hbox{\yoko \DIR}, \hbox{\tate\DIR},}
3. \texttt{\hbox{\dtou\DIR}, \hbox{\utod\DIR},}
4. \texttt{\hbox{\tate$\hbox{tate math: \DIR}$}}
5. \texttt{\setbox2=\hbox{\tate}\ltjgetparameter{boxdir}[2]}

21
7.1 Suppressing redefinitions

Sometimes redefining primitives by \LaTeX-ja causes a problem. For example, the \texttt{breqn} package (v0.98k) assumes that \texttt{\vadjust} and \texttt{\insert} have their primitive meanings. So, this package cannot by loaded after \LaTeX-ja by default.

\LaTeX-ja version 20210517.0 has features for that problem. Namely:

- Primitives which is listed in \texttt{\ltj@stop@overwrite@primitive} are retain their meanings at just before loading \LaTeX-ja.
- After loading \LaTeX-ja, one can specify primitives to \texttt{\ltj@overwrite@primitive}, to redefine them by \LaTeX-ja.

See Figure 2 for an example.

8 Font Metric and Japanese Font

8.1 \texttt{\jfont}

To load a font as a Japanese font (for horizontal direction), you must use \texttt{\jfont} instead of \texttt{\font}, while \texttt{\jfont} admits the same syntax used in \texttt{\font}. \LaTeX-ja automatically loads \texttt{luaotfload} package, so TrueType/OpenType fonts with features can be used for Japanese fonts:

\begin{verbatim}
\jfont\tradmc={IPAexMincho:script=latn;+trad;-kern;jfm=ujis} at 14pt
\end{verbatim}

It is required to specify a (horizontal) JFM in at each calling of \texttt{\jfont}. A JFM is a Lua script which contains measurements of characters and glues/kerns that are automatically inserted for Japanese typesetting. The structure of JFM will be described in the next subsection.
Figure 3. Example of **jfmvar** key

| ブルー: jfm-ujis.lua, ブラック: jfm-jis.lua, レッド: jfm-min.lua |
| ある日モモちゃんのがお使いで迷子になって泣きました。 | ちょっと！何 | ちょっと!!何 |
| 漢つ | 漢つ | 漢つ |

Note that the defined control sequence (**\tradmc** in the example above) using **\jfont** is not a *font_def* token, but a macro. Hence the input like **\fontname\tradmc** causes a error. We denote control sequences which are defined in **\jfont** by **⟨jfont cs⟩**.

### Specifying JFM

The general scheme for specifying a JFM is the following:

```latex
\jfont⟨jfont_cs⟩=...;jfms={⟨JFM name⟩}/⟨{⟨JFM features⟩}⟩;...;jfmsvar=⟨identifier⟩};...\jfont
```

- **⟨JFM name⟩** The name of a (horizontal) JFM. LuaTeX-ja searches and loads jfm-{⟨JFM name⟩}.lua.
- **⟨JFM features⟩** An optional comma-separated list of JFM options. Enclosing braces ({}) are optional, but this does not escape any characters. The contents of this list can be accessed by a table `luatexja.jfont.jfm_feature` from a JFM, at its loading. See Figure 4 for an example.

Note that any JFM files which is shipped with LuaTeX-ja does not use this feature.

- **⟨identifier⟩** An optional string.

LuaTeX-ja “does not distinguish” two Japanese fonts which uses same JFM and are the same size. Here “uses same JFM” means that all of **⟨JFM name⟩**. **⟨JFM features⟩** and **⟨identifier⟩** of two fonts agree.

For example, The first “)” and “【” in Figure 3 are typeset in different real fonts. However, because they use the same JFMs and their size are same, LuaTeX-ja inserts penalties, glues and kerns as if these two character are typeset in a same font. Namely, the glue between these characters is halfwidth, as in “）【”. However, this does not applies with **\F** and **\H** in Figure 3, because their **⟨identifier⟩** are different.

### Horizontal JFMs

The following horizontal JFMs are shipped with LuaTeX-ja.

- **jfm-ujis.lua** A standard horizontal JFM of LuaTeX-ja. This file is based on `upnmlminr-h.tfm`, a metric for UTF/OTF package that is used in upTeX. When you are going to use the luatexja-otf package, you should use this JFM.
\begin{verbatim}
\small
\ltxsetparameter{differentjfm=both} \tablcolsep=.5\zw
% \printjfmfeat is defined in the source of this document
\jfont\A=HaranoAjiMincho-Regular\:jfm=test\ at 9pt \printjfmfeat\A
\jfont\B=HaranoAjiMincho-Bold\:jfm=test\:kern=0.5,-ps,+kana\ at 9pt \printjfmfeat\B
\jfont\C=HaranoAjiGothic-Regular\:jfm=test\:kern=0.5,down=0.2\ at 9pt \printjfmfeat\C
\jfont\D=HaranoAjiGothic-Bold\:jfm=test\:down=0.2,kern=0.5\ at 9pt \printjfmfeat\D
\def\TEST#1{\string#1&\string#1\&\string#1\&\string#1\}
\vspace{-4\baselineskip}\hfill\ttfamily
\begin{tabular}{lllll}
1 & \string\A&\string\B&\string\C&\string\D\
\TEST\A
\TEST\B
\TEST\C
\TEST\D
\\%
% No space between \"漢" and \"イ" iff two Japanese fonts uses same JFM
\ltxsetparameter{differentjfm=paverage}
\end{tabular}
\end{verbatim}

Figure 4. Example of JFM features

\textit{jfm-jis.lua} A counterpart for \texttt{jis.tfm}, "JIS font metric" which is widely used in \TeX. A major difference between \texttt{jfm-ujis.lua} and this \texttt{jfm-jis.lua} is that most characters under \texttt{jfm-ujis.lua} are square-shaped, while that under \texttt{jfm-jis.lua} are horizontal rectangles.

\textit{jfm-min.lua} A counterpart for \texttt{min10.tfm}, which is one of the default Japanese font metric shipped with \TeX.

\textit{jfm-prop.lua} A JFM for proportional typesetting. This JFM doesn't have any information of character dimension (width, height, depth), nor glues/kerns information.

\textit{jfm-propw.lua} Another JFM for proportional typesetting. In contrast to \texttt{jfm-prop.lua}, this JFM has informations of character height and depth.

See Table 12 for the difference among \texttt{jfm-ujis.lua}, \texttt{jfm-jis.lua}, \texttt{jfm-min.lua}.

\textbf{■ Using kerning information in a font} Some fonts have information for inter-glyph spacing. Lua\TeX\-ja 20140324.0 or later treats kerning spaces like an italic correction; any glue and/or kern from the JFM and a kerning space can coexist. See Figure 5 for detail.

Note that in \texttt{\setmainjfont} etc. which are provided by luatexja-fontspec package, kerning option is set \texttt{off} \texttt{(Kerning=-0ff)} by default, because of the compatibility with previous versions of Lua\TeX\-ja.

\textbf{■ extend and slant} The following setting can be specified as OpenType font features:

\begin{verbatim}
\jfont\E=HaranoAjiMincho-Regular\:extend=1.5;jfm=ujis;-kern
\jfont\S=HaranoAjiMincho-Regular\:slant=1;jfm=ujis;-kern
\end{verbatim}

Note that Lua\TeX\-ja doesn't adjust JFMs by these \texttt{extend} and \texttt{slant} settings; one have to write new JFMs on purpose. For example, the following example uses the standard JFM \texttt{jfm-ujis.lua}, hence the letterspacing and the width of italic corrections are not correct:

\begin{verbatim}
\jfont\E=HaranoAjiMincho-Regular\:extend=1.5;jfm=ujis;-kern
\jfont\S=HaranoAjiMincho-Regular\:slant=1;jfm=ujis;-kern
\end{verbatim}

\textbf{■ \ltxksp "feature"} \texttt{kanjiskip\_natural}, \texttt{kanjiskip\_stretch}, \texttt{kanjiskip\_shrink} keys (Page ??) makes the Lua\TeX\-ja inserts not only a glue which is specified by a JFM, and also the natural width/stretch part/shrink part of \texttt{kanjiskip}. This functionality is disabled by \texttt{-ltxksp} specification, as shown in Figure 6.
Figure 5. Kerning information and \texttt{kanjiskip}

Figure 6. \texttt{ltjksp “feature”}
8.2 \texttt{\texttt{tfont}}

\texttt{\texttt{tfont}} loads a font as a Japanese font for vertical direction. This command admits the same syntax as in \texttt{\texttt{jfont}} and \texttt{\texttt{jfont}}. A font defined by \texttt{\texttt{tfont}} differs the following points from that by \texttt{\texttt{jfont}}:

- OpenType Feature vrt2\textsuperscript{2} is automatically activated, unless vert and/or vrt2 features are explicitly activated or deactivated (as the second line in the example below).

\begin{verbatim}
\texttt{\texttt{tfont}}\texttt{S=HaranoAjiMincho-Regular:jfm=ujisv \% vrt2 is automatically activated
\texttt{\texttt{tfont}}\texttt{T=HaranoAjiMincho-Regular:jfm=ujisv;-vert \% vert and vrt2 are not activated
\texttt{\texttt{tfont}}\texttt{U=file:ipaexm.ttf:jfm=ujisv
% vert is automatically activated, since this font does not have vrt2
\end{verbatim}

- Sometimes vert and/or vrt2 are not activated while one specified activation of these feature. This is because the font does not define these features in current combination of script tag and language system identifier.

In this situation, Lua\TeX-xja performs all replacements which is defined in vert feature for some scripts for some languages.

- Furthermore, a glyph is automatically rotated 90 degrees, if it is not replaced by vert feature for any script for any language, and if it is marked as ‘r’ or “Tr” in UAX #50.

- \texttt{\texttt{tfont}} uses a vertical JFM instead of a horizontal JFM. Lua\TeX-xja ships following vertical JFMs:

\begin{itemize}
  \item \texttt{jfm-ujisv.lua} A standard vertical JFM in Lua\TeX-xja. This JFM is based on upnmlminr-v.tfm, a metric for UTF/OTF package that is used in up\TeX.
  \item \texttt{jfm-tmin.lua} A counterpart for tmin10.tfm, which is one of the default Japanese font metric shipped with \TeX.
\end{itemize}

- If vert and/or vrt2 features are activated, one can specify jpotf to additional substitutions. By default, it substitutes ideographic comma/period for fullwidth comma/period, and double prime quotation marks for double quotation marks (See Figure 7). One can customize substitutions by lua function \texttt{luatexja.jfont.register_vert_replace} (see Japanese version of this manual).

8.3 Default Japanese fonts and JFMs

If following commands are defined at loading Lua\TeX-xja package, these change default Japanese fonts and JFMs for them:

\begin{itemize}
  \item \texttt{\texttt{jfont}{X=\{HaranoAjiMincho-Regular.otf\}:jfm=ujis}
  \item \texttt{\texttt{jfont}{U=\{HaranoAjiMincho-Regular.otf\}:jfm=ujisv}
  \item \texttt{\texttt{jfont}{V=\{HaranoAjiMincho-Regular.otf\}:jfm=ujisv;jpotf}
\end{itemize}

If the font does not define the vrt2 feature, vert is used instead.

\begin{figure}[h]
\centering
\texttt{\texttt{\ttfamily{centering}}}\texttt{\texttt{\TEST\yoko\X \quad \TEST\tate\U \quad \TEST\tate\V}}
\texttt{”引用,と句読点.”}
\caption{jpotf “feature”}
\end{figure}
The default Japanese font for the mincho family.

\ltj@stdmcfont The default Japanese font for the mincho family.

\ltj@stdgtfont The default Japanese font for the gothic family.

\ltj@stdyokojfm The default JFM for horizontal direction.

\ltj@stdtatejfm The default JFM for vertical direction.

For example,
\def\ltj@stdmcfont{IPAMincho}
\def\ltj@stdgtfont{IPAGothic}

makes that IPA Mincho and IPA Gothic will be used as default Japanese fonts, instead of Harano Aji fonts.

This feature is intended for classes which use special JFMs\textsuperscript{9}. It is recommended to use \texttt{\luatexja-preset} or \texttt{\luatexja-fontspec} package to select standard fonts in ordinary \LaTeX sources.

For compatibility with earlier versions, Lua\TeX-ja reads \texttt{\luatexja.cfg} automatically if it is found by Lua\TeX. One should not overuse this \texttt{\luatexja.cfg}; it will overwrite the definition of \texttt{\ltj@stdmcfont} and others.

### 8.4 Prefix psft

Besides "file" and "name" prefixes which are introduced in the luaotfload package, Lua\TeX-ja adds "psft" prefix in \texttt{\jfont} (and \texttt{\font}), to specify a "name-only" Japanese font which will not be embedded to PDF. Note that these non-embedded fonts under current Lua\TeX has Identity-H encoding, and this violates the standard ISO32000-1:2008 ([10]).

OpenType font features, such as "\texttt{+jp90}", have no meaning in name-only fonts using "psft" prefix, because we can’t expect what fonts are actually used by the PDF reader. Note that extend and slant settings (see above) are supported with psft prefix, because they are only simple linear transformations.

[\texttt{\jfont\testJ={psft:Ryumin-Light:cid=Adobe-Japan1-7;jfm=jis}}] % Japanese

[\texttt{\jfont\testD={psft:Ryumin-Light:jfm=jis}}] % default: Adobe-Japan1-7

[\texttt{\jfont\testC={psft:AdobeMingStd-Light:cid=Adobe-CNS1-7;jfm=jis}}] % Traditional Chinese

[\texttt{\jfont\testB={psft:SimSun:cid=Adobe-GB1-5;jfm=jis}}] % Simplified Chinese

[\texttt{\jfont\testT={psft:Batang:cid=Adobe-Korean1-2;jfm=jis}}] % Korean

[\texttt{\jfont\testR={psft:SourceHanSerifAKR9:cid=Adobe-KR-9;jfm=jis}}] % Korean

Note that the code above specifies \texttt{jfm=jis.lua}, which is for Japanese fonts, as JFM for Chinese and Korean fonts.

