# pxrubrica パッケージ

八登 崇之　(Takayuki YATO; aka “ZR”)

v1.3　[2017/04/27]

## 目次

# 1 パッケージ読込

\usepackage 命令を用いて読み込む。オプションは存在しない。

```
\usepackage{pxrubrica}
```

# 2 ルビ機能

## 2.1 用語集

本パッケージで独自の意味をもつ単語を挙げる。

- 突出： ルビ文字出力の端が親文字よりも外側になること。
- 進入： ルビ文字出力が親文字に隣接する文字の（水平）領域に配置されること。
- 和文ルビ： 親文字が和文文字であることを想定して処理されるルビ。
- 欧文ルビ： 親文字が欧文文字であることを想定して処理されるルビ。
- グループ： ユーザにより指定された、親文字列・ルビ文字列の処理単位。
- 《文字》: 均等割りにおいて不可分となる単位のこと。通常は、本来の意味での文字となるが、ユーザ指定で変更できる。
- ブロック： 複数の親文字・ルビ文字の集まりで、大域的な配置決定の処理の中で内部の相対位置が固定されているもの。

次の用語については、『日本語組版の要件』に従う。

ルビ、親文字、中付き、肩付き、モノルビ、グループルビ、熟語ルビ

## 2.2 ルビ用命令

- \ruby[⟨オプション⟩]{⟨親文字⟩}{⟨ルビ文字⟩}
  和文ルビの命令。すなわち、和文文字列の上側（横組）／右側（縦組）にルビを付す（オプションで逆側にもできる）。
  ここで、⟨オプション⟩ は以下の形式をもつ。

⟨前進入設定⟩⟨前補助設定⟩⟨モード⟩⟨後補助設定⟩⟨後進入設定⟩

⟨前補助設定⟩・⟨モード⟩・⟨後補助設定⟩ は複数指定可能で、排他な指定が併存した場合は後のものが有効になる。また、どの要素も省略可能で、その場合は `\rubysetup` で指定された既定値が用いられる。ただし、構文上曖昧な指定を行った場合の結果は保証されない。例えば、「前進入無し」のみ指定する場合は `|` ではなく `|-` とする必要がある。

⟨前進入設定⟩ は以下の値の何れか。

| | | | |
|---|---|---|---|
| `||` | 前突出禁止 | `<` | 前進入大 |
| `|` | 前進入無し | `(` | 前進入小 |

⟨前補助設定⟩ は以下の値の何れか。

| | | | |
|---|---|---|---|
| `:` | 和欧文間空白挿入 | `*` | 行分割禁止 |
| `.` | 空白挿入なし | `!` | 段落頭で進入許可 |

– 空白挿入量の既定値は和文間空白である。
– `*` 無指定の場合の行分割の可否は pLaTeX の標準の動作に従う。
– `!` 無指定の場合、段落冒頭では ⟨前進入設定⟩ の設定に関わらず進入が抑止される。
– ゴースト処理が有効の場合はここの設定は無視される。

⟨モード⟩ は以下の値の何れか。

| | | | |
|---|---|---|---|
| `-` | （無指定） | `P` (< *primary*) | 上側配置 |
| `c` (< *center*) | 中付き | `S` (< *secondary*) | 下側配置 |
| `h` (< *head*) | 肩付き | `e` (< *even-space*) | 親文字均等割り有効 |
| `H` | 拡張肩付き | `E` | 親文字均等割り無効 |
| `m` (< *mono*) | モノルビ | `f` (< *full-size*) | 小書き文字変換有効 |
| `g` (< *group*) | グループルビ | `F` | 小書き文字変換無効 |
| `j` (< *jukugo*) | 熟語ルビ | | |
| `M` | 自動切替モノルビ | | |
| `G` | 自動切替グループルビ | | |

