LuaTEX-ja パッケージ

LuaTEX-ja プロジェクトチーム

2011 年 11 月 20 日

目次

本ドキュメントはまだまだ未完成です．

第 I 部

ユーザーズマニュアル

1　はじめに

LuaTEX-ja パッケージは，次世代標準 TEX である LuaTEX の上で，pTEX と同等/それ以上の品質の日本語組版を実現させようとするマクロパッケージである．

1.1　背景

従来，「TEX を用いて日本語組版を行う」といったとき，エンジンとしては ASCII pTEX やそれの拡張物が用いられることが一般的であった．pTEX は TEX のエンジン拡張であり，（少々仕様上不便な点はあるものの）商業印刷の分野にも用いられるほどの高品質な日本語組版を可能としている．だが，それは弱点にもなってしまった：pTEX という（組版的に）満足なものがあったため，海外で行われている数々の TEX の拡張 例えば ε-TEX や pdfTEX や，TrueType, OpenType, Unicode といった計算機で日本語を扱う際の状況の変化に追従することを怠ってしまったのだ．

ここ数年，若干状況は改善されてきた．現在手に入る大半の pTEX バイナリでは外部 UTF-8 入力が利用可能となり，さらに Unicode 化を推進し，pTEX の内部処理まで Unicode 化した upTEX も開発されている．また，pTEX に ε-TEX 拡張をマージした ε-pTEX も登場し，TEX Live 2011 では pLATEX が ε-pTEX の上で動作するようになった．だが，pdfTEX 拡張（PDF 直接出力や micro-typesetting）を pTEX に対応させようという動きはなく，海外との gap は未だにあるのが現状である．

しかし，LuaTEX の登場で，状況は大きく変わることになった．Lua コードで‘callback’を書くことにより，LuaTEX の内部処理に割り込みをかけることが可能となった．これは，エンジン拡張という真似をしなくても，Lua コードとそれに関する TEX マクロを書けば，エンジン拡張とほぼ同程度のことができるようになったということを意味する．LuaTEX-ja は，このアプローチによって Lua コード・TEX マクロによって日本語組版を LuaTEX の上で実現させようという目的で開発が始まったパッケージである．

1.2　pTEX からの主な変更点

LuaTEX-ja は，pTEX に多大な影響を受けている．初期の開発目標は，pTEX の機能を Lua コードにより実装することであった．しかし，開発が進むにつれ，pTEX の完全な移植は不可能であり，また pTEX における実装がいささか不可解になっているような状況も発見された．そのため，**LuaTEX-ja** は，もはや **pTEX** の完全な移植は目標とはしない．**pTEX** における不自然な仕様・挙動があれば，そこは積極的に改める．

以下は pTEX からの主な変更点である．

- 和文フォントは「実際の」フォント，和文フォントメトリック（JFM と呼ぶ），そして ‘variation’ と呼ばれる文字列の組である．
- 日本語の文書中では改行はほとんどどこでも許されるので，pTEX では和文文字直後の改行は無視される（スペースが入らない）ようになっていた．しかし，LuaTEX-ja では LuaTEX の仕様のためにこの機能は完全には実装されていない．
- 2 つの和文文字の間，和文文字と欧文文字の間に入るグルー／カーン（**JAglue** と呼ぶ）の挿入処理が 0 から書き直されている．
  - LuaTEX の内部での文字の扱いが「ノードベース」になっているように（例えば，`of{}fice` で合字は抑制されない），**JAglue** の挿入処理も「ノードベース」である．

  - さらに，2 つの文字の間にある行末では効果を持たないノード（例えば\special ノード）や，イタリック補正に伴い挿入されるカーンは挿入処理中では無視される．
  - 注意：上の **2** つの変更により，従来 **JAglue** の挿入処理を分断するのに使われていたいくつかの方法は用いることができない．具体的には，次の方法はもはや無効である：

    ちょ{}っと　　ちょ\/っと

    もし同じことをやりたければ，空の hbox を間に挟めばよい：

    ちょ\hbox{}っと
  - 処理中では，2 つの和文フォントは，「実際の」フォントのみが異なる場合に同一視される．
- 現時点では，縦書きは LuaTEX-ja ではサポートされていない．

詳細については第 III 部を見よ．

1.3　用語と記法

本ドキュメントでは，以下の用語と記法を用いる：

- 文字は 2 種類に分けられる：
  - **JAchar**: ひらがな，カタカナ，漢字，和文用の約物といった和文文字のことを指す．
  - **ALchar**: アルファベットを始めとする，その他全ての文字を指す．

  そして，**ALchar** の出力に用いられるフォントを「欧文フォント」と呼び，**JAchar** の出力に用いられるフォントを「和文フォント」と呼ぶ．
- サンセリフ体で書かれた語（例：prebreakpenalty）は日本語組版用のパラメータを表し，これらは\ltjsetparameter コマンドのキーとして用いられる．
- 下線付きのタイプライタ体で書かれた語（例：fontspec）は LATEX のパッケージやクラスを表す．
- 「プリミティブ」という語を，LuaTEX のプリミティブだけではなく LuaTEX-ja のコアモジュールで定義されたコントロールシーケンスに対しても用いる．
- 本ドキュメントでは，自然数は 0 から始まる．

1.4　プロジェクトについて

プロジェクト Wiki　プロジェクト Wiki は構築中である．

- http://sourceforge.jp/projects/luatex-ja/wiki/FrontPage（日本語）
- http://sourceforge.jp/projects/luatex-ja/wiki/FrontPage%28en%29（英語）

本プロジェクトは SourceForge.JP のサービスを用いて運営されている．

開発メンバー

- 北川 弘典
- 黒木 裕介
- 本田 知亮
- 前田 一貴
- 阿部 紀行
- 齋藤 修三郎
- 八登 崇之
- 山本 宗宏

2 使い方

2.1 インストール

LuaTEX-ja パッケージのインストールには，次のものが必要である．

- LuaTEX（バージョン 0.65.0-beta 以降）とその支援パッケージ．TEX Live 2011 や W32TEX の最新版ならば問題ない．
- LuaTEX-ja のソースアーカイブ（もちろん :)）．

インストール方法は以下のようになる：

1. ソースアーカイブをダウンロードする．
   現時点では，LuaTEX-ja の公式リリースはないので，レポジトリからアーカイブを取得しなければならない．次のようにすることで，Git レポジトリを取得することができる：

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

   もしくは，master ブランチの HEAD アーカイブを以下からダウンロードすることができる：

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

   開発中の最新の成果は master ブランチには含まれていないかもしれないことに注意．
2. アーカイブを展開する．src/ をはじめとしたいくつかのディレクトリができる．
3. src/ の中身全てを自分の TEXMF ツリーにコピーする．
4. もし mktexlsr を実行する必要があればそうする．

