



Development of the LuaT_EX-ja package

Hironori Kitagawa 北川 弘典

LuaT_EX-ja project team h.kitagawa2001@yahoo.co.jp

KEYWORDS T_EX, pT_EX, LuaT_EX, LuaT_EX-ja, Japanese

ABSTRACT LuaT_EX-ja is a macro package for typesetting Japanese documents with LuaT_EX. It enjoys improved flexibility of LuaT_EX in typesetting T_EX documents, eliminating some unwanted features of pT_EX, the widely-used variant of T_EX for the Japanese language. In this paper, we describe the specifications, the current status, and some internal processing methods of LuaT_EX-ja.

1 Introduction

pT_EX [15], developed by ASCII Corp., is widely used in Japan to typeset T_EX documents written in Japanese. There have been other tools, such as Omega [3] and the CJK package [6], but they have received little attention by Japanese T_EX users. The main reason for the popularity of pT_EX is that it enjoys superior typesetting quality than alternative methods. Another reason is that pT_EX has already taken the dominant position in Japan.

However, pT_EX lacks some advanced features of modern extensions of T_EX, such as ϵ -T_EX and pdfT_EX. Moreover, it does not support full range of Unicode characters.

In recent years, there have been several attempts at improving pT_EX: ptexenc [13] by Nobuyuki Tsuchimura (土村展之), ϵ -pT_EX [18] by this author, and upT_EX [12] by Takuji Tanaka (田中琢爾). All of these attempts, including pT_EX itself, have a common approach of development, i.e., to implement a localized variant of T_EX for the Japanese language. As a result, we have multiple executable programs, and it is impossible to use features of different extensions at the same time. The approach using the LuaT_EX engine solves the inconvenience of using multiple executables. Features of these different extensions can be realized in the LuaT_EX approach using Lua callbacks that hook TeX's internal process.

Prior to LuaT_EX-ja [8] there were several experimental attempts at typesetting Japanese documents in LuaT_EX. Three of them are as follows.

- luaaums.sty [19] developed by the present author. The package allows LuaT_EX to typeset Japanese characters.
- luajalayout [22] by Kazuki Maeda (前田一貴), formerly known as jafontspec. The package is based on L^AT_EX_{2 ϵ} and fontspec [10].
- luajp-test [20] written by Atsuhito Kohda (香田温人) based on the article [4].

All of these attempts were based on L^AT_EX_{2 ϵ} , and they did not support enough controls required for Japanese typesetting. Moreover, it was inefficient to maintain similar packages separately. Development of LuaT_EX-ja was started initially by the author and Kazuki Maeda.¹

The initial aim of the LuaT_EX-ja project was to implement the features ('primitive's) of pT_EX as LuaT_EX macros. We wanted LuaT_EX-ja to be at least as flexible as pT_EX is in typesetting Japanese documents. Compared with the Japanese typesetting standard JIS X 4051 [5] and W3C Requirements for Japanese Text Layout [14], pT_EX is more flexible, with its tunable parameters such as `\kanjiskip` and `\prebreakpenalty`, and its customized JFM (Japanese TFM).

It turned out, however, that straightforward implementation of pT_EX features is neither sufficient nor desirable, as will be discussed in the next section.

The second aim is that LuaT_EX-ja is neither a mere re-implementation nor a porting of pT_EX. Technically and/or conceptually inconvenient features of pT_EX are modified. In Section 2 these features will be described in detail.

We now describe an outline of the process of LuaT_EX-ja in order.

`process_input_buffer callback` handles 'line-breaking' after Japanese characters (see Section 2.1).

`hyphenate callback` handles 'font replacement'. For each *glyph_node* p in the horizontal list, if the character represented by p is Japanese, the font for p will be replaced by the value of the attribute `\lj@curjfont`, 'the current Japanese font'. Furthermore, the subtype of p is subtracted by 1 to suppress hyphenation around p .

`pre_linebreak_filter` and `hpack_filter callbacks` handles the following processes.

1. LuaT_EX-ja has its own stack system, and the current horizontal list is traversed in this stage to determine what the level of LuaT_EX-ja's internal stack is at the end of the list (see Section 5.2).
2. In this stage, LuaT_EX-ja inserts glues/kerns for Japanese typesetting in the list. This is the core routine of LuaT_EX-ja (see Section 2.3).
3. While matching a font metric to a real font, adjustments to the positions of (Japanese) glyphs are necessary at times (see Section 5.3).

`mlist_to_hlist callback` handles Japanese characters in a math formula. This process is very similar to the position adjustments of glyphs in the previous process.