– 肩付き（`h`）の場合、ルビが短い場合にのみ、ルビ文字列と親文字列の頭を揃えて配置される。拡張肩付き（`H`）の場合、常に頭を揃えて配置される。
– `P` は親文字列の上側（横組）／右側（縦組）、`S` は親文字列の下側（横組）／左側（縦組）にルビを付す指定。
– `e` 指定時は、ルビが長い場合に親文字列をルビの長さに合わせて均等割りで配置する。`E` 指定時は、空きを入れずに中央揃えで配置する。なお、ルビが短い場合のルビ文字列の均等割りは常に有効である。
– `f` 指定時は、ルビ文字列中の（`{ }` の外にある）小書き仮名（ぁ ぃ ぅ ぇ ぉ っ ゃ ゅ ょ ゎ、およびその片仮名）を対応の非小書き仮名に変換する。`F` 指定はこの機能を無効にする。
– `M` および `J` の指定は「グループルビとモノ・熟語ルビの間で自動的に切り替える」設定である。具体的には、ルビのグループが 1 つしかない場合は `m` および `g`、複数ある場合は `g` と等価になる。

⟨後補助設定⟩ は以下の値の何れか。

| | | | |
|---|---|---|---|
| : | 和欧文間空白挿入 | * | 行分割禁止 |
| . | 空白挿入なし | ! | 段落末で進入許可 |

– 空白挿入量の既定値は和文間空白である。
– * 無指定の場合の行分割の可否は pLaTeX の標準の動作に従うのが原則だが、直後にあるものが文字でない場合、正しく動作しない（禁則が破れる）可能性がある。従って、不適切な行分割が起こりうる場合は適宜 * を指定する必要がある（なお、段落末尾で * を指定してはならない）。
– ! 無指定の場合、段落末尾では進入が抑止される。
– ゴースト処理が有効の場合はここの設定は無視される。

⟨後進入設定⟩ は以下の値。

| | | | |
|---|---|---|---|
| \|\| | 後突出禁止 | > | 後進入大 |
| \| | 後進入無し | ) | 後進入小 |

- \jruby[⟨オプション⟩]{⟨親文字⟩}{⟨ルビ文字⟩}
  \ruby 命令の別名。\ruby という命令名は他のパッケージとの衝突の可能性が高いので、LaTeX 文書の本文開始時（\begin{document}）に未定義である場合にのみ定義される。これに対して \jruby は常に定義される。なお、\ruby 以外の命令（\jruby を含む）が定義済であった（命令名の衝突）場合にはエラーとなる。
- \aruby[⟨オプション⟩]{⟨親文字⟩}{⟨ルビ文字⟩}
  欧文ルビの命令。すなわち、欧文文字列の上側（横組）／右側（縦組）にルビを付す。欧文ルビは和文ルビと比べて以下の点が異なる。
  – 常にグループルビと扱われる。（m、g、j の指定は無効。）
  – 親文字列の均等割りは常に無効である。（e 指定は無効。）
  – ルビ付き文字と前後の文字との間の空き調整や行分割可否は両者がともに欧文であるという想定で行われる。従って、既定では空き調整量はゼロ、行分割は禁止となる。
  – 空き調整を和欧文間空白（:）にした場合は、* が指定されるあるいは自動の禁則処理が働くのでない限り、行分割が許可される。
- \truby[⟨オプション⟩]{⟨親文字⟩}{⟨上側ルビ文字⟩}{⟨下側ルビ文字⟩}
  和文両側ルビの命令。横組の場合、親文字列の上側と下側にルビを付す。縦組の場合、親文字列の右側と左側にルビを付す。
  両側ルビで熟語ルビを使うことはできない。すなわち、⟨オプション⟩ 中で j、J は指定できない。
  ※ 1.1 版以前では常にグループルビの扱いであった。旧版との互換のため、両側ルビの場合には自動切替モノルビ（M）を既定値とする。[*1]
- \atruby[⟨オプション⟩]{⟨親文字⟩}{⟨上側ルビ文字⟩}{⟨下側ルビ文字⟩}
  欧文両側ルビの命令。欧文ルビであることを除き \truby と同じ。