2.2 注意点

- 原稿のソースファイルの文字コードは UTF-8 でなければならない．EUC-JP や Shift-JIS は使用できない．
- いくつかのパッケージと衝突する．
  例えば，**JAchar** の範囲の設定がデフォルトのままだと，現行のバージョンでは unicode-math パッケージと共存できない．以下の行をプリアンブルに追加することで数学記号が正しく出るようになるが，副作用としていくつかの和文文字が **ALchar** として扱われるようになってしまう：

  ```
  \ltjsetparameter{jacharrange={-3, -8}}
  ```

2.3 plain TEX で使う

LuaTEX-ja を plain TEX で使うためには，単に次の行をソースファイルの冒頭に追加すればよい：

```
\input luatexja.sty
```

これで（ptex.tex のように）日本語組版のための最低限の設定がなされる：

- 以下の 6 つの和文フォントが定義される：

字体	フォント名	**‘10 pt’**	**‘7 pt’**	**‘5 pt’**
明朝体	Ryumin-Light	`\tenmin`	`\sevenmin`	`\fivemin`
ゴシック体	GothicBBB-Medium	`\tengt`	`\sevengt`	`\fivegt`

  - ‘Q（級）’ は日本の写植で用いられる単位で，$1\,\mathrm{Q} = 0.25\,\mathrm{mm}$ である．この長さは `\jQ` に保持されている．
  - ‘Ryumin-Light’ と ‘GothicBBB-Medium’ は PDF ファイルに埋め込まずに名前参照のみで用いることが広く受け入れられており，この場合 PDF リーダーが適切な外部フォントで代用する（例えば，Adobe Reader では Ryumin-Light は小塚明朝で代替される）．そこで，これらをデフォルトのフォントとして採用する．
  - 欧文フォントの文字は和文フォントの文字よりも，同じ文字サイズでも一般に小さい．そこで，これらの和文フォントの実際のサイズは指定された値よりも小さくなるように設定されており，具体的には指定の 0.962216 倍にスケールされる．
- **JAchar** と **ALchar** の間に入るグルー (xkanjiskip) の量は次のように設定されている：

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

2.4 LaTeX で使う

LaTeX 2_ε　LaTeX 2_ε を用いる場合も基本的には同じである．日本語組版のための最低限の環境を設定するためには，`luatexja.sty` を読み込むだけでよい：

```
\usepackage{luatexja}
```

これで pLaTeX の `plfonts.dtx` と `pldefs.ltx` に相当する最低限の設定がなされる：

- `JY3` は和文フォント用のフォントエンコーディングである（横書き用）．
  将来的に，LuaTeX-ja で縦書きがサポートされる際には，`JT3` を縦書き用として用いる予定である．
- 2 つのフォントファミリ `mc` と `gt` が定義されている：

字体	ファミリ	`\mdseries`	`\bfseries`	スケール
明朝体	`mc`	Ryumin-Light	GothicBBB-Medium	0.962216
ゴシック体	`gt`	GothicBBB-Medium	GothicBBB-Medium	0.962216

  どちらのファミリにおいても，その bold シリーズはゴシック体の medium シリーズであることに注意．これは初期の DTP において和文フォントが 2 つ（それがちょうど Ryumin-Light, GothicBBB-Medium だった）しか利用できなかった時の名残であり，pLaTeX での標準設定とも同じである．
- 数式モード中の和文文字は `mc` ファミリで出力される．

しかしながら，上記の設定は日本語の文書にとって十分とは言えない．日本語文書を組版するためには，`article.cls`, `book.cls` といった欧文用のクラスファイルではなく，和文用のクラスファイルを用いた方がよい．現時点では，jclasses （pLaTeX の標準クラス）と jsclasses （奥村晴彦氏によるクラスファイル）に対応するものとして，ltjclasses，ltjsclasses がそれぞれ用意されている．

`\CID`，`\UTF` と OTF パッケージのマクロ　pLaTeX では，JIS X 0208 にない Adobe-Japan1-6 の文字を出力するために，齋藤修三郎氏による otf パッケージが用いられていた．このパッケージは広く用いられているため，LuaTeX-ja においても otf パッケージの機能の一部をサポートしている．これらの機能を用いるためには luatexja-otf パッケージを読み込めばよい．

```
森
\UTF{9DD7}外と内田百\UTF{9592}とが\UTF{9AD9}島
    屋に行く 。

\CID{7652}飾区の\CID{13706}野家 ,
葛飾区の吉野家
```

森鷗外と内田百閒とが髙島屋に行く。
葛飾区の吉野家，葛飾区の吉野家

2.5 フォントの変更

意見：数式モード中の和文文字　pTEX では数式中での和文文字の使用がサポートされていたので，以下のようなソースが存在した：

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

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

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

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

LuaTEX-ja プロジェクトでは，数式モード中での和文文字はそれらが識別子として用いられるときのみ許されると考えている．この観点から，

- 上記数式のうち 1, 2 行目は正しくない．なぜならば ‘ 高温 ’ が意味のあるラベルとして，‘ よって ’ が接続詞として用いられているからである．
- しかしながら，3 行目は ‘ 素 ’ が識別子として用いられているので正しい．

したがって，LuaTEX-ja プロジェクトの意見としては，上記の入力は次のように直されるべきである：

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

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

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

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

また LuaTEX-ja プロジェクトでは，和文文字が識別子として用いられることはほとんどないと考えており，したがってこの節では数式モード中の和文フォントを変更する方法については記述しない．この方法については第 II 部を参照のこと．

plain TEX　plain TEX で和文フォントを変更するためには，`\jfont` プリミティブを用いなければならない．第 II 部を参照せよ．

NFSS2　LATEX 2_ε については，LuaTEX-ja では単にフォント選択システムを pLATEX 2_ε (`plfonts.dtx`) から採用している．

- 2 つのコントロールシーケンス `\mcdefault` と `\gtdefault` がそれぞれ明朝体とゴシック体のデフォルトのフォントファミリを指定するために用いられる．初期値：`\mcdefault` は `mc`，`\gtdefault` は `gt`．
- `\fontfamily`, `\fontseries`, `\fontshape`, そして`\selectfont` が和文フォントの属性を変更するために使用できる．

	エンコーディング	ファミリ	シリーズ	シェープ	選択
欧文フォント	\romanencoding	\romanfamily	\romanseries	\romanshape	\useroman
和文フォント	\kanjiencoding	\kanjifamily	\kanjiseries	\kanjishape	\usekanji
両方	—	–	\fontseries	\fontshape	—
自動選択	\fontencoding	\fontfamily	—	—	\usefont

ここで，\fontencoding{<encoding>}は，引数により和文側か欧文側かのどちらかが切り替わる．例えば，\fontencoding{JY3}は和文フォントのエンコーディングを JY3 に変更し，\fontencoding{T1} は欧文フォント側を T1 へと変更する．\fontfamily も引数により和文側，欧文側，あるいは両方のフォントファミリが切り替わる．詳細は 7.1 節を参照すること．