The processes above will be described in detail in Section 2 and 5. Section 2 contains the major differences between pT_EX and LuaT_EX-ja. We concentrate on how to distinguish Japanese characters from other characters in Section 3. The current development status and the internal routines of LuaT_EX-ja will be described in Section 4 and 5, respectively.

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

1. Members of the LuaT_EX-ja 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 Differences between pTeX and LuaTeX-ja

In this section, we describe major differences between pTeX and LuaTeX-ja. For general information of Japanese typesetting and an overview of pTeX, we refer to [9].

pTeX added several primitives to Knuth's original T_EX82, which are difficult to implement as T_EX macros. For instance, `\prebreakpenalty⟨char_code⟩ [=] ⟨penalty⟩` inserts a penalty *⟨penalty⟩* before every occurrence of a character *⟨char_code⟩*. The primitive is also used in the form `\prebreakpenalty⟨char_code⟩` to retrieve the penalty value assigned to the character.

On the other hand, LuaTeX-ja provides new control sequences by modifying the internal callbacks in LuaTeX. This leads to some restrictions. We will discuss this topic in detail in Section 5.2.

LuaTeX-ja provides the following three control sequences for assignment and retrieval of parameters:

- `\ltjsetparameter{⟨name⟩=⟨value⟩, ...}` assigns *⟨value⟩* locally to a parameter *⟨name⟩*.
- `\ltjglobalsetparameter{⟨name⟩=⟨value⟩, ...}` assigns *⟨value⟩* globally to a parameter *⟨name⟩*.
- `\ltjgetparameter{⟨name⟩} [⟨optional argument⟩]` retrieves the value assigned to the parameter *⟨name⟩*. The returned value is always a string.

Note that the two control sequences above for assignment obey the value of the primitive `\globaldefs`.

2.1 Japanese characters at the end of an input line

In Japanese typesetting line-breaking is allowed after a Japanese character in contrast to European typesetting in which either line-breaking is allowed between words or hyphenation is used instead. So it is natural that the input processor of pTeX does not regard an end-line character following a Japanese character as a space. However, there is no way to customize the input processor of LuaTeX without hacking the CWEB source. A possible way of LuaTeX-ja is to modify an input line inside the `process_input_buffer` callback before LuaTeX processes the line.

LuaTeX-ja appends a *comment* letter 'U+FFFFF' in order to avoid an extra space after a Japanese character found at the end of the line.² The behavior, at first glance, looks the same as that of pTeX. But there is a slight difference between them as Figure 1 shows.

The assignment `'jacharrange={-6}'` given in the second line of Figure 1 makes LuaTeX-ja treat 'Japanese' characters as 'non-Japanese'. Hence, two Japanese characters 'あ' in the second and the third line are treated as non-Japanese. Then, why does no space appear after the first 'あ' in the second line? The reason is that the assignment in the second line comes into effect *after* the second *input* line is processed by the `process_input_buffer` callback, and then at that time LuaTeX-ja appends the comment letter at the end of the line as mentioned above.

2. Strictly speaking, one more condition is required, that is the case when the category code of the end-line character is 5 (*end-of-line*). It is useful for the feature not to work in the verbatim environment.

```

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

```

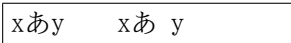


FIGURE 1. Different handling of LuaT_EX-ja from pT_EX: Japanese characters at the end of an input line.

2.2 Japanese font metrics

Traditional Japanese fonts are mainly fixed-width, with most glyphs designed on square canvases. Moreover, the font metric is common among fonts; there is no substantive difference among different JFMs. For example, `min10.tfm` and `goth10.tfm`, the two JFMs shipped with pT_EX for serif *mincho* and sans-serif *gothic* families, respectively, have the same contents except for the FAMILY name and the FACE name. As another example, `jis.tfm` and `jisg.tfm` used in `jsclasses` [16] by Haruhiko Okumura (奥村晴彦) are completely identical.

LuaT_EX-ja separates ‘real’ fonts and font metrics. The following example shows typical declarations of Japanese fonts in the style of plain T_EX.

```

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

```

Here are some remarks:

- A control sequence `\jfont` must be used for Japanese fonts, instead of `\font`.
- LuaT_EX-ja automatically loads the `luaotfload` package, so the prefixes, ‘file:’ and ‘name:’, and various font features can be used as shown in the first line of the example above.
- The ‘jfm’ key specifies the font metric associated with the font. In the example, the two fonts, `\foo` and `\bar`, will use the same font metric stored in the Lua script ‘jfm-ujis.lua’. It is a standard metric in LuaT_EX-ja, based on JFMs used in the `otf` package [21].
- The ‘psft:’ prefix specifies non-embedded fonts having only names. Non-embedded fonts, if included in a PDF file, will be replaced by actual fonts in the PDF reader.