At present, Lua\TeX-ja supports only 5 values written in the sample code above. Specifying other values, e.g.,

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

produces the following error:

\texttt{! Package luatexja Error: bad cid key `Adobe-Japan2'.}

\texttt{\par}

\texttt{?}

\texttt{?}

\texttt{I couldn’t find any non-embedded font information for the CID `Adobe-Japan2'. For now, I’ll use `Adobe-Japan1-6'. Please contact the Lua\TeX-ja project team.}

\texttt{?}

\textsuperscript{9}This is because commands has $@$ in their names.
8.5 Structure of a JFM file

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

```lua
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.

- **version**=⟨version⟩ (optional, default value is 1)
  The version JFM. Currently 1, 2, and, 3 are supported

- **dir**=⟨direction⟩ (required)
  The direction of JFM. ‘yoko’ (horizontal) or ‘tate’ (vertical) are supported.

- **zw**=⟨length⟩ (required)
  The amount of the length of the “full-width”.

- **zh**=⟨length⟩ (required)
  The amount of the “full-height” (height + depth).

- **kanjiskip**=⟨natural⟩, ⟨stretch⟩, ⟨shrink⟩} (optional)
  This field specifies the “ideal” amount of kanjiskip. As noted in Subsection 4.2, if the parameter kanjiskip is \texttt{\textbackslash maxdimen}, the value specified in this field is actually used (if this field is not specified in JFM, it is regarded as 0 pt). Note that ⟨stretch⟩ and ⟨shrink⟩ fields are in design-size unit too.

- **xkanjiskip**=⟨natural⟩, ⟨stretch⟩, ⟨shrink⟩} (optional)
  Like the kanjiskip field, this field specifies the “ideal” amount of xkanjiskip.

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

- **chars**=⟨character⟩, ...} (required except character class 0)
  This field is a list of characters which are in this character type $i$. This field is optional if $i = 0$, since all \texttt{JAchar} which do not belong any character classes other than 0 are in the character class 0 (hence, the character class 0 contains most of \texttt{JAchars}). In the list, character(s) can be specified in the following form:
  - a Unicode code point
  - the character itself (as a Lua string, like ‘あ’)
  - a string like ‘あ*’ (the character followed by an asterisk)
  - several “imaginary” characters (We will describe these later.)

- **width**=⟨length⟩, **height**=⟨length⟩, **depth**=⟨length⟩, **italic**=⟨length⟩ (required)
  Specify the width of characters in character class $i$, the height, the depth and the amount of italic correction. All characters in character class $i$ are regarded that its width, height, and depth are as values of these fields. The default values are shown in Table 13.

### Table 13. Default values of width field and other fields

<table>
<thead>
<tr>
<th>Direction of JFM</th>
<th>'yoko' (horizontal)</th>
<th>'tate' (vertical)</th>
</tr>
</thead>
<tbody>
<tr>
<td>width field</td>
<td>the width of the “real” glyph</td>
<td>0.0</td>
</tr>
<tr>
<td>height field</td>
<td>the height of the “real” glyph</td>
<td>0.0</td>
</tr>
<tr>
<td>depth field</td>
<td>the depth of the “real” glyph</td>
<td>0.0</td>
</tr>
<tr>
<td>italic field</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>
Consider a Japanese character node which belongs to a character class whose the align field is 'middle'.

- The black rectangle is the imaginary body 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) first.
- 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 8. The position of the real glyph (horizontal Japanese fonts)

(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 8 and Figure 9.

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=⟨ratio⟩]⟩}, ...

glue={⟨j⟩=⟨(width), ⟨stretch⟩, ⟨shrink⟩, [ratio=⟨ratio⟩], ...⟩}, ...

Specifies the amount of kern or glue which will be inserted between characters in character class i and those in character class j.

⟨ratio⟩ specifies how much the glue is originated in the “right” character. It is a real number between 0 and 1, and treated as 0.5 if omitted. For example, The width of a glue between an ideographic full stop “。” and a fullwidth middle dot “・” is three-fourth of fullwidth, namely halfwidth from the ideographic full stop, and quarter-width from the fullwidth middle dot. In this case, we specify ⟨ratio⟩ to 0.25/(0.5 + 0.25) = 1/3.

In case of glue, one can specify following additional keys in each [j] subtable:

priority=⟨priority⟩ An integer in [−4, 3] (treated as 0 if omitted), or a pair of these integers {⟨(stretch), ⟨shrink⟩⟩} (version 2 or later). This is used only in line adjustment with priority by luatexja-adjust (see Subsection 13.3). Higher value means the glue is easy to stretch, and is also easy to shrink.

kanjiskip natural=⟨num⟩, kanjiskip stretch=⟨num⟩, kanjiskip shrink=⟨num⟩

These keys specifies the amount of the natural width of kanjiskip (the stretch/shrink part, respectively) which will be inserted in addition to the original JFM glue. Default values of them are all 0.

As an example, in jfm-ujis.lua, the standard JFM in horizontal writing, we have
Between an ordinal letter “あ” and an ideographic opening bracket, we have a glue whose natural part and shrink part are both half-width, while its stretch part is zero. However, this glue also can be stretched as much as the stretch part of \texttt{kanjiskip} times the value of \texttt{kanjiskip\_stretch} key (1 in this case).

Between an ideographic closing brackets (including the ideographic comma “，”) and an ordinal letter (including an \texttt{ALchar} “f”), we have the same glue. Again, this glue also can be stretched as much as the stretch part of \texttt{kanjiskip} times the value of \texttt{kanjiskip\_stretch} key (1 in this case).

Between an ideographic opening bracket and an ordinal letter and between an ordinal letter and an ideographic closing bracket, we have a glue whose natural part and stretch part are both zero, while its shrink part as much as the shrink part of \texttt{kanjiskip}.

Hence we have the following result:

\begin{verbatim}
\leavevmode\let\vrule\vrule
\let\jsetparameter{\setparameter}
\vrule\hbox spread 7.0zw{あ「い」「う,えお」f}\vrule
\vrule\hbox{あ「い」「う,えお」f}\vrule\par
\vrule\hbox spread -2.5zw{あ「い」「う,えお」f}\vrule
\end{verbatim}

\vrule\hbox spread 7.0zw{あ「い」「う,えお」f}\vrule
\vrule\hbox{あ「い」「う,えお」f}\vrule\par
\vrule\hbox spread -2.5zw{あ「い」「う,えお」f}\vrule

end\_stretch=(kern), end\_shrink=(kern) (optional, version 1 only)
end\_adjust={\langle(kern), (kern), ...\rangle} (optional, version 2 or later)
round\_threshold={\langle(float)\rangle} (optional, version 3 or later, only available in character class 0)

\section*{Character to character classes}

We explain how the character class of a character is determined, using \texttt{jfm-test.lua} which contains the following:

\begin{verbatim}
[0] = {
  chars = { '漢' },
  align = 'left', left = 0.0, down = 0.0,
  width = 1.0, height = 0.88, depth = 0.12, italic=0.0,
},
[2000] = {
  chars = { 'ヒ', 'ト' },
  align = 'left', left = 0.0, down = 0.0,
  width = 0.5, height = 0.88, depth = 0.12, italic=0.0,
},
\end{verbatim}

Now consider the following input/output:

\begin{verbatim}
\jfont\a=IPAexMincho;jfm-test++;hwidth
\setbox0=\hbox{\a ヒ漢}\the\wd0
\end{verbatim}

Now we look why the above source outputs 15 pt.

1. The character “ヒ” is converted to its half width form “ヒ” by \texttt{hwidth} feature.
2. According to the JFM, the character class of “ヒ” is 2000, hence its width is halfwidth.
3. The character class of “漢” is zero, hence its width is fullwidth.
4. Hence the width of \texttt{hbox} equals to 15 pt.

This example shows that the character class of a character is generally determined \textit{after applying font features by luaotfload}.

However, if the class determined by the glyph after application of features is zero, LuaTeX-X-ja adopts the class determined by the glyph \textit{before} application of features. The following input is an example.
### Imaginary characters
As described before, one can specify several “imaginary characters” in \texttt{chars} field. The most of these characters are regarded as the characters of class 0 in \LaTeX. As a result, \LaTeX-ja can control typesetting finer than \LaTeX. The following is the list of imaginary characters:

- \texttt{'boxbdd'}
  The beginning/ending of a hbox, and the beginning of a noindented (i.e., began by \texttt{\noindent}) paragraph. If a hbox begins (resp. ends) a glue or kern between this “character” and a JAchar, JAglue won’t be inserted before (resp. after) the hbox. \texttt{kanjiskip} and \texttt{xkanjiskip} around a hbox.

- \texttt{'parbdd'}
  The beginning of an (indented) paragraph.

- \texttt{'jcharbdd'}
  A boundary between JAchar and anything else.

- \texttt{'alchar', 'nox_alchar'}
  (version 3 or later) A boundary between JAchar and ALchar.

- \texttt{'glue'}
  (version 3 or later) A boundary between JAchar, and, a glue or kern.

\[-1\text{ The left/right boundary of an inline math formula.}\]

#### Porting JFM from \LaTeX
See Japanese version of this manual.

### 8.6 Math font family
\LaTeX handles fonts in math formulas by 16 font families\footnote{Omega, Aleph, \LaTeX and \(\varepsilon\) \LaTeX can handle 256 families, but an external package is needed to support this in plain \LaTeX and \LaTeX.}, and each family has three fonts: \texttt{\textfont}, \texttt{\scriptfont} and \texttt{\scriptscriptfont}.

\LaTeX-ja’s handling of Japanese fonts in math formulas is similar; Table 14 shows counterparts to \LaTeX’s primitives for math font families. There is no relation between the value of \texttt{\fam} and that of \texttt{\jfam}; with appropriate settings, one can set both \texttt{\fam} and \texttt{\jfam} to the same value. Here \texttt{(jfont_cs)} in the argument of \texttt{jatextfont} etc. is a control sequence which is defined by \texttt{jfont}, i.e., a horizontal Japanese font.
8.7 Callbacks

LuaTeX-ja also has several callbacks. These callbacks can be accessed via \texttt{luatexbase.add_to_callback} function and so on, as other callbacks.

\textbf{luatexja.load\_jfm callback}
With this callback, one can overwrite JFMs. This callback is called when a new JFM is loaded.

\begin{verbatim}
function (\texttt{table} \texttt{jfm\_info}, \texttt{string} \texttt{jfm\_name})
  return \texttt{table} \texttt{new\_jfm\_info}
end
\end{verbatim}

The argument \texttt{jfm\_info} contains a table similar to the table in a JFM file, except this argument has \texttt{chars} field which contains character codes whose character class is not 0.

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

\textbf{luatexja.define\_jfont callback}
This callback and the next callback form a pair, and you can assign characters which do not have fixed code points in Unicode to non-zero character classes. This \texttt{luatexja.define\_font} callback is called just when new Japanese font is loaded.

\begin{verbatim}
function (\texttt{table} \texttt{jfont\_info}, \texttt{number} \texttt{font\_number})
  return \texttt{table} \texttt{new\_jfont\_info}
end
\end{verbatim}

\texttt{jfont\_info} has the following fields, \textit{which may not overwritten by a user}:

- \texttt{size} The font size specified at \texttt{\jfont} in scaled points ($1 \text{ sp} = 2^{-16} \text{ pt}$).
- \texttt{zw}, \texttt{zh}, \texttt{kanjiskip}, \texttt{xkanjiskip} These are scaled value of those specified by the JFM, by the font size.
- \texttt{jfm} The internal number of the JFM.
- \texttt{var} The value of \texttt{jfmvar} key, which is specified at \texttt{\jfont}. The default value is the empty string.
- \texttt{chars} The mapping table from character codes to its character classes.
  The specification \texttt{[i].chars={\langle character\rangle, \ldots}} in the JFM will be stored in this field as \texttt{chars={\langle character\rangle=i, \ldots}}.
- \texttt{char\_type} For \texttt{i} ∈ \texttt{ω}, \texttt{char\_type[i]} is information of characters whose class is \texttt{i}, and has the following fields:
  - \texttt{width}, \texttt{height}, \texttt{depth}, \texttt{italic}, \texttt{down}, \texttt{left} are just scaled value of those specified by the JFM, by the font size.
  - \texttt{align} is a number which is determined from \texttt{align} field in the JFM:
    \begin{align*}
    &1 \quad ('right' in JFM), \\
    &0.5 \quad ('middle' in JFM), \\
    &0 \quad (otherwise).
    \end{align*}

  For \texttt{i}, \texttt{j} ∈ \texttt{ω}, \texttt{char\_type[i][j]} stores a kern or a glue which will be inserted between character class \texttt{i} and class \texttt{j}.

The returned table \texttt{new\_jfont\_info} also should include these fields, but you are free to add more fields (to use them in the \texttt{luatexja.find\_char\_class} callback). The \texttt{font\_number} is a font number.

A good example of this and the next callbacks is the \texttt{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.

\textbf{luatexja.find\_char\_class callback}
This callback is called just when LuaTEx-ja is trying to determine which character class a character \texttt{chr\_code} belongs. A function used in this callback should be in the following form:
function (num char_class, table jfont_info, num chr_code)
    if char_class~=0 then return char_class
    else
        ....
        return (num new_char_class or 0)
    end
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.

function (table shift_info, table jfont_info, table char_type)
    return table new_shift_info
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 Open-Type 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

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

9 Parameters

9.1 \ltjsetparameter

As described before, \ltjsetparameter and \ltjglobalsetparameter are commands for accessing most parameters of LuaTEX-ja. One of the main reason that LuaTEX-ja didn’t adopted the syntax similar to that of p\TeX{} (e.g., \prebreakpenalty\n{10000}) is the position of hpack_filter callback in the source of LuaTEX, see Section 14.

\ltjsetparameter and \ltjglobalsetparameter are commands for assigning parameters. These take one argument which is a key-value list. The difference between these two commands is the scope of assignment; \ltjsetparameter does a local assignment and \ltjglobalsetparameter does a global one by default. They also obey the value of \globaldefs, like other assignments.

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

- “∗”: values at the end of a paragraph or a hbox are adopted in the whole paragraph or the whole hbox.
- “†”: assignments are always global.

jcharwidowpenalty = (penalty)∗ [\jcharwidowpenalty]
Penalty value for suppressing orphans. This penalty is inserted just after the last JAchar which is not regarded as a (Japanese) punctuation mark.
\texttt{kcatcode =\{(chr\_code),\{natural\_number\}\}^*}

An additional attribute which each character whose character code is \langle \text{chr\_code} \rangle has. At version 20120506.0 or later, the lowermost bit of \langle \text{natural\_number} \rangle indicates whether the character is considered as a punctuation mark (see the description of \texttt{jcharwidowpenalty} above).

\texttt{prebreakpenalty =\{(chr\_code),\{penalty\}\}^* \texttt{[prebreakpenalty]}}

Set a penalty which is inserted automatically before the character \langle \text{chr\_code} \rangle, to prevent a line starts from this character. For example, a line cannot started with one of closing brackets “]”, so \LaTeX-ja sets

\begin{verbatim}
\ltjsetparameter{prebreakpenalty={`]},10000}}
\end{verbatim}

by default.

\TeX has following restrictions on \texttt{\prebreakpenalty} and \texttt{\postbreakpenalty}, but they don’t exist in \LaTeX-ja:

- Both \texttt{\prebreakpenalty} and \texttt{\postbreakpenalty} cannot be set for the same character.
- We can set \texttt{\prebreakpenalty} and \texttt{\postbreakpenalty} up to 256 characters.

\texttt{postbreakpenalty =\{(chr\_code),\{penalty\}\}^* \texttt{[postbreakpenalty]}}

Set a penalty which is inserted automatically after the character \langle \text{chr\_code} \rangle, to prevent a line ends with this character.

\texttt{jatextfont =\{(jfam),\{jfont\_cs\}\}^* \texttt{[\textfont in \TeX]}}

\texttt{jascriptfont =\{(jfam),\{jfont\_cs\}\}^* \texttt{[\scriptfont in \TeX]}}

\texttt{javascriptscriptfont =\{(jfam),\{jfont\_cs\}\}^* \texttt{[\scriptscriptfont in \TeX]}}

\texttt{yjabaselineshift =\{\dimen\}}

\texttt{yalbaselineshift =\{\dimen\} \texttt{[\ybaselineshift]}}

\texttt{tjabaselineshift =\{\dimen\}}

\texttt{talbaselineshift =\{\dimen\} \texttt{[\tbaselineshift]}}

\texttt{jaxspmode =\{(chr\_code),\{mode\}\}^*}

Set whether inserting \texttt{xkanjiskip} is allowed before/after a \texttt{JAchar} whose character code is \langle \text{chr\_code} \rangle. The followings are allowed for \langle \text{mode} \rangle:

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

This parameter is similar to the \texttt{\inhibitxspcode} primitive of \TeX, but not compatible with \texttt{\inhibitxspcode}.

\texttt{alxspmode =\{(chr\_code),\{mode\}\}^* \texttt{[\xspcode]}}

Set whether inserting \texttt{xkanjiskip} is allowed before/after a \texttt{ALchar} whose character code is \langle \text{chr\_code} \rangle. The followings are allowed for \langle \text{mode} \rangle:

- **0, inhibit** Insertion of \texttt{xkanjiskip} is inhibited before the character, nor after the character.
- **1, preonly** Insertion of \texttt{xkanjiskip} is allowed before the character, but not after.
- **2, postonly** Insertion of \texttt{xkanjiskip} is allowed after the character, but not before.
- **3, allow** Insertion of \texttt{xkanjiskip} is allowed both before the character and after the character. This is the default value.
Note that parameters \texttt{jaxspmode} and \texttt{alxspmode} share a common table, hence these two parameters are synonyms of each other.

\begin{verbatim}
autospacing = (bool) \{\[autospacing\]
autoxspacing = (bool) \{\[autoxspacing\]
kanjiskip = \langle \text{skip} \rangle \* \[\kanjiskip\]
    The default glue which inserted between two \texttt{JAchar}s. Changing current Japanese font does not alter this parameter, as \TeX.
    If the natural width of this parameter is \texttt{\maxdimen}, Lua\TeX\-ja uses the value which is specified in the JFM for current Japanese font (See Subsection 8.5).
\xkanjiskip = \langle \text{skip} \rangle \* \[xkanjiskip\]
    The default glue which inserted between a \texttt{JAchar} and an \texttt{ALchar}. Changing current font does not alter this parameter, as \TeX.
    As \texttt{kanjiskip}, if the natural width of this parameter is \texttt{\maxdimen}, Lua\TeX\-ja uses the value which is specified in the JFM for current Japanese font (See Subsection 8.5).
differentjfm = \langle \text{mode} \rangle^†
    Specify how glues/kerns between two \texttt{JAchar}s whose JFM (or size) are different. The allowed arguments are the followings:
        \texttt{average}, \texttt{both}, \texttt{large}, \texttt{small}, \texttt{pleft}, \texttt{pright}, \texttt{paverage}
    The default value is \texttt{paverage}. ...
\jacharrange = \langle \text{ranges} \rangle
\kansujichar = \{\langle \text{digit} \rangle, \langle \text{chr_code} \rangle\} \* \[\kansujichar\]
direction = \langle \text{dir} \rangle (always local)
    Assigning to this parameter has the same effect as \texttt{\yoko} (if \langle \text{dir} \rangle = 4), \texttt{\tate} (if \langle \text{dir} \rangle = 3), \texttt{\dtou} (if \langle \text{dir} \rangle = 1) or \texttt{\utod} (if \langle \text{dir} \rangle = 11). If the argument \langle \text{dir} \rangle is not one of 4, 3, 1 nor 11, the behavior of this assignment is undefined.
\end{verbatim}

9.2 \texttt{\ltjgetparameter}

\texttt{\ltjgetparameter} is a control sequence for acquiring parameters. It always takes a parameter name as first argument.

\begin{verbatim}
\ltjgetparameter{differentjfm},
\ltjgetparameter{autospacing},
\ltjgetparameter{kanjiskip},
\ltjgetparameter{prebreakpenalty}{`}
\end{verbatim}

The return value of \texttt{\ltjgetparameter} is always a string, which is outputted by \texttt{tex.write()}. Hence any character other than space “ ” (0x0020) has the category code 12 (other), while the space has 10 (space).

- If first argument is one of the following, no additional argument is needed.
    \begin{itemize}
    \item \texttt{\ltjgetparameter{\charwidowpenalty}}, \texttt{\yjabaselineshift}, \texttt{\yalbaselineshift}, \texttt{\autospacing}, \texttt{\autoxspacing}, \texttt{\kanjiskip}, \texttt{\xkanjiskip}, \texttt{\differentjfm}, \texttt{\direction}
    \end{itemize}
    Note that \texttt{\ltjgetparameter{\autospacing}} and \texttt{\ltjgetparameter{\autoxspacing}} returns 1 or 0, not true nor false.

- If first argument is one of the following, an additional argument—a character code, for example—is needed.
    \texttt{\kcatcode}, \texttt{\prebreakpenalty}, \texttt{\postbreakpenalty}, \texttt{\jaxspmode}, \texttt{\alxspmode}

35
\ltjgetparameter{jaxspmode}{...} and \ltjgetparameter{alxspmode}{...} returns 0, 1, 2, or 3, instead of preonly etc.

- \ltjgetparameter{jacharrange}{\langle range\rangle} returns 0, 1, 2, or 3, instead of preonly etc. Although there is no character range \(-1\), specifying \(-1\) to \langle range\rangle does not cause an error (returns 1).

- For an integer \langle digit\rangle between 0 and 9, \ltjgetparameter{kansujichar}{\langle digit\rangle} returns the character code of the result of \kansuji{\langle digit\rangle}.

- \ltjgetparameter{adjustdir} returns a integer which represents the direction of the surrounding vertical list. As direction, the return value 1 means dtou direction, 3 means tate direction (vertical typesetting), and 4 means yoko direction (horizontal typesetting).

- For an integer \langle reg num\rangle between 0 and 65535, \ltjgetparameter{boxdir}{\langle reg num\rangle} returns the direction of \box{\langle reg num\rangle}. If this box register is void, the returned value is zero.

- The following parameter names cannot be specified in \ltjgetparameter.
  jatextfont, jascriptfont, jascriptscriptfont, jacharrange

- \ltjgetparameter{chartorange}{\langle chr code\rangle} returns the range number which \langle chr code\rangle belongs to (although there is no parameter named "chartorange"). If \langle chr code\rangle is between 0 and 127, this \langle chr code\rangle does not belong to any character range. In this case, \ltjgetparameter{chartorange}{\langle chr code\rangle} returns \(-1\).

  Hence, one can know whether \langle chr code\rangle is JChar or not by the following:

  \ltjgetparameter{jacharrange}{\ltjgetparameter{chartorange}{\langle chr code\rangle}}

  \% 0 if JChar, 1 if ALChar

- Because the returned value is string, the following conditionals do not work if kanjiskip (or xkanjiskip) has the stretch part or the shrink part.

  \ifdim\ltjgetparameter{kanjiskip}>\z@ ... \fi
  \ifdim\ltjgetparameter{xkanjiskip}>\z@ ... \fi

  The correct way is using a temporary register.

  \@tempskipa=\ltjgetparameter{kanjiskip} \ifdim@tempskipa>\z@ ... \fi
  \@tempskipa=\ltjgetparameter{xkanjiskip} \ifdim@tempskipa>\z@ ... \fi

\section*{9.3 Alternative Commands to \ltjsetparameter}

The basic method to set parameters of Lua\TeX\-ja is to use \ltjsetparameter or \ltjglobalsetparameter. However, these commands are slow, because they parse a key-value list, so several alternative commands are used in Lua\TeX\-ja. \textit{This subsection is not for general Lua\TeX\-ja users.}

\begin{itemize}
  \item **Setting kanjiskip or xkanjiskip** In \ltjclasses, every size-changing command such as \Large changes \kanjiskip and \xkanjiskip. But a simple implementation, as the code below, is slow since two key-value lists are parsed by \ltjsetparameter:

  \begin{verbatim}
  \ltjsetparameter{kanjiskip=0\zw plus .1\zw minus .01\zw}
  \@tempskipa=\ltjgetparameter{xkanjiskip}
  \ifdim@tempskipa>\z@ \if@slide
  \ltjsetparameter{xkanjiskip=0.1em}
  \else
  \ltjsetparameter{xkanjiskip=0.25em plus 0.15em minus 0.06em}
  \fi
  \fi
  \end{verbatim}

  Hence, Lua\TeX\-ja defines more primitive commands, namely \ltjsetpar@global, \ltjsetkanjiskip, and \ltjsetxkanjiskip. Here
and \@setpar\global\@setkanjiskip 10pt has the same effect. The actual code of \ltxclasses is shown below:
\begin{verbatim}
\@setpar\global
\setkanjiskip{\z@ plus .1\zw minus .01\zw}
@tempskipa=\getparameter{xkanjiskip}
\ifdim\@tempskipa>\z@
\if@slide
\setxkanjiskip.1em
\else
\setxkanjiskip.25em plus .15em minus .06em
\fi
\fi
\end{verbatim}

Note that using \setkanjiskip or \setxkanjiskip alone, that is, without executing \@setpar\global in advance, is not supported.

10 Other Commands for plain \TeX\ and \LaTeX\ $2\epsilon$

10.1 Commands for compatibility with \p\TeX\ $2\epsilon$

The following commands are implemented for compatibility with \p\TeX. Note that the former five commands don’t support JIS X 0213, but only JIS X 0208. The last \kansuji converts an integer into its Chinese numerals.

\begin{verbatim}
\kuten, \jis, \euc, \sjis, \ucs, \kansuji
\end{verbatim}

These six commands takes an internal integer, and returns a string.

\begin{verbatim}
\newcount\hoge
\hoge=2423 %
\the\hoge, \kansuji\hoge\kansuji
\jis\hoge, \char\jis\hoge\jis\hoge\jis
\kansuji1701\kansuji
\end{verbatim}

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

\begin{verbatim}
\setparameter{kansujichar={1,`壹}}
\setparameter{kansujichar={7,`漆}}
\setparameter{kansujichar={0,`零}}
\kansuji1701
\end{verbatim}

10.2 \inhibitglue, \disinhibitglue

\inhibitglue suppresses the insertion of a glue/kern specified in JFM at the place. 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".

\begin{verbatim}
\jfont\g=HaranoAjiMincho-Regular:jfm-test \g
\fbox{\hbox{あウあ}}\inhibitglue
\inhibitglue あ
\par\inhibitglue あ
\par\noindent\inhibitglue
\par\noindent あ
\par\rule{0pt}{\ht\strutbox}\inhibitglue
\end{verbatim}

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

- The call of \inhibitglue in the (internal) vertical mode is simply ignored.
- \inhibitglue does not suppress \kanjiskip or \xkanjiskip.
• 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.

\disinhibitglue suppresses the effect of \inhibitglue. In other words, \disinhibitglue allows the insertion of a glue/kern specified by JFM. If \inhibitglue and \disinhibitglue both specified at the same time, the latest one is effective. This command is added in the version 20201224.0.

Note that \disinhibitglue also cancels ligatures and kernings.

10.3 \ltjfakeboxbdd, \ltjfakeparbegin

Sometimes 'parbdd' and 'boxbdd' specifications look like "fail", especially in paragraphs inside list environments. This is because \everypar inserts some nodes such as boxes and kerns, so the "first letter" in a paragraph is in fact not the first letter.