- 和文フォントファミリの定義には \DeclareFontFamily の代わりに\DeclareKanjiFamily を用いる．しかし，現在の実装では\DeclareFontFamily を用いても問題は生じない．

fontspec パッケージと同様の機能を和文フォントに対しても用いるためには，luatexja-fontspec パッケージをプリアンブルで読み込む必要がある．このパッケージは必要ならば自動で luatexja パッケージと fontspec パッケージを読み込む．

luatexja-fontspec パッケージでは，以下の 7 つのコマンドを fontspec パッケージの元のコマンドに対応するものとして定義している：

和文フォント	\jfontspec	\setmainjfont	\setsansjfont	\newjfontfamily
欧文フォント	\fontspec	\setmainfont	\setsansfont	\newfontfamily
和文フォント	\newjfontface	\defaultjfontfeatures	\addjfontfeatures	
欧文フォント	\newfontface	\defaultfontfeatures	\addfontfeatures	

```
\fontspec[Numbers=OldStyle]{TeX Gyre Termes}
\jfontspec{IPAexMincho}
JIS~X~0213:2004→辻

\addjfontfeatures{CJKShape=JIS1990}
JIS~X~0208:1990→辻
```

JIS X 0213:2004→ 辻

JIS X 0208:1990→ 辻

和文フォントについては全ての和文文字のグリフがほぼ等幅であるのが普通であるため，\setmonojfont コマンドは存在しないことに注意．また，これらの和文用の 7 つのコマンドでは Kerning feature はデフォルトでは off となっている．これはこの feature が **JAglue** と衝突するためである（4.1 節を見よ）．

3 パラメータの変更

LuaTEX-ja には多くのパラメータが存在する．そして LuaTEX の振る舞いのために，その多くは TEX のレジスタにではなく，LuaTEX-ja 独自の方法で保持されている．そのため，これらのパラメータを設定・取得するためには \ltjsetparameter と\ltjgetparameter を用いなければならない．

3.1 JAchar の範囲の設定

JAchar の範囲を設定するためには，まず文字範囲に 0 より大きく 217 より小さい自然数を割り当てる必要がある．これには \ltjdefcharrange プリミティブを用いる．例えば，次のように書くことで追加多言語面 (SMP) にある全ての文字と‘漢’の範囲番号が 100 に設定される．

```
\ltjdefcharrange{100}{"10000-"1FFFF,`漢}
```

この文字範囲への番号の割り当てはいつもグローバルであり，したがって文書の途中でこの操作をするべきではない．

もし指定されたある文字がある非零番号の範囲に属していたならば，これは新しい設定で上書きされる．例えば，SMP は全て LuaTEX-ja のデフォルトでは 4 番の文字範囲に属しているが，上記の指定を行えば SMP は 100 番に属すようになり，4 番からは除かれる．

文字範囲に番号を割り当てた後は，jacharrange パラメータが **JAchar** として扱われる文字の範囲を設定するために用いられる．例えば，以下は LuaTEX-ja の初期設定である：

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

jacharrange パラメータには整数のリストを与える．リスト中の負の整数 $-n$ は「文字範囲 n に属する文字は **ALchar** として扱われる」ことを意味し，正の整数 $+n$ は **JAchar** として扱うことを意味する．

初期設定　LuaTEX-ja では 8 つの文字範囲を設定している．これらは以下のデータに基づいて決定している．

- Unicode 6.0 のブロック．
- `Adobe-Japan1-UCS2` による Adobe-Japan1-6 の CID と Unicode の間のマッピング．
- 八登崇之氏による upTEX 用の `PXbase` バンドル．

以下ではこれら 8 つの文字範囲について記述する．番号のあとのアルファベット ‘J’ と ‘A’ はデフォルトで **JAchar** として扱われるかどうかを示す．これらの設定は `PXbase` バンドルで定義されている `prefercjk` と類似のものである．

範囲 8^{J}　ISO 8859-1 の下半分（ラテン 1 補助）と JIS X 0208 の共通部分にある記号．この文字範囲は以下の文字で構成される：

- § (U+00A7, 節記号)
- ¨ (U+00A8, トレマ)
- ° (U+00B0, 度)
- ± (U+00B1, 正又は負符号)
- ´ (U+00B4, アキュート・アクセント)
- ¶ (U+00B6, 段落記号)
- × (U+00D7, 乗算記号)
- ÷ (U+00F7, 除算記号)

範囲 1^{A}　ラテン文字．一部は Adobe-Japan1-6 にも含まれている．この範囲は以下の Unicode のブロックから構成されている．ただし，範囲 **8** は除く．

- U+0080–U+00FF: ラテン 1 補助
- U+0100–U+017F: ラテン文字拡張 A
- U+0180–U+024F: ラテン文字拡張 B
- U+0250–U+02AF: IPA 拡張（国際音声記号）
- U+02B0–U+02FF: 前進を伴う修飾文字
- U+0300–U+036F: ダイアクリティカルマーク（合成可能）
- U+1E00–U+1EFF: ラテン文字拡張追加

範囲 2^{J}　ギリシャ文字とキリル文字．JIS X 0208（したがって多くの和文フォント）はこれらの文字を持つ．

- U+0370–U+03FF: ギリシア文字及びコプト文字
- U+0400–U+04FF: キリル文字
- U+1F00–U+1FFF: キリル文字補助

範囲 3^{J}　句読点と記号類．ブロックのリストは表 1 に示してある．

範囲 4^{A}　通常和文フォントには含まれていない文字．この範囲は他の範囲にないほとんど全ての Unicode ブロックで構成されている．したがって，ブロックのリストを示す代わりに，範囲の定義そのものを示す：

```
\ltjdefcharrange{4}{%
  "500-"10FF, "1200-"1DFF, "2440-"245F, "27C0-"28FF, "2A00-"2AFF,
```

表 1. 文字範囲 3 に指定されている Unicode ブロック.

U+2000–U+206F	一般句読点	U+2070–U+209F	上付き・下付き
U+20A0–U+20CF	通貨記号	U+20D0–U+20FF	記号用ダイアクリティカルマーク（合成可能）
U+2100–U+214F	文字様記号	U+2150–U+218F	数字に準じるもの
U+2190–U+21FF	矢印	U+2200–U+22FF	数学記号（演算子）
U+2300–U+23FF	その他の技術用記号	U+2400–U+243F	制御機能用記号
U+2500–U+257F	罫線素片	U+2580–U+259F	ブロック要素
U+25A0–U+25FF	幾何学模様	U+2600–U+26FF	その他の記号
U+2700–U+27BF	装飾記号	U+2900–U+297F	補助矢印 B
U+2980–U+29FF	その他の数学記号 B	U+2B00–U+2BFF	その他の記号及び矢印
U+E000–U+F8FF	私用領域（外字領域）		

表 2. 文字範囲 6 に指定されている Unicode ブロック.

