The LuaTeX-ja package

The LuaTeX-ja project team

November 3, 2015
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 LaTeX-ja package is a macro package for typesetting high-quality Japanese documents when using LaTeX.

1.1 Backgrounds

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

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

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

1.2 Major changes from \texttt{p\TeX}

The LaTeX-ja package is under much influence of \texttt{p\TeX} engine. The initial target of development was to implement features of \texttt{p\TeX}. However, implementing all feature of \texttt{p\TeX} is impossible, since all process of LaTeX-ja must be implemented only by Lua and \texttt{T\TeX} macros. Hence LaTeX-ja is not a just porting of \texttt{p\TeX}; unnatural specifications/behaviors of \texttt{p\TeX} were not adopted.

The followings are major changes from \texttt{p\TeX}. For more detailed information, see Part III or other sections of this manual.

\textbf{Command names} \texttt{p\TeX} addes several primitives, such as \texttt{\kanjiskip}, \texttt{\prebreakpenalty}, and, \texttt{\ifydir}. They can be used as follows:

\begin{verbatim}
\kanjiskip=10pt \dimen0=\kanjiskip
\tbaselineshift=0.1zw
\dimen0=\tbaselineshift
\prebreakpenalty=100
\ifydir ... \fi
\end{verbatim}

However, we cannot use them under \texttt{Lua\TeX}-ja. Instead of them, we have to write as the following.

\begin{verbatim}
\ltjsetparameter{kanjiskip=10pt} \dimen0=\ltjgetparameter{kanjiskip}
\ltjsetparameter{talbaselineshift=0.1\zw}
\dimen0=\ltjgetparameter{talbaselineshift}
\ltjsetparameter{prebreakpenalty={`,100}}
\ifnum\ltjgetparameter{direction}=4 ... \fi
\end{verbatim}

Note that \texttt{p\TeX} adds new two useful units, namely \texttt{zw} and \texttt{zh}. As shown above, they are changed by \texttt{\zw} and \texttt{\zh} respectively, in \texttt{Lua\TeX}-ja.

\textbf{Linebreak after a Japanese character} In \texttt{p\TeX}, 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, \texttt{Lua\TeX}-ja doesn’t have this feature completely, because of a specification of \texttt{Lua\TeX}. For the detail, see Section 13.
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 JAglue) is rewritten from scratch.

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

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

• Caution: due to above two points, many methods which did for the dividing the process of the insertion of JAglue in p\TeX are not effective anymore. In concrete terms, the following two methods are not effective anymore:

    ちょっと ちょっと

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

    ちょっと

• 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 not be confused with Omega-style direction support of LuaTeX itself. Due to implementation, the dimension returned by \wd, \ht, or \dp depends on the content of the register only. This is major difference with p\TeX.

\discretionary  Japanese characters in discretionary break (\discretionary) is not supported.

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

From this version, characters which belongs both ISO 8859-1 and JIS X 0208, such as ¶ and §, are now typeset in alphabetic fonts. This means that without the \fontspec (and luatexja-fontspec) package, these characters are not typeset correctly.

1.3 Notations

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

• Characters are classified into following two types. Note that the classification can be customized by a user (see Subsection 4.1).

  – JAchar: standing for characters which is used in Japanese typesetting, such as Hiragana, Katakana, Kanji, and other Japanese punctuation marks.
  – ALchar: standing for all other characters like latin alphabets.

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

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

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

• In this document, natural numbers start from zero. \omega denotes the set of all natural numbers.
1.4 About the project

Project Wiki  Project Wiki is under construction.

- [https://osdn.jp/projects/luatex-ja/wiki/FrontPage%28en%29](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 Lua\TeX-ja package.

- Lua\TeX beta-0.80.0 (or later)
- luaotfload v2.5 (or later)
- adobemapping (Adobe cmap and pdfmapping files)
- everytex (if you want to use Lua\TeX-ja with \LaTeX 2\epsilon)
- fontspec v2.4
- IPAex fonts (http://ipafont.ipa.go.jp/)

In summary, this version of Lua\TeX-ja no longer supports \TeX Live 2014 (or older version).

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

- MiKTeX (in luatexja.tar.lzma); see the next subsection
- \TeX Live (in texmf-dist/tex/luatex/luatexja)
- W32\TeX (in luatexja.tar.xz)

IPAex fonts are also available in these distributions.

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:
     
     $ git clone git://git.osdn.jp/gitroot/luatex-ja/luatexja.git
   
   - Download the tar.gz archive of HEAD in the master branch from
     
     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 src/ and several other sub-directories. But only the contents in 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 and ltj-kinsoku.lua (the file which stores default “kinsoku” parameters):
   
   $ cd src
   $ lualatex ltjclasses.ins
   $ lualatex ltjsclasses.ins
   $ lualatex ltjltxdoc.ins
   $ luatex ltj-kinsoku_make.tex

   Do not forget The last line (processing ltj-kinsoku_make.tex).*.{dtx,ins} and ltj-kinsoku_make.tex used here are not needed in regular use.

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

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

For changes from p\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.

- L\a\TeX-ja is very slower than p\TeX. Generally speaking, LuaJIT\TeX processes L\a\TeX-ja about 30\% faster than L\a\TeX, but not always.

- (Outdated) note for MiK\TeX{} users  L\a\TeX-ja requires that several CMap files must be found from L\a\TeX. Strictly speaking, those CMaps are needed only in the first run of L\a\TeX-ja after installing or updating. But it seems that MiK\TeX 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-xlrmgbm.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 L\a\TeX-ja in this directory, and finally delete the temporary directory.

2.3 Using in plain \TeX

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

\input luatexja.sty

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

- The following 12 Japanese fonts are preloaded:

| direction | classification | font name | "10 pt" | "7 pt" | "5 pt"
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>yoko (horizontal)</td>
<td>mincho</td>
<td>IPAex Mincho</td>
<td>\tenmin</td>
<td>\sevenmin</td>
<td>\fivemin</td>
</tr>
<tr>
<td>gothic</td>
<td>IPAex Gothic</td>
<td>\tengt</td>
<td>\sevengt</td>
<td>\fivegt</td>
<td></td>
</tr>
<tr>
<td>tate (vertical)</td>
<td>mincho</td>
<td>IPAex Mincho</td>
<td>\tentmin</td>
<td>\seventmin</td>
<td>\fivemin</td>
</tr>
<tr>
<td>gothic</td>
<td>IPAex Gothic</td>
<td>\tentgt</td>
<td>\seventgt</td>
<td>\fivegt</td>
<td></td>
</tr>
</tbody>
</table>

- With luatexja.cfg, one can use other fonts as "default" Japanese fonts (Subsection 3.5).
- 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

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

2.4 Using in \La\TeX

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

\usepackage{luatexja}

It also does minimal settings (counterparts in p\TeX{} are p\La fonts.dtx and p\La\ldef ls.ltx):

- Font encodings for Japanese fonts is JY3 (for horizontal direction) and JT3 (for vertical direction).
- Traditionally, Japanese documents use two typeface categories: 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.

\footnote{UniJIS2004-UTF32-\{H,V\} and Adobe-Japan1-UCS2.}
• By default, the following fonts are used for \textit{mincho} and \textit{gothic}:

<table>
<thead>
<tr>
<th>classification</th>
<th>family</th>
<th>\textit{mdseries}</th>
<th>\textit{bfseries}</th>
<th>\textit{scale}</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{mincho} (明朝体)</td>
<td>mc</td>
<td>IPAex Mincho</td>
<td>IPAex Gothic</td>
<td>0.962216</td>
</tr>
<tr>
<td>\textit{gothic} (ゴシック体)</td>
<td>gt</td>
<td>IPAex Gothic</td>
<td>IPAex Gothic</td>
<td>0.962216</td>
</tr>
</tbody>
</table>

Note that the bold series in both family are same as the medium series of \textit{gothic} family. There is no italic nor slanted shape for these \textit{mc} and \textit{gt}.

• Japanese characters in math mode are typeset by the font family \textit{mc}.

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

\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 \texttt{article}.cls, \texttt{book}.cls, and so on. At the present, we have the counterparts of \texttt{jclasses} (standard classes in \texttt{pL\TeX}) and \texttt{jsclasses} (classes by Haruhiko Okumura), namely, \texttt{ltjclasses}\footnote{\texttt{ltjarticle.cls}, \texttt{ltjbook.cls}, \texttt{ltjreport.cls}, \texttt{ltjtarticle.cls}, \texttt{ltjbook.cls}, \texttt{ltjreport.cls}. The latter \texttt{ltjt*.cls} are for vertically writtened Japanese documents.} and \texttt{ltjsclasses}\footnote{\texttt{ltjsarticle.cls}, \texttt{ltjsbook.cls}, \texttt{ltjskiyou.cls}.}

\section*{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.
\@begindvi ->\unvbox \@begindvibox \global \let \@begindvi \@empty
\end{verbatim}

Now, Lua\textsc{TeX}-ja automatically applies the patch \texttt{lltjp-geometry} to the geometry package, when the direction of the document is \textit{tate} (vertical writing). This patch \texttt{lltjp-geometry} also can be used in \texttt{pL\TeX}; for the detail, please refer \texttt{lltjp-geometry.pdf} (Japanese).

\section{Changing Fonts}
\subsection{plain \textsc{TeX} and \textsc{Pdf}\textsc{TeX} 2\textsubscript{\textit{e}}}

\section*{plain \textsc{TeX}}  
To change Japanese fonts in plain \textsc{TeX}, you must use the command \texttt{\jfont} and \texttt{\tfont}. So please see Subsection 7.1.

\section*{\textsc{Pdf}\textsc{TeX} 2\textsubscript{\textit{e}} (NFSS2)}  
For \textsc{Pdf}\textsc{TeX} 2\textsubscript{\textit{e}}, Lua\textsc{TeX}-ja adopted most of the font selection system of \textsc{pL\TeX} 2\textsubscript{\textit{e}} (in \texttt{plfonts.dtx}).

• Commands \texttt{\fontfamily}, \texttt{\fontseries}, and \texttt{\fontshape} can be used to change attributes of Japanese fonts.
\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 10.1.

- 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}{bx}{n}{<-> s*KozMinPr6N-Bold:jfm=ujis;-kern}{}
% Kozuka Mincho Pr6N Bold

\begin{remark}{Japanese characters in math mode} Since \TeX supports Japanese characters in math mode, there are sources like the following:
\begin{itemize}
\item $f_{\text{高温}}$ \quad ($f_{\text{high temperature}}$).
\item $y=(x-1)^2+2$ \quad よって \quad $y>0$
\item $5\in \text{素} : = \{ p \in \mathbb{N} : \text{$p$ is a prime} \}$.
\end{itemize}
\end{remark}

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

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

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

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

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 7.5.