```
\parindent1\zw
\noindent ああああああああ
% for comparison
「ああああああ
par % normal paragraph
「ああああああ
\everypar{\null}
「ああああああ
par % ???
```

\ltjfakeboxbdd and \ltjfakeparbegin primitives resolve this situation.

• \ltjfakeparbegin creates a node which indicates "beginning of an indented paragraph" to the insertion process of JAglue.

• \ltjfakeboxbdd creates a node which indicates "beginning/ending of a box" to the insertion process of JAglue.

As an example, the example above can be improved as follows:

```
\parindent1\zw
\noindent ああああああああ
% for comparison
「ああああああ
par % normal paragraph
「ああああああ
\everypar{\null\ltjfakeparbegin}
「ああああああ
par % ???
```

10.4 \insertxkanjiskip, \insertkanjiskip

There are some situations which one wants to insert xkanjiskip manually. A simple approach is to use \hskip{\ltjgetparameter{xkanjiskip}}, but this approach has several weak points. To cope with these weak points, LuaTeX-ja defines a command \insertxkanjiskip which inserts xkanjiskip glue manually, from the version 20201224.0.

• "\insertxkanjiskip" (without any keyword) uses the value of xkanjiskip at the place.

• "\insertxkanjiskip late" (with "late" keyword) uses the value of xkanjiskip at the end of a paragraph/hbox.

See the example below.