U+2460–U+24FF	囲み英数字	U+2E80–U+2EFF	CJK 部首補助
U+3000–U+303F	CJK の記号及び句読点	U+3040–U+309F	平仮名
U+30A0–U+30FF	片仮名	U+3190–U+319F	漢文用記号（返り点）
U+31F0–U+31FF	片仮名拡張	U+3200–U+32FF	囲み CJK 文字・月
U+3300–U+33FF	CJK 互換用文字	U+3400–U+4DBF	CJK 統合漢字拡張 A
U+4E00–U+9FFF	CJK 統合漢字	U+F900–U+FAFF	CJK 互換漢字
U+FE10–U+FE1F	縦書き形	U+FE30–U+FE4F	CJK 互換形
U+FE50–U+FE6F	小字形	U+20000–U+2FFFF	（追加多言語面）

表 3. 文字範囲 7 に指定されている Unicode ブロック.

U+1100–U+11FF	ハングル字母	U+2F00–U+2FDF	康熙部首
U+2FF0–U+2FFF	漢字構成記述文字	U+3100–U+312F	注音字母 （注音符号）
U+3130–U+318F	ハングル互換字母	U+31A0–U+31BF	注音字母拡張
U+31C0–U+31EF	CJK の筆画	U+A000–U+A48F	イ文字
U+A490–U+A4CF	イ文字部首	U+A830–U+A83F	共通インド数字に準じるもの
U+AC00–U+D7AF	ハングル音節文字	U+D7B0–U+D7FF	ハングル字母拡張 B

```
"2C00-"2E7F, "4DC0-"4DFF, "A4D0-"A82F, "A840-"ABFF, "FB50-"FE0F,
"FE20-"FE2F, "FE70-"FEFF, "FB00-"FB4F, "10000-"1FFFF} % non-Japanese
```

範囲 5^{A}　代用符号と補助私用領域.

範囲 6^{J}　日本語で用いられる文字．ブロックのリストは表 2 に示す.

範囲 7^{J}　CJK 言語で用いられる文字のうち，Adobe-Japan1-6 に含まれていないもの．ブロックのリストは表 3 に示す.

3.2 kanjiskip と xkanjiskip

JAglue は以下の 3 つのカテゴリに分類される：

- JFM で指定されたグルー／カーン．もし \inhibitglue が和文文字の周りで発行されていれば，このグルーは挿入されない.
- デフォルトで 2 つの **JAchar** の間に挿入されるグルー（kanjiskip）.
- デフォルトで **JAchar** と **ALchar** の間に挿入されるグルー（xkanjiskip）.

kanjiskip や xkanjiskip の値は以下のようにして変更可能である.

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

JFM は「望ましい kanjiskip の値」や「望ましい xkanjiskip の値」を持っていることがある．これらのデータを使うためには，kanjiskip や xkanjiskip の値を\maxdimen の値に設定すればよい．

3.3 xkanjiskip の設定の挿入

xkanjiskip がすべての **JAchar** と **ALchar** の境界に挿入されるのは望ましいことではない．例えば，xkanjiskip は開き括弧の後には挿入されるべきではない（‘(あ ’ と ‘(あ ’ を比べてみよ）．

LuaTEX-ja では xkanjiskip をある文字の前／後に挿入するかどうかを，**JAchar** に対しては jaxspmode を，**ALchar** に対しては alxspmode をそれぞれ変えることで制御することができる．

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

p あq！う

2 つ目のの引数の preonly は「xkanjiskip の挿入はこの文字の前でのみ許され，後では許さない」ことを意味する．他に指定可能な値は postonly, allow, inhibit である．pTEX との互換性のために，0 から 3 の自然数を 2 つ目の引数として指定することも可能である[*1]．

もし全ての kanjiskip と xkanjiskip の挿入を有効化／無効化したければ，それぞれ autospacing と autoxspacing を true/false に設定すればよい．

3.4 ベースラインの移動

和文フォントと欧文フォントを合わせるためには，時々どちらかのベースラインの移動が必要になる．pTEX ではこれは \ybaselineshift を非零の長さに設定することでなされていた（欧文フォントのベースラインが下がる）．しかし，日本語が主ではない文書に対しては，欧文フォントではなく和文フォントのベースラインを移動した方がよい．このため，LuaTEX-ja では欧文フォントのベースラインのシフト量（yalbaselineshift パラメータ）と和文フォントのベースラインのシフト量（yjabaselineshift パラメータ）を独立に設定できるようになっている．

```
1 \vrule width 150pt height 0.4pt depth 0pt\
    hskip-120pt
2 \ltjsetparameter{yjabaselineshift=0pt,
    yalbaselineshift=0pt}abc あいう
3 \ltjsetparameter{yjabaselineshift=5pt,
    yalbaselineshift=2pt}abc あいう
```

abc あいう abc あいう

上の例において引かれている水平線がベースラインである．

この機能には面白い使い方がある：2 つのパラメータを適切に設定することで，サイズの異なる文字を中心線に揃えることができるのだ．以下は一つの例である（値はあまり調整されていないことに注意）：

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

xyz 漢字 XYZ ひらがな abc かな

[*1] しかし，これは推奨されない：なぜならば 1 と 2 は jaxspmode と alxspmode で逆の意味になるから．

3.5 トンボ

トンボは用紙の四つ角と水平／垂直方向の中心を表す印である．pLaTeX と LuaTeX-ja ではトンボの出力をサポートしている．トンボを出力するためには以下の手順が必要である：

1. まず，用紙の左上に印刷されるバナーを定義する．これは `\@bannertoken` にトークンリストを与えることでなされる．
   例えば，以下はバナーとして ‘`filename (YYYY-MM-DD hh:mm)`’ を設定する:

```
\makeatletter

\hour\time \divide\hour by 60 \@tempcnta\hour \multiply\@tempcnta 60\relax
\minute\time \advance\minute-\@tempcnta
\@bannertoken{%
   \jobname\space(\number\year-\two@digits\month-\two@digits\day
   \space\two@digits\hour:\two@digits\minute)}%
```

2. ...

第 II 部
リファレンス

4 フォントメトリックと日本語フォント

4.1 `\jfont` プリミティブ

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

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

當／體／醫／區

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

Prefix `psft` Besides `file:` and `name:` prefixes, `psft:` can be used a prefix in `\jfont` (and `\font`) primitive. Using this prefix, you can specify a ‘name-only’ Japanese font which will be not embedded to PDF. Typical use of this prefix is to specify the ‘standard’ Japanese fonts, namely, ‘Ryumin-Light’ and ‘GothicBBB-Medium’. For kerning or other information, that of Kozuka Mincho Pr6N Regular (this is a font by Adobe Inc., and included in Japanese Font Packs for Adore Reader) will be used.