The specification of a font metric for LuaT_EX-ja is similar to that of a JFM.³ 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. It is possible to develop a program that *converts* a JFM to a font metric for LuaT_EX-ja. LuaT_EX-ja provides three font metrics by default; `jfm-ujis.lua` and `jfm-jis.lua` based on the *jis* font metric, and `jfm-min.lua` based on the old `min10.tfm`.

Note that the font feature ‘-kern’ is important, because the kerning information from a real font will clash with the glue/kern information from the font metric.

3. For the specification of JFM, see [9].

2.3 Glues and kerns for Japanese characters

As described in [7] Lua \TeX handles kerning and ligature in a totally different way compared to \TeX 82. \TeX 82 processes kerning and ligature whenever a (sequence of) character is appended to the current list. On the other hand, Lua \TeX is *node-based*, in other words, Lua \TeX do the process at the end of a horizontal box or a paragraph. This is the reason why ‘f{}irm’ and ‘firm’ yield the same output in Lua \TeX .

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

- A glue (or kern) from the Japanese font metrics, called ‘JFM glue’ for short.
- Default glue between a Japanese character and an alphabetic character, called ‘*xkanjiskip*’. The length is, in usual, 1/4 of the full-width (*shibuaki*) with some stretch and shrink.
- Default glue between two consecutive Japanese characters, called ‘*kanjiskip*’. The main role of this glue is to allow line-breaking after Japanese characters. In most cases, the length is zero with some stretch and shrink.

These three categories are handled differently in p \TeX . A JFM glue is inserted when a (sequence of) Japanese character is appended to the current list, in the same way as a glue is inserted after a word of alphabetic characters in \TeX 82. Hence, {} prevents p \TeX from inserting a glue. A *xkanjiskip* is inserted just before the process of ‘hpack’ or the process of breaking lines in a paragraph. The processing time is quite similar to that of the kerning process of Lua \TeX . Finally, a *kanjiskip* is not appeared as a node anywhere. It appears implicitly in calculating the width of a horizontal box, the process of breaking lines and DVI output. These specifications have made p \TeX ’s behavior very hard to understand.

Lua \TeX -ja handles all three categories at the same time inside the Lua \TeX callbacks, `hpack_filter` and `pre_linebreak_filter`. It makes the process of Lua \TeX -ja more clear and simple than p \TeX .

We now investigate the process of glues and kerns in detail.

Nodes ignored in the process of glues and kerns

In the process of glues and kerns, Lua \TeX -ja ignores anything that does not make a node, as shown in the example (1) of Table 1.⁴ Moreover, any nodes that contribute nothing to the current horizontal list (*ins_node*, *adjust_node*, *mark_node*, *whatsit_node* and *penalty_node*) are also ignored, as shown in the example (4) of the same table.

It happens that some nodes are attached to a *glyph_node*, for example, an accent or a kern for italic correction.⁵ Lua \TeX -ja ignores these attached nodes inside the process. It is the same behavior as that of p \TeX (version p3.2), see the example (2) in Table 1.

How do the users of Lua \TeX -ja prevent ignoring nodes? One solution is to put an empty horizontal box ‘\hbox{ }’ at the appropriate place, as the example (3) in Table 1.

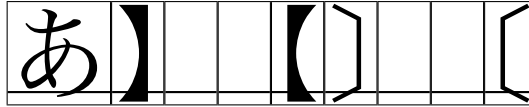
4. In the detailed output of the example (1) in Table 1, notice that p \TeX inserts two half-width glues between ‘J’ and ‘I’. The left glue is from ‘J’ and the right glue is from ‘I’.

5. \TeX 82 (and Lua \TeX) does not distinguish a kern for italic correction from an explicit kern. To distinguish them, p \TeX requires an additional subtype for a kern. On the other hand, Lua \TeX -ja uses an additional attribute and redefines ‘\’ to set the attribute.

TABLE 1. Different behavior of LuaT_EX-ja in handling glues.

Input	(1)	(2)	(3)	(4)
	あ】{ } [\ / (い』 \ / a	う) \ hbox{ } (え] \ special{ } [
pT _E X	あ】 [] [い』 a	う) (え] [
LuaT _E X-ja	あ】 [] [い』 a	う) (え] [

The figure below is the detailed output of the example (1) produced by pT_EX.



Glues between Japanese characters

Consider the following input and its output processed by pT_EX and by LuaT_EX-ja.