```
\ltjgetparameter{xkanjiskip=0.25}\zw
あ (% 0.5\zw (from JFM)
あ\insertxkanjiskip (% 0.25\zw (xkanjiskip at here)
あ\insertxkanjiskip late (% 0.25\zw (xkanjiskip at EOP)
\% 1.25\zw (xkanjiskip at EOP)
\%
\ltjgetparameter{xkanjiskip=1.25}\zw
あ\insertxkanjiskip (% 1.25\zw (xkanjiskip at here)
あ\insertxkanjiskip (% 1.25\zw (xkanjiskip at EOP)
%% At the end of the paragraph (EOP), xkanjiskip is 1.25\zw.
```

38
There is a similar command \insertkanjiskip (\kanjiskip instead of \xkanjiskip) is also defined. Note that any shorthand form of \insert[x]kanjiskip are not defined by Lua\TeX-ja.

10.5 \ltjdeclarealtfont

Using \ltjdeclarealtfont, one can “compose” more than one Japanese fonts. This \ltjdeclarealtfont uses in the following form:

\ltjdeclarealtfont\langle base_font_cs\rangle\langle alt_font_cs\rangle\{\langle range\rangle\}

where \langle base_font_cs\rangle and \langle alt_font_cs\rangle are defined by \jfont. Its meaning is

If the current Japanese font is \langle base_font_cs\rangle, characters which belong to \langle range\rangle is typeset by another Japanese font \langle alt_font_cs\rangle, instead of \langle base_font_cs\rangle.

Here \langle range\rangle is a comma-separated list of character codes, but also accepts negative integers: $-n \ (n \geq 1)$ means that all characters of character classes $n$, with respect to JFM used by \langle base_font_cs\rangle. Note that characters which do not exist in \langle alt_font_cs\rangle are ignored.

For example, if \hoge uses jfm-ujis.lua, the standard JFM of Lua\TeX-ja, then

\ltjdeclarealtfont\hoge\piyo\{"3000--30FF, {-1}--{-1}\}

does

If the current Japanese font is \hoge, U+3000–U+30FF and characters in class 1 (ideographic opening brackets) are typeset by \piyo.

Note that specifying negative numbers needs specification like \{-1}--{-1}, because simple “-1” is treated as the range between 0 and 1.

11 Commands for \LaTeXe

11.1 Loading Japanese fonts in \LaTeXe

From version 20190107, Lua\TeX-ja does not load Japanese fonts for horizontal direction and that for vertical direction at same time, to reduce the number of loaded fonts. This will save time for typesetting and memory consumption of Lua side ([11]).

- \selectfont loads (and chooses) only the Japanese font for the current direction, and does not load the Japanese font for other direction (Lua\TeX-ja only detects its size and JFM, to calculate the amount of shifting the baseline).

- Direction changing commands (\yoko, \tate, \dtou, \utod) are patched to include the following process:

If the Japanese font for new direction is not loaded, Lua\TeX-ja loads it automatically.

Original commands are saved as \ltj@@orig@yoko etc.

- Specifying Japanese font command which is defined by \jfont, \tfont, or \DeclareFixedFont directly actually loads (and selects) the Japanese font. For example, \jachar\s in \box0 will be typeset in \HOGE, in the following code:

\begin{verbatim}
\setbox0=\hbox{\tateあいう}
\end{verbatim}

% in horizontal direction (\yoko)
\DeclareFixedFont\HOGE{JT3}{gt}{m}{n}{12} % JT3: for vertical direction
\HOGE

11.2 Patch for NFSS2

Japanese patch for NFSS2 in LuaTeX-ja is based on p1fonts.dtx which plays the same role in p\TeX 2\epsilon.

We will describe commands which are not described in Subsection 3.1.

additional dimensions

Like \TeX \epsilon, LuaTEX-ja defines the following dimensions for information of current Japanese font:

\cht
\cdp
\chT
\cws
\cvs
\cbs

and its \normalsize version:

\Cht
\Cdp
\Cws
\Cvs
\Chs

Note that \cws and \Cws may differ from \zw and \zh respectively. On the one hand the former
dimensions are determined from a character whose character class is zero, but on the other hand
\zw and \zh are specified by JFM.

\DeclareYokoKanjiEncoding{{encoding}}{{text-settings}}{{math-settings}}
\DeclareTateKanjiEncoding{{encoding}}{{text-settings}}{{math-settings}}

In NFSS2 under LuaTeX-ja, distinction between alphabetic fonts and Japanese fonts are only made by
their encodings. For example, encodings OT1 and T1 are encodings for alphabetic fonts, and Japanese
fonts cannot have these encodings. These command define a new encoding scheme for Japanese font
families.

\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}}
\SetRelationFont

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

\userelfont

(Only) at the next call of \selectfont, 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.

The following is an example of \SetRelationFont and \userelfont:

\makeatletter
\SetRelationFont{JY3}{\k@family}{m}{n}{TU}{lmss}{m}{n}
% \k@family: current Japanese font family
\userelfont
\selectfontあいう abc
\adjustbaseline

In p\TeX 2\epsilon, \adjustbaseline sets $\baselineshift$ to match the vertical center of ”M” and that of
”漢” in vertical typesetting:

$$\baselineshift \leftarrow \frac{(h_M + d_M) - (h_å + d_å)}{2} + d_å - d_M,$$

where $h_a$ and $d_a$ denote the height of ”a” and the depth, respectively. In LuaTeX-ja, this
\adjustbaseline does similar task, namely setting the \talbaselineshift parameter (a Japanese character
whose character class is zero is used, instead of ”漢”).
日本国民は、正当に選挙された国会における代表者を通じて行動し、……

Figure 10. An example of \DeclareAlternateKanjiFont

\fontfamily{(family)}
As in \TeX\, this command changes current font family (alphabetic, Japanese, or both) to \texttt{(family)}. See Subsection 11.3 for detail.

\fontshape{(shape)}\textbf{,} \fontshapeforce{(shape)}
As in \TeX\, this command changes current alphabetic font shape according to shape change rules. Traditionally, \texttt{\fontshape} changes also current Japanese font shape always. However, this leads a lot of \TeX\ font warning like

\begin{verbatim}
Font shape `JY3/mc/m/it' undefined using `JY3/mc/m/n' instead on ....
\end{verbatim}

when \texttt{\itshape} is called, because almost all Japanese fonts only have shape "n", and \texttt{\itshape} calls \texttt{\fontshape}.

Lua\TeX\-ja 20200323.0 change the behavior. Namely, \texttt{\fontshape{(shape)}} and \texttt{\fontshapeforce{(shape)}} change current Japanese font shape, only if the required shape (according to shape changing rules) or \texttt{(shape)} is available in current Japanese font family/series. When this is not the case, an info such as

\begin{verbatim}
Kanji font shape JY3/mc/m/it' undefined No change on ....
\end{verbatim}

is issued instead of a warning.

\texttt{\kanjishape{(shape)}}\textbf{,} \texttt{\kanjishapeforce{(shape)}}
\texttt{\kanjishape{(shape)}} changes current Japanese font shape according to shape change rules, and \texttt{\kanjishapeforce{(shape)}} changes current Japanese font shape to \texttt{(shape)}, regardless of the rules. Hence \texttt{\kanjishape{it}} produces a warning

\begin{verbatim}
Font shape `JY3/mc/m/it' undefined using `JY3/mc/m/n' instead on ....
\end{verbatim}

which is not produced by \texttt{\fontshape{it}}.

\texttt{\DeclareAlternateKanjiFont{\texttt{(base-encoding)}}{\texttt{(base-family)}}{\texttt{(base-series)}}{\texttt{(base-shape)}}
\texttt{\{\texttt{(alt-encoding)}\texttt{\{\texttt{(alt-family)}\texttt{\{\texttt{(alt-series)}\texttt{\{\texttt{(alt-shape)}\texttt{\texttt{(range)}}}
As \texttt{\ltjdeclarealtfont} (Subsection 10.5), characters in \texttt{(range)} of the Japanese font (we say the base font) which specified by first 4 arguments are typeset by the Japanese font which specified by fifth to eighth arguments (we say the alternate font). An example is shown in Figure 10.

- In \texttt{\ltjdeclarealtfont}, the base font and the alternate font must be already defined. But this \texttt{\DeclareAlternateKanjiFont} is not so. In other words, \texttt{\DeclareAlternateKanjiFont} is effective only after current Japanese font is changed, or only after \texttt{\selectfont} is executed.
- ...

41
Furthermore, Lua\TeX-ja applies patches which enables NFSS2 commands, such as `\DeclareSymbolFont` and `\SetSymbolFont`, to specify Japanese fonts as math fonts.

Specifying `disablejfam` option in `\usepackage` prevents applying these patches. Hence one cannot write Japanese Characters in math mode directly if `disablejfam` option is specified. The code below does not work either:

```latex
\DeclareSymbolFont{mincho}{JY3}{mc}{m}{n}
\DeclareSymbolFontAlphabet{\mathmc}{mincho}
```

## 11.3 Detail of `\fontfamily` command

In this subsection, we describe when `\fontfamily{(family)}` changes current Japanese/alphabetic font family. Basically, current Japanese font family is changed to `(family)` if it is recognized as a Japanese font family, and similar with alphabetic font family. There is a case that current Japanese/alphabetic font family are both changed to `(family)`, and another case that `(family)` isn’t recognized as a Japanese/alphabetic font family either.

**Recognition as Japanese font family** First, whether Japanese font family will be changed is determined in following order. This order is very similar to `\fontfamily` in p\TeX\ 2\varepsilon, but we re-implemented in Lua. We use an auxiliary list $N_J$.

1. If the family `(family)` has been defined already by `\DeclareKanjiFamily`, `(family)` is recognized as a Japanese font family. Note that `(family)` need not be defined under current Japanese font encoding.
2. If the family `(family)` has been listed in a list $N_J$, this means that `(family)` is not a Japanese font family.
3. If the luatexja-fontspec package is loaded, we stop here, and `(family)` is not recognized as a Japanese font family.
4. Now Lua\TeX-ja looks whether there exists a Japanese font encoding `(enc)` such that a font definition named `(enc)(family).fd` (the file name is all lowercase) exists. If so, `(family)` is recognized as a Japanese font family (the font definition file won’t be loaded here). If not, `(family)` is not a Japanese font family, and `(family)` is appended to the list $N_J$.

**Recognition as alphabetic font family** Next, whether alphabetic font family will be changed is determined in following order. We use auxiliary lists $F_A$ and $N_A$.

1. If the family `(family)` has been listed in a list $F_A$, `(family)` is recognized as an alphabetic font family.
2. If the family `(family)` has been listed in a list $N_A$, this means that `(family)` is not an alphabetic font family.
3. If there exists an alphabetic font encoding such that the family `(family)` has been defined under it, `(family)` is recognized as an alphabetic font family, and to memorize this, `(family)` is appended to the list $F_A$.
4. Now Lua\TeX-ja looks whether there exists an alphabetic font encoding `(enc)` such that a font definition named `(enc)(family).fd` (the file name is all lowercase) exists. If so, current alphabetic font family will be changed to `(family)` (the font definition file won’t be loaded here). If not, current alphabetic font family won’t be changed, and `(family)` is appended to the list $N_A$.

Also, each call of `\DeclareFontFamily` after loading of Lua\TeX-ja makes the second argument (family) is appended to the list $F_A$.

The above order is very similar to `\fontfamily` in p\TeX\ 2\varepsilon, but more complicated (clause 3.). This is because p\TeX\ 2\varepsilon is a format however Lua\TeX-ja is not, hence Lua\TeX-ja does not know calls of `\DeclareFontFamily` before itself is loaded.
### Remarks

Of course, there is a case that \textit{family} is not recognized as a Japanese font family, nor an alphabetic font family. In this case, Lua\TeX{}-ja treats “the argument \textit{family} is wrong”, so set both current alphabetic and Japanese font family to \textit{family}, to use the default family for font substitution.

### Notes on \DeclareTextSymbol

From \LaTeX{} 2017/01/01, the standard encoding of Lua\TeX{} is changed to the \texttt{TU} encoding. This means that symbols defined by \texttt{T1} and \texttt{TS1} encodings can be used without loading any package. To produce these symbols in alphabetic fonts in Lua\TeX{}-ja, Lua\TeX{}-ja patches \texttt{\DeclareTextSymbol}, and reloads \texttt{tuenc.def}.

Under original definition of \texttt{\DeclareTextSymbol}, internal commands which is defined by \texttt{\DeclareTextSymbol} (such as \texttt{\T1\textquotedblleft}) are \texttt{chardef} tokens. However, this no longer holds in Lua\TeX{}-ja; for example, the meaning of \texttt{\TU\textquotedblleft} is \texttt{|tjalchar8220|}.

### \strutbox

As \LaTeX{} (2017/04/08 or later), \texttt{\strutbox} is a macro which is expanded to one of \texttt{\ystrutbox}, \texttt{\tstrutbox}, and \texttt{\dstrutbox} (all of them are shown in Table 15), according to the current direction. Similarly, \texttt{\ strut} now uses one of these boxes.

### expl3 interface

This section describes expl3 interfaces provided by Lua\TeX{}-ja. All of them belong to the \texttt{platex} module, since they are provided for compatibility with Japanese \LaTeX{}. Note that commands which are marked with dagger (“†”) are additions by Lua\TeX{}-ja.

\begin{verbatim}
\texttt{\platex_direction_yoko}: \texttt{\platex_direction_tate}: \texttt{\platex_direction_dtou}:
    Synonyms for \texttt{\yoko}, \texttt{\tate} and \texttt{\dtou}, respectively.

\texttt{\platex_if_direction_yoko_p}:
\texttt{\platex_if_direction_yoko:TF} \{\texttt{true code}\}\{\texttt{false code}\}
    Tests if the current direction is \texttt{yoko} (horizontal writing).

\texttt{\platex_if_direction_tate_nomath_p}:
\texttt{\platex_if_direction_tate_nomath:TF} \{\texttt{true code}\}\{\texttt{false code}\}
    Tests if the current direction is \texttt{tate} (vertical writing).

\texttt{\platex_if_direction_tate_math_p}:
\texttt{\platex_if_direction_tate_math:TF} \{\texttt{true code}\}\{\texttt{false code}\}
    Tests if the current direction is \texttt{utod}.

\texttt{\platex_if_direction_tate_p}:
\texttt{\platex_if_direction_tate:TF} \{\texttt{true code}\}\{\texttt{false code}\}
    Tests if the current direction is \texttt{tate or utod}.

\texttt{\platex_if_direction_dtou_p}:
\texttt{\platex_if_direction_dtou:TF} \{\texttt{true code}\}\{\texttt{false code}\}
    Tests if the current direction is \texttt{dtou}.
\end{verbatim}


\begin{verbatim}
\fjfontspec
\YokoFeatures={Color=FF1900}, \TateFeatures={Color=003FFF},
\TateFont=HaranoAjiGothic-Regular
\{HaranoAjiMincho-Regular\}
\hbox{\yoko 横組のテスト}\hbox{\tate 縦組のテスト}
\addjfontfeatures{Color=00AF00}
\hbox{\yoko 横組}\hbox{\tate 縦組}
\end{verbatim}

Figure 11. An example of TateFeatures etc.

\texttt{\platex_if_box_yoko_p:N} \langle box \rangle
\texttt{\platex_if_box_yoko:NTF\{\langle box \rangle\}{{\{true code\}}}{\{false code\}}}
Tests if the direction of \langle box \rangle is yoko.

\texttt{\platex_if_box_tate_nomath_p:N} \langle box \rangle
\texttt{\platex_if_box_tate_nomath:NTF\{\langle box \rangle\}{{\{true code\}}}{\{false code\}}}
Tests if the direction of \langle box \rangle is tate.

\texttt{\platex_if_box_tate_math_p:N} \langle box \rangle
\texttt{\platex_if_box_tate_math:NTF\{\langle box \rangle\}{{\{true code\}}}{\{false code\}}}
Tests if the direction of \langle box \rangle is utod.

\texttt{\platex_if_box_tate_p:N} \langle box \rangle
\texttt{\platex_if_box_tate:NTF\{\langle box \rangle\}{{\{true code\}}}{\{false code\}}}
Tests if the direction of \langle box \rangle is tate or utod.

\texttt{\platex_if_box_dtou_p:N} \langle box \rangle
\texttt{\platex_if_box_dtou:NTF\{\langle box \rangle\}{{\{true code\}}}{\{false code\}}}
Tests if the direction of \langle box \rangle is dtou.

13 Addon packages

LuaTeX-ja has several addon packages. These addons are written as \TeX packages, but \luatexja-otf and \luatexja-adjust can be loaded in plain \TeX by \texttt{\input}.

13.1 luatexja-fontspec

As described in Subsection 3.2, this optional package provides the counterparts for several commands defined in the fontspec package (requires fontspec v2.4). In addition to OpenType font features in the original fontspec, the following "font features" specifications are allowed for the commands of Japanese version:

\texttt{\CID=(name), JFM=(name), JFM-var=(name)}
These 3 keys correspond to \texttt{cid}, \texttt{jfm} and \texttt{jfmvar} keys for \texttt{\jfont} and \texttt{\tfont} respectively. See Subsections 8.1 and 8.4 for details of \texttt{cid}, \texttt{jfm} and \texttt{jfmvar} keys.

The \texttt{CID} key is effective only when with \texttt{NoEmbed} described below. The same \texttt{JFM} cannot be used in both horizontal Japanese fonts and vertical Japanese fonts, hence the \texttt{JFM} key will be actually used in \texttt{YokoFeatures} and \texttt{TateFeatures} keys.

\texttt{NoEmbed}
By specifying this key, one can use "name-only" Japanese font which will not be embedded in the output PDF file. See Subsection 8.4.

\texttt{Kanjiskip=(bool)}
日本国民は、正当に選挙された国会における代表者を通じて行動し、われらとわれらの子孫のために、
諸国民との協和による成果と、わが国全土にわたつて自由のもたらす恵沢を確保し、……

日本国民は、正当に選挙された国会における代表者を通じて行動し、われらとわれらの子孫のために、
諸国民との協和による成果と、わが国全土にわたつて自由のもたらす恵沢を確保し、……

Figure 12. An example of AltFont

TateFeatures={⟨features⟩}, TateFont=⟨font⟩
The TateFeatures key specifies font features which are only turned on in vertical writing, such as
Style=VerticalKana (vkna feature). Similarly, the TateFont key specifies the Japanese font which
will be used only in vertical writing. A demonstrarion is shown in Figure 11.

YokoFeatures={⟨features⟩}
The YokoFeatures key specifies font features which are only turned on in horizontal writing. A
demonstration is shown in Figure 11.

AltFont
As \ltjdeclarealtfont (Subsection 10.5) and \DeclareAlternateKanjiFont (Subsection 11.2), with
this key, one can typeset some Japanese characters by a different font and/or using different features.
The AltFont feature takes a comma-separated list of comma-separated lists, as the following:
AltFont = { ...
  { Range=⟨range⟩, ⟨features⟩ },
  { Range=⟨range⟩, Font=⟨font name⟩, ⟨features⟩ },
  { Range=⟨range⟩, Font=⟨font name⟩ },
  …
}

Each sublist should have the Range key (sublist which does not contain Range key is simply ignored).
A demonstrarion is shown in Figure 12.

Remark on AltFont, YokoFeatures, TateFeatures keys
In AltFont, YokoFeatures, TateFeatures keys, one cannot specify per-shape settings such as
BoldFeatures. For example,
AltFont = {
  { Font=HogeraMin-Light, BoldFont=HogeraMin-Bold,
    Range="3000-"30FF, BoldFeatures={Color=FF1900} } 
}

does not work. Instead, one have to write
UprightFeatures = {
  AltFont = { { Font=HogeraMin-Light, Range="3000-"30FF, } },
},
BoldFeatures = {
  AltFont = { { Font=HogeraMin-Bold, Range="3000-"30FF, Color=FF1900 } },
}

On the other hand, YokoFeatures, TateFeatures and TateFont keys can be specified in each list in
the AltFont key. Also, one can specify AltFont inside YokoFeatures, TateFeatures.
The value of \texttt{kanjiskip} is 0 pt $+ \tfrac{1}{5}$ em in this figure, for making the difference obvious.

Figure 13. Line adjustment

Note that features which are specified in \texttt{YokoFeatures} and \texttt{TateFeatures} are always interpreted \textit{after} other "direction-independent" features. This explains why \texttt{\addfontfeatures} at line 6 in Figure 11 has no effect, because a color specification is already done in \texttt{YokoFeatures} and \texttt{TateFeatures} keys.

13.2 \texttt{luatexja-otf}

This optional package supports typesetting glyphs by specifying a CID number. The package \texttt{luatexja-otf} offers the following 2 low-level commands:

\begin{verbatim}
\CID{(number)}
\end{verbatim}

Typeset a glyph whose CID number is \texttt{\{number\}}. If the Japanese font is neither Adobe-Japan1, Adobe-GB1, Adobe-CNS1, Adobe-Korea1, nor Adobe-KR CID-keyed font, \LaTeX\-ja treats that \texttt{\{number\}} is a CID number of Adobe-Japan1 character collection, and tries to typeset a "most suitable glyph".

Note that if the Japanese font is loaded using the HarfBuzz library, this \texttt{\CID} command does not work.

\begin{verbatim}
\UTF{(hex_number)}
\end{verbatim}

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

This package automatically loads \texttt{luatexja-ajmacros.sty}, which is slightly modified version of \texttt{ajmacros.sty}\footnote{Useful macros by INOUE Koich!, for the \texttt{japanese-otf} package.}. Hence one can use macros which are defined in \texttt{ajmacros.sty}, such as \texttt{\aj半角}.

\section*{Remarks} Characters by \texttt{\CID} and \texttt{\UTF} commands are different from ordinary characters in the following points:

- Always treated as \texttt{JAchars}.

- In vertical direction, \texttt{vert}/\texttt{vrt2} feature are automatically applied to characters by \texttt{\UTF}, regardless these feature are not activated in current Japanese font.

- Processes for supporting other OpenType features (for example, glyph replacement and kerning) by the \texttt{luaotfload} package is not performed to these characters.

\section*{Additional syntax of JFM} The package \texttt{luatexja-otf} extends the syntax of JFM; the entries of \texttt{chars} table in JFM now allows a string in the form \texttt{"AJ1-xxx"}, which stands for the character whose CID number in Adobe-Japan1 is \texttt{xxx}.

This extended notation is used in the standard JFM \texttt{jfm-ujis.1ua} to typeset halfwidth Hiragana glyphs (CID 516–598) in halfwidth.

13.3 \texttt{luatexja-adjust} (see Japanese version of this manual)
13.4 luatexja-ruby

This addon package provides functionality of “ruby” (furigana) annotations using callbacks of LuaTeX-ja. There is no detailed manual of luatexja-ruby.sty in English. (Japanese manual is another PDF file, luatexja-ruby.pdf.)

Group-ruby By default, ruby characters (the second argument of \ruby) are attached to base characters (the first argument), as one object. This type of ruby is called group-ruby.

```
1 東西線ruby{妙典}{みようでん}駅は……
2 東西線のruby{妙典}{みようでん}駅は……
3 東西線のruby{葛西}{かさい}駅は……
4 東西線ruby{葛西}{かさい}駅は……
```

As the above example, ruby hangover is allowed on the Hiragana before/after its base characters.

Mono-ruby To attach ruby characters to each base characters (mono-ruby), one should use \ruby multiple times:

```
1 東西線のruby{妙}{みよう}ruby{典}{でん}駅は……
2 東西線のruby{妙}{みよう}ruby{典}{でん}駅は……
3 東西線のruby{妙}{みよう}ruby{典}{でん}駅は……
4 東西線のruby{妙}{みよう}ruby{典}{でん}駅は……
```

Jukugo-ruby Vertical bar | denotes a boundary of groups.

```
1 \ruby{妙}{みよう}{でん}
2 \ruby{葛}{かさい}
3 \ruby{妙}{みよう}{でん}
```

If there are multiple groups in one \ruby call, A linebreak between two groups is allowed.

```
1 vbox{\hspace=6zw}\noindent
2 hbox to 2.5zw}{ruby{京|急|蒲}{田}|{けい|きゆう|かま|た}
3 hbox to 2.5zw}{ruby{京|急|蒲}{田}|{けい|きゆう|かま|た}
4 hbox to 3zw}{ruby{京|急|蒲}{田}|{けい|きゆう|かま|た}
5 }
```

If the width of ruby characters are longer than that of base characters, \ruby automatically selects the appropriate form among the line-head form, the line-middle form, and the line-end form.

```
1 vbox{\hspace=8zw}\noindent
2 null}{kern3zw}{ruby{承}{うけたまわ}る
3 kern1zw}{ruby{承}{うけたまわ}る
4 null}{kern5zw}{ruby{承}{うけたまわ}る
5 }
```

13.5 lltjext.sty

p\TeX supplies additional macros for vertical writing in the plext package. The lltjext package which we want to describe here is the Lua\TeX-ja counterpart of the plext package.

tabular, array, minipage environments

These environments are extended by <dir>, which specifies the direction, as follows:

```
\begin{tabular}<dir>{pos}{table spec} ... \end{tabular}
\begin{array}<dir>{pos}{table spec} ... \end{array}
\begin{minipage}<dir>{width} ... \end{minipage}
```

This option permits one of the following five values. If none of them is specified, the direction inside the environment is same as that outside the environment.

y yoko direction (horizontal writing)
tate direction (vertical writing)

utod direction if direction outside the env. is tate.

dou direction

utod direction

\parbox<\langle \text{dir} \rangle>\{\langle \text{pos} \rangle\}\{\langle \text{width} \rangle\}\{\langle \text{contents} \rangle\}
\parbox command is also extended by \langle \text{dir} \rangle.

\pbox<\langle \text{dir} \rangle>\{\langle \text{width} \rangle\}\{\langle \text{pos} \rangle\}\{\langle \text{contents} \rangle\}
This commands typeset \langle \text{contents} \rangle in LR-mode, in \langle \text{dir} \rangle direction. If \langle \text{width} \rangle is positive, the width of the box becomes this \langle \text{width} \rangle. In this case, \langle \text{contents} \rangle will be aligned to left (when \langle \text{pos} \rangle is 1), center (c), or right (r).

picture environment
picture environment also extended by \langle \text{dir} \rangle, as follows:
\begin{picture}<\langle \text{dir} \rangle>(x_{\text{size}}, y_{\text{size}})(x_{\text{offset}},y_{\text{offset}})
...\end{picture}
\rensujivi\{\langle \text{pos} \rangle\}\{\langle \text{contents} \rangle\}, \rensujiskip
\Kanji\{\langle \text{counter} \rangle \text{name} \}
\kasen\{\langle \text{contents} \rangle\}, \bou\{\langle \text{contents} \rangle\}, \boutenchar

参照番号

13.6 \texttt{luatexja-preset}

As described in Subsection 3.3, One can load the luatexja-preset package to use several “presets” of Japanese fonts. This package provides functions in a part of japanese-otf package (changing fonts) and a part of PXchfon package (presets) by Takayuki Yato.

Options which are given in \usepackage but not described in this subsection are simply passed to the luatexja-fontspec\textsuperscript{12}. For example, the line 5 in below example is equivalent to lines 1–3.
\usepackage[no-math]{fontspec}
\usepackage[match]{luatexja-fontspec}
\usepackage[kozuka-pr6n]{luatexja-preset}
\%
\usepackage[no-math,match,kozuka-pr6n]{luatexja-preset}

13.6.1 General Options

\texttt{fontspec} (enabled by default)
With this option, Japanese fonts are selected using functionality of the luatexja-fontspec package. This means that the fontspec package is automatically loaded by this package.

If you need to pass some options to fontspec, you can load fontspec manually before luatexja-preset:
\usepackage[no-math]{fontspec}
\usepackage[...]{luatexja-preset}

\textsuperscript{12}if \texttt{nfssonly} option is \textit{not} specified; in this case these options are simply ignored.
nfssonly
With this option, selecting Japanese fonts won’t be performed using the functionality of the fontspec package, but only standard NFSS2 (hence without \addfontfeatures etc.). This option is ignored when luatexja-fontspec package is loaded.

When this option is specified, fontspec and luatexja-fontspec are not loaded by default. Nevertheless, the packagefontspec can coexist with the option, as the following:
\usepackage[fontspec]
\usepackage[hiragino-pron,nfssonly]{luatexja-preset}

In this case, one can use \setmainfont etc. to select alphabetic fonts.

match
If this option is specified, usual family-changing commands such as \rmfamily, \texttt, \sffamily, ... also change Japanese font family. This option is passed to luatexja-fontspec, if fontspec option is specified.

nodeluxe (enabled by default)
The negation of deluxe option. Use one-weighted mincho and gothic font families. This means that \mcfamily\bfseries, \gtfamily\bfseries and \gtfamily\mdseries use the same font.

deluxe
Use the mincho family with three weights (light, medium, and bold), the gothic family with three weights (medium, bold, and extra bold), and rounded gothic\(^{13}\). Mincho light and gothic extra bold can be by \mcfamily\ltseries and \gtfamily\ebseries, respectively.

- Some presets do not have the light weight of mincho. In this case, we substitute the medium weight for the light weight.
- luatexja-preset does not produce an error (only produces a warning), even if (one of) fonts for \mcfamily\ltseries, \gtfamily\ebseries, \mgfamily do not exist.

expert
Use horizontal/vertical kana alternates, and define a command \rubyfamily to use kana characters designed for ruby.

bold
Substitute bold series of gothic for bold series of mincho. If nodeluxe option is enabled, medium series of gothic is also changed, since we use same font for both series of gothic.

jis90, 90jis
Use JIS X 0208:1990 glyph variants if possible.

jis2004, 2004jis
Use JIS X 0213:2004 glyph variants if possible.

\texttt{\texttt{jfm\_yoko=\texttt{jfm}}}
Use the JFM \texttt{jfm-\texttt{jfm}}.lua for horizontal direction, instead of \texttt{jfm-ujis\_lua} (default JFM).

\texttt{\texttt{jfm\_tate=\texttt{jfm}}}
Use the JFM \texttt{jfm-\texttt{jfm}}.lua for vertical direction, instead of \texttt{jfm-ujisv\_lua} (default JFM).

jis Same as \texttt{jfm\_yoko=jis}.

Note that jis90, 90jis, jis2004 and 2004jis only affect with mincho, gothic (and, possibly rounded gothic) families defined by this package. We didn’t taken account of when more than one options among them are specified.

\(^{13}\)Provided by \\mgfamily and \textmg, because “rounded gothic” is called maru gothic (丸ゴシック) in Japanese.
13.6.2 Presets which support multi weights

Besides bizud, haranoaji, morisawa-pro, and morisawa-pr6n presets, fonts are specified by font name, not by file name. In following tables, starred fonts (e.g. KozGoPro-Regular) are used for medium series of gothic, if and only if deluxe option is specified.

**kozuka-pro** Kozuka Pro (Adobe-Japan1-4) fonts.
**kozuka-pr6** Kozuka Pr6 (Adobe-Japan1-6) fonts.
**kozuka-pr6n** Kozuka Pr6N (Adobe-Japan1-6, JIS04-savvy) fonts.

Kozuka Pro/Pr6N fonts are bundled with Adobe’s software, such as Adobe InDesign. There is not rounded gothic family in Kozuka fonts.

<table>
<thead>
<tr>
<th>family</th>
<th>series</th>
<th>kozuka-pro</th>
<th>kozuka-pr6</th>
<th>kozuka-pr6n</th>
</tr>
</thead>
<tbody>
<tr>
<td>mincho</td>
<td>light</td>
<td>KozMinPro-Light</td>
<td>KozMinProVI-Light</td>
<td>KozMinPr6N-Light</td>
</tr>
<tr>
<td></td>
<td>medium</td>
<td>KozMinPro-Regular</td>
<td>KozMinProVI-Regular</td>
<td>KozMinPr6N-Regular</td>
</tr>
<tr>
<td></td>
<td>bold</td>
<td>KozMinPro-Bold</td>
<td>KozMinProVI-Bold</td>
<td>KozMinPr6N-Bold</td>
</tr>
<tr>
<td>gothic</td>
<td>medium</td>
<td>KozGoPro-Regular*</td>
<td>KozGoProVI-Regular*</td>
<td>KozGoPr6N-Regular*</td>
</tr>
<tr>
<td></td>
<td>bold</td>
<td>KozGoPro-Bold</td>
<td>KozGoProVI-Bold</td>
<td>KozGoPr6N-Bold</td>
</tr>
<tr>
<td></td>
<td>extra bold</td>
<td>KozGoPro-Heavy</td>
<td>KozGoProVI-Heavy</td>
<td>KozGoPr6N-Heavy</td>
</tr>
<tr>
<td>rounded gothic</td>
<td></td>
<td>KozGoPro-Heavy</td>
<td>KozGoProVI-Heavy</td>
<td>KozGoPr6N-Heavy</td>
</tr>
</tbody>
</table>

**hiragino-pro** Hiragino Pro (Adobe-Japan1-5) fonts.
**hiragino-pron** Hiragino ProN (Adobe-Japan1-5, JIS04-savvy) fonts.

Hiragino fonts (except Hiragino Mincho W2) are bundled with Mac OS X 10.5 or later. Note that fonts for gothic extra bold (HiraKakuStd[N]-W8) only contains characters in Adobe-Japan1-3 character collection, while others contains those in Adobe-Japan1-5 character collection.

<table>
<thead>
<tr>
<th>family</th>
<th>series</th>
<th>hiragino-pro</th>
<th>hiragino-pron</th>
</tr>
</thead>
<tbody>
<tr>
<td>mincho</td>
<td>light</td>
<td>Hiragino Mincho Pro W2</td>
<td>Hiragino Mincho ProN W2</td>
</tr>
<tr>
<td></td>
<td>medium</td>
<td>Hiragino Mincho Pro W3</td>
<td>Hiragino Mincho ProN W3</td>
</tr>
<tr>
<td></td>
<td>bold</td>
<td>Hiragino Mincho Pro W6</td>
<td>Hiragino Mincho ProN W6</td>
</tr>
<tr>
<td>gothic</td>
<td>medium</td>
<td>Hiragino Kaku Gothic Pro W3*</td>
<td>Hiragino Kaku Gothic ProN W3*</td>
</tr>
<tr>
<td></td>
<td>bold</td>
<td>Hiragino Kaku Gothic Pro W6</td>
<td>Hiragino Kaku Gothic ProN W6</td>
</tr>
<tr>
<td></td>
<td>extra bold</td>
<td>Hiragino Kaku Gothic Std W8</td>
<td>Hiragino Kaku Gothic StdN W8</td>
</tr>
<tr>
<td>rounded gothic</td>
<td></td>
<td>Hiragino Maru Gothic Pro W4</td>
<td>Hiragino Maru Gothic ProN W4</td>
</tr>
</tbody>
</table>

**bizud** BIZ UD fonts (by Morisawa Inc.) bundled with Windows 10 October 2018 Update.

<table>
<thead>
<tr>
<th>family</th>
<th>series</th>
</tr>
</thead>
<tbody>
<tr>
<td>mincho</td>
<td>BIZ-UDMinchoM.ttc</td>
</tr>
<tr>
<td>gothic</td>
<td>BIZ-UDGothicR.ttc</td>
</tr>
<tr>
<td></td>
<td>BIZ-UDGothicB.ttc</td>
</tr>
<tr>
<td></td>
<td>BIZ-UDGothicB.ttc</td>
</tr>
<tr>
<td>rounded gothic</td>
<td>BIZ-UDGothicB.ttc</td>
</tr>
</tbody>
</table>
morisawa-pro Morisawa Pro (Adobe-Japan1-4) fonts.
morisawa-pr6n Morisawa Pr6N (Adobe-Japan1-6, JIS04-savvy) fonts.

<table>
<thead>
<tr>
<th>family</th>
<th>series</th>
<th>morisawa-pro</th>
<th>morisawa-pr6n</th>
</tr>
</thead>
<tbody>
<tr>
<td>mincho</td>
<td>medium</td>
<td>A-OTF-RyuminPro-Light.otf</td>
<td>A-OTF-RyuminPr6N-Light.otf</td>
</tr>
<tr>
<td></td>
<td>bold</td>
<td>A-OTF-FutoMinA101Pro-Bold.otf</td>
<td>A-OTF-FutoMinA101Pr6N-Bold.otf</td>
</tr>
<tr>
<td>gothic</td>
<td>medium</td>
<td>A-OTF-GothicBBBPro-Medium.otf</td>
<td>A-OTF-GothicBBBPr6N-Medium.otf</td>
</tr>
<tr>
<td></td>
<td>bold</td>
<td>A-OTF-FutoGooB101Pro-Bold.otf</td>
<td>A-OTF-FutoGooB101Pr6N-Bold.otf</td>
</tr>
<tr>
<td></td>
<td>extra bold</td>
<td>A-OTF-MidashiGoPro-MB31.otf</td>
<td>A-OTF-MidashiGoPr6N-MB31.otf</td>
</tr>
<tr>
<td>rounded gothic</td>
<td></td>
<td>A-OTF-Jun101Pro-Light.otf</td>
<td>A-OTF-ShinMGoPr6N-Light.otf</td>
</tr>
</tbody>
</table>

yu-win Yu fonts bundled with Windows 8.1.
yu-win10 Yu fonts bundled with Windows 10.
yu-osx Yu fonts bundled with OSX Mavericks.

<table>
<thead>
<tr>
<th>family</th>
<th>series</th>
<th>yu-win</th>
<th>yu-win10</th>
<th>yu-osx</th>
</tr>
</thead>
<tbody>
<tr>
<td>mincho</td>
<td>light</td>
<td>YuMincho-Light</td>
<td>YuMincho-Light</td>
<td>(YuMincho Medium)</td>
</tr>
<tr>
<td></td>
<td>medium</td>
<td>YuMincho-Regular</td>
<td>YuMincho-Regular</td>
<td>YuMincho Medium</td>
</tr>
<tr>
<td></td>
<td>bold</td>
<td>YuMincho-DemiBold</td>
<td>YuMincho-DemiBold</td>
<td>YuMincho Demibold</td>
</tr>
<tr>
<td>gothic</td>
<td>medium</td>
<td>YuGothic-Regular*</td>
<td>YuGothic-Regular*</td>
<td>YuGothic Medium*</td>
</tr>
<tr>
<td></td>
<td>bold</td>
<td>YuGothic-Bold</td>
<td>YuGothic-Bold</td>
<td>YuGothic Bold</td>
</tr>
<tr>
<td></td>
<td>extra bold</td>
<td>YuGothic-Bold</td>
<td>YuGothic-Bold</td>
<td>YuGothic Bold</td>
</tr>
<tr>
<td>rounded gothic</td>
<td></td>
<td>YuGothic-Bold</td>
<td>YuGothic-Bold</td>
<td>YuGothic Bold</td>
</tr>
</tbody>
</table>

moga-mobo MogaMincho, MogaGothic, and MoboGothic.
moga-mobo-ex MogaExMincho, MogaExGothic, and MoboExGothic.

These fonts can be downloaded from http://yozvox.web.fc2.com/.

<table>
<thead>
<tr>
<th>family</th>
<th>series</th>
<th>default, 90jis option</th>
<th>jis2004 option</th>
</tr>
</thead>
<tbody>
<tr>
<td>mincho</td>
<td>medium</td>
<td>Moga90Mincho</td>
<td>MogaMincho</td>
</tr>
<tr>
<td></td>
<td>bold</td>
<td>Moga90Mincho Bold</td>
<td>MogaMincho Bold</td>
</tr>
<tr>
<td>gothic</td>
<td>medium</td>
<td>Moga90Gothic</td>
<td>MogaGothic</td>
</tr>
<tr>
<td></td>
<td>bold</td>
<td>Moga90Gothic Bold</td>
<td>MogaGothic Bold</td>
</tr>
<tr>
<td></td>
<td>extra bold</td>
<td>Moga90Gothic Bold</td>
<td>MogaGothic Bold</td>
</tr>
<tr>
<td>rounded gothic</td>
<td></td>
<td>Mobo90Gothic</td>
<td>MoboGothic</td>
</tr>
</tbody>
</table>

When moga-mobo-ex is specified, the font “MogaEx90Mincho” etc. are used.

ume Ume Mincho and Ume Gothic.