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

表 4. Differences between JFMs shipped with LuaTEX-ja

	jfm-ujis.lua	jfm-jis.lua	jfm-min.lua
Example 1	◆◆◆◆◆◆◆ ある日モモちゃんがお使いで迷子になって泣きました.	◆◆◆◆◆◆◆ ある日モモちゃんがお使いで迷子になって泣きました.	◆◆◆◆◆◆◆ ある日モモちゃんがお使いで迷子になって泣きました.
Example 2	ちょっと！ 何	ちょっと！何	ちょっと！何
Bounding Box	漢	漢	漢

`jfm=`⟨*name*⟩ Specify the name of JFM. A file named `jfm-`⟨*name*⟩`.lua` will be searched and/or loaded.

The followings are JFMs 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, please use this JFM.

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

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

`jfmvar=`⟨*string*⟩ Sometimes there is a need that

Note: kern feature Some fonts have information for inter-glyph spacing. However, this information is not well-compatible with LuaTEX-ja. More concretely, this kerning space from this information are inserted *before* the insertion process of **JAglue**, and this causes incorrect spacing between two characters when both a glue/kern from the data in the font and it from JFM are present.

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

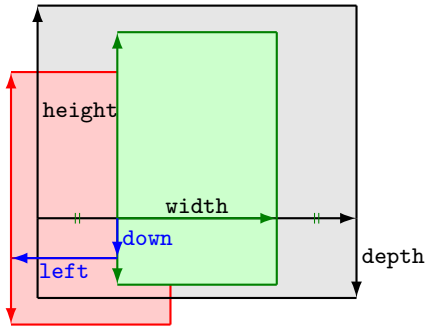
4.2 JFM ファイルの構造

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

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

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

`dir=`⟨*direction*⟩ (required)



Consider a node containing Japanese character whose value of the `align` field is `'middle'`.

- The black rectangle is a frame of the node. Its width, height and depth are specified by JFM.
- Since the `align` field is `'middle'`, the 'real' glyph is centered horizontally (the green rectangle).
- Furthermore, the glyph is shifted according to values of fields `left` and `down`. The ultimate position of the real glyph is indicated by the red rectangle.

図 1. The position of the 'real' glyph.

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

`zw=`⟨*length*⟩ (required)

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

`zh=`⟨*length*⟩ (required)

`kanjiskip={`⟨*natural*⟩`,` ⟨*stretch*⟩`,` ⟨*shrink*⟩`}` (optional)

This field specifies the 'ideal' amount of kanjiskip. As noted in Subsection 3.2, 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.

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

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

This field is a list of characters which are in this character type i. This field is not required if $i = 0$, since all **JAchar** which are not in any character class other than 0 (hence, the character class 0 contains most of **JAchar**s). In the list, a character can be specified by its code number, or by the character itself (as a string of length 1). Moreover, there are 'imaginary characters' which specified in the list. We will describe these later.

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

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

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

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

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

`kern={[`j`]=`⟨*kern*⟩`, ...}`

glue={[*j*]={⟨*width*⟩, ⟨*stretch*⟩, ⟨*shrink*⟩}, ...}

上で説明した通り，chars フィールド中にはいくつかの「特殊文字」も指定可能である．これらは，大半が pTEX の JFM グルーの挿入処理ではみな「文字クラス 0 の文字」として扱われていた文字であり，その結果として pTEX より細かい組版調整ができるようになっている．以下のその一覧を述べる：

'lineend' 行の終端を表す．
'diffmet'
'boxbdd' hbox の先頭と末尾，及びインデントされていない（\noindent で開始された）段落の先頭を表す．
'parbdd' 通常の（\noindent で開始されていない）段落の先頭．
'jcharbdd' 和文文字と「その他のもの」（欧文文字，glue，kern 等）との境界．
−1　行中数式と地の文との境界．

pTEX 用和文フォントメトリックの移植　以下に，pTEX 用和文フォントメトリックを LuaTEX-ja 用に移植する場合の注意点を挙げておく．

- 実際に出力される和文フォントのサイズが design size となる．このため，例えば 1 zw が design size の 0.962216 倍である JIS フォントメトリック等を移植する場合は，
  - JFM 中の全ての数値を 1/0.962216 倍しておく．
  - TEX ソース中で使用するところで，サイズ指定を 0.962216 倍にする．LATEX でのフォント宣言なら，例えば次のように：
    ```
    \DeclareFontShape{JY3}{mc}{m}{n}{<-> s*[0.962216] psft:Ryumin-Light:jfm=jis}{}
    ```
- 上に述べた特殊文字は，'boxbdd' を除き文字クラスを全部 0 とする（JFM 中に単に書かなければよい）．
- 'boxbdd' については，それのみで一つの文字クラスを形成し，その文字クラスに関しては glue/kern の設定はしない．
  これは，pTEX では，hbox の先頭・末尾とインデントされていない（\noindent で開始された）段落の先頭には JFM グルーは入らないという仕様を実現させるためである．
- pTEX の組版を再現させようというのが目的であれば以上の注意を守れば十分である．
  ところで，pTEX では通常の段落の先頭に JFM グルーが残るという仕様があるので，段落先頭の開き括弧は全角二分下がりになる．全角下がりを実現させるには，段落の最初に手動で\inhibitglue を追加するか，あるいは\everypar の hack を行い，それを自動化させるしかなかった．
  一方，LuaTEX-ja では，'parbdd' によって，それが JFM 側で調整できるようになった．例えば，LuaTEX-ja 同梱の JFM のように，'boxbdd' と同じ文字クラスに'parbdd' を入れれば全角下がりとなる．

```
1 \jfont\g=psft:Ryumin-Light:jfm=test \g
2 \parindent1\zw\noindent{}◆◆◆◆◆
3 \par{}「◆◆←二分下がり
4 \par{}【◆◆←全角下がり
5 \par{}〔◆◆←全角二分下がり
```

◆◆◆◆◆
「◆◆←二分下がり
【◆◆←全角下がり
〔◆◆←全角二分下がり

4.3　数式フォントファミリ

TEX handles fonts in math formulas by 16 font families[*2], and each family has three fonts: \textfont, \scriptfont and \scriptscriptfont.

LuaTEX-ja's handling of Japanese fonts in math formulas is similar; Table 5 shows counterparts to TEX's

[*2] Omega, Aleph, LuaTEX and ε-(u)pTEXcan handles 256 families, but an external package is needed to support this in plain TEX and LATEX.

表 5. Primitives for Japanese math fonts.

