For submission to The Asian Journal of TEX
Draft of December 4, 2011

Development of LuaTEX-ja package

Hironori Kitagawa 北川 弘典
LuaTEX-ja project team h_kitagawa2001@yahoo.co.jp

KEYWORDS TEX, pTEX, LuaTEX, LuaTEX-ja, Japanese

ABSTRACT LuaTEX-ja package is a macro package for typesetting Japanese documents under LuaTEX. The package has more flexibility of typesetting than pTEX, which is widely used Japanese extension of TEX, and has corrected some unwanted features of pTEX. In this paper, we describe specifications, the current status and some internal processing methods of LuaTEX-ja.

1 Introduction

1.1 History

To typeset Japanese documents with TEX, ASCII pTEX [2] has been widely used in Japan. There are other methods—for example, using Omega and OTP [4], or with the CJK package—to do so, however, these alternative methods did not become majority. The author thinks that this is because pTEX enables us to produce high-quality documents (e.g., supporting vertical typesetting), and the appearance of pTEX is earlier than that of alternatives described above.

However, pTEX has been left behind from the extensions of TEX such as ε-TEX and pdfTEX, and the diffusion of UTF-8 encoding. In recent years, the situation has become better, by development of `ptexenc` [19] by Nobuyuki Tsuchimura (土村展之), ε-pTEX [7] by the author, and upTEX [18] by Takuji Tanaka (田中琢爾). However, continuing this approach, namely, to develop an engine extension localized for Japanese, is not wise. This approach needs lots of work for *each* engine. In addition, if we use LuaTEX, the necessity of an engine extension is getting smaller because LuaTEX has an ability to hook TEX's internal process by using Lua callbacks.

Before our LuaTEX-ja project, there were several experimental attempts to typeset Japanese documents with LuaTEX. Here we cite three examples:

– `luaums.sty` [8] developed by the author. This experimental package is for creating a certain Japanese-based presentation with LuaTEX.

– the *luajalayout* package [12], formerly known as the *jafontspec* package, by Kazuki Maeda (前田一貴). This package is based on LATEX 2$_\varepsilon$ and *fontspec* package.

– the *luajp-test* package [11], a test package made by Atsuhito Kohda (香田温人), based on articles on the web page [5].

However, these packages are based on LaTeX 2$_\varepsilon$, and do not have much ability to control the typesetting rule. And it is inefficient that more than one person separately develop similar packages. Development of the LuaTeX-ja package is started initially by the author and Kazuki Maeda, because of these situations.

1.2 Development policy of LuaTeX-ja

The first aim of LuaTeX-ja project was to implement features (from the 'primitive' level) of pTeX as macros under LuaTeX, therefore LuaTeX-ja is much affected by pTeX. However, as development proceeded, some technical/conceptual difficulties arose. Hence we changed the aim of the project as follows:

- *LuaTeX-ja offers at least the same flexibility of typesetting that pTeX has.*
  We are not satisfied with the ability of producing (PDF) outputs conformed to JIS X 4051 [6], the Japanese Industrial Standard for typesetting, or to a technical note [20] by W3C; if one wants to produce very incoherent outputs for some reason, it should be possible. In this point, previous attempts of Japanese typesetting with LuaTeX which we cited in the previous subsection are inadequate.
  pTeX has some flexibility of typesetting, by changing internal parameters such as `\kanjiskip` or `\prebreakpenalty`, and by using custom JFM (Japanese TFM). Therefore we decided to include these functionality to LuaTeX-ja.
- *LuaTeX-ja isn't mere re-implementation or porting of pTeX; some (technically and/or conceptually) inconvenient features of pTeX are modified.*
  We describe this point in more detail at the next section.

1.3 Overview of the processes

We describe an outline of LuaTeX-ja's process in order.

- In the `process_input_buffer` callback: treatment of line-break after a Japanese character (in Subsection 2.2).
- In the `hyphenate` callback: font replacement.
  LuaTeX-ja looks into for each *glyph_node* p in the horizontal list. If the character represented by p is considered as a Japanese character, the font used at p is replaced by the value of `\ltj@curjfnt`, an attribute for 'the current Japanese font' at p.
  Furthermore, the subtype of p is subtracted by 1 to suppress hyphenation around p by LuaTeX, because later processes of LuaTeX-ja take care of all things about Japanese characters.
- In `pre_linebreak_filter` and `hpack_filter` callbacks:
  1. LuaTeX-ja has its own stack system, and the current horizontal list is traversed in this stage to determine what the level of LuaTeX-ja's internal stack at the end of the list is. We will discuss it in Subsection 5.2.
  2. In this stage, LuaTeX-ja inserts glues/kerns for Japanese typesetting in the list. This is the core routine of LuaTeX-ja. We will discuss it in Subsections 2.4 and 2.5 .

3. To make a match between a metric and a real font, sometimes adjustment of the position of (Japanese) glyphs are performed. We will discuss it in Subsection 5.3.

– In the `mlist_to_hlist` callback: treatment of Japanese characters in math formulas. This stage is similar to adjustment of the position of glyphs (see above), so we omit to describe this stage from this paper.

In this paper, a *alphabetic character* means a non-Japanese character. Similarly, we use the word an *alphabetic font* as the counterpart of a Japanese font.