[*1] つまり、旧来の使用ではグループルビと扱われるため、ルビのグループは 1 つにしているはずで、これは新版でもそのままグループルビと扱われる。一方で、モノルビを使いたい場合はグループを複数にするはずで、この時は自動的にモノルビになる。なので結局、基底モード（g、m）を指定する必要は無いことになる。

## 2.3 入力文字列のグループの指定

入力文字列（親文字列[*2]・ルビ文字列）の中で「|」はグループの区切りとみなされる（ただし { } の中にあるものは文字とみなされる）。

例えば、ルビ文字列

じゆく|ご

は 2 つのグループからなり、最初のものは 3 文字、後のものは 1 文字からなる。

長さを合わせるために均等割りを行う場合、その分割の単位は通常は文字であるが、{ } で囲ったものは 1 文字とみなされる（本文書ではこの単位のことを《文字》と記す）。例えば

```
ベクタ{\<（ー）\<}
```

は 1 つのグループからなり、それは 4 つの《文字》からなる。

グループや《文字》の指定はルビの付き方に影響する。その詳細を説明する。なお、非拡張機能では親文字のグループは常に 1 つに限られる。

- モノルビ・熟語ルビでは親文字列の 1 つの《文字》にルビ文字列の 1 つのグループが対応する。例えば、
  ```
  \ruby[m]{熟語}{じゆく|ご}
  ```
  は、「熟 + じゆく」「語 + ご」の 2 つのブロックからなる。
- （単純）グループルビではルビ文字列のグループも 1 つに限られ、親文字とルビ文字の唯一のグループが対応する。例えば、
  ```
  \ruby[g]{五月雨}{さみだれ}
  ```
  は、「五月雨 + さみだれ」の 1 つのブロックからなる。

拡張機能では、親文字列が複数グループをもつような使用法が存在する予定である。

## 2.4 ゴースト処理

「和文ゴースト処理」とは以下のようなものである：

> 和文ルビの親文字列出力の前後に全角空白文字を挿入する（ただしその空きを打ち消すように負の空きを同時に入れる）ことで、親文字列全体が、その外側から見たときに、全角空白文字（大抵の JFM ではこれは漢字と同じ扱いになる）と同様に扱われるようにする。例えば、前に欧文文字がある場合には自動的に和欧文間空白が挿入される。

「欧文ゴースト処理」も対象が欧文であることと除いて同じである。（こちらは、「複合語記号（compound word mark）」というゼロ幅不可視の欧文文字を用いる。ルビ付文字列全体が単一欧文文字のように扱われる。）なお、「ゴースト（ghost）」というのは Omega の用

[*2] 後述の通り、現在の版では親文字列を複数グループにする使用法は存在しないため、親文字列中では「|」は使われない。

語で、「不可視であるが（何らかの性質において）特定の可視の文字と同等の役割をもつオブジェクト」のことである。

ゴースト処理を有効にすると次のようなメリットがある。

- 和欧文間空白が自動的に挿入される。
- 行分割禁止（禁則処理）が常に正しく機能する。
- 特殊な状況（例えば段落末）でも異常動作を起こしにくい。
- （実装が単純化され、バグ混入の余地が少なくなる。）

ただし、次のような重要なデメリットがある。

- pTEX エンジンの仕様上の制約により、ルビ出力の進入と共存できない。（従って共存するような設定を試みるとエラーになる。）

このため、既定ではゴースト処理は無効になっている。有効にするには、`\rubyusejghost`（和文）／`\rubyuseaghost`（欧文）を実行する。

なお、⟨前補助設定⟩／⟨後補助設定⟩ で指定される機能は、ゴースト処理が有効の場合には無効化される。これらの機能の目的が自動処理が失敗するのを捕逸するためだからである。