	Japanese fonts	alphabetic fonts
font family	`\jfam` $\in [0, 256)$	`\fam`
text size	jatextfont={⟨*jfam*⟩,⟨*jfont_cs*⟩}	`\textfont`⟨*fam*⟩=⟨*font_cs*⟩
script size	jascriptfont={⟨*jfam*⟩,⟨*jfont_cs*⟩}	`\scriptfont`⟨*fam*⟩=⟨*font_cs*⟩
scriptscript size	jascriptscriptfont={⟨*jfam*⟩,⟨*jfont_cs*⟩}	`\scriptscriptfont`⟨*fam*⟩=⟨*font_cs*⟩

primitives for math font families. There is no relation between the value of `\fam` and that of `\jfam`; with appropriate settings, you can set both `\fam` and `\jfam` to the same value.

4.4 コールバック

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

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

```
function (<table> 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 `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`. This callback doesn't replace any code of LuaTEX-ja.

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

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

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

`jfm` The index number of JFM.

`size` Font size in a scaled point ($= 2^{-16}$ pt).

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

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

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

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

```
1 function (<number> char_class, <table> jfont_info, <number> chr_code)
```

```
  if char_class~=0 then return char_class
  else
    ....
    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. This callback doesn't replace any code of LuaTEX-ja.

`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, <number> char_class)
  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.

良い例が test/valign.lua である．このファイルが読み込まれた状態では，JFM 内で規定された文字クラス 0 の文字における (高さ) : (深さ) の比になるように，実際のフォントの出力上下位置が自動調整される．例えば，

- JFM 側の設定：(高さ) $= 88x$, (深さ) $= 12x$（和文 OpenType フォントの標準値）
- 実フォント側の数値：(高さ) $= 28y$, (深さ) $= 5y$（和文 TrueType フォントの標準値）

となっていたとする．すると，実際の文字の出力位置は，

$$\frac{88x}{88x+12x}(28y+5y)-28y=\frac{26}{825}y=0.031\dot{5}y$$

だけ上にずらされることになる，

5 パラメータ

5.1 \ltjsetparameter プリミティブ

As noted before, `\ltjsetparameter` and `\ltjgetparameter` are primitives for accessing most parameters of LuaTEX-ja. One of the main reason that LuaTEX-ja didn't adopted the syntax similar to that of pTEX (*e.g.*, `\prebreakpenalty`) =10000`) is the position of `hpack_filter` callback in the source of LuaTEX, see Section 9.

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

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

```
\ltjgetparameter{differentjfm},
\ltjgetparameter{autospacing},
\ltjgetparameter{prebreakpenalty}{`) }.
```

average, 1, 10000.

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

5.2 パラメーター一覧

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

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

jcharwidowpenalty =⟨*penalty*⟩ [`\jcharwidowpenalty`]

Penalty value for suppressing orphans. This penalty is inserted just after the last **JAchar** which is not regarded as a (Japanese) punctuation mark.

kcatcode ={⟨*chr_code*⟩,⟨*natural number*⟩}

An additional attributes having each character whose character code is ⟨*chr_code*⟩. At the present version, the lowermost bit of ⟨*natural number*⟩ indicates whether the character is considered as a punctuation mark (see the description of jcharwidowpenalty above).

prebreakpenalty ={⟨*chr_code*⟩,⟨*penalty*⟩} [`\prebreakpenalty`] 文字コード ⟨*chr_code*⟩ の **JAchar** が行頭にくることを抑止するために，この文字の前に挿入/追加されるペナルティの量を指定する．

例えば閉じ括弧「〗」は絶対に行頭にきてはならないので，標準で読み込まれる `luatexja-kinsoku.tex` において

```
\ltjsetparameter{prebreakpenalty={`〗,10000}}
```

と，最大値の 10000 が指定されている．他にも，小書きのカナなど，絶対禁止というわけではないができれば行頭にはきて欲しくない場合に，0 と 10000 の間の値を指定するのも有用であろう．

```
\ltjsetparameter{prebreakpenalty={`か,150}}
```

postbreakpenalty ={⟨*chr_code*⟩,⟨*penalty*⟩} [`\postbreakpenalty`] 文字コード ⟨*chr_code*⟩ の **JAchar** が行末にくることを抑止するために，この文字の後に挿入/追加されるペナルティの量を指定する．

pTEX では，`\prebreakpenalty`, `\postbreakpenalty` において，

- 一つの文字に対して，pre, post どちらか一つしか指定することができなかった（後から指定した方で上書きされる）．
- pre, post 合わせて 256 文字分の情報を格納することしかできなかった．

という制限があったが，LuaTEX-ja ではこれらの制限は解消されている．

jatextfont ={⟨*jfam*⟩,⟨*jfont_cs*⟩} [`\textfont` in TEX]

jascriptfont ={⟨*jfam*⟩,⟨*jfont_cs*⟩} [`\scriptfont` in TEX]

jascriptscriptfont ={⟨*jfam*⟩,⟨*jfont_cs*⟩} [`\scriptscriptfont` in TEX]

yjabaselineshift =⟨*dimen*⟩∗

yalbaselineshift =⟨*dimen*⟩∗ [`\ybaselineshift`]

jaxspmode ={⟨*chr_code*⟩,⟨*mode*⟩} [`\inhibitxspcode`]

Setting whether inserting xkanjiskip is allowed before/after a **JAchar** whose character code is ⟨*chr_code*⟩. The followings are allowed for ⟨*mode*⟩:

0, `inhibit` Insertion of xkanjiskip is inhibited before the character, nor after the character.

2, `preonly` Insertion of xkanjiskip is allowed before the character, but not after.

1, `postonly` Insertion of xkanjiskip is allowed after the character, but not before.

3, `allow` Insertion of xkanjiskip is allowed before the character and after the character. This is the default value.

alxspmode={⟨*chr_code*⟩,⟨*mode*⟩} [\xspcode]

Setting whether inserting xkanjiskip is allowed before/after a **ALchar** whose character code is ⟨*chr_code*⟩. The followings are allowed for ⟨*mode*⟩:

0, `inhibit` Insertion of xkanjiskip is inhibited before the character, nor after the character.

1, `preonly` Insertion of xkanjiskip is allowed before the character, but not after.

2, `postonly` Insertion of xkanjiskip is allowed after the character, but not before.

3, `allow` Insertion of xkanjiskip is allowed both before the character and after the character. This is the default value.