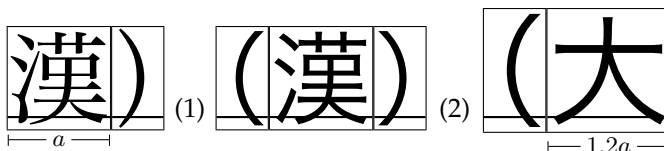
1 明朝 \gt (ゴシック 明朝) (ゴシック) (pT_EX) 明朝 (ゴシック) (LuaT_EX-ja)

Note that the Japanese characters on the left of ‘\gt’ are typeset by a serif Japanese font and the characters on the right are typeset by a sans-serif font. Even though two different Japanese fonts are used, they use the same font metric as mentioned before. In the case of pT_EX, two half-width glues are inserted between the Japanese characters. On the other hand, LuaT_EX-ja inserts only one glue because it treats Japanese fonts with the same font metric as one font. It is also possible to change the default behavior in LuaT_EX-ja. We leave it to the manual [8].

The following example shows the case of adjacent Japanese characters having different font metrics and/or different size.

1 漢 \gt (漢) \large (大) (漢) (漢) (大) (pT_EX)

The default behavior of LuaT_EX-ja in this case is different from that of pT_EX. The size of a glue that LuaT_EX-ja inserts between Japanese characters is the *average* of the two glues associated to the left and the right character. The following figure shows the output of LuaT_EX-ja.



The width of the glue (1) is $(\frac{a}{2} + \frac{a}{2})/2 = 0.5a$, and the width of the glue (2) is $(\frac{a}{2} + \frac{1.2a}{2})/2 = 0.55a$. The default behavior can be changed by the ‘differentjfm’ parameter in LuaT_EX-ja.

kanjiskip and xkanjiskip

In pT_EX, the value of *xkanjiskip* is controlled by the control sequence \xkanjiskip. One defect of the implementation of pT_EX is that the value of *xkanjiskip* is not related to the size of the current Japanese font. It seems that the parameters, EXTRASPACE,

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)

EXTRASTRETCH, and EXTRASHRINK in a JFM are reserved for specifying the default value of *xkanjiskip* in a unit of the design size, but p \TeX never use the parameters.

On the contrary, Lua \TeX -ja uses the value of *xkanjiskip* specified in a font metric. If the value of *xkanjiskip* on the user side (that is the value of `xkanjiskip` parameter of `\ltjsetparameter`) is `\maxdimen`, then Lua \TeX -ja extracts the value of *xkanjiskip* from the current font metric. This description is also applied to *kanjiskip*.

3 Distinguishing Japanese characters

Since Lua \TeX can handle Unicode natively, it becomes an important problem to distinguish Japanese characters from alphabetic characters. For example, the multiplication sign (U+00D7) exists both in ISO-8859-1 (hence in Latin-1 Supplement in Unicode) and in JIS X 0208. It is not desirable that this character is always treated as an alphabetic character, because it is often used in the meaning of ‘negative’ in Japan.

3.1 Character ranges

We first review how up \TeX [12] distinguishes Japanese characters. up \TeX extends the `\kcatcode` primitive in p \TeX , in order to divide Unicode characters, for example, alphabetic characters (15), *kanji* (16), *kana* (17), *Hangul* (17), and *other CJK characters* (18). It is also possible to assign a `\kcatcode` number to a Unicode block.⁶

Lua \TeX -ja adopts a different approach. There are many Unicode blocks in ‘Basic Multilingual Plane’ which are not included in Japanese fonts. Furthermore, JIS X 0208 is not a union of Unicode blocks; for example, see the intersection of JIS X 0208 and Latin-1 Supplement in Table 2. In Lua \TeX -ja one has to define in advance the ranges of character codes in the source to customize the range of Japanese characters.

We note that Lua \TeX -ja provides two additional control sequences, `\ltjjachar` and `\ltjalchar`, which are similar to the primitive `\char`. The control sequence `\ltjjachar` yields a Japanese character provided its argument is more than or equal to 128. On the other hand, `\ltjalchar` always yields an alphabetic character regardless of the argument.

3.2 Predefined ranges

In the patches for plain \TeX and L \TeX 2 ϵ , Lua \TeX -ja predefines eight character ranges, as shown in Table 3. Almost all of these ranges are just the union of Unicode blocks, and are determined from the ‘Adobe-Japan1-6’ character collection [1] and JIS X 0208. Among the eight ranges, the ranges 2, 3, 6, 7, and 8 are considered as the ranges of Japanese characters, and others are considered as the ranges of alphabetic characters.

We make a remark on the range 2 and 8:

6. There are some exceptions. For example, U+FF00–FFEF (halfwidth and fullwidth forms) are divided into three blocks in the recent version of up \TeX .

TABLE 3. Predefined ranges in LuaT_EX-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.