1.4 Contents of this paper

Here we describe the contents of the rest of this paper briefly. In Section 2, we describe major differences between pTEX and LuaTEX-ja. The next section, Section 3, is concentrated on a problem how we distinguish between Japanese characters and alphabetic characters. In Section 4, we show current development status of the package. Finally, in Section 5, we describe some internal routines of LuaTEX-ja.

1.5 General information of the project

This LuaTEX-ja project is hosted by SourceForge.jp. The official wiki is located on `http://sourceforge.jp/projects/luatex-ja/wiki/`. There is no stable version on October 22, 2011, however a set of developer sources can be obtained from the git repository. Members of the project team are as follows (in random order): Hironori Kitagawa, Kazuki Maeda, Takayuki Yato, Yusuke Kuroki, Noriyuki Abe, Munehiro Yamamoto, Tomoaki Honda, and Shuzaburo Saito.

2 Major differences with pTEX

In this section, we explain several major differences between pTEX and our LuaTEX-ja. For general information of Japanese typesetting and the overview of pTEX, please see Okumura [14].

2.1 Names of control sequences

Because pTEX is an engine modification of Knuth's original TEX82 engine, some of the additional primitives take a form that is very difficult to be simulated by a macro. For example, an additional primitive `\prebreakpenalty`⟨*char_code*⟩`[=]`⟨*penalty*⟩ in pTEX sets the amount of penalty inserted before a character whose code is ⟨*char_code*⟩ to ⟨*penalty*⟩, and this form `\prebreakpenalty`⟨*char_code*⟩ can be also used for retrieving the value.

Moreover, there are some internal parameters of pTEX which values of them at the end of a horizontal box or that of a paragraph are valid in whole box or paragraph. However, the implementation of these parameters in LuaTEX-ja is not so easy; we will discuss it in Subsection 5.2.

From the two problems discussed above, the assignment and retrieval of most parameters in LuaTEX-ja are summarized into the following three control sequences:

xあy　　xあ y

```
1 \font\x=IPAMincho \x
2 \ltjsetparameter{jacharrange={-6}}x あ
3 y\qquad x あ
4 y
```

FIGURE 1. A notable sample showing the treatment of a line-break after a Japanese character.

- `\ltjsetparameter{`⟨*name*⟩`=`⟨*value*⟩`,...}`: for local assignment.
- `\ltjglobalsetparameter`: for global assignment. Note that these two control sequences obey the value of `\globaldefs` primitive.
- `\ltjgetparameter{`⟨*name*⟩`}[{`⟨*optional argument*⟩`}]`: for retrieval. The returned value is always a string.

2.2 Line-break after a Japanese character

Japanese texts can break lines almost everywhere, in contrast with alphabetic texts can break lines only between words (or use hyphenation). Hence, pTEX's input processor is modified so that a line-break after a Japanese character doesn't emit a space. However, there is no way to customize the input processor of LuaTEX, other than to hack its CWEB-source. All a macro package can do is to modify an input line before when LuaTEX begin to process it, inside the `process_input_buffer` callback.

Hence, in LuaTEX-ja, a comment letter (we reserve U+FFFFF for this purpose) will be appended to an input line, if this line ends with a Japanese character.[1] One might jump to a conclusion that the treatment of a line-break by pTEX and that of LuaTEX-ja are totally same, however they are different in the respect that LuaTEX-ja's judgment whether a comment letter will be appended the line is done *before* the line is actually processed by LuaTEX.

Figure 1 shows an example of this situation; the command at the first line marks most of Japanese characters as 'non-Japanese characters'. In other words, from that command onward, the letter 'あ' will be treated as an alphabetic character by LuaTEX-ja. Then, it is natural to have a space between 'あ' and 'y' in the output (as the second example in the figure), where the actual output of the first example in the figure does not so. This is because 'あ' at the input line 2 is considered a Japanese character by LuaTEX-ja, when LuaTEX-ja does the decision whether U+FFFFF will be added to the input line 2.

2.3 Separation between 'real' fonts and metrics

Traditionally, most Japanese fonts used in typesetting are not proportional, that is, most glyphs have same size (in most cases, square-shaped). Hence, it is not rare that the contents of different JFMs are essentially same, and only differ in their names. For example, `min10.tfm` and `goth10.tfm`, which are JFMs shipped with pTEX for seriffed

1. Strictly speaking, it also requires that the catcode of the end-line character is 5 (*end-of-line*). This condition is useful under the verbatim environment.

```
\jfont\foo=file:ipam.ttf:jfm=ujis;script=latn;-kern;+jp04 at 12pt
\jfont\bar=psft:Ryumin-Light:jfm=ujis at 10pt
```

FIGURE 2. Typical declarations of Japanese fonts.

mincho family and sans-seriffed *gothic* family, differ their `FAMILY` and `FACE` only. Moreover, `jis.tfm` and `jisg.tfm`, which is included in the *jis* font metric, which is used in *jsclasses* [13] by Haruhiko Okumura (奥村晴彦), are totally same as binary files. Considering this situation, we decided to separate 'real' fonts and metrics used for them in LuaTEX-ja. Typical declarations of Japanese fonts in the style of plain TEX are shown in Figure 2. We would like to add several remarks:

- A control sequence `\jfont` must be used for Japanese fonts, instead of `\font`.
- LuaTEX-ja automatically loads the *luaotfload* package, so `file:` and `name:` prefixes, and various font features can be used as the first line in Figure 2.
- The `jfm` key specifies the metric for the font. In Figure 2, `\foo` and `\bar` will use a metric stored in a Lua script named `jfm-ujis.lua`. This metric is the standard metric in LuaTEX-ja, and is based on JFMs used in the *otf* package [16] (hence almost all characters are square-shaped).
- The `psft:` prefix can be used to specify name-only, non-embedded fonts. When one displays a pdf with these fonts, actual fonts which will be used for them depend on a pdf reader.