Note that parameters jaxspmode and alxspmode use a common table.

autospacing=⟨*bool*⟩* [\autospacing]

autoxspacing=⟨*bool*⟩* [\autoxspacing]

kanjiskip=⟨*skip*⟩ [\kanjiskip]

xkanjiskip=⟨*skip*⟩ [\xkanjiskip]

differentjfm=⟨*mode*⟩† Specify how glues/kerns between two **JAchar**s whose JFM (or size) are different. The allowed arguments are the followings:

`average`

`both`

`large`

`small`

jacharrange=⟨*ranges*⟩*

kansujichar={⟨*digit*⟩, ⟨*chr_code*⟩} [\kansujichar]

6 その他のプリミティブ

6.1 互換プリミティブ

The following primitives are implemented for compatibility with pTEX:

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

6.2 \inhibitglue プリミティブ

The primitive `\inhibitglue` suppresses the insertion of **JAglue**. 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 ’.

```
1 \jfont\g=psft:Ryumin-Light:jfm=test \g
2 あウあ\inhibitglue{}ウ\inhibitglue\par
3 あ\par\inhibitglue{}あ
4 \par\inhibitglue\hrule{}あoff\inhibitglue ice
```

あ　ウあウ

あ

あ

あ office

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

- The call of `\inhibitglue` in the (internal) vertical mode is effective at the beginning of the next paragraph. This is realized by hacking `\everypar`.
- 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 line 4 of above example.
- The call of `\inhibitglue` in math mode is just ignored.

7 LaTeX 2ε 用のコントロールシーケンス

7.1 NFSS2 へのパッチ

As described in Subsection 2.4, LuaTeX-ja simply adopted `plfonts.dtx` in pLaTeX 2ε for the Japanese patch for NFSS2. For an convenience, we will describe commands which are not described in Subsection 2.5.

`\DeclareYokoKanjiEncoding{`⟨*encoding*⟩`}{`⟨*text-settings*⟩`}{`⟨*math-settings*⟩`}`

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

`\DeclareKanjiEncodingDefaults{`⟨*text-settings*⟩`}{`⟨*math-settings*⟩`}`

`\DeclareKanjiSubstitution{`⟨*encoding*⟩`}{`⟨*family*⟩`}{`⟨*series*⟩`}{`⟨*shape*⟩`}`

`\DeclareErrorKanjiFont{`⟨*encoding*⟩`}{`⟨*family*⟩`}{`⟨*series*⟩`}{`⟨*shape*⟩`}{`⟨*size*⟩`}`

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

`\reDeclareMathAlphabet{`⟨*unified-cmd*⟩`}{`⟨*al-cmd*⟩`}{`⟨*ja-cmd*⟩`}`

和文・欧文の数式用フォントファミリを一度に変更する命令を作成する．具体的には，欧文数式用フォントファミリ変更の命令 ⟨*al-cmd*⟩ と，和文数式用フォントファミリ変更の命令 ⟨*ja-cmd*⟩ の 2 つを同時に行う命令として ⟨*unified-cmd*⟩ を（再）定義する．実際の使用では ⟨*unified-cmd*⟩ と ⟨*al-cmd*⟩ に同じものを指定する，すなわち，⟨*al-cmd*⟩ に和文側も変更させるようにするのが一般的と思われる．

本コマンドの使用については，pLaTeX 配布中の `plfonts.dtx` に詳しく注意点が述べられているので，そちらを参照されたい．

`\DeclareRelationFont{`⟨*ja-encoding*⟩`}{`⟨*ja-family*⟩`}{`⟨*ja-series*⟩`}{`⟨*ja-shape*⟩`}`
`{`⟨*al-encoding*⟩`}{`⟨*al-family*⟩`}{`⟨*al-series*⟩`}{`⟨*al-shape*⟩`}`

いわゆる「従属欧文」を設定するための命令である．前半の 4 引数で表される和文フォントファミリに対して，そのフォントに対応する「従属欧文」フォントファミリを後半の 4 引数により与える．

`\SetRelationFont`

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

`\userelfont`

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

`\adjustbaseline`

...

`\fontfamily{`⟨*family*⟩`}`

As in LaTeX 2ε, this command changes current font family (alphabetic, Japanese, *or both*) to ⟨*family*⟩. Which family will be changed is determined as follows:

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

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

```
\gtfamily{}あいうabc
\SetRelationFont{JY3}{gt}{m}{n}{OT1}{pag}{m}{
    n}
\userelfont\selectfont{}あいうabc
```

あいう abc あいう abc

7.2 トンボ

8 拡張

8.1 luatexja-fontspec.sty

8.2 luatexja-otf.sty

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

`\CID{`⟨*number*⟩`}` Typeset a character whose CID number is ⟨*number*⟩.

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

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

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

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

第 III 部
実装

9　パラメータの保持

9.1　用いられる寸法レジスタ，属性レジスタ，whatsit ノード

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

`\jQ` (dimension) As explained in Subsection 2.3, `\jQ` is equal to 1 Q = 0.25 mm, where 'Q' (also called '級') is a unit used in Japanese phototypesetting. So one should not change the value of this dimension.

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

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

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

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

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

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

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

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

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

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

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

- *italic* (1)　Glues from an italic correction (`\/`). This distinction of origins of glues (from explicit `\kern`, or from `\/`) is needed in the insertion process of xkanjiskip.
- *packed* (2)
- *kinsoku* (3)　Penalties inserted for the word-wrapping process of Japanese characters (*kinsoku*).
- *from_jfm* (4)　Glues/kerns from JFM.
- *line_end* (5)　Kerns for ...
- *kanji_skip* (6)　Glues for kanjiskip.
- *xkanji_skip* (7)　Glues for xkanjiskip.
- *processed* (8)　Nodes which is already processed by ...
- *ic_processed* (9)　Glues from an italic correction, but also already processed.
- *boxbdd* (15)　Glues/kerns that inserted just the beginning or the ending of an hbox or a paragraph.

`\ltj@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 **JAchar**s.

Furthermore, LuaTEX-ja uses several 'user-defined' whatsit nodes for typesetting. All those nodes store a natural number (hence the node's `type` is 100).

30111　Nodes for indicating that `\inhibitglue` is specified. The `value` field of these nodes doesn't matter.

30112　Nodes for LuaTEX-ja's stack system (see the next subsection). The `value` field of these nodes is current group.

30113　Nodes for Japanese Characters which the callback process of luaotfload won't be applied, and the

character code is stored in the `value` field. Each node having this `user_id` is converted to a 'glyph_node' *after* the callback process of luaotfload.

These whatsits will be removed during the process of inserting **JAglue**s.

9.2 LuaTEX-ja のスタックシステム

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:

```
\ltjsetparameter{kanjiskip=0pt}ふがふが.%
\setbox0=\hbox{\ltjsetparameter{kanjiskip=5pt
    }ほげほげ}
\box0.ぴよぴよ\par
```

ふがふが. ほ げ ほ げ. ぴよぴよ

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

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

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

The method The code of stack system is based on that in a post of Dev-luatex mailing list[*3].

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

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

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

- If there is no *stack_flag* node in the list of hbox, then no assignment was occurred inside the hbox. Hence values of parameters at the end of the hbox are stored in the stack level s.
- If there is a *stack_flag* node whose value is $t+1$, then an assignment was occurred just inside the hbox group. Hence values of parameters at the end of the hbox are stored in the stack level $s+1$.
- If there are *stack_flag* nodes but all of their values are more than $t+1$, then an assignment was occurred in the box, but it is done is 'more internal' group. Hence values of parameters at the end of the hbox are stored in the stack level s.