The range 2 JIS X 0208 includes Greek and Cyrillic letters, however, these letters cannot be used for typesetting Greek or Russian. It is reasonable that Greek and Cyrillic letters form another character range.

The range 8 To use 8-bit TFMs, such as T1 or TS1 encodings, the range 8 should be marked as a range of alphabetic characters as follows:

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

Some 8-bit TFMs have a glyph in this range; for example, the character ‘ \mathcal{E} ’ is located at "D7 in the T1 encoding.

3.3 Unicode characters

The ‘fontspec’ package provides various control sequences producing Unicode characters.⁷ However, these control sequences do not work correctly with the predefined range settings of LuaT_EX-ja. For example, `\textquotedblleft` is just an abbreviation of `\char"201C\relax`, and the character ‘U+201C (LEFT DOUBLE QUOTATION MARK)’ is treated as a Japanese character, because it belongs to the range 3. This problem can be resolved by using `\ltjalchar` instead of the primitive `\char`. It is included in the optional package ‘`luatexja-fontspec.sty`’. The following example shows several ways to typeset a character, both as a Japanese character and as an alphabetic character.

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

×, ×, ×, ×, ×

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 a synonym of a character. For example, the meaning of `\otimes` is just the character ‘U+2297 (CIRCLED TIMES)’ 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}...` should be permitted.

LuaT_EX-ja does not give a satisfactory solution to this problem. The following candidates are being tested for a solution:

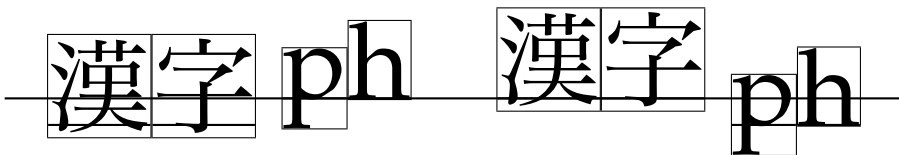
7. More precisely, the role is played by the ‘`xunicode`’ package, originally a package for X_YT_EX and automatically loaded by the `fontspec` package.

- Lua \TeX -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 character codes mentioned in the unicode-math package will be included in the list.

4 Current status of Lua \TeX -ja

We investigate the current status of Lua \TeX -ja part by part.⁸ The development of the lowest part of Lua \TeX -ja is almost finished, which corresponds to the implementation of p \TeX as an *extension* of \TeX . However, Lua \TeX -ja does not support yet the vertical typesetting called '*tategaki*' in Japanese.

We now explain the feature 'baseline shift' of Lua \TeX -ja in detail. To achieve better balance between Japanese and alphabetic characters it is required to shift appropriately the baseline of alphabetic characters. The dimension `\ybaselineshift` in p \TeX controls how much the baseline of alphabetic characters shift down. Lua \TeX -ja provides two parameters, '`yjabaselineshift`' and '`yalbaselineshift`', for the baseline shift of Japanese and alphabetic characters, respectively.



In the figure above, the left half shows the baseline shift (down) of Japanese characters when `yjabaselineshift` is positive. On the other hand, the right half shows the baseline shift (down) of alphabetic characters in the case of positive `yalbaselineshift`. An interesting application of these parameters is shown in the following figure.



4.1 Patches for plain \TeX and L \TeX 2 ϵ

p \TeX provides patches to support the plain \TeX macros and L \TeX 2 ϵ macros. The default setting of Japanese hyphenation called '*kinsoku shori*' is also provided in the file '`kinsoku.tex`'. These patches are ported to Lua \TeX -ja except for the codes related to vertical typesetting.

We remark on the behavior of `\fontfamily` that changes in pL \TeX 2 ϵ the current alphabetic and/or Japanese font family. More precisely, `\fontfamily{<arg>}` changes the current alphabetic font family to `<arg>` if and only if one of the following conditions is satisfied:

- An alphabetic font family `<arg>` in *some* alphabetic encoding is already defined in the document.

8. At the moment, Lua \TeX -ja runs under plain \TeX and under L \TeX 2 ϵ . In order to typeset Japanese characters only, it is enough to load `luatexja.sty` using `\input` or `\usepackage` in L \TeX 2 ϵ .

- An alphabetic encoding $\langle enc \rangle$ is already defined in the document and a font definition file ' $\langle enc \rangle \langle arg \rangle .fd$ ' (all lowercase) exists.

The same criterion is also used for changing the current Japanese font family.