The specification of a metric for LuaTEX-ja is similar to that of a JFM (see [14]); characters are grouped into several classes, the size information of characters are specified for each class, and glue/kern insertions are specified for each pair of classes. Although the author have not tried, it may be possible to develop a program that 'converts' a JFM to a metric for LuaTEX-ja. LuaTEX-ja offers three metrics by default; `jfm-ujis.lua`, `jfm-jis.lua` based on the *jis* font metric, and `jfm-min.lua` based on old `min10.tfm`.

Note that `-kern` in features is important, because kerning information from a real font itself will clash with glue/kern information from the metric.

2.4 Insertion of glues/kerns for Japanese typesetting: timing

As described in [9], LuaTEX's kerning and ligaturing processes are totally different from those of TEX82. TEX82's process is done just when a (sequence of) character is appended to the current list. Thus we can interrupt this process by writing as `f{}irm`. However, LuaTEX's process is *node-based*, that is, the process will be done when a horizontal box or a paragraph is ended, so `f{}irm` and `firm` yield same outputs under LuaTEX.

The situation for Japanese characters is more complicated. Glues (and kerns) which are needed for Japanese typesetting are divided into the following three categories:

- Glue (or kern) from the metric of Japanese fonts (*JFM glue*, for short).
- Default glue between a Japanese character and an alphabetic character (we say *xkanjiskip*, for short), usually 1/4 of full-width (*shibuaki*) with some stretch and shrink for justifying each line.

TABLE 1. Examples of differences between pTEX and LuaTEX-ja.

	(1)	(2)	(3)	(4)
Input	`あ】{}【〕\/〔`	`い』\/a`	`う）\hbox{}（`	`え］\special{}［`
pTEX	あ】 【〕 〔	い』a	う） （	え］ ［
LuaTEX-ja	あ】【〕〔	い』a	う） （	え］［

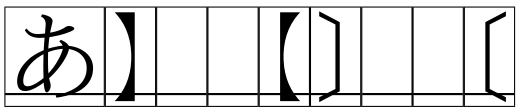

FIGURE 3. Detail of the output of pTEX in the input (1) in Table 1.

– Default glue between two consecutive Japanese characters (*kanjiskip*, for short). The main reason of this glue is to enable breaking lines almost everywhere in Japanese texts. In most cases, its natural width is zero, and some stretch/shrink for justifying each line.

In pTEX, these three kinds of glues are treated differently. A JFM glue is inserted when a (sequence of) Japanese character is appended to the current list, same as the case of alphabetic characters in TEX82. This means that one can interrupt the insertion process by saying `{}`. A *xkanjiskip* is inserted just before 'hpack' or line-breaking of a paragraph; this timing is somewhat similar to that of LuaTEX's kerning process. Finally, A *kanjiskip* is not appeared as a node anywhere; only appears implicitly in calculation of the width of a horizontal box, that of breaking lines, and the actual output process to a DVI file. These specifications have made pTEX's behavior very hard to understand.

LuaTEX-ja inserts glues in all three categories simultaneously inside `hpack_filter` and `pre_linebreak_filter` callbacks. The reasons of this specification are to behave like alphabetic characters in LuaTEX (as described in the first paragraph in this subsection), and to clarify the specification for LuaTEX-ja's process.

2.5 Insertion of glues/kerns for Japanese typesetting: specification

Now we will take a look at the insertion process itself through four points.

Ignored nodes As noted in the previous subsection, the insertion process in pTEX can be interrupted by saying `{}` or anything else.[2] This leads the second row in Table 1, or Figure 3. Here 'the process is interrupted' means that pTEX does not think the letter '】' is followed by '【', hence two half-width glues are inserted between '】' and '【', where the left one is from '】' and the right one is from '【'.

On the other hand, in LuaTEX-ja, the process is done inside `hpack_filter` and `pre_linebreak_filter` callbacks. Hence, *anything that does not make any node will be ignored* in LuaTEX-ja, as shown in (1) in Table 1. LuaTEX-ja also ignores any nodes which does not make any contribution to current horizontal list—*ins_node*, *adjust_node*, *mark_node*, *whatsit_node* and *penalty_node*—, as shown in (4).

2. This is why some tricks like ちょ{}っと for `min10.tfm` and other 'old' JFMs work.

By the way, around a *glyph_node* p there may be some nodes attached to p. These are an accent and kerns for moving it to the right place, and a kern from the italic correction[3] for p. It is natural that these attachments should be ignored inside the process. Hence LuaTEX-ja takes this approach, as the latest version of pTEX (version p3.2). This explains (2) in the Table 1.

Summarizing the above, one should put an empty horizontal box `\hbox{}` to where he/she wants to interrupt the insertion process in LuaTEX-ja as (3) in the Table 1.