### 3.2 luatexja-fontspec package

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

\usepackage[⟨options⟩]{luatexja-fontspec}

This luatexja-fontspec package automatically loads luatexja and fontspec packages, if needed.

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

<table>
<thead>
<tr>
<th>Japanese fonts</th>
<th>\jfontspec</th>
<th>\setmainjfont</th>
<th>\setsansjfont</th>
<th>\setsanjfont</th>
<th>\setmonojfont</th>
</tr>
</thead>
<tbody>
<tr>
<td>alphabetic fonts</td>
<td>\newjfontfamily</td>
<td>\newjfontface</td>
<td>\defaultjfontfeatures</td>
<td>\addjfontfeatures</td>
<td>\addjfontfeatures</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Japanese fonts</th>
<th>\newfontfamily</th>
<th>\newfontface</th>
<th>\defaultfontfeatures</th>
<th>\addfontfeatures</th>
</tr>
</thead>
<tbody>
<tr>
<td>alphabetic fonts</td>
<td>\newfontfamily</td>
<td>\newfontface</td>
<td>\defaultfontfeatures</td>
<td>\addfontfeatures</td>
</tr>
</tbody>
</table>
The package option of luatexja-fontspec are the followings:

**match**
If this option is specified, usual family-changing commands such as \rmfamily, \texttrm, \sffamily,... also change Japanese font family.

Note that \setmonojfont is defined if and only if this match option is specified.

**pass=(opts)**
(Obsoleted) Specify options (opts) which will be passed to the fontspec package.

**scale=(float)**
Override the ratio of the font size of Japanese fonts to that of alphabetic fonts. The default value is calculated automatically (for example, about 0.924865 when the \ltjsarticle class is used).

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}

The reason that \setmonojfont is not defined by default is that it is popular for Japanese fonts that nearly all Japanese glyphs have same widths. Also 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 Lua\TeX\-ja (see 7.1).

Below is an example of \jfontspec.

1 \jfontspec[CJKShape=NLC]{KozMinPr6N-Regular}  
2 JIS~X~0213:2004 →辻
3 \jfontspec[CJKShape=JIS1990]{KozMinPr6N-Regular}  
4 JIS~X~0208:1990 →辻

### 3.3 Presets of Japanese fonts

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 and a part of PXchfon package by Takayuki Yato.

One can specified other options other than listed in this subsection. These are simply passed to the luatexja-fontspec\footnote{If nfssonly option is not specified; in this case these options are simply ignored.}. 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}

#### General options

**fontspec**
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. This option is enabled by default.

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

**nfssonly**
With this option, selecting Japanese fonts won’t be performed using the functionality of the fontspec package, but only standard NFSS2 (hence without \addjfontfeatures 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:
In this case, one can use \setmainfont etc. to select alphabetic fonts.

nodeluxe
Use one-weighted mincho and gothic font families. This means that \mcfamily\bfseries, \gfamily\bfseries and \gtfamily\mdseries use the same font. This option is enabled by default.

deluxe
Use mincho with two weights (medium and bold), gothic with three weights (medium, bold and heavy), and rounded gothic. The heavy weight of gothic can be used by "changing the family" \gtebfamily, or \textgteb{...}. This is because the fontspec package can handle only medium (\mdseries) and bold (\bfseries).

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.

90jis
Use 90JIS glyph variants if possible.

jis2004
Use JIS2004 glyph variants if possible.

jis
Use the JFM jfm-jis.1ua, instead of jfm-ujis.1ua, which is the default JFM of LuaTeX-ja.

Note that 90jis and jis2004 only affect with mincho, gothic (and possibly rounded gothic) defined by this package. We didn’t taken account of when both 90jis and jis2004 are specified.

Presets for multi weight
Besides morisawa-pro and morisawa-pr6n presets, fonts are specified by font name, not by file name. In following tables, starred fonts (e.g. KozGo...-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>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></td>
<td>KozGoPro-Medium</td>
<td>KozGoProVI-Medium</td>
<td>KozGoPr6N-Medium</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>heavy</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 are bundled with Mac OS X 10.5 or later. Some editions of a Japanese word-processor “一太郎 2012” includes Hiragino ProN fonts. Note that the heavy weight of gothic family only supports Adobe-Japan1-3 character collection (Std/StdN).

Provided by \mgfamily and \textmg, because rounded gothic is called maru gothic (丸ゴシック) in Japanese.
### family series | morisawa-pro | morisawa-pr6n
---|---|---
mincho medium | A-OTF-RyuminPro-Light.otf | A-OTF-RyuminPr6N-Light.otf
bold | A-OTF-FutoMinA101Pro-Bold.otf | A-OTF-FutoMinA101Pr6N-Bold.otf

### family series | yu-win | yu-osx
---|---|---
mincho medium | YuMincho-Regular | YuMincho Medium
bold | YuMincho-Demibold | YuMincho Demibold

### family series | default, 90jis option | jis2004 option
---|---|---
mincho medium | Moga90Mincho | MogaMincho
bold | Moga90Mincho Bold | MogaMincho Bold

gothic medium | Moga90Gothic | MogaGothic
bold | Moga90Gothic Bold | MogaGothic Bold

MogaMincho, MogaGothic, and MoboGothic. These fonts can be downloaded from [http://yozvox.web.fc2.com/](http://yozvox.web.fc2.com/).
Using HG fonts

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

<table>
<thead>
<tr>
<th></th>
<th>ipa-hg</th>
<th>ipaex-hg</th>
<th>ms-hg</th>
</tr>
</thead>
<tbody>
<tr>
<td>mincho medium</td>
<td>IPA Mincho</td>
<td>IPAex Mincho</td>
<td>MS Mincho</td>
</tr>
<tr>
<td>mincho bold</td>
<td>HG Mincho E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gothic medium</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>without deluxe</td>
<td>IPA Gothic</td>
<td>IPAex Gothic</td>
<td>MS Gothic</td>
</tr>
<tr>
<td>with jis2004</td>
<td>IPA Gothic</td>
<td>IPAex Gothic</td>
<td>MS Gothic</td>
</tr>
<tr>
<td>otherwise</td>
<td>HG Gothic M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gothic bold</td>
<td>HG Gothic E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gothic heavy</td>
<td>HG Soei Kaku Gothic UB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rounded gothic</td>
<td>HG Maru Gothic PRO</td>
<td></td>
<td></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.).
- **90jis** by file name (hgrme.ttc, hgrge.ttc, hgrsgu.ttc, hgrsmp.ttf).
- **jis2004** by file name (hgrme04.ttc, hgrge04.ttc, hgrsgu04.ttc, hgrsmp04.ttf).

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

Under \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.

- \fjfontspec{KozMinPr6N-Regular.otf}
- \UTF{9DD7} 森
- \UTF{9592} 外と内田百閒とが
- \UTF{9AD9} 島屋に行く。
- \CID{7652}飾区の
- \CID{13706} 野家,
- \CID{1481} 城市, 葛西駅,
- \CID{8705}\UTF{FA11} 高崎と
- \CID{1219}半角{はんかくカタカナ}

### 3.5 Changing default Japanese fonts

If \luaexja.cfg can be seen from Lua\TeX, Lua\TeX-ja automatically reads it. The main use of \luaexja.cfg is for changing default Japanese fonts, when IPAex fonts cannot be installed in \TeX system. One should not overuse this \luaexja.cfg; fonts which will be used in a document should be specified in its source.

For example,

\begin{verbatim}
\def\ltj@stdmcfont{IPAMincho}
\def\ltj@stdgtfont{IPA Gothic}
\end{verbatim}

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

For another example, the following two lines makes that non-embedded fonts Ryumin-Light and GothicBBB-Medium as default Japanese fonts (as the earlier version of Lua\TeX-ja):

\begin{verbatim}
\def\ltj@stdmcfont{psft:Ryumin-Light}
\def\ltj@stdgtfont{psft:GothicBBB-Medium}
\end{verbatim}
4 Changing Internal Parameters

There are many internal parameters in LuaTeX-ja. And due to the behavior of LuaTeX, most of them are not stored as internal register of \TeX, but as an original storage system in LuaTeX-ja. Hence, to assign or acquire those parameters, you have to use commands \texttt{\textbackslash ltjsetparameter} and \texttt{\textbackslash 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 \texttt{\textbackslash ltjdefcharrange}. The next line adds whole characters in Supplementary Ideographic Plane and the character “漢” to the character range 100.

\texttt{\textbackslash 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 you execute the above line, then SIP will belong to the range 100 and be removed from the range 4.

The distinction between \texttt{ALchar} and \texttt{JAchar} is performed by character ranges. This can be edited by setting the \texttt{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 \texttt{ALchar},
- a character which belongs character ranges 2, 3, 6, and 7 is \texttt{JAchar}.

\texttt{\textbackslash ltjsetparameter(jacharrange={-1, +2, +3, -4, -5, +6, +7, -8})}

The argument to \texttt{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 \texttt{ALchar}”, and positive integer $+n$ means that “... is a \texttt{JAchar}”.

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

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

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

Now we describe these eight ranges. The superscript "J" or "A" after the number shows whether each character in the range is treated as \texttt{JAchars} or not by default. These settings are similar to the \texttt{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\textsuperscript{A} The intersection of the upper half of ISO 8859-1 (Latin-1 Supplement) and JIS X 0208 (a basic character set for Japanese). This character range consists of the following characters:

- § ($U+00A7$, Section Sign)
- ’ ($U+00B4$, Spacing acute)
- ” ($U+00A8$, Diaeresis)
- ¶ ($U+00B6$, Paragraph sign)
- ‘ ($U+00B0$, Degree sign)
- × ($U+00D7$, Multiplication sign)
- ± ($U+00B1$, Plus-minus sign)
- ⊗ ($U+00F7$, Division Sign)

Range 1\textsuperscript{A} Latin characters that some of them are included in Adobe-Japan1-6. This range consists of the following Unicode ranges, \textit{except characters in the range 8 above:}

---

14
Table 1. Unicode blocks in predefined character range 3.

<table>
<thead>
<tr>
<th>Block Range</th>
<th>Block Name</th>
<th>Block Range</th>
<th>Block Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>U+2000–U+206F</td>
<td>General Punctuation</td>
<td>U+2070–U+209F</td>
<td>Superscripts and Subscripts</td>
</tr>
<tr>
<td>U+20A0–U+20CF</td>
<td>Currency Symbols</td>
<td>U+2000–U+20FF</td>
<td>Comb. Diacritical Marks for Symbols</td>
</tr>
<tr>
<td>U+2100–U+214F</td>
<td>Letterlike Symbols</td>
<td>U+2150–U+218F</td>
<td>Number Forms</td>
</tr>
<tr>
<td>U+2190–U+21FF</td>
<td>Arrows</td>
<td>U+2200–U+22FF</td>
<td>Mathematical Operators</td>
</tr>
<tr>
<td>U+2300–U+23FF</td>
<td>Miscellaneous Technical</td>
<td>U+2400–U+243F</td>
<td>Control Pictures</td>
</tr>
<tr>
<td>U+2500–U+25FF</td>
<td>Box Drawing</td>
<td>U+2580–U+259F</td>
<td>Block Elements</td>
</tr>
<tr>
<td>U+2900–U+29FF</td>
<td>Supplemental Arrows-B</td>
<td>U+2B00–U+2BFF</td>
<td>Miscellaneous Symbols and Arrows</td>
</tr>
</tbody>
</table>

Table 2. Unicode blocks in predefined character range 6.