Notice that a list of all (alphabetic) encodings defined in the document is required for this behavior of `\fontfamily`. But there is no way of LuaT_EX-ja having such a list, since LuaT_EX-ja is loaded as a macro package. LuaT_EX-ja adopts a different approach, that is, `\fontfamily{arg}` changes the current alphabetic font family to $\langle arg \rangle$ if and only if:

- An alphabetic font family $\langle arg \rangle$ in the current alphabetic encoding $\langle enc \rangle$ is already defined in the document.
- A font definition file ' $\langle enc \rangle \langle arg \rangle .fd$ ' (all lowercase) exists.

4.2 Classes and packages for Japanese documents

To produce 'high-quality' Japanese documents, we need not only Japanese characters correctly placed but also a class file for Japanese documents. Two major families of classes are widely used in Japan. One is 'jclasses' distributed with the official pL^AT_EX_{2 ϵ} macros, and the other is 'jsclasses'. LuaT_EX-ja contains their counterparts, 'ltjclasses' and 'ltjsclasses'. However, the policy on class files is not settled yet.⁹

Apart from the patches for the kernel of L^AT_EX_{2 ϵ} and class files for Japanese documents, we need to make a patch for some macro packages.

The 'fontspec' package is built on NFSS2, so the control sequences provided by the package, such as `\setmainfont`, are only effective for alphabetic fonts if LuaT_EX-ja is loaded. The counterpart for Japanese fonts is provided by '`luatexja-fontspec.sty`' (not automatically loaded), with an additional 'j' in the name of each control sequence, for example `\setmainjfont`. It also includes a patch for control sequences producing Unicode characters (see Section 3.3).

The 'otf' package is widely used in pT_EX for (1) typesetting characters not included in JIS X 0208, and (2) using more than one font weights for *mincho* and *gothic* font families. LuaT_EX-ja supports these features by loading '`luatexja-otf.sty`' manually. In order to avoid the callbacks called by the `luaotfload` package, that characters given by `\UTF` and `\CID` are not appended to the current list as a *glyph_node*. We note that `\CID` does not work with TrueType fonts, since it uses a conversion table between CID and the glyph order of the current Japanese font.

The patch '`jlisting.sty`' for the 'listings' package is well known for pT_EX users, which allows Japanese characters in the `lstlisting` environment. The patch can also be used in LuaT_EX-ja. But a Japanese character following a space is not processed by the listings package; it is inconvenient when we use the `showexpl` package.

There is another way in [2] to use characters whose code are above 256 with the listings package. However, this method is not suitable for Japanese characters, since the number of Japanese characters is very large.

⁹ We hope that another family of class files is available for commercial printing. On the other hand, `ltjclasses` should be useful as an example for porting class files from pT_EX to LuaT_EX-ja.

```

1 void package(int c)
2 {
3     ...
4     d = box_max_depth;
5     unsave();
6     save_ptr -= 4;
7     if (cur_list.mode_field == -hmode) {
8         cur_box = filtered_hpack(cur_list.head_field,
9                                 cur_list.tail_field, saved_value(1),
10                                saved_level(1), grp, saved_level(2));
11         subtype(cur_box) = HLIST_SUBTYPE_HBOX;
12     } else {

```

FIGURE 2. A part of the CWEB source `tex/packaging.w` in LuaTeX (SVN revision 4358).

5 Implementation

5.1 Handling Japanese fonts

pTeX uses three slots to maintain the current font, one for an alphabetic font as TeX82, and the other two for Japanese fonts in horizontal and vertical direction, respectively. How do we implement the concept of the current Japanese fonts in LuaTeX that has only one slot for the current alphabetic font?

There are three approaches to implement this feature. One is a mapping table in which each alphabetic font corresponds to specific Japanese fonts. Here, we don't assume that NFSS2 is available. The second one is a composite font consisting of alphabetic fonts and Japanese fonts. LuaTeX-ja follows the third approach storing the information of the current Japanese font as an *attribute* provided by LuaTeX.

As the example in Section 2.2 shows, LuaTeX-ja uses the control sequence `\jfont` to define Japanese fonts in the same way as pTeX. However, the control sequences defined by `\jfont` (e.g., `\foo` and `\bar` in the example) cannot be used to extract font information by means of an ordinary way of TeX, for instance, as an argument of `\the`, `\fontname`, and `\textfont`.

The callbacks called by the `luaotfload` package (e.g., replacement of glyphs according to OpenType font features) are performed immediately after the process 'examination of stack level' (see Section 5.2). After these callbacks were called, character class is calculated for each Japanese character.

5.2 Stack management

It is not possible by TeX macros to implement a parameter (e.g., *kanjiskip*) the value of which at the end of a horizontal box (or paragraph) is applied to every place in the box (or paragraph). We describe how to implement this parameter in LuaTeX-ja.