Fonts with the same metric Recall that LuaTEX-ja separates 'real' fonts and metrics, as in Subsection 2.3. Consider the following input, where all Japanese fonts use same metric (in LuaTEX-ja), and `\gt` selects *gothic* family for the current Japanese font family:

```
明朝）\gt （ゴシック
```

If the above input is processed by pTEX, because the insertion process is interrupt by `\gt`, the result looks like

明朝） （ゴシック

However this seems to be unnatural, since two Japanese fonts in the output use the same metric, i.e., the same typesetting rule. Hence, we decided that Japanese fonts with the same metric are treated as one font in the insertion process of LuaTEX-ja. Thus, the output from the above input in LuaTEX-ja looks like:

明朝）（ゴシック

One might have the situation that this default behavior is not suitable. LuaTEX-ja offers a way to handle this situation, but we leave it to the manual [10].

Fonts with different metrics The case where two adjacent Japanese characters use different metrics and/or different size is similar. Consider the following input where the *mincho* family and the *gothic* family use different metrics:

```
漢）\gt （漢）\large （大
```

As the previous paragraph, this input yields the following, by pTEX:

漢） （漢） （大

We had thought that amounts of spaces between parentheses in the above output are too much. Hence we have changed the default behavior of LuaTEX-ja, so that the amount of a glue between two Japanese characters with different metrics is the *average* of a glue from the left character and that from the right character. For example, Figure 4 shows the output from the above input. The width of glue indicated '(1)' is $(a/2 + a/2)/2 = 0.5a$, and the width of glue indicated '(2)' is $(a/2 + 1.2a/2)/2 = 0.55a$. This default behavior can be changed by differentjfm parameter of LuaTEX-ja.

kanjiskip *and* xkanjiskip In pTEX, the value of *xkanjiskip* is controlled by a skip named `\xkanjiskip`. A well-known defect of this implementation is that the value of

3. TEX82 (and LuaTEX) does not distinguish between explicit kern and a kern for italic correction. To distinguish them, an additional subtype for a kern is introduced in pTEX. On the other hand, LuaTEX-ja uses an additional attribute and redefines `\/` to set this attribute.

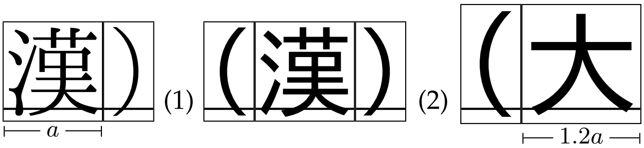

FIGURE 4. Fonts with different metrics.

xkanjiskip is not connected with the size of the current Japanese font. It seems that `EXTRASPACE`, `EXTRASTRETCH`, `EXTRASHRINK` parameters in a JFM are reserved for specifying the default value of *xkanjiskip* in a unit of the design size, but pTEX did not use these parameters, actually.

Considering this situation of pTEX, LuaTEX-ja can use the value of *xkanjiskip* that specified in a metric. If the value of *xkanjiskip* on user side (this is the value of xkanjiskip parameter of `\ltjsetparameter`) is `\maxdimen`, then LuaTEX-ja uses the specification from the current used metric as the actual value of *xkanjiskip*. This description also applies for *kanjiskip*.

3 Distinction of characters

Since LuaTEX can handle Unicode characters natively, it is a major problem that how we distinguish Japanese characters and alphabetic characters. For example, the multiplication sign (U+00D7) exists both in ISO-8859-1 (hence in Latin-1 Supplement in Unicode) and in the basic Japanese character set JIS X 0208. It is not desirable that this character is always treated as an alphabetic character, because this symbol is often used in the sense of 'negative' in Japan.

3.1 Character ranges

Before we describe the approach taken in LuaTEX-ja, we review the approach taken by upTEX. upTEX extends the `\kcatcode` primitive in pTEX, to use this primitive for setting how a character is treated among alphabetic characters (15), *kanji* (16), *kana* (17), *Hangul* (17), or *other CJK characters* (18). The assignment to `\kcatcode` can be done by a Unicode block.[4]

LuaTEX-ja adopted a different approach. There are many Unicode blocks in Basic Multilingual Plane which are not included in Japanese fonts, therefore it is inconvenient if we process by a Unicode block. Furthermore, JIS X 0208 are not just union of Unicode blocks; for example, the intersection of JIS X 0208 and Latin-1 Supplement is shown in Table 2. Considering these two points, to customize the range of Japanese characters in LuaTEX-ja, one has to define ranges of character codes in his/her source in advance.

We note that LuaTEX-ja offers two additional control sequences, `\ltjjachar` and `\ltjalchar`. They are similar to `\char` primitive, however `\ltjjachar` always yields a Japanese character, provided that the argument is more than or equal to 128, and `\ltjalchar` always yields an alphabetic character, regardless of the argument.

4. There are some exceptions. For example, U+FF00–FFEF (Halfwidth and Fullwidth Forms) are divided into three blocks in recent upTEX.

TABLE 2. Intersection of JIS X 0208 and Latin-1 Supplement.

§ (U+00A7),	¨ (U+00A8),	° (U+00B0),	± (U+00B1),
´ (U+00B4),	¶ (U+00B6),	× (U+00D7),	÷ (U+00F7)

TABLE 3. Predefined ranges in LuaTEX-ja.

1	(Additional) Latin characters which are not belonged in the range 8.
2	Greek and Cyrillic letters.
3	Punctuations and miscellaneous symbols.
4	Unicode blocks which does not intersect with Adobe-Japan1-6.
5	Surrogates and supplementary private use Areas.
6	Characters used in Japanese typesetting.
7	Characters possibly used in CJK typesetting, but not in Japanese.
8	Characters in Table 2.