<table>
<thead>
<tr>
<th>Block Range</th>
<th>Block Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>U+2460–U+24FF</td>
<td>Enclosed Alphanumerics</td>
</tr>
<tr>
<td>U+3000–U+303F</td>
<td>CJK Symbols and Punctuation</td>
</tr>
<tr>
<td>U+30A0–U+30FF</td>
<td>Katakana</td>
</tr>
<tr>
<td>U+31F0–U+31FF</td>
<td>Katakana Phonetic Extensions</td>
</tr>
<tr>
<td>U+3300–U+33FF</td>
<td>CJK Compatibility</td>
</tr>
<tr>
<td>U+4E00–U+4E9F</td>
<td>CJK Unified Ideographs</td>
</tr>
<tr>
<td>U+FE10–U+FE1F</td>
<td>Vertical Forms</td>
</tr>
<tr>
<td>U+FE50–U+FE6F</td>
<td>Small Form Variants</td>
</tr>
<tr>
<td>U+E0100–U+E01EF</td>
<td>Variation Selectors Supplement</td>
</tr>
</tbody>
</table>

- U+0080–U+00FF: Latin-1 Supplement
- U+0100–U+01FF: Latin Extended-A
- U+0180–U+024F: Latin Extended-B
- U+0250–U+02FF: IPA Extensions
- U+02B0–U+02FF: Spacing Modifier Letters

Range 2[1]: Greek and Cyrillic letters. JIS X 0208 (hence most of Japanese fonts) has some of these characters.

- U+0370–U+03FF: Greek and Coptic
- U+0400–U+04FF: Cyrillic

Range 3[1]: Punctuations and Miscellaneous symbols. The block list is indicated in Table 1.

Range 4[1]: 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:

\[\texttt{utf8charrange}(4)(\%\)
\[\texttt{"500-"10FF, "1200-"1FF, "2440-"245F, "27C0-"27FF, "2A00-"2AFF,}
\["2C00-"2E7F, "4DC0-"4DFF, "A4D0-"A82F, "A840-"ABFF, "FB00-"FE0F,}
\["FE20-"FE2F, "FE70-"FEFF, "10000-"1FFFF, "E000-"F8FF\} \% non-Japanese

Range 5[1]: Surrogates and Supplementary Private Use Areas.

Range 6[1]: Characters used in Japanese. The block list is indicated in Table 2.

Range 7[1]: Characters used in CJK languages, but not included in Adobe-Japan1-6. The block list is indicated in Table 3.

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 textcomp package or the marvosym package.

For example, the codepoint \textparagraph which is provided by the textcomp package is 182. This codepoint corresponds ¶ (U+00B6) in Unicode. Similarly, \Frowny which is provided by the marvosym
Table 3. Unicode blocks in predefined character range 7.

<table>
<thead>
<tr>
<th>Codepoints</th>
<th>Name</th>
<th>Codepoints</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>U+1100–U+11FF</td>
<td>Hangul Jamo</td>
<td>U+2F00–U+2FDF</td>
<td>Kangxi Radicals</td>
</tr>
<tr>
<td>U+2FF0–U+2FFF</td>
<td>Ideographic Description Characters</td>
<td>U+3100–U+312F</td>
<td>Bopomofo</td>
</tr>
<tr>
<td>U+3130–U+318F</td>
<td>Hangul Compatibility Jamo</td>
<td>U+31A0–U+31BF</td>
<td>Bopomofo Extended</td>
</tr>
<tr>
<td>U+31C0–U+31EF</td>
<td>CJK Strokes</td>
<td>U+A000–U+A48F</td>
<td>Yi Syllables</td>
</tr>
<tr>
<td>U+A490–U+A4CF</td>
<td>Yi Radicals</td>
<td>U+A830–U+A83F</td>
<td>Common Indic Number Forms</td>
</tr>
<tr>
<td>U+AC00–U+D7AF</td>
<td>Hangul Syllables</td>
<td>U+D7B0–U+D7FF</td>
<td>Hangul Jamo Extended-B</td>
</tr>
</tbody>
</table>

The package has the same codepoint as § (U+00A7). Hence, as previous versions of LuaTeX-ja, if these characters are treated as JAchars, then \textparagraph produces “ltjchar¶” (in a Japanese font), and \Frowny produces “§” (in a Japanese font).

To avoid such situations, the default setting of LuaTeX-ja is changed in this release so that all characters U+0080–U+00FF are treated as ALchar.

If you want to output a character as ALchar and JAchar regardless the range setting, you can use \ltjalchar and \ltjjachar respectively, as the following example.

```
1 \gtfamily\large % default, ALchar, JAchar
2 ¶, \ltjalchar¶, \ltjjachar¶\ % default: ALchar
3 α, \ltjalcharα, \ltjjacharα % default: JAchar
```

4.2 kanjiskip and xkanjiskip

JAglue is divided into the following three categories:

- Glues/kerns specified in JFM. If \inhibitglue is issued around a JAchar, this glue will not be inserted at the place.
- The default glue which inserted between two JAchars (kanjiskip).
- The default glue which inserted between a JAchar and an ALchar (xkanjiskip).

The value (a skip) of kanjiskip or xkanjiskip can be changed as the following. Note that only their values at the end of a paragraph or a hbox are adopted in the whole paragraph or the whole hbox.

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

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

The value of these parameter can be get by \ltjgetparameter. Note that the result by \ltjgetparameter is not the internal quantities, but a string (hence the cannot be prefixed).

```
kanjiskip: \ltjgetparameter{kanjiskip}, % kanjiskip: 0.0pt plus 0.4pt minus 0.4pt,
    xkanjiskip: \ltjgetparameter{xkanjiskip} % xkanjiskip: 2.40555pt plus 1.0pt minus 1.0pt
```

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

4.3 Insertion setting of xkanjiskip

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

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

pよqいiう
The second argument \texttt{preonly} means that the insertion of \texttt{\textasciitilde kanjiskip} is allowed before this character, but not after. The other possible values are \texttt{postonly}, \texttt{allow}, and \texttt{inhibit}.

\texttt{jxspmode} 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}
\ltjsetparameter{alxspmode={`` founding, preonly}, jxspmode={`` founding, postonly})
\end{verbatim}

One can use also numbers to specify these two parameters (see Subsection 8.1).

If you want to enable/disable all insertions of \texttt{kanjiskip} and \texttt{\textasciitilde kanjiskip}, 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 \TeX, this is achieved by setting \texttt{\baselineskip} (or \texttt{\baselineshift}) to a non-zero length (the baseline of \texttt{\textasciitilde 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, \LaTeX-ja can independently set the shifting amount of the baseline of alphabetic fonts and that of Japanese fonts.


<table>
<thead>
<tr>
<th>Horizontal writing (yoko direction) etc.</th>
<th>Vertical writing (tate direction)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Alphabetic fonts</strong></td>
<td><strong>Japanese fonts</strong></td>
</tr>
<tr>
<td>\texttt{yalbaselineshift} parameter</td>
<td>\texttt{yjabaselineshift} parameter</td>
</tr>
<tr>
<td>\texttt{talbaselineshift} 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}
1 \vrule width 150pt height 0.2pt depth 0.2pt \hskip-120pt
2 \ltjsetparameter{yjabaselineshift=0pt, yalbaselineshift=0pt}abcあいう
3 \ltjsetparameter{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}
1 \vrule width 150pt height 4.417pt depth -4.217pt %
2 \kern-150pt
3 \large xyz漢字
4 \{\scriptsize
5 \ltjsetparameter{yjabaselineshift=-1.757pt, yalbaselineshift=-1.757pt}
6 \large xyz漢字
7 \}あいう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{\textasciitilde 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, \LaTeX-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 \texttt{\lpcode} and \texttt{\rpcode} are both non-zero.

This problem does not apply for \texttt{yjabaselineshift} nor \texttt{tjabaselineshift}, because a \texttt{\textasciitilde 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*),

\[ \text{jaspmode, axsmpmode, prebreakpenalty, postbreakpenalty and kcatcode} \]

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 hwid feature. However, the penalty inserted after it is 10 which is the postbreakpenalty of “ア”, not 20.

\begin{verbatim}
\setparameter{postbreakpenalty={ア, 10}}
\setparameter{postbreakpenalty={ア, 20}}
\newcommand\showpostpena[1]{\leavevmode\setbox0=\hbox{#1\hbox{}}\unhbox0\setbox0=\lastbox\the\lastpenalty}
\showpostpena{ア}, \showpostpena{ア}, \addjfontfeatures{CharacterWidth=Half}\showpostpena{ア}}
\end{verbatim}
5 \texttt{\catcode} in \LaTeX-ja

5.1 Preliminaries: \texttt{\catcode} in \TeX and \LaTeX

In \TeX and \LaTeX, the value of \texttt{\catcode} determines whether a Japanese character can be used in a control word. For the detail, see Table 4.

\texttt{\catcode} can be set by a row of JIS X 0208 in \TeX, and generally by a Unicode block\footnote{\LaTeX divides U+FF00--U+FFEF (Halfwidth and Fullwidth Forms) into three subblocks, and \texttt{\catcode} can be set by a subblock.} in \LaTeX. So characters which can be used in a control word slightly differ between \TeX and \LaTeX.

5.2 Case of \LaTeX-ja

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

- Distinction between \texttt{\char} or \texttt{\charji} 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{\catcode} to 11 (enabled) or 12 (disabled), as usual.
- Whether \texttt{\jcharwidowpenalty} can be inserted before the character is controlled by the lowermost bit of the \texttt{\catcode} parameter.
- Linebreak after a \texttt{\char} does not produce a space.