Figure 2 shows a part of the function 'package()' that is called immediately after an explicit horizontal box or a vertical box is ended by LuaTeX. The `hpack_filter` callback and then the actual 'hpack' process are to be called from 'filtered_hpack()' in the eighth line of the source. Notice that the function 'unsave()' in the fifth line is called before `filtered_hpack()`. It is the reason why we can retrieve only the values

of registers *outside* the box, even in the `hpack_filter` callback.

LuaT_EX-ja implements its own stack system to solve this problem, based on the Lua codes in [11]. Furthermore, *whatsit* nodes whose *user_id* is 30112 (called *stack_node*, for short) are appended to the current horizontal list whenever the current stack level increases and the value is the same as that of `\currentgrouplevel` at that time. In the beginning of the `hpack_filter` callback, the list in question is traversed to determine whether the two stack levels at the end of the list and outside the box coincides.

Let x be the value of `\currentgrouplevel` and let y be the current stack level, both inside the `hpack_filter` callback (or outside a horizontal box). Consider a list containing the elements in the box.

- A *stack_node* whose value is $x + 1$ in the list corresponds to an assignment related to the stack system in the top level of the list, for example,

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

Because all elements in the box are included in a group `\hbox{...}`, the value of `\currentgrouplevel` inside the box is at least $x + 1$. 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,

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

creates a *stack_node* whose value is $x + 3 = (x + 1) + 2$.

Therefore, we 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

It happens in usual that the size of a glyph specified in a font metric differs from the one specified in a real font. For example, the letter ‘`!`’ is half-width in `jfm-ujis.lua` or `jis.tfm`, while it is full-width in many TrueType fonts used in Japanese typesetting, such as IPA Mincho. Hence we need to adjust the positions of such glyphs. Virtual fonts are used in pT_EX for this process of adjustment.

On the other hand, LuaT_EX-ja does the adjustment by encapsulating a glyph into a horizontal box. There are two main reasons why this method was adopted. The first reason is the size of the Lua codes that coexist with the callbacks called by the `luaotfload` package if virtual fonts are used. The other reason is to cope with the baseline shift of characters at the same time.

Figure 3 shows the adjustment process. The large square M is the imaginary body of a Japanese character specified in the font metric, and the rectangle inside M is the imaginary body of the real glyph. First of all, the real glyph is horizontally aligned with respect to the width of M . In the figure, it is aligned horizontally in the center. This alignment is useful for the full-width middle dot ‘`・`’. There are another adjustments, the ‘left’ and the ‘right’ shift. The real glyph aligned horizontally is shifted according to the values of `left` and `down` that are specified in the font metric for fine refinement. The final position of the real glyph is shown in the figure by the gray rectangle R . Furthermore, M and hence the real glyph are shifted if the base line shift is required.

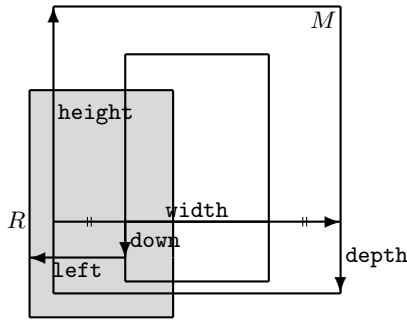


FIGURE 3. Adjustment of the position of a Japanese character.

A brief remark on the vertical positioning of a real glyph is in order. A JFM (or a font metric used in LuaTeX-ja) and a real font may have different heights or depths for a specific character.¹⁰ In this case, it may be better if the real glyph is shifted vertically to match the height-depth ratio specified in the metric. We are going to implement this feature after the implementation details are agreed upon.

5.4 Further notes on font metrics in LuaTeX-ja

Proportional Japanese fonts

There are proportional Japanese fonts in which each glyph has its own width. One example is ‘IPA P Mincho’. It is very hard to use these fonts in pTeX, because one needs to make a dedicated JFM for a real font.

LuaTeX-ja supports proportional Japanese fonts. If the width of a character class is specified as ‘prop’ in a font metric, then the width of all characters in the character class comes from that of the corresponding glyph in the real font. If no JFM glue is needed, it is enough to use `jfm-prop.lua`. In the following example, the first line shows the fixed-width, on the contrary, the second one shows proportional width.

<pre>1 \jfont\pr=file:ipamp.ttf:jfm=prop at 3.25mm 2 あいうえお\pr{あいうえお}</pre>	あいうえお あいうえお
--	----------------

Scaling by font metrics

Traditional JFM of pTeX, such as `min10.tfm` and `jis.tfm`, specify the width of a fullwidth glyph as 0.962216 times the design size. This means that in the default 10pt setting, the Japanese fullwidth glyph is 9.62216pt wide. If one wants to use 3.25 mm (13Q or 13 quarter-mm) Japanese fonts and 10pt alphabetic fonts in pTeX, for example, we need to scale Japanese fonts by

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

in the declaration of a Japanese font.