3.2 Default setting of ranges

Patches for plain TEX and LATEX 2_ε of LuaTEX-ja predefine eight character ranges, as shown in Table 3. Almost of these ranges are just the union of Unicode blocks, and determined from the Adobe-Japan1-6 character collection [1], and JIS X 0208. Among these eight ranges, the ranges 2, 3, 6, 7, and 8 are considered ranges of Japanese characters, and others are considered ranges of alphabetic characters.[5] We remark on ranges 2 and 8:

The range 2 JIS X 0208 includes Greek letters and Cyrillic letters, however, these letters cannot be used for typesetting Greek or Russian, of course. Hence it is reasonable that Greek letters and Cyrillic consist another character range.

The range 8 If one wants to use 8-bit TFMs, such as T1 or TS1 encodings, he should mark this range 8 as a range of alphabetic characters by

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

This is because some 8-bit TFMs have a glyph in this range; for example, the character 'Œ' is located at `"D7` in the T1 encoding.

3.3 Control sequences producing Unicode characters

The *fontspec* package[6] offers various control sequences that produce Unicode characters. However, these control sequences as it stands cannot work correctly with the default range setting of LuaTEX-ja. For example, `\textquotedblleft` is just an abbreviation of `\char"201C\relax`, and the character U+201C (LEFT DOUBLE QUOTATION MARK) is treated as an Japanese character, because it belongs to the range 3. This problem is resolved by using `\ltjalchar` instead of the `\char` primitive. It is included

5. Note that ranges 3 and 8 are considered ranges of alphabetic characters in this paper.
6. Preciously saying, it is the *xunicode* package, originally a package for XƎTEXand automatically loaded by the *fontspec* package.

×, ×, ×, ×, ×

```
×, \char`×,   % depend on range setting
\ltjalchar`×, % alphabetic char
\ltjjachar`×, % Japanese char
\texttimes    % alph. char (by fontspec)
```

FIGURE 5. Control sequences producing a Unicode character.

in an optional package named `luatexja-fontspec.sty`. Figure 5 shows several ways to typeset a character, both as a Japanese character and as an alphabetic characters.

The situation looks similar in math formulas, but in fact it differs. Each control sequence that represents an ordinary symbol defined by the *unicode-math* package is just synonym of a character. For example, the meaning of `\otimes` is just the character U+2297 (CIRCLED TIMES), which is included in the range 3. However, it is difficult to define a control sequence like `\ltjalUmathchar` as a counterpart of `\Umathchar`, since an input like '`\sum^\ltjalUmathchar ...`' has to be permitted.

However, we couldn't develop a satisfactory solution to this problem in time for this paper, due to a lack of time. We are just testing a solution below:

- LuaTeX-ja has a list of character codes which will be always treated as alphabetic characters in math mode. Considering 8-bit TFMs for math symbols, this list includes natural numbers between `"80` and `"FF` by default.
- Redefine internal commands defined in the *unicode-math* package so that codes of characters which are mentioned in the *unicode-math* package will be included in the list.

We would like to extend treatments described in this subsection to 8-bit font encodings, but we leave it to further development too.

4 Current status of development

At the moment, LuaTeX-ja can be used under plain TeX, and under LaTeX 2_ε. Generally speaking, one only has to read `luatexja.sty`, by `\input` command or `\usepackage` (in LaTeX 2_ε), if you merely want to typeset Japanese characters. We look more details by parts.

4.1 'Engine extension'

The lowest part of LuaTeX-ja corresponds to the pTeX extension as *an engine extension of TeX*. We, the project members, think that this part is almost done. There is one more feature of LuaTeX-ja which we are going to explain:

Shifting baseline In order to make a match between Japanese fonts and alphabetic fonts, sometimes shifting the baseline of alphabetic characters may be needed. pTeX has a dimension `\ybaselineshift`, which corresponds to the amount of shifting down the baseline of alphabetic characters. This is useful for Japanese-based documents, but not for documents mainly in languages with alphabetic characters.

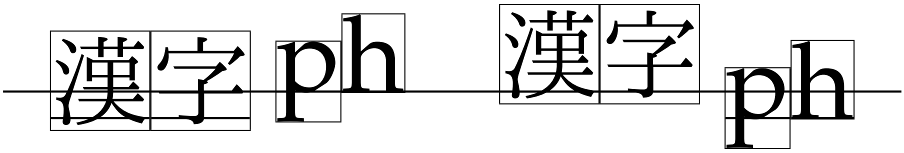

FIGURE 6. First example of shifting baseline.

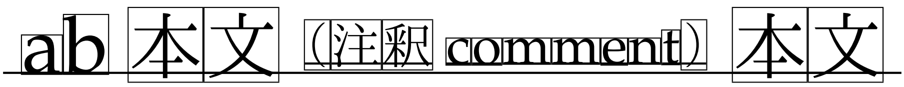

FIGURE 7. Second example of shifting baseline.

Hence, LuaTeX-ja extends pTeX's `\ybaselineshift` to Japanese characters. Namely, LuaTeX-ja offers two parameters, yjabaselineshift and yalbaselineshift, for the amount of shifting the baseline of Japanese characters and that of alphabetic characters, respectively.

An example output is shown in Figure 6. The left half is the output when yjabaselineshift is positive, hence the baseline of Japanese characters is shifted down. On the other hand, the right half is the output when yalbaselineshift is positive, hence the baseline of alphabetic characters is shifted down. Figure 7 shows an interesting use of these parameters.