These fonts can be downloaded from https://ja.osdn.net/projects/ume-font/wiki/FrontPage.
sourcehan  Source Han Serif and Source Han Sans fonts (Language-specific OTF or OTC)
sourcehan-jp  Source Han Serif JP and Source Han Sans JP fonts (Region-specific Subset OTF)

<table>
<thead>
<tr>
<th>family</th>
<th>series</th>
<th>sourcehan</th>
<th>sourcehan-jp</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>light</td>
<td>Source Han Serif Light</td>
<td>Source Han Serif JP Light</td>
</tr>
<tr>
<td></td>
<td>medium</td>
<td>Source Han Serif Regular</td>
<td>Source Han Serif JP Regular</td>
</tr>
<tr>
<td></td>
<td>bold</td>
<td>Source Han Serif Bold</td>
<td>Source Han Serif JP Bold</td>
</tr>
<tr>
<td>mincho</td>
<td>medium</td>
<td>Source Han Sans Regular*</td>
<td>Source Han Sans JP Regular*</td>
</tr>
<tr>
<td></td>
<td>bold</td>
<td>Source Han Sans Medium</td>
<td>Source Han Sans JP Medium</td>
</tr>
<tr>
<td></td>
<td>extra bold</td>
<td>Source Han Sans Heavy</td>
<td>Source Han Sans JP Heavy</td>
</tr>
<tr>
<td>gothic</td>
<td>medium</td>
<td>Source Han Sans Medium</td>
<td>Source Han Sans JP Medium</td>
</tr>
<tr>
<td></td>
<td>bold</td>
<td>Source Han Sans Bold</td>
<td>Source Han Sans JP Bold</td>
</tr>
<tr>
<td></td>
<td>extra bold</td>
<td>Source Han Sans Heavy</td>
<td>Source Han Sans JP Heavy</td>
</tr>
<tr>
<td>rounded gothic</td>
<td></td>
<td>Source Han Sans Medium</td>
<td>Source Han Sans JP Medium</td>
</tr>
</tbody>
</table>

noto-otc  Noto Serif CJK and Noto Sans CJK fonts (OTC)
noto-otf  Noto Serif CJK and Noto Sans CJK fonts (Language-specific OTF)

<table>
<thead>
<tr>
<th>family</th>
<th>series</th>
<th>noto-otc</th>
<th>noto-otf</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>light</td>
<td>Noto Serif CJK Light</td>
<td>Noto Serif CJK JP Light</td>
</tr>
<tr>
<td></td>
<td>medium</td>
<td>Noto Serif CJK Regular</td>
<td>Noto Serif CJK JP Regular</td>
</tr>
<tr>
<td></td>
<td>bold</td>
<td>Noto Serif CJK Bold</td>
<td>Noto Serif CJK JP Bold</td>
</tr>
<tr>
<td>mincho</td>
<td>medium</td>
<td>Noto Sans CJK Regular*</td>
<td>Noto Sans CJK JP Regular*</td>
</tr>
<tr>
<td></td>
<td>bold</td>
<td>Noto Sans CJK Medium</td>
<td>Noto Sans CJK JP Medium</td>
</tr>
<tr>
<td></td>
<td>extra bold</td>
<td>Noto Sans CJK Bold</td>
<td>Noto Sans CJK JP Bold</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Noto Sans CJK Black</td>
<td>Noto Sans CJK JP Black</td>
</tr>
<tr>
<td>gothic</td>
<td>medium</td>
<td>Noto Sans CJK Medium</td>
<td>Noto Sans CJK JP Medium</td>
</tr>
<tr>
<td></td>
<td>bold</td>
<td>Noto Sans CJK Bold</td>
<td>Noto Sans CJK JP Bold</td>
</tr>
<tr>
<td></td>
<td>extra bold</td>
<td>Noto Sans CJK Black</td>
<td>Noto Sans CJK JP Black</td>
</tr>
<tr>
<td>rounded gothic</td>
<td></td>
<td>Noto Sans CJK Medium</td>
<td>Noto Sans CJK JP Medium</td>
</tr>
</tbody>
</table>

haranoaji  Harano Aji Fonts.
These fonts can be downloaded from [https://github.com/trueroad/HaranoAjiFonts](https://github.com/trueroad/HaranoAjiFonts). There is not rounded gothic family in Harano Aji Fonts.

<table>
<thead>
<tr>
<th>family</th>
<th>series</th>
<th>haranoaji</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>light</td>
<td>HaranoAjiMincho-Light.otf</td>
</tr>
<tr>
<td></td>
<td>medium</td>
<td>HaranoAjiMincho-Regular.otf</td>
</tr>
<tr>
<td></td>
<td>bold</td>
<td>HaranoAjiMincho-Bold.otf</td>
</tr>
<tr>
<td>mincho</td>
<td>medium</td>
<td>HaranoAjiGothic-Regular.otf*</td>
</tr>
<tr>
<td></td>
<td>bold</td>
<td>HaranoAjiGothic-Bold.otf</td>
</tr>
<tr>
<td></td>
<td>extra bold</td>
<td>HaranoAjiGothic-Heavy.otf</td>
</tr>
<tr>
<td>gothic</td>
<td>medium</td>
<td>HaranoAjiGothic-Medium.otf</td>
</tr>
<tr>
<td></td>
<td>bold</td>
<td>HaranoAjiGothic-Bold.otf</td>
</tr>
<tr>
<td></td>
<td>extra bold</td>
<td>HaranoAjiGothic-Heavy.otf</td>
</tr>
<tr>
<td>rounded gothic</td>
<td></td>
<td>HaranoAjiGothic-Medium.otf</td>
</tr>
</tbody>
</table>

13.6.3  Presets which do not support multi weights

Next, we describe settings for using only single weight.

<table>
<thead>
<tr>
<th>noembed</th>
<th>series</th>
<th>ipa</th>
<th>ipaex</th>
<th>ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>mincho</td>
<td>Ryumin-Light (non-embedded)</td>
<td>IPA Mincho</td>
<td>IPAex Mincho</td>
<td>MS Mincho</td>
</tr>
<tr>
<td>gothic</td>
<td>GothicBBB-Medium (non-embedded)</td>
<td>IPA Gothic</td>
<td>IPAex Gothic</td>
<td>MS Gothic</td>
</tr>
</tbody>
</table>
13.6.4 Presets which use HG fonts

We can use HG fonts bundled with Microsoft Office for realizing multiple weights. In the table below, starred fonts (e.g., IPA Gothic*) are used only if jis2004 or nodeluxe option is specified.

<table>
<thead>
<tr>
<th>family</th>
<th>series</th>
<th>ipa-hg</th>
<th>ipaex-hg</th>
<th>ms-hg</th>
</tr>
</thead>
<tbody>
<tr>
<td>mincho</td>
<td>medium</td>
<td>IPA Mincho</td>
<td>IPAex Mincho</td>
<td>MS Mincho</td>
</tr>
<tr>
<td></td>
<td>bold</td>
<td>HG Mincho E</td>
<td>HG Mincho E</td>
<td>HG Mincho E</td>
</tr>
<tr>
<td>gothic</td>
<td>medium</td>
<td>IPA Gothic*</td>
<td>IPAex Gothic*</td>
<td>MS Gothic*</td>
</tr>
<tr>
<td></td>
<td>bold</td>
<td>HG Gothic M</td>
<td>HG Gothic M</td>
<td>HG Gothic M</td>
</tr>
<tr>
<td></td>
<td>extra bold</td>
<td>HG Soei Kaku Gothic UB</td>
<td>HG Soei Kaku Gothic UB</td>
<td>HG Soei Kaku Gothic UB</td>
</tr>
<tr>
<td>rounded gothic</td>
<td></td>
<td>HG MaruGothic M PRO</td>
<td>HG MaruGothic M PRO</td>
<td>HG MaruGothic M PRO</td>
</tr>
</tbody>
</table>

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.).
- **jis90, 90jis** by file name (hgrme.ttc, hgrge.ttc, hgrsgu.ttc, hgrsmp.ttf).
- **jis2004, 2004jis** by file name (hgrme04.ttc, hgrge04.ttc, hgrsgu04.ttc, hgrsmp04.ttf).

13.6.5 Define/Use Custom Presets

From version 20170904.0, one can define new presets using `\ltjnewpreset`, and use them by `\ltjapplypreset`. These two commands can only be used in the preamble.

`\ltjnewpreset{⟨name⟩}{⟨specification⟩}`

Define new preset ⟨name⟩. This ⟨name⟩ cannot be same as other presets, options described in Subsubsection 13.6.1, nor following 11 strings:

- `mc` mc-l `mc-m` mc-b `mc-bx` gt `gt-m` `gt-b` `gt-bx` `gt-eb` mg-m

⟨specification⟩ is a comma-separated list which consists of other presets and/or the following keys:

- `mc-l=(font)` mincho light
- `mc-m=(font)` mincho medium
- `mc-b=(font)` mincho bold
- `mc-bx=(font)` synonym for `mc-b=(font)`
- `gt-m=(font)` gothic medium
- `gt-b=(font)` gothic bold
- `gt-bx=(font)` synonym for `gt-b=(font)`
- `gt-eb=(font)` gothic extra bold
- `mg-m=(font)` rounded gothic
- `mc=(font)` Same as `mc-l=(font), mc-m=(font), mc-b=(font)`
- `gt=(font)` Same as `gt-m=(font), gt-b=(font), gt-eb=(font)`

If `deluxe` is not specified at loading the package, only `mc` and `gt` keys (among above 11 keys) have a meaning.

`\ltjnewpreset*{⟨name⟩}{⟨specification⟩}`

Almost same as `\ltjnewpreset`. However, if ⟨name⟩ matches a preset which already defined, this command simply overwrite it.
\texttt{\textbackslash ltjapplypreset}\{\textit{name}\}\}

Set Japanese font families using preset \textit{name}.  

Note that \texttt{\textbackslash ltjnewpreset} does not "expand" the definition to define a preset. This means that one can write as the following:

\begin{verbatim}
\texttt{\textbackslash ltjnewpreset\{hoge\}\{piyo,mg-b-HiraMinProN-W6\}}  
\texttt{\textbackslash ltjnewpreset\{piyo\}\{mg-m-HiraMaruProN-W4\}}  
\texttt{\textbackslash ltjapplypreset\{hoge\}}
\end{verbatim}

\textbf{Restrictions} \hspace{1em} Presets which are defined by \texttt{\textbackslash ltjnewpreset} have following restrictions:

\begin{itemize}
  \item One cannot specify non-embedded fonts (such as Ryumin-Light).
  \item Some presets, such as \texttt{ipa-hg}, have a feature that fonts are changed according to whether \texttt{90jis} or \texttt{jis2004} is specified. This feature is not usable in presets which are defined by \texttt{\textbackslash ltjnewpreset}.
\end{itemize}
Part III
Implementations

14  Storing Parameters

14.1  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 Q = 0.25 mm, where "Q" (also called "級") is a unit used in Japanese phototypesetting. So one should not change the value of this dimension.

\jH (dimension) There is also a unit called "歯" which equals to 0.25 mm and used in Japanese phototypesetting. This \jH is the same \dimen register as \jQ.

\ltj@dimen@zw (dimension) A temporal register for the “full-width” of current Japanese font. The command \zw sets this register to the correct value, and “return” this register itself.

\ltj@dimen@zh (dimension) A temporal register for the “full-height” (usually the sum of height of imaginary body and its depth) of current Japanese font. The command \zh sets this register to the correct value, and “return” this register itself.

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

\ltj@curjfnt (attribute) If this attribute is a positive number, it stores the font number of current Japanese font for horizontal direction. If this attribute is negative, it means that the Japanese font for horizontal direction is not loaded—LuaTeX-ja only knows its size and JFM.

\ltj@curtfnt (attribute) Similar to \ltj@curjfnt, but with current Japanese font for vertical direction.

\ltj@charclass (attribute) The character class of a \texttt{JAchar}. This attribute is only set on a \texttt{glyph_node} which contains a \texttt{JAchar}.

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

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

\ltj@tablshift (attribute)
\ltj@tkblshift (attribute)
\ltj@autospc (attribute) Whether the auto insertion of \texttt{kanjiskip} is allowed at the node.
\ltj@autoxspc (attribute) Whether the auto insertion of \texttt{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:

\texttt{italic} \((1)\) Kerns from italic correction (\texttt{/}), or from kerning information of a Japanese font. These kerns are “ignored” in the insertion process of \texttt{JAgue}, unlike explicit \texttt{\kern}.

\texttt{packed} \((2)\)
\texttt{kinsoku} \((3)\) Penalties inserted for the word-wrapping process (\texttt{kinsoku shori}) of Japanese characters.
\texttt{from_jfm}–\texttt{(from_jfm + 63)} \((4–67)\) Glues/kerns from JFM.
\texttt{kanji\_skip} \((68)\), \texttt{kanji\_skip\_jfm} \((69)\) Glues from \texttt{kanjiskip}.
\texttt{xkanji\_skip} \((70)\), \texttt{xkanji\_skip\_jfm} \((71)\) Glues from \texttt{xkanjiskip}.
\texttt{processed} \((73)\) Nodes which is already processed by ....
Glues from an italic correction, but already processed in the insertion process of JAglues.

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

Glues from \insert[x]kanjiskip.

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

\ltj@dir (attribute) dir\_node\_auto (128)
\ltj@dir (attribute) dir\_node\_manual (256)

Furthermore, LuaTEX-ja uses several user-defined whatsit nodes for internal processing. All those nodes except direction whatsis store a natural number (hence its type is 100). direction whatsis store a node list, hence its type is 110. Their user id (used for distinguish user-defined whatsis) are allocated by luatexbase.newuserwhatsitid.

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

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

Nodes for JAchar which processes by luaotfload won’t be applied, and the character code is stored in the value field. Each node of this type are converted to a glyph node after processes by luaotfload. Nodes of this type is used in \CID and \UTF.

Similar to char\_by\_cid whatsis above. These nodes are for ALchar which the callback process of luaotfload won’t be applied.

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 ...

direction

These whatsis will be removed during the process of inserting JAglues.

14.2 Stack system of LuaTEX-ja

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:

```latex
1 \ltjsetparameter{kanjiskip=0pt}ふがふが。
2 \setbox0=\hbox{
3 \ltjsetparameter{kanjiskip=5pt}ほげほげ}
4 \box0。
5 ぴよぴよ
```

As described in Subsection 9.1, the only effective value of kanjiskip in an hbox is the latest value, so the value of kanjiskip which applied in the entire hbox should be 5pt. However, by the implementation method of LuaTEX, this "5pt" cannot be known from any callbacks. In the tex/packaging.w, which is a file in the source of LuaTEX, there are the following codes:

```c
1226 void package(int c)
1227 {
1228     scaled h; /* height of box */
1229     halfword p; /* first node in a box */
1230     scaled d; /* max depth */
1231     int grp;
1232     grp = cur_group;
```
\[d = \text{box\_max\_depth};\]
\[\text{unsave}();\]
\[\text{save\_ptr} = 4;\]
\[\text{if (cur\_list.mode\_field == -hmode)} \{
\quad \text{cur\_box} = \text{filtered\_hpack}(\text{cur\_list.head\_field},\n\quad \quad \text{cur\_list.tail\_field}, \text{saved\_value}(1),\n\quad \quad \text{saved\_level}(1), \text{grp}, \text{saved\_level}(2));\n\quad \text{subtype}(%s) = \text{HLIST\_SUBTYPE\_HBOX};\]

Notice that \text{unsave}() is executed before \text{filtered\_hpack}(), where \text{hpack\_filter} callback is executed) here. So "5 pt" in the above source is orphaned at \text{unsave}(), and hence it can't be accessed from \text{hpack\_filter} callback.

\section*{Implementation} The code of stack system is based on that in a post of Dev-luatex mailing list\(^{14}\).

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

To resolve the problem mentioned in above paragraph "Background", Lua\TeX-ja uses another trick. When the stack level is about to be increased, a \texttt{whatsit} node whose type, subtype and value are 44 (user\_defined), stack\_marker and the current group level respectively is appended to the current list (we refer this node by stack\_flag). This enables us to know whether assignment is done just inside a hbox. Suppose that the stack level is \(s\) and the \TeX 's group level is \(t\) just after the hbox group, then:

- If there is no stack\_flag node in the list of the contents of the hbox, then no assignment was occurred inside the hbox. Hence values of parameters at the end of the hbox are stored in the stack level \(s\).
- If there is a stack\_flag node whose value is \(t + 1\), then an assignment was occurred just inside the hbox group. Hence values of parameters at the end of the hbox are stored in the stack level \(s + 1\).
- If there are stack\_flag nodes but all of their values are more than \(t + 1\), then an assignment was occurred in the box, but it is done in more internal group. Hence values of parameters at the end of the hbox are stored in the stack level \(s\).

Note that to work this trick correctly, assignments to \texttt{\ltj@stack} and \texttt{\ltj@group@level} have to be local always, regardless the value of \texttt{\globaldefs}. To solve this problem, we use another trick: the assignment \texttt{\directlua{\text{tex.globaldefs}=0}} is always local.

\subsection{Lua functions of the stack system}

In this subsection, we will see how a user use Lua\TeX-ja's stack system to store some data which obeys the grouping of \TeX.

The following function can be used to store data into a stack:
\texttt{luatexja.stack.set\_stack\_table(index, <any> data)}

Any values which except \texttt{nil} and \texttt{NaN} are usable as \texttt{index}. However, a user should use only negative integers or strings as \texttt{index}. Since natural numbers are used by Lua\TeX-ja itself. Also, whether \texttt{data} is stored locally or globally is determined by \texttt{luatexja.isglobal} (stored globally if and only if \texttt{luatexja.isglobal == 'global'}).

Stored data can be obtained as the return value of
\texttt{luatexja.stack.get\_stack\_table(index, <any> default, <number> level)}

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

\subsection{Extending Parameters}

Keys for \texttt{\ltjsetparameter} and \texttt{\ltjgetparameter} can be extended, as in \texttt{luatexja-adjust}.

\footnote{\texttt{Dev-luatex} \texttt{tex.currentgrouplevel}, a post at 2008/8/19 by Jonathan Sauer.}
Figure 14. Definition of parameter setting commands

**Setting parameters**  Figure 14 shows the most outer definition of two commands, \ltjsetparameter and \ltjglobalsetparameter. Most important part is the last \setkeys, which is offered by the xkeyval package.

Hence, to add a key in \ltjsetparameter, one only have to add a key whose prefix is ltj and whose family is japaram, as the following.
\begin{verbatim}
define@key[ltj]{japaram}{...}{...}
\end{verbatim}
\ltjsetparameter and \ltjglobalsetparameter automatically sets luatexja.isglobal. Its meaning is the following.

\begin{equation}
\text{luatexja.isglobal} = \begin{cases} 
'global' & \text{(global assignment)}, \\
'\ ' & \text{(local assignment)}.
\end{cases}
\end{equation}

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

**Getting parameters** \ltjgetparameter is implemented by a Lua script.

For parameters that do not need additional arguments, one only have to define a function in the table luatexja.unary_pars. For example, with the following function, \ltjgetparameter{hoge} returns a string 42.
\begin{verbatim}
function luatexja.unary_pars.hoge (t) 
return 42 
end
\end{verbatim}

Here the argument of luatexja.unary_pars.hoge is the stack level of LuaTeX-ja’s stack system (see Subsection 14.2).

On the other hand, for parameters that need an additional argument (this must be an integer), one have to define a function in luatexja.binary_pars first. For example,
\begin{verbatim}
function luatexja.binary_pars.fuga (c, t) 
return tostring(c) .. ', ' .. tostring(42) 
end
\end{verbatim}

Here the first argument 𝑡 is the stack level, as before. The second argument 𝑐 is just the second argument of \ltjgetparameter.

For parameters that need an additional argument, one also have to execute the TeX code like \ltj@@decl@array@param{fuga}
to indicate that "the parameter fuga needs an additional argument".

15 Linebreak after a Japanese Character

15.1 Reference: behavior in \pTeX

In \pTeX, a line break after a Japanese character doesn’t emit a space, since words are not separated by spaces in Japanese writings. However, this feature isn’t fully implemented in Lua\TeX-ja due to the specification of
We omitted about category codes 9 (ignored), 14 (comment), and 15 (invalid) from the above diagram. We also ignored the input like "~A" or "~df".

When a character whose category code is 0 (escape character) is seen by \TeX, the input processor scans a control sequence (scan a c.s.). These paths are not shown in the above diagram.

After that, the state is changed to State S (skipping blanks) in most cases, but to State M (middle of line) sometimes.

Figure 15. State transitions of \TeX’s input processor

15.2 Behavior in \LaTeX-ja

States in the input processor of \LaTeX is the same as that of \TeX, and they can’t be customized by any callbacks. Hence, we can only use \texttt{process\_input\_buffer} and \texttt{token\_filter} callbacks for to suppress a space by a line break which is after Japanese characters.

However, \texttt{token\_filter} callback cannot be used either, since a character in category code 5 (end-of-line) is converted into a space token in the input processor. So we can use only the \texttt{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 \LaTeX-ja are as follows:

A character whose character code is \texttt{\ljlineendcomment} is appended to an input line, before \LaTeX actually process it, if and only if the following three conditions are satisfied:

1. The category code of \texttt{\endlinechar} is 5 (end-of-line).
2. The category code of \texttt{\ljlineendcomment} itself is 14 (comment).

\textsuperscript{15}Its default value is "\texttt{FFFFF}, so \texttt{UFFFFF} is used. The category code of \texttt{UFFFFF} is set to 14 (comment) by \LaTeX-ja.

\textsuperscript{16}Usually, it is \texttt{(return)} (whose character code is 13).
3. The input line matches the following "regular expression":

\((\text{any char})^*\text{(J}\text{A}\text{c}\text{h}\text{a}\text{r})(\{\text{catcode }= 1\} \cup \{\text{catcode } = 2\})^*\)

\[\begin{align*}
\text{Remark} & \quad \text{The following example shows the major difference from the behavior of p\TeX.} \\
\text{Remark} & \quad \text{The following example shows the major difference from the behavior of p\TeX.} \\
\text{Remark} & \quad \text{The following example shows the major difference from the behavior of p\TeX.} \\
\text{Remark} & \quad \text{The following example shows the major difference from the behavior of p\TeX.} \\
\text{Remark} & \quad \text{The following example shows the major difference from the behavior of p\TeX.}
\end{align*}\]

It is not strange that "あ" does not printed in the above output. This is because \TeX Gyre Termes does not contain "あ", and because "あ" in line 3 is considered as an \texttt{ALchar}.

Note that there is no space before "ゆ" in the output, but there is a space before "う". This follows from following reasons:

- When line 3 is processed by \texttt{process_input_buffer} callback, "あ" is considered as an \texttt{JAchar}. Since line 3 ends with an \texttt{JAchar}, the comment character (whose character code is \texttt{\textbackslash jlineendcomment}) is appended to this line, and hence the linebreak immediately after this line is ignored.

- When line 4 is processed by \texttt{process_input_buffer} callback, "い" is considered as an \texttt{ALchar}. Since line 4 ends with an \texttt{ALchar}, the linebreak immediately after this line emits a space.

### 15.3 Composition of Kana from Combining Character Sequences

In (u)p\TeX, Hiragana and Katakana in the NFD form\textsuperscript{17} are normalized to precomposed characters before (u)p\TeX looks into the input line. Character sequences which are involved in this normalization are shown in Table 16.

The above normalization process is also performed in Lua\TeX-ja version 20220103.0 or later, because fonts might not have these transformation as an OpenType feature.

### 16 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 \texttt{listings}([4]) resolves the problem forcibly.

In Lua\TeX-ja, the problem is resolved by using the \texttt{process_input_buffer} callback. The callback function inserts the output command (active character \texttt{\textbackslash jlineendcomment}) before each letter above \texttt{U\textbackslash e8888}. This method can omits the process to make all Japanese characters active (most of the activated characters are not used in many cases).

If the listings package and Lua\TeX-ja were loaded, then the patch \texttt{ltj\textendash listings} is loaded automatically at \texttt{\begin{document}}.

\textsuperscript{17}namely, character sequences which contains "COMBINING KATAKANA-HIRAGANA VOICED SOUND MARK" (U+3899) and "COMBINING KATAKANA-HIRAGANA SEMI-VOICED SOUND MARK" (U+389A).
16.1 Notes and additional keys

- **Variation selectors** \littj-listings adds two keys, namely \texttt{vsraw} and \texttt{vscmd}, which specify how variation selectors are treated in \texttt{lstlisting} or other environments. Note that these additional keys are not usable in the preamble, since \littj-listings is loaded at \texttt{\begin{document}}.

\texttt{vsraw} is a key which takes a boolean value, and its default value is false.

- If the \texttt{vsraw} key is true, then variation selectors are “combined” with the previous character.

\begin{lstlisting}[vsraw=true]
葛
城市, 葛
飾区, 葛西
\end{lstlisting}

- If the \texttt{vsraw} key is false, then variation selectors are typeset by an appropriate command, which is specified by the \texttt{vscmd} key. The default setting of the \texttt{vscmd} key produces the following.

\begin{lstlisting}[vsraw=false, vscmd=\ltjlistingsvsstdcmd]
葛
城市, 葛
飾区, 葛西
\end{lstlisting}

For example, the following code is the setting of the \texttt{vscmd} key in this document.

\begin{verbatim}
def\IVSA#1#2#3#4#5{\hbox to1em{\hss\textcolor{blue}{\raisebox{3.5pt}{\normalfont\ttfamily\fboxsep=0.5pt\fbox{\hbox to0.75em{\hss\tiny \oalign{0#1#2\crcr#3#4#5\crcr}}\hss}}}}\hss}
\}
def\IVSB#1{\expandafter\IVSA\directlua{local cat_str = luatexbase.catcodetables['string']\tex.sprint(cat_str, string.format('%X', 0xE00EF+#1))}}\}
\lstset{vscmd=\IVSB}
\end{verbatim}

The default output command of variation selectors is stored in \texttt{\ltjlistingsvsstdcmd}.

- **The doubleletterspace key** Even the column format is \texttt{[c]} fixed, sometimes characters are not vertically aligned. The following example is typeset with \texttt{basewidth=2em}, and you’ll see the leftmost “H” are not vertically aligned.

\begin{verbatim}
: H :
: H H H H :
\end{verbatim}

\littj-listing adds the \texttt{doubleletterspace} key (not activated by default, for compatibility) to improve the situation, namely doubles inter-character space in each output unit. With this key, the above input now produces better output.

\begin{verbatim}
: H :
: H H H H :
\end{verbatim}

16.2 Class of characters

Roughly speaking, the listings package processes input as follows:

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

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

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

<table>
<thead>
<tr>
<th></th>
<th>Letter</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\lst@ifletter</code></td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td><code>\lst@ifkanji</code></td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

**Meaning**
- `\lst@ifletter` char in an identifier
- `\lst@ifkanji` other alphabet

<table>
<thead>
<tr>
<th></th>
<th>Kanji</th>
<th>Open</th>
<th>Close</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\lst@ifletter</code></td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td><code>\lst@ifkanji</code></td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>

**Meaning**
- `\lst@ifletter` most of Japanese char
- `\lst@ifkanji` opening brackets
- `\lst@ifkanji` closing brackets

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

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

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

<table>
<thead>
<tr>
<th></th>
<th>Letter</th>
<th>Other</th>
<th>Kanji</th>
<th>Open</th>
<th>Close</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Next</strong></td>
<td>going collects</td>
<td>outputs</td>
<td>going outputs</td>
<td>going collects</td>
<td></td>
</tr>
<tr>
<td><strong>Prev</strong></td>
<td>going outputs</td>
<td>going collects</td>
<td>going outputs</td>
<td>going collects</td>
<td></td>
</tr>
</tbody>
</table>

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).

Characters above or equal to `U+0080` except Variation Selectors are classified into above 5 classes by the following rules:

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

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

This classification process is executed every time a character appears in the `lstlisting` environment or other environments/commands.
17 Cache Management of LuaT\exp\-ja

LuaT\exp\-ja creates some cache files to reduce the loading time, in a similar way to the luaotfload package:

- Cache files are usually stored in (and loaded from) \texttt{TEXMFVAR/luatexja/}.
- In addition to caches of the text form (the extension is ".lua.gz", because they are compressed by gzip), caches of the binary (bytecode) form are supported.
  - In loading a cache, the binary cache precedes the text form.
  - When LuaT\exp\-ja updates a compressed text cache \texttt{hoge.lua.gz}, its binary version is also updated.

17.1 Use of cache

LuaT\exp\-ja uses the following cache:

- \texttt{ltj-cid-auto-adobe-japan1.{lua.gz,luc}}
  - The font table of a CID-keyed non-embedded Japanese font. This is loaded in every run. It is created from three CMaps, \texttt{UniJIS2004-UTF32-\{H,V\}} and \texttt{Adobe-Japan1-UCS2}, and this is why these two CMaps are needed in the first run of LuaT\exp\-ja.
  - Similar caches are created as Table 17, if you specified \texttt{cid key} in \texttt{\jfont} to use other CID-keyed non-embedded fonts for Chinese or Korean, as in Page 27.

- \texttt{ltj-kinsoku.luc}
  - The bytecode cache which default \texttt{kinsoku} parameters are stored.

- \texttt{ltj-jisx0208.luc}
  - The bytecode version of \texttt{ltj-jisx0208.lua}. This is the conversion table between JIS X 0208 and Unicode which is used in Kanji-code conversion commands for compatibility with \LaTeX.

- \texttt{ltj-ivd_aj1.luc}
  - The bytecode version of \texttt{ltj-ivd_aj1.lua}.

- \texttt{extra_***.{lua.gz,luc}}
  - This file contains some information (especially for vertical typesetting) about the font ‘***’.

17.2 Internal

Cache management system of LuaT\exp\-ja is stored in \texttt{luatexja.base} (\texttt{ltj-base.lua}). There are four public functions for cache management in \texttt{luatexja.base}, where \texttt{\{filename\}} stands for the file name 	extit{without suffix}:

- \texttt{save_cache\{\{filename\}, \{data\}\}}
  - Save a non-nil table \texttt{\{data\}} into a cache \texttt{\{filename\}}. Both the compressed text form \texttt{\{filename\}.lua.gz} and its binary version are created or updated.

- \texttt{save_cache_luc\{\{filename\}, \{data\}[, \{serialized_data\}]\}}
  - Same as \texttt{save_cache}, except that only the binary cache is updated. The third argument \texttt{\{serialized_data\}} is not usually given. But if this is given, it is treated as a string representation of \texttt{\{data\}}.
load_cache(filename, outdate)

Load the cache (filename). (outdate) is a function which takes one argument (the contents of the cache), and its return value is whether the cache is outdated.

load_cache first tries to read the binary cache (filename).luc. If its contents is up-to-date, load_cache returns the contents. If the binary cache is not found or its contents is outdated, load_cache tries to read the compressed text form (filename).lua.gz. Hence, the return value of load_cache is non-nil, if and only if the updated cache is found.

remove_cache(filename)

Remove the cache (filename).
References