Default setting of \texttt{\catcode} of Unicode characters are located in

\texttt{plain \LaTeX} \texttt{luatex-unicode-letters.tex}, which is based on \texttt{unicode-letters.tex} (for Xe\TeX).

\texttt{Luatex} now included in \LaTeX kernel as \texttt{unicode-letters.def}.

However, the default setting of \texttt{\catcode} differs between Xe\TeX and \LaTeX, by the following reasons:

- (plain format) \texttt{luatex-unicode-letters.tex} is based on old \texttt{unicode-letters.tex}.
- The latter half of \texttt{unicode-letters.tex} and \texttt{unicode-letters.def} sets \texttt{\catcode} of several characters to 11, via setting \texttt{\xetexcharclass}. However, this latter half does not exist (plain case), or not executed (\LaTeX case) in \LaTeX-ja.

In other words,

\texttt{plain \LaTeX} Kanji nor kana characters cannot be used in a control word, in the default setting of plain \LaTeX.

\texttt{Luatex} in recent (2015-10-01 or later) \LaTeX, kanji and kana characters in a control word is supported (these \texttt{\catcode} are 11), but not fullwidth alphanumerics and several other characters.

This would be inconvenient for \TeX users to shifting to \LaTeX-ja, since several control words containing Kanji or other fullwidth characters, such as \\texttt{\xen} or \\texttt{\xenn} are used in \TeX. Hence, \LaTeX-ja have a counterpart of \texttt{unicode-letters.tex} for \LaTeX, \texttt{to match the \catcode setting with that of Xe\TeX}.
Table 4. $\texttt{kcatcode}$ in up$\TeX$

<table>
<thead>
<tr>
<th>$\texttt{kcatcode}$</th>
<th>meaning</th>
<th>control word</th>
<th>widow penalty</th>
<th>linebreak</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>non-cjk</td>
<td>(treated as usual $\TeX$)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>kanji</td>
<td>Y</td>
<td>Y</td>
<td>ignored</td>
</tr>
<tr>
<td>17</td>
<td>kana</td>
<td>Y</td>
<td>Y</td>
<td>ignored</td>
</tr>
<tr>
<td>18</td>
<td>other</td>
<td>N</td>
<td>N</td>
<td>ignored</td>
</tr>
<tr>
<td>19</td>
<td>hangul</td>
<td>Y</td>
<td>Y</td>
<td>space</td>
</tr>
</tbody>
</table>

Table 5. 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>p$\TeX$</th>
<th>up$\TeX$</th>
<th>Lua$\TeX$-ja</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>1</td>
<td>12</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>1</td>
<td>16</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>1</td>
<td>17</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>1</td>
<td>18</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>1</td>
<td>23</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>1</td>
<td>24</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>2</td>
<td>26</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>1</td>
<td>27</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>1</td>
<td>28</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>1</td>
<td>31</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>1</td>
<td>32</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>

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 p$\TeX$, in up$\TeX$, and in Lua$\TeX$-ja. Table 5 shows the difference. Except for four characters "・", "゛", "゜", "゠", Lua$\TeX$-ja admits more characters in a control word than up$\TeX$.

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

6 Directions

Lua$\TeX$ supports four $\Omega$-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.

Lua$\TeX$-ja supports four directions, as shown in Table 6. The second column (yoko direction) is just horizontal writing, and the third column (tate direction) is vertical writing. The fourth column (dtou direction) is actually a hidden feature of p$\TeX$. We implemented this for debugging purpose. The fifth column (utod direction) corresponds the "tate (math) direction" of p$\TeX$.

Directions can be changed by \texttt{\yoko}, \texttt{\tate}, \texttt{\dtou}, \texttt{\utod}, only when the current list is null. Also, the direction of a math formula is changed to \texttt{utod}, when the direction outside the math formula is \texttt{tate} (vertical writing).

6.1 Boxes in different direction

As in p$\TeX$, one can use boxes of different direction in one document. The below is an example.

Table 6. 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>Left</td>
<td>Right</td>
</tr>
<tr>
<td>Beginning of the line</td>
<td>Left</td>
<td>Top</td>
<td>Bottom</td>
<td>Top</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>

Example

銀は、Ag

(Notation used in Ω)

| TLT | RTR, RTT | LBL | RTR |

Table 7 shows how a box is arranged when the direction inside the box and that outside the box differ.

\begin{itemize}
\item \textbf{\\vbox and direction} In \LaTeX, \wd, \ht, \dp means the dimensions of a box register \textit{with respect to the current direction}. This means that the value of \wd0 etc. might differ when the current direction is different, even if \box0 stores the same box. However, this no longer applies in LuaTeX-ja.
\end{itemize}

\begin{verbatim}
  \setbox0=\hbox to 20pt{foo}
  \the\wd0, \hbox{\tate\vrule\the\wd0}
  \wd0=100pt
  \the\wd0, \hbox{\tate \the\wd0}
\end{verbatim}

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

\begin{verbatim}
\ltjgetwd(num), \ltjgetht(num), \ltjgetdp(num)
\end{verbatim}

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

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

The following is an example.

\begin{verbatim}
  \setindent0pt
  \setbox32767=\hbox{\yokoよこぐみ}
  \fboxsep=0mm\fbox{\copy32767}
  \hsize=20mm
  \yoko YOKO \the\ltjgetwd32767, \ytate TATE \the\ltjgetwd32767, \ydtou DTOU \the\ltjgetwd32767, \yutod UTOD \the\ltjgetwd32767
\end{verbatim}

表 7 shows how a box is arranged when the direction inside the box and that outside the box differ.

\begin{itemize}
\item \textbf{\vbox and direction} In \LaTeX, \wd, \ht, \dp means the dimensions of a box register \textit{with respect to the current direction}. This means that the value of \wd0 etc. might differ when the current direction is different, even if \box0 stores the same box. However, this no longer applies in LuaTeX-ja.
\end{itemize}

\begin{verbatim}
\setbox0=\hbox to 20pt{foo}
\the\wd0, \hbox{\tate\vrule\the\wd0}
\wd0=100pt
\the\wd0, \hbox{\tate \the\wd0}
\end{verbatim}

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

\begin{verbatim}
\ltjgetwd(num), \ltjgetht(num), \ltjgetdp(num)
\end{verbatim}

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

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

The following is an example.
Table 7. 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>( W_y = h_y + d_y )</td>
<td>( W_t = h_t + d_t )</td>
<td>( W_d = h_d + d_d )</td>
</tr>
<tr>
<td>( H_y = w_y )</td>
<td>( H_t = w_t )</td>
<td>( H_d = w_d )</td>
</tr>
<tr>
<td>( D_y = 0 ) pt</td>
<td>( D_t = w_t/2 )</td>
<td>( D_d = h_d )</td>
</tr>
</tbody>
</table>

\ltjsetwd(num)\langle\dimen\rangle, \ltjsetht(num)\langle\dimen\rangle, \ltjsetdp(num)\langle\dimen\rangle

These commands set the dimension of \box\langle num \rangle. One does not need to group the argument \langle num \rangle; four calls of \ltjsetwd below have the same meaning.

\ltjsetwd42 20pt, \ltjsetwd42=20pt, \ltjsetwd=42 20pt, \ltjsetwd=42=20pt

6.2 Getting current direction

The direction parameter returns the current direction, and the boxdir parameter (with the argument \langle num \rangle) returns the direction of a box register \box\langle num \rangle. The returned value of these parameters are a 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>1</td>
<td>11</td>
<td>0</td>
</tr>
</tbody>
</table>

\begin{verbatim}
\leavevmode\def\DIR{\ltjgetparameter{direction}}
\hbox{yoko \DIR}, \hbox{tate\DIR}, \hbox{dtou\DIR}, \hbox{utod\DIR}, \hbox{tate math: \DIR}$
\setbox2=\hbox{tate math: \DIR}$
\setbox2=\hbox[tate]\ltjgetparameter{boxdir}$
\end{verbatim}

6.3 Overridden box primitives

To cope with multiple directions, the following primitives are overridden by LuaTeX-ja, using \protected\def.
Table 8. Differences between horizontal JFMs shipped with LuaTeX-ja

<table>
<thead>
<tr>
<th>Blue: jfm-ujis.lua</th>
<th>Black: jfm-jis.lua</th>
<th>Red: jfm-min.lua</th>
</tr>
</thead>
<tbody>
<tr>
<td>ある日モモちゃんがお使いで迷子になって泣きました。</td>
<td>ちょっと！何</td>
<td>ちょっと！！何何</td>
</tr>
<tr>
<td>漢 つ</td>
<td>漢 つ</td>
<td>漢 つ</td>
</tr>
</tbody>
</table>

(Blue: jfm-ujis.lua, Black: jfm-jis.lua, Red: jfm-min.lua)

\unhbox\langle num\rangle, \unvbox\langle num\rangle, \unhcopy\langle num\rangle, \unvcopy\langle num\rangle

\vadjust{\langle material\rangle}

\insert\langle number\rangle{\langle material\rangle}

\lastbox

\raise\langle dimen\rangle\langle box\rangle, \lower\langle dimen\rangle\langle box\rangle etc., \vcenter

\vcenter

7 Font Metric and Japanese Font

7.1 \jfont

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

\jfont\tradgt={file:KozMinPr6N-Regular.otf:script=latn;\% +trad;-kern;jfm=ujis} at 14pt
\tradgt 当／体／医／区

Note that the defined control sequence (\tradgt in the example above) using \jfont is not a font_def
token, but a macro. Hence the input like \fontname\tradgt causes an error. We denote control sequences
which are defined in \jfont by \langle jfont_cs \rangle.

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

\jfm=\langle name\rangle

Specify the name of (horizontal) JFM. If specified JFM has not been loaded, LuaTeX-ja search and load
a file named jfm-\langle name\rangle.lua.