Note that LuaTeX-ja doesn't support vertical typesetting, *tategaki*, for now.

4.2 Patches for plain TeX and LaTeX 2ε

pTeX has a patch for plain TeX, namely `ptex.tex`, that for LaTeX 2ε macro (this patch and LaTeX 2ε consist *pLaTeX 2ε*), and `kinsoku.tex` which includes the default setting of *kinsoku shori*, the Japanese hyphenation. We ported them to LuaTeX-ja, except the codes related to vertical typesetting, because LuaTeX-ja doesn't support vertical typesetting yet. We remark one point related to the porting:

Behavior of `\fontfamily` The control sequence `\fontfamily` in pLaTeX 2ε changes the current alphabetic font family and/or the current Japanese font family, depending the argument. More concretely, `\fontfamily{`⟨*arg*⟩`}` changes the current alphabetic font family to ⟨*arg*⟩, if and only if one of the following conditions are satisfied:

- An alphabetic font family named ⟨*arg*⟩ in *some* alphabetic encoding is already defined in the document.
- There exists an alphabetic encoding ⟨*enc*⟩ already defined in the document such that a font definition file ⟨*enc*⟩⟨*arg*⟩`.fd` (all lowercase) exists.

The same criterion is used for changing Japanese font family.

To work this behavior well, it is required that a list of all (alphabetic) encodings defined already in the document. However, since LuaTeX-ja is loaded as a package, LuaTEX-ja cannot have this list. Hence LuaTEX-ja adopted a different approach, namely `\fontfamily{`⟨*arg*⟩`}` changes the current alphabetic font family to ⟨*arg*⟩, if and only if:

- An alphabetic font family named ⟨*arg*⟩ in the current alphabetic encoding ⟨*enc*⟩ is already defined in the document.
- A font definition file ⟨*enc*⟩⟨*arg*⟩`.fd` (all lowercase) exists.

4.3 Classes for Japanese documents

To produce 'high-quality' Japanese documents, we need not only that Japanese characters are correctly placed, but also class files for Japanese documents. Two major families of classes are widely used in Japan: *jclasses* which is distributed with the official pLATEX 2$_\varepsilon$ macros, and *jsclasses*. At the present, LuaTEX-ja simply contains their counterparts: *ltjclasses* and *ltjsclasses*. However, the policy on classes is not determined now, and we hope to have another family of classes which are useful for commercial printing. In the author's opinion, *ltjclasses* is better to stay as an example of porting of class files for pTEX to LuaTEX-ja.

4.4 Patches for packages

Apart from patches for the LATEX 2$_\varepsilon$ kernel and classes for Japanese documents, we need to make patches for several packages. At the present, we considered the following packages, and made patches or porting for the former two packages.

The fontspec *package* The *fontspec* package is built on NFSS2, hence control sequences offered by the *fontspec* package, such as `\setmainfont`, are only effective for alphabetic fonts if LuaTEX-ja is loaded. `luatexja-fontspec.sty` (not automatically loaded) offers these counterparts for Japanese fonts, with additional 'j' in the name of control sequences, such as `\setmainjfont`. As described in Subsection 3.3, it also includes a patch for control sequences producing Unicode characters.

The otf *package* This package is widely used in pTEX for typesetting characters which is not in JIS X 0208, and for using more than one weight in *mincho* and *gothic* font families. Therefore LuaTEX-ja supports features in the *otf* package, by loading `luatexja-otf.sty` manually. Note that characters by `\UTF{}` and `\CID{}` are not appended to the current list as a *glyph_node*, to avoid from callbacks by the *luaotfload* package. We have another remark; `\CID` does not work with TrueType fonts, since `\CID` uses the conversion table between CID and the glyph order of the current Japanese font.

The listings *package* It is known for users of pTEX that there is a patch `jlisting.sty` for the *listings* package, to use Japanese characters in the `lstlisting` environment. Generally speaking, it also can be used in LuaTEX-ja. However, it seems to be that a Japanese character after a space does not receive any process of the *listings* package; this is inconvenient when we use the *showexpl* package.

There is another way to use characters whose code are above 256 with the *listings* package (described in [3]). However, this method is not suitable for Japanese, since the number of Japanese characters is very large. We hope that the *listings* package will be able to handle all characters above 256 without any patch, in the future.

5 Implementation

5.1 Handling of Japanese fonts

In pTeX, there are three slots for maintaining current fonts, namely `\font` for alphabetic fonts, `\jfont` for Japanese fonts (in horizontal direction) and `\tfont` for Japanese fonts (in vertical direction). With these slots, we can manage the current font for alphabetic characters and that for Japanese characters separately in pTeX. However, LuaTeX has only one slot for maintaining the current font, as TeX82. This situation leads a problem: how can we maintain the 'current Japanese font'?

There are three approaches for this problem. One approach is to make a mapping table from alphabetic fonts to corresponding Japanese fonts (here we don't assume that NFSS2 is available). Another approach is that we always use composite fonts with alphabetic fonts and Japanese fonts. The third approach is that the information of the current Japanese font is stored in an attribute. We adopted the third approach, since LuaTeX-ja is much affected by pTeX as we noted in Subsection 1.2.