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

10 和文文字直後の改行

10.1 参考: pTEX の挙動

欧文では文章の改行は単語間でしか行わない．そのため，TEX では，（文字の直後の）改行は空白文字と同じ扱いとして扱われる．一方，和文ではほとんどどこでも改行が可能なため，pTEX では和文文字の直後の改行は単純に無視されるようになっている．

このような動作は，pTEX が TEX からエンジンとして拡張されたことによって可能になったことである．pTEX の入力処理部は，TEX におけるそれと同じように，有限オートマトンとして記述することができ，以下に述べるような 4 状態を持っている．

- State N: 行の開始．
- State S: 空白読み飛ばし．
- State M: 行中．
- State K: 行中（和文文字の後）．

また，状態遷移は，図 2 のようになっており，図中の数字はカテゴリーコードを表している．最初の 3 状態は TEX の入力処理部と同じであり，図中から状態 K と「j」と書かれた矢印を取り除けば，TEX の入力処理部と同じものになる．

この図から分かることは，

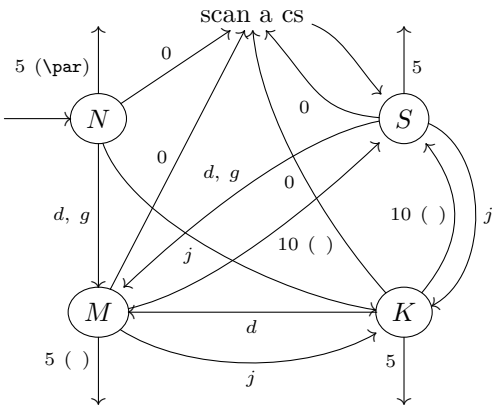
> 行が和文文字（とグループ境界文字）で終わっていれば，改行は無視される

ということである．

10.2 LuaTEX-ja の挙動

LuaTEX の入力処理部は TEX のそれと全く同じであり，callback によりユーザがカスタマイズすることはできない．このため，改行抑制の目的でユーザが利用できそうな callback としては，`process_input_buffer` や `token_filter` に限られてしまう．しかし，TEX の入力処理部をよく見ると，後者も役には経たないことが分かる：改行文字は，入力処理部によってトークン化される時に，カテゴリーコード 10 の 32 番文字へと置き換えられてしまうため，`token_filter` で非標準なトークン読み出しを行おうとしても，空白文字由来のトークンと，改行文字由来のトークンは区別できないのだ．

すると，我々のとれる道は，`process_input_buffer` を用いて LuaTEX の入力処理部に引き渡される前に入力文字列を編集するというものしかない．以上を踏まえ，LuaTEX-ja における「和文文字直後の改行抑制」の処理は，次のようになっている：



$d := \{3, 4, 6, 7, 8, 11, 12, 13\}, \quad g := \{1, 2\}, \quad j :=$ (Japanese characters)

- Numbers represent category codes.
- Category codes 9 (ignored), 14 (comment) and 15 (invalid) are omitted in above diagram.

図 2. State transitions of pTEX's input processor.

各入力行に対し，その入力行が読まれる前の内部状態で以下の 2 条件が満たされている場合，LuaTEX-ja は U+FFFFF 番の文字[*4]を末尾に追加する．よって，その場合に改行は空白とは見做されないこととなる．

1. 改行文字（文字コード 13 番）のカテゴリーコードが 5 (end-of-line) である．
2. 入力行は次の「正規表現」にマッチしている：

$$(\text{any char})^*(\textbf{JAchar})\big(\{\text{catcode} = 1\} \cup \{\text{catcode} = 2\}\big)^*$$

この仕様は，前節で述べた pTEX の仕様にできるだけ近づけたものとなっている．最初の条件は，verbatim 系環境などの日本語対応マクロを書かなくてすませるためのものである．しかしながら，完全に同じ挙動が実現できたわけではない．差異は，次の例が示すように，和文文字の範囲を変更した行の改行において見られる：

```
\ltjsetparameter{autoxspacing=false}
\ltjsetparameter{jacharrange={-6}}x あ
y\ltjsetparameter{jacharrange={+6}}z あ
u
```

xyzあ u

もし pTEX とまったく同じ挙動を示すならば，出力は「x yzあu」となるべきである．しかし，実際には上のように異なる挙動となっている．

- 2 行目は「あ」という和文文字で終わる（2 行目を処理する前の時点では，「あ」は和文文字扱いである）ため，直後の改行文字は無視される．
- 3 行目は「あ」という欧文文字で終わる（2 行目を処理する前の時点では，「あ」は欧文文字扱いである）ため，直後の改行文字は空白に置き換わる．

このため，トラブルを避けるために，和文文字の範囲を\ltjsetparameter で編集した場合，その行はそこで改行するようにした方がいいだろう．

[*4] この文字はコメント文字として扱われるように LuaTEX-ja 内部で設定をしている．

11　JFM グルーの挿入，kanjiskip と xkanjiskip

11.1　概要

LuaTEX-ja における和文処理グルーの挿入方法は，pTEX のそれとは全く異なる．pTEX では次のような仕様であった：

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

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

LuaTEX-ja における **JAglue** 挿入処理では，下の図**??**のように「塊」を単位にして行われる．大雑把にいうと，「塊」は文字とそれに付随するノード達（アクセント位置補正用の kern や，イタリック補正）をまとめたものであり，2 つの塊の間には，ペナルティ，`\vadjust`，whatsit など，行組版には関係しないものがある．そのため，……

11.2　「クラスタ」の定義

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

1. Nodes whose value of `\ltj@icflag` is in $[3, 15)$. These nodes come from a hbox which is already packaged, by unpackaging (`\unhbox`). The *id* is *id_pbox*.
2. A inline math formula, including two *math_node*s at the boundary of it: HOGE The *id* is *id_math*.
3. A *glyph_node* with nodes which relate with it: HOGE The *id* is *id_jglyph* or *id_glyph*, according to whether the *glyph_node* represents a Japanese character or not.
4. An box-like node, that is, an hbox, an vbox and an rule (`\vrule`). The *id* is *id_hlist* if the node is an hbox which is not shifted vertically, or *id_box_like* otherwise.
5. A glue, a kern whose subtype is not 2 (*accent*), and a discretionary break. The *id* is *id_glue*, *id_kern* and *id_disc*, respectively.

We denote a cluster by *Np*, *Nq* and *Nr*.

Internally, a cluster is represented by a table *Np* with the following fields.

first, last The first/last node of the cluster.
id The *id* in above definition.
nuc
auo_kspc, auto_xspc
xspc_before, xspc_after
pre, post
char
class
lend

met, var