The following JFMs are shipped with LuaTeX-ja:

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

\jfm-jis.lua  A counterpart for jis.tfm, "JIS font metric" which is widely used in pTeX. A ma-
jor difference between jfm-ujis.lua and this jfm-jis.lua is that most characters under
jfm-ujis.lua are square-shaped, while that under jfm-jis.lua are horizontal rectangles.
1 \ltjsetparameter{differentjfm=both}
2 \jfont\F=file:KozMinPr6N-Regular.otf:jfm=ujis
3 \jfont\G=file:KozGoPr6N-Medium.otf:jfm=ujis
4 \jfont\H=file:KozGoPr6N-Medium.otf:jfm=ujis;jfmvar=hoge
5 \F)(\G【】(%
6 \H『』(%
7 ほげ, \G「ほげ」(ほげ)\par
8 ほげ, \H「ほげ」(ほげ) % pTeX-like
9 \ltjsetparameter{differentjfm=paverage}
10

Figure 1. Example of jfmvar key

ダイナミックダイクマ|ダイナミックダイクマ|
ダイナミックダイクマ|ダイナミックダイクマ|
ダイナミックダイクマ|ダイナミックダイクマ|
ダイナミックダイクマ|ダイナミックダイクマ|

11 \newcommand\test\vrule ダイナミックダイクマ\vrule\}
12 \jfont\KMFW = KozMinPr6N-Regular:jfm=prop; Kern at 17pt
13 \jfont\KMFK = KozMinPr6N-Regular:jfm=prop at 17pt % kern is activated
14 \jfont\KMPW = KozMinPr6N-Regular:jfm=prop;script=dflt;+pwid; Kern at 17pt
15 \jfont\KMPK = KozMinPr6N-Regular:jfm=prop;script=dflt;+pwid;+kern at 17pt
16 \begin{multicols}{2}
17 \ltjsetparameter{kanjiskip=0pt}
18 \KMFW\test \KMFK\test \KMPW\test \KMPK\test\}
19 \ltjsetparameter{kanjiskip=3pt}
20 \KMFW\test \KMFK\test \KMPW\test \KMPK\test\}
21 \end{multicols}
22

Figure 2. Kerning information and kanjiskip

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

The difference among these three JFMs is shown in Table 8.

\begin{verbatim}
jfmvar=(\texttt{string})
Sometimes there is a need that ....
\end{verbatim}
\end{verbatim}

Using kerning information in a font Some fonts have information for inter-glyph spacing. This version of Lua\TeX{}-ja treats kerning spaces like an italic correction; any glue and/or kern from the JFM and a kerning space can coexist. See Figure 2 for detail.

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

\begin{verbatim}
\texttt{extend} and \texttt{slant} The following setting can be specified as OpenType font features:
\begin{verbatim}
\texttt{extend=(extend)} expand the font horizontally by \texttt{(extend)}.
\texttt{slant=(slant)} slant the font.
\end{verbatim}
\end{verbatim}

24
Note that LuaTeX-ja doesn’t adjust JFMs by these extend and slant settings; you have to write new JFMs on purpose. For example, the following example uses the standard JFM jfm-ujis.lua, hence letter-spacing and the width of italic correction are not correct:

\begin{verbatim}
\jfont\E=KozMinPr6N-Regular:extend=1.5;jfm=ujis;-kern
\Eあいうえお
\jfont\S=KozMinPr6N-Regular:slant=1;jfm=ujis;-kern
\SあいうABC
\end{verbatim}

### 7.2 \jfont

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

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

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

\begin{verbatim}
\jfont\testJ={psft:Ryumin-Light:cid=Adobe-Japan2;jfm=jis}
\end{verbatim}

produces the following error:

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

See the luatexja package documentation for explanation.

Type H <return> for immediate help.

<to be read again>
7.4 Structure of a JFM file

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

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

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

- `dir=(direction)` (required)
  
  The direction of JFM. '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 `\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 ∈ ω` 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 `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 `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 10.
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 3. The position of the real glyph (horizontal Japanese fonts)

Figure 4. The position of the real glyph (vertical 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 Figures 3 and 4.

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={i={kern}, j'={kern}, [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] suitable:
priority=⟨priority⟩ An integer in [−2, 2] (treated as 0 if omitted), and this is used only in line adjustment with priority by luatexja-adjust (see Subsection 11.3). 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 kanjiskip times the value of kanjiskip_stretch key (1 in this case).
- Between an ideographic closing brackets (the ideographic comma “，” is included) and an ordinal letter, we have the same glue. Again, this glue also can be stretched as much as the stretch part of kanjiskip times the value of kanjiskip_stretch key (1 in this case).

Hence we have the following result:

```
\leavevmode
\ltjsetparameter{kanjiskip=0pt plus 3\zw}         あ 「い」 う, え お
\vrule\hbox to 15\zw{あ「い」う, えお}\vrule
```

end_stretch=(kern), end_shrink=(kern)

■Character to character classes  We explain how the character class of a character is determined, using jfm-test.lua which contains the following:

```
[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 = { '。', '，' },
   align = 'left', left = 0.0, down = 0.0,
   width = 0.5, height = 0.88, depth = 0.12, italic=0.0,
},
```

Now consider the following input/output:

```
\jfont\a=file:KozMinPr6N-Regular.otf;jfm=test;+hwid
\setbox0\hbox{あ 漢}
\the\wd0
```

Now we look why the above source outputs 15 pt.

1. The character “ヒ” is converted to its half width form “ヒ” by hwid 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 \hbox equals to 15 pt.

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

However, if the class determined by the glyph after application of features is zero, LuaTeX-ja adopts the class determined by the glyph before application of features. The following input is an example.

```
\jfont\a=file:KozMinPr6N-Regular.otf;jfm=test;+vert
\a 漢。\inhibitglue 漢
```

28
Here, the character class of the ideographic full stop “。” (U+3002) is determined as follows:

1. As the case of “ヒ”, the ideographic full stop “。” is converted to its vertical form “︒” (U+FE12) by \textfeature.

2. The character class of “︒”, according to the JFM is zero.

3. However, LuaTeX-ja remembers that this “︒” is obtained from “。” by font features. The character class of “。” is non-zero value, namely, 2000.

4. Hence the ideographic full stop “。” in above belongs the character class 2000.

### Imaginary characters

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

- `{boxbdd}`
  - The beginning/ending of a hbox, and the beginning of a noindented (i.e., began by \textfamily{\noindent}) paragraph.

- `{parbdd}`
  - The beginning of an (indented) paragraph.

- `{jcharbdd}`
  - A boundary between \textfamily{JAchar} and anything else (such as \textfamily{ALchar}, kern, glue, ...).

- `{−1}`
  - The left/right boundary of an inline math formula.

### Porting JFM from \textfamily{pTeX}

See Japanese version of this manual.

#### 7.5 Math font family

\textfamily{pTeX} handles fonts in math formulas by 16 font families\textsuperscript{7}, and each family has three fonts: \textfamily{textfont}, \textfamily{scriptfont} and \textfamily{scriptscriptfont}.

LuaTeX-ja’s handling of Japanese fonts in math formulas is similar; Table 11 shows counterparts to \textfamily{pTeX}’s primitives for math font families. There is no relation between the value of \textfamily{fam} and that of \textfamily{jfam}; with appropriate settings, you can set both \textfamily{fam} and \textfamily{jfam} to the same value. Here \textfamily{(jfont_cs)} in the argument of \textfamily{jatextfont} etc. is a control sequence which is defined by \textfamily{jfont}, i.e., a horizontal Japanese font.

#### 7.6 Callbacks

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

\texttt{luatexja.load_jfm} callback

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

\textsuperscript{7} Omega, Aleph, LuaTeX and \texttt{\varepsilon-(u)pTeX} can handles 256 families, but an external package is needed to support this in plain \textfamily{pTeX} and \textfamily{\EPIC}.
function (jfm_info, string jfm_name)
    return table.new_jfm_info
end

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

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

**luatexja.define_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 `luatexja.define_font` callback is called just when new Japanese font is loaded.

function (jfont_info, number font_number)
    return table.new_jfont_info
end

jfont_info has the following fields, which may not overwritten by a user:

- **size** The font size specified at \jfont in scaled points (1 sp = 2^{-16} pt).
- **zw, zh, kanjiskip, xkanjiskip** These are scaled value of those specified by the JFM, by the font size.
- **jfm** The internal number of the JFM.
- **var** The value of jfmvar key, which is specified at \jfont. The default value is the empty string.
- **chars** The mapping table from character codes to its character classes.
  
  The specification `[i].chars={⟨character⟩, ...} in the JFM will be stored in this field as `chars={[(character)]=i, ...}`.

- **char_type**
  
  For `i ∈ ω`, `char_type[i]` is information of characters whose class is `i`, and has the following fields:
  
  - **width, height, depth, italic, down, left** are just scaled value of those specified by the JFM, by the font size.
  - **align** is a number which is determined from align field in the JFM:

  \[
  \begin{align*}
  0 & \text{ 'left' (default)} \\
  0.5 & \text{ 'middle'} \\
  1 & \text{ 'right'}
  \end{align*}
  \]

  - For `j ∈ ω`, `[j] stores a kern or a glue which will be inserted between character class `i` and class `j`.

  If a kern will be inserted, the value of this field is `[j]={false, ⟨kern_node⟩, ⟨ratio⟩}`, where ⟨kern_node⟩ is a node. If a glue will be inserted, we have `[j]={false, ⟨spec_node⟩, ⟨ratio⟩, ⟨icflag⟩}`, where ⟨spec_node⟩ is also a node, and ⟨icflag⟩ = from_jfm + ⟨priority⟩.

The returned table `new_jfont_info` also should include these fields, but you are free to add more fields (to use them in the `luatexja.find_char_class` callback). The `font_number` is a font number.

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

**luatexja.find_char_class callback**

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

function (char_class, jfont_info, chr_code)
    if char_class==0 then return char_class
    else...

8This version of LuaTeX-ja uses "direct access model" for accessing nodes, if possible.
return (<number> 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 OpenType fonts);
• The value of the real font: (height) = 28y, (depth) = 5y (the standard values of Japanese TrueType fonts).

Then, the position of glyphs is shifted up by

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

8 Parameters

8.1 \ltjsetparameter
As described before, \ltjsetparameter and \ltjgetparameter 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 pHijX (e.g., \prebreakpenalty = 10000) is the position of hpack_filter callback in the source of LuaTeX, see Section 12.

\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. 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. \[\text{[\textit{cas}]\] indicates the counterpart in pHijX, 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)* \[\text{\jcharwidowpenalty}\]
Penalty value for suppressing orphans. This penalty is inserted just after the last JAChar which is not regarded as a (Japanese) punctuation mark.

kcatcode={(chr_code),(natural number)}*
An additional attributes which each character whose character code is ⟨chr_code⟩ has. At the present version, the lowermost bit of ⟨natural number⟩ indicates whether the character is considered as a punctuation mark (see the description of \jcharwidowpenalty above).
\texttt{prebreakpenalty}=$\langle(\text{chr\_code}),\langle\text{penalty}\rangle)\rangle$ * [$\backslash$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 $"\rangle", so \LaTeX-ja sets
\begin{verbatim}
\ltjsetparameter{prebreakpenalty={"\rangle",10000}}
\end{verbatim}
by default.