As in Figure 2, LuaTeX-ja uses `\jfont` for defining Japanese fonts, as pTeX. However, because the information of the current Japanese font is stored into an attribute, control sequences defined by `\jfont` (e.g., `\foo` and `\bar` in Figure 2) is not representing a font by the means of TeX82. In other words, each of these control sequences is just an assignment to an attribute, therefore they cannot be an argument of `\the`, `\fontname`, nor `\textfont`.

Callbacks by the *luaotfload* package, e.g., replacement of glyphs according to OpenType font features, are performed just after 'Examination of stack level' (see Subsections 1.3 and 5.2). Also note that calculation of character classes for each Japanese character is done *after* the these callbacks for now.

5.2 Stack management

As we noted in Subsection 2.1, parameters that the values at the end of a horizontal box or that of a paragraph are valid in whole box or paragraph, such as *kanjiskip*, cannot be implemented by internal integers or registers of other types in TeX. We explain it in this subsection.

Figure 8 is an extract of a CWEB-source `tex/packaging.w` of LuaTeX (SVN revision 4358). This function is called just when an explicit `\hbox{...}` or `\vbox{...}` is ended, and the function `filtered_hpack()` is where the `hpack_filter` and then the actual 'hpack' process are performed. Notice that the `unsave()` function is called before `filtered_hpack()`. This is the problem; because of `unsave()`, we can retrieve only the values of registers *outside* the box, even in the `hpack_filter` callback.

To cope with this problem, LuaTeX-ja has its own stack system, based on Lua codes in [17]. Furthermore, *whatsit* nodes whose *user_id* is 30112 (*stack_node*, for short) will be appended to the current horizontal list each time the current stack level is incremented, and their values are the values of `\currentgrouplevel` at that time. In the beginning

```
void package(int c)
{
    ...
    d = box_max_depth;
    unsave();
    save_ptr -= 4;
    if (cur_list.mode_field == -hmode) {
        cur_box = filtered_hpack(cur_list.head_field,
                                 cur_list.tail_field, saved_value(1),
                                 saved_level(1), grp, saved_level(2));
        subtype(cur_box) = HLIST_SUBTYPE_HBOX;
    } else {
```

FIGURE 8. An extract of a CWEB-source `tex/packaging.w` of LuaTEX.

of the `hpack_filter` callback, the list in question is traversed to determine whether the stack level at the end of the list and that outside the box coincides.

Let x be the value of `\currentgrouplevel`, and y be the current stack level, both inside the `hpack_filter` callback, i.e., outside a horizontal box. Consider a list which represents the content of the box, then we have:

– A *stack_node* whose value is $x+1$ (because all materials in the box are included in a group `\hbox{...}`, the value of `\currentgrouplevel` inside the box is at least $x+1$) in the list corresponds to an assignment related to the stack system in just top-level of the list, like

  `\hbox{...(assignment)...}`

  In this case, the current stack level is incremented to $y+1$ after the assignment.

– A *stack_node* whose value is more than $x+1$ in the list corresponds to an assignment inside another group contained in the box. For example, the following input creates a *stack_node* whose value is $x+3=(x+1)+2$:

  `\hbox{...{...{...(assignment)}...}...}`

Thus, we can conclude that the stack level at the end of the list is $y+1$, if and only if there is a *stack_node* whose value is $x+1$. Otherwise, the stack level is just y.

5.3 Adjustment of the position of Japanese characters

The size of a glyph specified in a metric and that of a real font usually differ. For example, the letter '【' is half-width in `jfm-ujis.lua` or `jis.tfm`, while this letter is full-width like '【' in most TrueType fonts used in Japanese typesetting, such as IPA Mincho. Hence the adjustment of position of such glyphs is needed. In the context of pTEX, this process was performed using virtual fonts.

On the other hand, LuaTEX-ja does the adjustment by encapsulating a glyph into a horizontal box. There are two main reasons why we adopted this method; one is that we feared Lua codes for coexisting with callbacks by the `luaotfload` package would be large if we use virtual fonts, and the other is to cope with shifting of the baseline of characters at the same time.

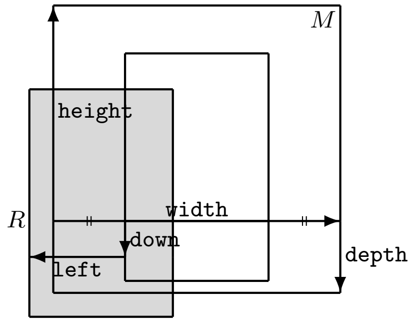

FIGURE 9. The position of the 'real' glyph.

Figure 9 shows the adjustment process. A large square M is the imaginary body specified in the metric, and a vertical rectangle is the imaginary body of a real glyph. First, the real glyph is aligned with respect to the width of M. In the figure, the real glyph is aligned 'middle'; this setting is useful for the full-width middle dot '・'. We have other settings, 'left' and 'right'. Furthermore, it is shifted according to the value of `left` and `down`, which are specified in the metric, for fine adjustment. The final position of the real glyph is shown by the gray rectangle R. If the amount of shifting the baseline is not zero, M (and hence the real glyph) is shifted by that amount.

We would like to remark briefly on the vertical position of a real glyph. A JFM (or a metric used in LuaTEX-ja) and a real font used for it may have different height or depth. In that case, it may look better if the real glyph is shifted vertically to match the height-depth ratio specified in the metric, while any vertical adjustment except the adjustment by the `down` value does not performed in the present implementation of LuaTEX-ja . This situation is carefully studied by Otobe [15]. Here the policy on this problem is not determined now, however we would like to offer several solutions in future development.