LuaTeX-ja didn’t support such scaling of glyphs by font metrics, so one has to adjust manually the size argument of `\jfont`. Continuing the previous example, to

10. Otake [17] carefully discusses this issue.

use 3.25 mm Japanese fonts and 10 pt alphabetic fonts in LuaT_EX-ja, we need to scale a Japanese font by $3.25 \text{ mm}/10 \text{ pt} \simeq 0.925$.

6 Conclusion

We have discussed LuaT_EX-ja, a macro package for typesetting Japanese documents in LuaT_EX. Even though LuaT_EX-ja is highly affected by pT_EX, it is more powerful than pT_EX, because of the flexibility coming from the Lua programming language embedded in LuaT_EX. LuaT_EX-ja is still at the experimental stage, and many refinements are required for regular use. The author hopes that this paper and the LuaT_EX-ja project contribute to typesetting of Japanese language and possibly other Asian languages.

Acknowledgements

The author would like to thank Ken Nakano and Hideaki Togashi for their development and management of ASCII pT_EX. The author is very grateful to Haruhiko Okumura for his leadership in the Japanese T_EX community. The author is also very grateful to the members of the LuaT_EX-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. John Baker, *Typesetting UTF8 APL code with the L^AT_EX lstlisting package*. <http://bakerjd99.wordpress.com/2011/08/15/>
3. Jin-Hwan Cho and Haruhiko Okumura, *Typesetting CJK Languages with Omega, T_EX, XML, and Digital Typography*, Lecture Notes in Computer Science, vol. 3130, Springer, 2004, 139–148.
4. Yannis Haralambous, *The Joy of LuaT_EX*. <http://luatex.bluwiki.com>
5. Japanese Industrial Standards Committee, *JIS X 4051: Formatting rules for Japanese documents*, 1993, 1995, 2004.
6. Werner Lemberg, *The CJK package for L^AT_EX*. CTAN:language/chinese/CJK/
7. LuaT_EX development team, *The LuaT_EX reference*. <http://www.luatex.org/svn/trunk/manual/luatexref-t.pdf>
8. LuaT_EX-ja project team, *The LuaT_EX-ja package*. <http://sourceforge.jp/projects/luatex-ja/> Documentations (not completed yet) written in English and Japanese are available in the Git repository.
9. Haruhiko Okumura, *pT_EX and Japanese Typesetting*, The Asian Journal of T_EX 2 (2008), 43–51.
10. Will Robertson and Khaled Hosny, *The fontspec package*. CTAN:macros/latex/contrib/fontspec/
11. Jonathan Sauer, *[Dev-luatex] tex.currentgrouplevel*. <http://www.ntg.nl/pipermail/dev-luatex/2008-August/001765.html>

12. Takuji Tanaka, *upTeX, upL^AT_EX—unicode version of pTeX, pL^AT_EX*.
http://homepage3.nifty.com/ttk/comp/tex/uptex_en.html
13. Nobuyuki Tsuchimura and Yusuke Kuroki, *Development of Japanese T_EX Environment*, *The Asian Journal of T_EX* 2 (2008), 53–62.
14. W3C Working Group, *Requirements for Japanese Text Layout*. <http://www.w3.org/TR/jlreq/>
15. アスキー・メディアワークス, アスキー日本語 T_EX (pT_EX).
<http://ascii.asciimw.jp/pb/ptex/>
16. 奥村晴彦, pL^AT_EX2_ε 新ドキュメントクラス.
<http://oku.edu.mie-u.ac.jp/~okumura/jsclasses/>
17. 乙部巖己, min10 フォントについて.
<http://argent.shinshu-u.ac.jp/~otobe/tex/files/min10.pdf>
18. 北川弘典, ϵ -pT_EX についての wiki.
<http://sourceforge.jp/projects/eptex/wiki/FrontPage>
19. 北川弘典, LuaT_EX で日本語.
<http://oku.edu.mie-u.ac.jp/tex/mod/forum/discuss.php?d=378>
20. 香田温人, LuaT_EX と日本語.
<http://www1.pm.tokushima-u.ac.jp/~kohda/tex/luatex-old.html>
21. 齋藤修三郎, Open Type Font 用 VF. <http://psitau.kitunebi.com/otf.html>
22. 前田一貴, lua_lajalayout パッケージ—LuaL^AT_EX による日本語組版—. <http://www-is.amp.i.kyoto-u.ac.jp/lab/kmaeda/lualatex/luajalayout/>