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

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

\texttt{postbreakpenalty}=$\langle(\text{chr\_code}),\langle\text{penalty}\rangle)\rangle$ * [$\backslash$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}=$\langle(\text{jfam}),\langle\text{jfont\_cs}\rangle)\rangle$ * [$\backslash$textfont in \TeX]
\texttt{jascriptfont}=$\langle(\text{jfam}),\langle\text{jfont\_cs}\rangle)\rangle$ * [$\backslash$scriptfont in \TeX]
\texttt{jascriptscriptfont}=$\langle(\text{jfam}),\langle\text{jfont\_cs}\rangle)\rangle$ * [$\backslash$scriptscriptfont in \TeX]
\texttt{yjabaselineshift}=$\langle\text{dimen}\rangle$
\texttt{yalbaselineshift}=$\langle\text{dimen}\rangle$ [$\text{ybaselineshift}$]
\texttt{tjabaselineshift}=$\langle\text{dimen}\rangle$
\texttt{talbaselineshift}=$\langle\text{dimen}\rangle$ [$\text{tbaselineshift}$]
\texttt{jaxspmode}=$\langle(\text{chr\_code}),\langle\text{mode}\rangle)\rangle$

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

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

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

\texttt{alxspmode}=$\langle(\text{chr\_code}),\langle\text{mode}\rangle)\rangle$ * [$\backslash$xspcode]

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

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

Note that parameters \texttt{jaxspmode} and \texttt{alxspmode} share a common table, hence these two parameters are synonyms of each other.

\texttt{autospan}=(\text{bool}) * [$\backslash$autospan]
\texttt{autoxspan}=(\text{bool}) * [$\backslash$autoxspan]
kanjiskip = (skip) ∗ [\kanjiskip]

The default glue which inserted between two JAChars. 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 7.4).

xkanjiskip = (skip) ∗ [\xkanjiskip]

The default glue which inserted between a JAChar and an ALChar. Changing current font does not alter this parameter, as \TeX.

As 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 7.4).

differentjfm = (mode)†

Specify how glues/kerns between two JAChars whose JFM (or size) are different. The allowed arguments are the followings:

average, both, large, small, pleft, pright, paverage

The default value is paverage, ...

jacharrange = (ranges)

kansujichar = \{\langle digit\rangle, \langle chr_code\rangle\} ∗ [\kansujichar]

direction = (dir) (always local)

Assigning to this parameter has the same effect as \texttt{\yoko} (if \langle \texttt{dir} \rangle = 4), \texttt{\tate} (if \langle \texttt{dir} \rangle = 3), \texttt{\dtou} (if \langle \texttt{dir} \rangle = 1) or \texttt{\utod} (if \langle \texttt{dir} \rangle = 11). If the argument \langle \texttt{dir} \rangle is not one of 4, 3, 1 nor 11, the behavior of this assignment is undefined.

8.2 \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 “ ” (U+0020) 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.
  
  jcharwidowpenalty, yjabaselineshift, yalbaselineshift, autospacing, autoxspacing, kanjiskip, xkanjiskip, differentjfm, direction
  
  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.

  kcatcode, prebreakpenalty, postbreakpenalty, jaxspmode, alxspmode

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

- \texttt{\ltjgetparameter{jacharrange}\{(range)\}} returns 0 if “characters which belong to the character range (range) are JAChar”, 1 if “... are ALChar”. Although there is no character range −1, specifying −1 to (range) 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 an integer which represents the direction of the surrounding vertical list. As direction, the return value 1 means down-to-up 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{boxdim}{\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\).

\begin{verbatim}
\jatextfont, jascriptfont, jascriptscriptfont, jacharrange
\end{verbatim}

• \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 J\text{Achar} or not by the following:

\begin{verbatim}
\ltjgetparameter{jacharrange}{\ltjgetparameter{chartorange}{\langle chr\_code\rangle}}
\%
\end{verbatim}

% 0 if J\text{Achar}, 1 if AL\text{char}

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

\begin{verbatim}
\ifdim\ltjgetparameter{kanjiskip}>\z@ \ldots \fi
\ifdim\ltjgetparameter{xkanjiskip}>\z@ \ldots \fi
\end{verbatim}

The correct way is using a temporary register.

\begin{verbatim}
\@tempskipa=\ltjgetparameter{kanjiskip} \ifdim\@tempskipa>\z@ \ldots \fi
\@tempskipa=\ltjgetparameter{xkanjiskip}\ifdim\@tempskipa>\z@ \ldots \fi
\end{verbatim}

\section{9 Other Commands for plain \LaTeX and \LaTeXe}

\subsection{9.1 Commands for compatibility with p\LaTeX}

The following commands are implemented for compatibility with p\LaTeX. 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, \jis, \jis, \kansuji
\end{verbatim}

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

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

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

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

\addcontentsline{toc}{section}{9 Other Commands for plain \LaTeX and \LaTeXe}
9.2 \inhibitglue

\inhibitglue suppresses the insertion of JAgue. 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=file:KozMinPr6N-Regular.otf:jfm=test \g
\fbox{\hbox{あウあ\inhibitglue ウ}}
\inhibitglue\par\noindent あ1
\inhibitglue\par\noindent あ2
\inhibitglue\par\noindent あ3
\par\hrule \noindent office
\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.
- 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.

9.3 \ltjdeclarealtfont

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

\begin{verbatim}
\ltjdeclarealtfont\langle base_font_cs\rangle\langle alt_font_cs\rangle\{(range)\}
\end{verbatim}

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 LuaTeX-ja, then

\begin{verbatim}
\ltjdeclarealtfont\hoge\piyo\{“3000-“30FF, {-1}-{-1}\}
\end{verbatim}

does

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

10 Commands for \LaTeX 2ε

10.1 Patch for NFSS2

Japanese patch for NFSS2 in LuaTeX-ja is based on p1fonts.dtx which plays the same role in p1\LaTeX 2ε. We will describe commands which are not described in Subsection 3.1.

additional dimensions

Like p1\LaTeX 2ε, LuaTeX-ja defines the following dimensions for information of current Japanese font:

\begin{verbatim}
\cht (height), \cdp (depth), \cHT (sum of former two),
\cwd (width), \cvs (lineskip), \chs (equals to \cwd)
\end{verbatim}

and its \normalsize version:
\Cht (height), \Cdp (depth), \Cwd (width), \
\Cvs (equals to \baselineskip), \Chs (equals to \cws).

Note that \cws and \cvt may differ from \zw and \zh respectively. On the one hand the former dimensions are determined from the character "あ", 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 Lua\TeX-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.

\DeclareMathAlphabet{(unified-cmd)}{(al-cmd)}{(ja-cmd)}
\DeclareRelationFont{(ja-encoding)}{(ja-family)}{(ja-series)}{(ja-shape)}
\DeclareRelationFont{(al-encoding)}{(al-family)}{(al-series)}{(al-shape)}

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.

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

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

\adjustbaseline
In p\TeX\ e, \adjustbaseline sets \baselineshift to match the vertical center of “M” and that of “あ” in vertical typesetting:

\baselineshift = \frac{(h_M + d_M) - (h_a + d_a)}{2} + d_a - d_M,

where \h_\text{a} and \d_\text{a} denote the height of “a” and the depth, respectively. In Lua\TeX-ja, this \adjustbaseline does same task, namely setting the \tbaselineshift parameter.

\fontfamily{(family)}
As in p\TeX\ e, this command changes current font family (alphabetic, Japanese, or both) to \textit{family}. See Subsection 10.2 for detail.

\DeclareAlternateKanjiFont{(base-encoding)}{(base-family)}{(base-series)}{(base-shape)}
\DeclareAlternateKanjiFont{(al-encoding)}{(al-family)}{(al-series)}{(al-shape)}{(range)}

As \ltjdeclarealtfont (Subsection 9.3), characters in \textit{range} of the Japanese font (we say the \textit{base font}) which specified by first 4 arguments are typeset by the Japanese font which specified by fifth to eighth arguments (we say the \textit{alternate font}). An example is shown in Figure 5.

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

• ...

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

日本国民族は、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 5. An example of \DeclareAlternateKanjiFont

10.2 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\LaTeX, but we re-implemented in Lua. We use an auxiliary list \textit{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 \textit{N}_J, this means that \{family\} is not a Japanese font family.

3. If the \texttt{luatexja-fontspec} package is loaded, we stop here, and \{family\} is not recognized as a Japanese font family.

   If the \texttt{luatexja-fontspec} package is not loaded, now Lua\TeX\-ja looks whether there exists a Japanese font encoding \texttt{enc} such that a font definition named \texttt{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 \textit{N}_J.

■ Recognition as alphabetic font family  Next, whether alphabetic font family will be changed is determined in following order. We use auxiliary lists \textit{F}_A and \textit{N}_A.

1. If the family \{family\} has been listed in a list \textit{F}_A, \{family\} is recognized as an alphabetic font family.

2. If the family \{family\} has been listed in a list \textit{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 \textit{F}_A.

4. Now Lua\TeX\-ja looks whether there exists an alphabetic font encoding \texttt{enc} such that a font definition named \texttt{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 \textit{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\ e, but more complicated (clause 3.). This is because p\TeX\ e is a format however Lua\TeX-ja is not, hence Lua\TeX-ja does not know calls of `\DeclareFontFamily` before itself is loaded.

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

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

\subsection*{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:

\begin{itemize}
  \item \texttt{CID=\langle name\rangle, JFM=\langle name\rangle, JFM-var=\langle name\rangle}
      These 3 keys correspond to cid, jfm and jfmvar keys for \texttt{\jfont} and \texttt{\tfont} respectively. See Subsections 7.1 and 7.3 for details of cid, jfm and jfmvar keys.
      The CID key is effective only when with \texttt{NoEmbed} described below. The same JFM cannot be used in both horizontal Japanese fonts and vertical Japanese fonts, hence the JFM key will be actually used in YokoFeatures and TateFeatures keys.
  \item \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 7.3.
  \item \texttt{Kanjiskip=\langle bool\rangle}
  \item \texttt{TateFeatures=\langle\{features\}\rangle, TateFont=\langle font\rangle}
      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 6.
  \item \texttt{YokoFeatures=\langle\{features\}\rangle}
      The YokoFeatures key specifies font features which are only turned on in horizontal writing. A demonstrarion is shown in Figure 6.
\end{itemize}
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 7. An example of AltFont

AltFont
As \ltjdeclarealtfont (Subsection 9.3) and \DeclareAlternateKanjiFont (Subsection 10.1), 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:

\begin{quote}
  \[
  \text{AltFont} = \{
  \quad \ldots
  \quad \{ \text{Font} = \text{font name}, \ (\text{features}) \},
  \quad \{ \text{Range} = \langle \text{range} \rangle, \ (\text{features}) \},
  \quad \{ \text{Range} = \langle \text{range} \rangle, \text{Font} = \text{font name} \},
  \quad \ldots
  \}
  \end{quote}

Each sublist should have the Range key (sublist which does not contain Range key is simply ignored). A demonstration is shown in Figure 7.