5.4 Further notes on metrics for LuaTEX-ja

Proportional typesetting Some fonts are proportional, that is, each glyphs in those fonts have its own width. An example of proportional fonts is IPA P Mincho. Using these fonts in pTEX is very hard, since one needs to make a dedicated JFM for a real font.

LuaTEX-ja supports these proportional fonts; specifying the `width` of a character class in a metric to `"prop"` makes the width of each character in this class that of a glyph in a real font. If no JFM glue is needed, one simply has to use `jfm-prop.lua`. The following is an example:

あいうえお あいうえお

```
\jfont\pr=file:ipamp.ttf:jfm=prop at 3.25mm
あいうえお\\\pr{}あいうえお
```

Scaling by metrics Because of virtual fonts, even if one specifies to use `min10.tfm` or `jis.tfm` at 10 pt in pTEX, the actual size of real fonts used in dviwares for these JFMs are 9.62216 pt. Hence, for example, if one wants to use 3.25 mm Japanese fonts and 10 pt alphabetic fonts in pTEX, he/she needs to scale a Japanese font by

$$\frac{3.25\,\mathrm{mm}}{10\,\mathrm{pt}\cdot 0.962216} \simeq 0.961$$

in declarations of Japanese fonts.

LuaTEX-ja didn't support such scaling of glyphs by metrics, so one has to adjust the size argument for `\jfont` manually. Continuing the previous example, for using 3.25 mm Japanese fonts and 10 pt alphabetic fonts in LuaTEX-ja, he/she needs to scale a Japanese font by $3.25\,\mathrm{mm}/10\,\mathrm{pt} \simeq 0.92487$.

6 Conclusion

We have discussed about our LuaTEX-ja package, which is much affected by pTEX. For now, it can be used for experimental use, however there are much refinements which are needed for regular use. The author hopes that this paper and LuaTEX-ja project contribute the typesetting Japanese, and possibly other Asian languages, under LuaTEX.

Acknowledgements

The author would like to thank Ken Nakano and Hideaki Togashi for their development and management of ASCII pTEX. The author is very grateful to Haruhiko Okumura for his leadership in the Japanese TEX community. The author is also very grateful to members of LuaTEX-ja project team for their valuable cooperation in development.

References

1. Adobe Systems Incorporated, *Adobe-Japan1-6 Character Collection for CID-Keyed Fonts*, Technical Note #5078, 2004. http://partners.adobe.com/public/developer/en/font/5078.Adobe-Japan1-6.pdf
2. ASCII MEDIA WORKS, アスキー日本語 TEX (pTEX). http://ascii.asciimw.jp/pb/ptex/
3. John Baker, *Typesetting UTF8 APL code with the LATEX lstlisting package*. http://bakerjd99.wordpress.com/2011/08/15/
4. Jin-Hwan Cho and Haruhiko Okumura, *Typesetting CJK Languages with Omega*, TEX, XML, and Digital Typography, Lecture Notes in Computer Science, vol. 3130, Springer, 2004, 139–148.
5. Yannis Haralambous. *The Joy of LuaTEX*. http://luatex.bluwiki.com/
6. Japanese Industrial Standards Committee. *JIS X 4051: Formatting rules for Japanese documents*, 1993, 1995, 2004.
7. 北川弘典, ε-pTEX についての wiki. http://sourceforge.jp/projects/eptex/wiki/FrontPage
8. 北川弘典, LuaTEX で日本語. http://oku.edu.mie-u.ac.jp/tex/mod/forum/discuss.php?d=378
9. LuaTEX development team, *The LuaTEX reference*. http://www.luatex.org/svn/trunk/manual/luatexref-t.pdf (snapshot of SVN trunk)

10. LuaTEX-ja project team, *The LuaTEX-ja package*. Not completed for now. Available at doc/man-en.pdf (in English) or doc/man-ja.pdf (in Japanese) in the Git repository.
11. 香田温人, LuaTEX と日本語. http://www1.pm.tokushima-u.ac.jp/~kohda/tex/luatex-old.html
12. 前田一貴, luajalayout パッケージ—LuaLATEX による日本語組版—. http://www-is.amp.i.kyoto-u.ac.jp/lab/kmaeda/lualatex/luajalayout/
13. 奥村晴彦, pLATEX 2_ε新ドキュメントクラス. http://oku.edu.mie-u.ac.jp/~okumura/jsclasses/
14. Haruhiko Okumura, *pTEX and Japanese Typesetting*, The Asian Journal of TEX **2** (2008), 43–51.
15. 乙部厳己, min10 フォントについて. http://argent.shinshu-u.ac.jp/~otobe/tex/files/min10.pdf
16. 齋藤修三郎, Open Type Font 用 VF. http://psitau.kitunebi.com/otf.html
17. Jonathan Sauer, *[Dev-luatex] tex.currentgrouplevel*. http://www.ntg.nl/pipermail/dev-luatex/2008-August/001765.html
18. Takuji Tanaka, *upTEX, upLATEX—unicode version of pTEX, pLATEX*. http://homepage3.nifty.com/ttk/comp/tex/uptex_en.html
19. Nobuyuki Tsuchimura and Yusuke Kuroki, *Development of Japanese TEX Environment*, The Asian Journal of TEX **2** (2008), 53–62.
20. W3C Working Group, *Requirements for Japanese Text Layout*. http://www.w3.org/TR/jlreq/