■Remark on AltFont, YokoFeatures, TateFeatures keys In AltFont, YokoFeatures, TateFeatures keys, one cannot specify per-shape settings such as BoldFeatures. For example,

\begin{quote}
  \text{AltFont} = \{ \text{Font} = \text{HogeraMin-Light}, \text{BoldFont} = \text{HogeraMin-Bold}, \text{Range} = "3000-"30FF, \text{BoldFeatures} = \{ \text{Color} = 007F00 \} \}
  \end{quote}

\text{does not work. Instead, one have to write}

\begin{quote}
  \text{UprightFeatures} = \{ \text{AltFont} = \{ \text{Font} = \text{HogeraMin-Light}, \text{Range} = "3000-"30FF, \} \}, \text{BoldFeatures} = \{ \text{AltFont} = \{ \text{Font} = \text{HogeraMin-Bold}, \text{Range} = "3000-"30FF, \text{Color} = 007F00 \} \} \}
  \end{quote}

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.

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

11.2 luatexja-otf

This optional package supports typesetting characters in Adobe-Japan1 character collection (or other CID character collection, if the font is supported). The package luatexja-otf offers the following 2 low-level commands:
\CID{(number)}
Typeset a character whose CID number is \textit{(number)}.

\UTF{(hex\_number)}
Typeset a character whose character code is \textit{(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 japanese-otf package.}. Hence one can use macros which are defined in \texttt{ajmacros.sty}, such as \texttt{\aj半角}.

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

\begin{itemize}
\item Always treated as JAchars.
\item Processing codes for supporting OpenType features (e.g., glyph replacement and kerning) by the \texttt{luaotfload} package is not performed to these characters.
\end{itemize}

\textbf{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 'AJ1-xxx', which stands for the character whose CID number in Adobe-Japan1 is xxx.

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

\textbf{IVS support} Recent fonts support Ideographic Variation Selector (IVS). It seems that \texttt{luaotfload} and \texttt{fontspec} packages do not support IVS, so we implemented IVS support in \texttt{luatexja-otf}. \textit{IVS support is experimental; if you want to enable this, load \texttt{luatexja-otf} and execute the following:}

\begin{verbatim}
\directlua{luatexja.otf.enable_ivs()}
\end{verbatim}

After executing the command above, you can use IVS like the following:

\begin{verbatim}
\Large
\jfontspec{KozMinPr6N-Regular}
奈良県葛城市と, 東京都葛飾区.
こんにちは, 渡邉.
\end{verbatim}

Specifying glyph variants by IVS precedes glyph replacement by font features. For example, only “葛” in “葛西” is changed by font features \texttt{jp78} or \texttt{jp90}, which does not followed by any variation selector.

\begin{verbatim}
\def\TEST#1{%
{\jfontspec[#1]{KozMinPr6N-Regular}}% 指定なし: 葛, 葛, 葛, 葛
指定なし: 葛城市, 葛飾区, 葛西\}
\texttt{jp78}: \TEST{CJKShape=JIS1990}
\texttt{jp90}: \TEST{CJKShape=JIS1990}
\end{verbatim}

\section{11.3 luatexja-adjust}
(see Japanese version of this manual)
The above principle is often called the **inclusion-exclusion principle**.

---

### 11.4 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{妙典}{みようでん}という駅……
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}{典}{でん}駅は……
2. 東西線の\ruby{妙}{みよう}{典}{でん}駅は……
```

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

```
1. \ruby{妙}{みよう}{典}{でん}
2. \ruby{葛}{かさい}{西}{せい}
3. \ruby{神楽}{かぐら}{坂}{ざか}
```

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

```
1. \vbox{\hsize=6\zw\noindent}
2. \hbox to 2.5\zw{\ruby{京急}{けいき}{蒲}{かま}{田}{た}}
3. \hbox to 2.5\zw{\ruby{京急}{けいき}{蒲}{かま}{田}{た}}
4. \hbox to 3\zw{\ruby{京急}{けいき}{蒲}{かま}{田}{た}}
```

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{\hsize=8\zw\noindent}
2. \null\kern3\zw{……を 承る}
3. \null\kern3\zw{……を 承る}
4. \null\kern3\zw{……を 承る}
```

---

### 11.5 lltxjext.sty

p\TeX supplies additional macros for vertical writing in the plex package. The lltxjext package which we want to describe here is the Lua\TeX-ja counterpart of the plex package.
tabular, array, minipage environments
These environments are extended by \texttt{<dir>}, which specifies the direction, as follows:

\begin{tabular}\texttt{<dir>}[\texttt{pos}]{\texttt{table spec}} \ldots \end{tabular}
\begin{array}\texttt{<dir>}[\texttt{pos}]{\texttt{table spec}} \ldots \end{array}
\begin{minipage}\texttt{<dir>}[\texttt{pos}]{\texttt{width}} \ldots \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.

\begin{itemize}
\item \texttt{y} yoko direction (horizontal writing)
\item \texttt{t} tate direction (vertical writing)
\item \texttt{z} utod direction if direction outside the env. is tate.
\item \texttt{d} dtou direction
\item \texttt{u} utod direction
\end{itemize}

\texttt{\parbox}\texttt{<dir>}[\texttt{pos}]{\texttt{width}}{\texttt{contents}}
\parbox\texttt{command is also extended by \texttt{<dir>}}.

\texttt{\pbox}\texttt{<dir>}{\texttt{width}}{\texttt{pos}}{\texttt{contents}}
This command typeset \texttt{contents} in LR-mode, in \texttt{dir} direction. If \texttt{width} is positive, the width of the box becomes this \texttt{width}. In this case, \texttt{contents} will be aligned ...

picture environment

\texttt{\rensuj}\texttt{[\texttt{pos}]}{\texttt{contents}}, \texttt{\rensujkip}

\texttt{\Kanji[\texttt{counter_name}]}

\texttt{\kasen}\texttt{[\texttt{contents}]}, \texttt{\bou}\texttt{[\texttt{contents}]}, \texttt{\boutenchar}

参照番号

Part III
Implementations

12 Storing Parameters

12.1 Used dimensions, attributes and whatsit nodes
Here the following is the list of dimensions and attributes which are used in Lua\TeX-ja.

\texttt{\jQ} (dimension) \texttt{\jQ} is equal to 1 \texttt{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.

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

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

\texttt{\ltj\@zh} (dimension) A temporal register for the “full-height” (usually the sum of height of imaginary body and its depth) of current Japanese font. The command \texttt{\zh} sets this register to the correct value, and “return” this register itself.
\jtj@curjfnt (attribute) The font index of current Japanese font for horizontal direction.
\jtj@curtfnt (attribute) The font index of current Japanese font for vertical direction.
\jtj@charclass (attribute) The character class of a JAchar. This attribute is only set on a glyph_node which contains a JAchar.
\jtj@yablshift (attribute) The amount of shifting the baseline of alphabetic fonts in scaled point (\(2^{-16}\) pt).
\jtj@ykblshift (attribute) The amount of shifting the baseline of Japanese fonts in scaled point (\(2^{-16}\) pt).
\jtj@tablshift (attribute)
\jtj@tkblshift (attribute)
\jtj@autospc (attribute) Whether the auto insertion of kanjiskip is allowed at the node.
\jtj@autoxspc (attribute) Whether the auto insertion of xkanjiskip is allowed at the node.
\jtj@icflag (attribute) An attribute for distinguishing "kinds" of a node. One of the following value is assigned to this attribute:
italic (1) Kerns from italic correction (\(\sqrt{\ )\)}, or from kerning information of a Japanese font. These kerns are "ignored" in the insertion process of JAgue, unlike explicit \kern.
packed (2)
kinsoku (3) Penalties inserted for the word-wrapping process (kinsoku shori) of Japanese characters.
(from_jfm – 2)–(from_jfm + 2) (4–8) Glues/kerns from JFM.
kanji.skip (9), kanji.skip.jfm (10) Glues from kanjiskip.
xkanji.skip (11), xkanji.skip.jfm (12) Glues from xkanjiskip.
processed (13) Nodes which is already processed by ....
ie.processed (14) Glues from an italic correction, but already processed in the insertion process of JAgule.
boxed (15) Glues/kerns that inserted just the beginning or the ending of an hbox or a paragraph.
\jtj@kcat\(i\) (attribute) Where \(i\) is a natural number which is less than 7. These 7 attributes store bit vectors indicating which character block is regarded as a block of JAchars.
\jtj@dir (attribute) dir_node.auto (128)
dir_node.manual (256)

Furthermore, Lua\TeX\-ja uses several user-defined whatsit nodes for internal processing. All those nodes except direction whatsits store a natural number (hence its type is 100). direction whatsits store a node list, hence its type is 110. Their user_id (used for distinguish user-defined whatsits) are allocated by luatexbase.newuserwhatsitid.
inhibitglue Nodes for indicating that \inhibitglue is specified. The value field of these nodes doesn’t matter.
stack.marker Nodes for Lua\TeX\-ja’s stack system (see the next subsection). The value field of these nodes is current group level.
char by cid Nodes for JAchar which the callback process of 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 the callback process of luaotfload. Nodes of this type is used in \CID, \UTF and IVS support.
**replace**

Similar to char by cid what sits above. These nodes are for ALchar which the callback process of luaotfload won’t be applied.

**begin par**

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

**direction**

These what sits will be removed during the process of inserting JAgIues.

### 12.2 Stack system of LuaTEx-ja

**Background**

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

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

As described in Subsection 8.1, 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 “5 pt” cannot be known from any callbacks. In the tex/packaging.\u, 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;
1233     d = box_max_depth;
1234     unsave();
1235     save_ptr -= 4;
1236     if (cur_list.mode_field == -hmode) {
1237         cur_box = filtered_hpack(cur_list.head_field,
1238             cur_list.tail_field, saved_value(1),
1239             saved_level(1), grp, saved_level(2));
1240         subtype(cur_box) = HLIST_SUBTYPE_HBOX;
1241     }  
```

Notice that unsave() is executed before filtered_hpack(), where hpack filter callback is executed here. So “5 pt” in the above source is orphaned at unsave(), and hence it can’t be accessed from hpack_filter callback.

**Implementation**

The code of stack system is based on that in a post of Dev-luatex mailing list\textsuperscript{10}.

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

To resolve the problem mentioned in above paragraph “Background”, LuaTEx-ja uses another trick. When the stack level is about to be increased, a what sit 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.

\textsuperscript{10}[Dev-luatex] tex.currentgrouplevel, a post at 2008/8/19 by Jonathan Sauer.
• If there is a stack flag node whose value is $t + 1$, then an assignment was occurred just inside the hbox group. Hence values of parameters at the end of the hbox are stored in the stack level $s + 1$.

• If there are stack flag nodes but all of their values are more than $t + 1$, then an assignment was occurred in the box, but it is done in more internal group. Hence values of parameters at the end of the hbox are stored in the stack level $s$.

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

12.3 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:
\begin{verbatim}
luatexja.stack.set_stack_table(index, <any> data)
\end{verbatim}

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

Stored data can be obtained as the return value of
\begin{verbatim}
luatexja.stack.get_stack_table(index, <any> default, <number> level)
\end{verbatim}

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

12.4 Extending Parameters

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

**Setting parameters** Figure 9 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 \directlua{luatexja.isglobal}. Its meaning is the following.
\begin{equation}
\text{\directlua{luatexja.isglobal}} = \begin{cases} 
\text{'global'} & \text{global} \\
\text{'} & \text{local}
\end{cases} \tag{1}
\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.

```lua
function luatexja.unary_pars.hoge (t)
    return 42
end
```

Here the argument of \luatexja.unary_pars.hoge is the stack level of LuaTeX-ja’s stack system (see Subsection 12.2).

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,

```lua
function luatexja.binary_pars.fuga (c, t)
    return tostring(c) .. ', ' .. tostring(42)
end
```

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”.

### 13 Linebreak after a Japanese Character

#### 13.1 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 LuaTeX-ja due to the specification of callbacks in LuaTeX. To clarify the difference between \pTeX and LuaTeX, We briefly describe the handling of a line break in \pTeX, in this subsection.

\pTeX’s input processor can be described in terms of a finite state automaton, as that of \TeX in Section 2.5 of [1]. The internal states are as follows:

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

The first three states—\( N \), \( S \), and \( M \)—are as same as \TeX’s input processor. State \( K \) is similar to state \( M \), and is entered after Japanese characters. The diagram of state transitions are indicated in Figure 10. Note that \pTeX doesn’t leave state \( K \) after “beginning/ending of a group” characters.

#### 13.2 Behavior in Lua\TeX-ja

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

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

Considering these situations, handling of an end-of-line in Lua\TeX-ja are as follows:
• 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 10. State transitions of p\TeX’s input processor

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

1. The category code of \texttt{\endlinechar}\(^{11}\) is 5 (end-of-line).
2. The category code of \text{U+FFFFF} itself is 14 (comment).
3. The input line matches the following "regular expression":

\[
(\text{any char})^* (\text{JAchar})\left(\text{\{catcode = 1\}} \cup \text{\{catcode = 2\}}\right)^*
\]

**Remark** The following example shows the major difference from the behavior of p\TeX.

\begin{verbatim}
\fontspec[Ligatures=TeX]{Linux Libertine O}
\ltjsetparameter{autoxspacing=false}
\ltjsetparameter{jacharrange={-6}}xyzあいう
\ltjsetparameter{jacharrange={+6}}いu
\end{verbatim}

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 "y" in the output, but there is a space before "u". 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 \text{U+FFFFF} 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.

\(^{11}\)Usually, it is \texttt{(return)} (whose character code is 13).
14 Patch for the listings Package

It is well-known that the listings package outputs weird results for Japanese input. The listings package makes most of letters active and assigns output command for each letter ([2]). But Japanese characters are not included in these activated letters. For \LaTeX series, there is no method to make Japanese characters active; a patch jlisting.sty ([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 \U{FFFF}) before each letter above \U{0080}. This method can omit the process to make all Japanese characters active (most of the activated characters are not used in many cases).

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

14.1 Notes and additional keys

\textbf{Escaping to \LaTeX} We used the \texttt{process\_input\_buffer} callback to output JAchars. But it has a drawback; any commands whose name contains a JAchar cannot be used in any “escape to \LaTeX”.

Consider the following input:

\begin{lstlisting}[,escapechar=\#]
\#\ほげ \びよ#
\end{lstlisting}

The line 2 is transformed by the callback to

\#\U{FFFF} \ほ \びよ#

before the line is actually processed. In the escape (between the character “\#”), the category code of \U{FFFF} is set to 9 (\texttt{ignore}). Hence the control symbol \texttt{\#} will be executed, instead of \texttt{\#\ほげ}.

\textbf{Variation selectors} \texttt{lttjp-listings} add 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 \texttt{lttjp-listings} is loaded at \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=\ltjlistingsvstcmd]
葛市,葛飾区,葛西
\end{lstlisting}

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

\begin{lstlisting}
\def\IVSA#1#2#3#4#5{%
\textcolor{blue}{\raisebox{3.5pt}{\tt%
\fboxsep=0.5pt\fbox{\tiny \oalign{0#1#2\crcr#3#4#5\crcr}}}}%
,}
{\catcode\%=11
\gdef\IVSB#1{\expandafter\IVSA\directlua{
local cat_str = luatexbase.catcodetables['string']
\tex.sprint(cat_str,string.format('%X',0xE00EF+#1))}}}
\lstset{vscmd=\IVSB}
\end{lstlisting}

The default output command of variation selectors is stored in \texttt{ltjlistingsvstcmd}.
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}
1 : H :
2 : H H H H :
\end{verbatim}

lltj-listing adds the 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}
1 : H :
2 : H H H H :
\end{verbatim}

14.2 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 \texttt{\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 \texttt{lltj-listings} introduces a new flag \texttt{\lst@ifkanji}, which indicates whether the previous character is a Japanese character or not. For illustration, we introduce following classes of characters:

\begin{verbatim}
\lst@ifletter T F T F T
\lst@ifkanji F F T T F

Meaning char in an identifier other alphabet most of Japanese char opening brackets closing brackets
\end{verbatim}

Note that \textit{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:

\begin{verbatim}
<table>
<thead>
<tr>
<th>Next</th>
<th>Letter</th>
<th>Other</th>
<th>Kanji</th>
<th>Open</th>
<th>Close</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{\lst@ifletter}</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>\texttt{\lst@ifkanji}</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Prev</th>
<th>Letter</th>
<th>Other</th>
<th>Kanji</th>
<th>Open</th>
<th>Close</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{\lst@ifletter}</td>
<td>collects</td>
<td>outputs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\texttt{\lst@ifkanji}</td>
<td>outputs</td>
<td>collects</td>
<td></td>
<td></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 \texttt{\lstlisting} environment or other environments/commands.

## 15 Cache Management of \texttt{LuaTeX-ja}

\texttt{LuaTeX-ja} creates some cache files to reduce the loading time. In a similar way to the \texttt{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"), caches of the binary, precompiled form are supported.
  - We cannot share same binary cache for \texttt{LuaTeX} and \texttt{LuaJITTeX}. Hence we distinguish them by their extension, ".luc" for \texttt{LuaTeX} and ".lub" for \texttt{LuaJITTeX}.
  - In loading a cache, the binary cache precedes the text form.
  - When \texttt{LuaTeX-ja} updates a cache \texttt{hoge.lua}, its binary version is also updated.

### 15.1 Use of cache

\texttt{LuaTeX-ja} uses the following cache:

\texttt{ltj-cid-auto-adobe-japan1.lua}

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 \texttt{LuaTeX-ja}.

Similar caches are created as Table 12, if you specified \texttt{cid} key in \texttt{\jfont} to use other CID-keyed non-embedded fonts for Chinese or Korean, as in Page 25.

\texttt{extra_***.lua}

This file stores the table which stores the following.

- unicode variants in a font "***"
- vertical width of glyphs, if it is not equal to the sum of the height of ascender and the depth of descender
- vertical variants

The following is the structure of the that table.

\texttt{return { }
\texttt{
[10955]={  -- U+2ACB "Subset Of Above Not Equal To"
[65024]=983879, -- <2ACB FE00>
["vwidth"]=0.98, -- vertical width
},
[37001]={  -- U+9089 "邉"
[0]=37001, -- <9089 E0100>
}}
Table 12. cid key and corresponding files

<table>
<thead>
<tr>
<th>cid key</th>
<th>name of the cache</th>
<th>used CMaps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adobe-Japan1-*</td>
<td>ltj-cid-auto-adobe-japan1.lua</td>
<td>UniJIS2004-UTF32-*</td>
</tr>
<tr>
<td>Adobe-Korea1-*</td>
<td>ltj-cid-auto-adobe-korea1.lua</td>
<td>UniKS-UTF32-*</td>
</tr>
<tr>
<td>Adobe-GB1-*</td>
<td>ltj-cid-auto-adobe-gb1.lua</td>
<td>UniGB-UTF32-*</td>
</tr>
<tr>
<td>Adobe-CNS1-*</td>
<td>ltj-cid-auto-adobe-cns1.lua</td>
<td>UniCNS-UTF32-*</td>
</tr>
</tbody>
</table>

991049, -- <9089 E0101>
...  
{"vert"}=995025, -- vertical variant
},
...

{"chksum"}="FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF", -- checksum of the fontfile
{"version"}=2, -- version of the cache
}

ltj-jisx0208.{luc|lub}
The binary version of 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 pifX.

15.2 Internal
Cache management system of LuaTeX-ja is stored in luatexja.base (ltj-base.lua). There are three public functions for cache management in luatexja.base, where ⟨filename⟩ stands for the file name without suffix:

save_cache(⟨filename⟩, ⟨data⟩)
Save a non-nil table ⟨data⟩ into a cache ⟨filename⟩. Both the text form ⟨filename⟩.lua and its binary version are created or updated.

save_cache_luc(⟨filename⟩, ⟨data⟩ [, ⟨serialized_data⟩])
Same as save_cache, except that only the binary cache is updated. The third argument ⟨serialized_data⟩ is not usually given. But if this is given, it is treated as a string representation of ⟨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{lub}. 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 text form ⟨filename⟩.lua. Hence, the return value of load_cache is non-nil, if and only if the updated cache is found.
References