## 2.5 パラメタ設定命令

基本的設定。

- `\rubysetup{`⟨オプション⟩`}`
  オプションの既定値設定。［既定 = `|cjPeF|`］
  - これ自体の既定値は「突出許可、進入無し、中付き、熟語ルビ、上側配置、親文字均等割り有効、小書き文字変換無効」である。
  - ⟨前補助設定⟩／⟨後補助設定⟩ の既定値は変更できない。`\rubysetup` でこれらのオプション文字を指定しても無視される。
  - `\rubysetup` での設定は累積する。例えば、初期状態から、`\rubysetup{hmf}` と `\rubysetup{<->}` を実行した場合、既定値設定は `<hmPef>` となる。
  - この設定に関わらず、両側ルビでは「自動切替モノルビ（`M`）」が既定として指定される。
- `\rubyfontsetup{`⟨命令⟩`}`
  ルビ用のフォント切替命令を設定する。例えば、ルビは必ず明朝体で出力したいという場合は、以下の命令を実行すればよい。
  ```
  \rubyfontsetup{\mcfamily}
  ```
- `\rubybigintrusion{`⟨実数⟩`}`
  「大」の進入量（ルビ全角単位）。［既定 = 1］
- `\rubysmallintrusion{`⟨実数⟩`}`
  「小」の進入量（ルビ全角単位）。［既定 = 0.5］
- `\rubymaxmargin{`⟨実数⟩`}`
  ルビ文字列の方が短い場合の、ルビ文字列の端の親文字列の端からの距離の上限値

（親文字全角単位）。［既定 = 0.75］

- `\rubyintergap{`⟨実数⟩`}`
  ルビと親文字の間の空き（親文字全角単位）。［既定 = 0］
- `\rubyusejghost`／`\rubynousejghost`
  和文ゴースト処理を行う／行わない。［既定 = 行わない］
- `\rubyuseaghost`／`\rubynouseaghost`
  欧文ゴースト処理を行う／行わない。［既定 = 行わない］

詳細設定。通常はこれらの既定値を変える必要はないだろう。

- `\rubysafemode`／`\rubynosafemode`
  安全モードを有効／無効にする。［既定 = 無効］
  - 本パッケージがサポートするエンジンは (u)pTEX、XeTeX、LuaTEX である。「安全モード」とは、これらのエンジンを必要とする一部の機能[*3]を無効化したモードである。つまり、安全モードに切り替えることで、“サポート対象”でないエンジン（pdfTEX 等）でも本パッケージの一部の機能が使える可能性がある。
  - 使用中のエンジンが pdfTEX である場合、既定で安全モードが有効になる。
- `\rubysizeratio{`⟨実数⟩`}`
  ルビサイズの親文字サイズに対する割合。［既定 = 0.5］
- `\rubystretchprop{`⟨$X$⟩`}{`⟨$Y$⟩`}{`⟨$Z$⟩`}`
  ルビ用均等割りの比率の指定。［既定 = 1, 2, 1］
- `\rubystretchprophead{`⟨$Y$⟩`}{`⟨$Z$⟩`}`
  前突出禁止時の均等割りの比率の指定。［既定 = 1, 1］
- `\rubystretchpropend{`⟨$X$⟩`}{`⟨$Y$⟩`}`
  後突出禁止時の均等割りの比率の指定。［既定 = 1, 1］
- `\rubyyheightratio{`⟨実数⟩`}`
  横組和文の高さの縦幅に対する割合。［既定 = 0.88］
- `\rubytheightratio{`⟨実数⟩`}`
  縦組和文の「高さ」の「縦幅」に対する割合（pTEX の縦組では「縦」と「横」が実際の逆になる）。［既定 = 0.5］

# 3 圏点機能

## 3.1 圏点用命令

- `\kenten[`⟨オプション⟩`]{`⟨親文字⟩`}`
  和文文字列の上側（横組）／右側（縦組）に圏点を付す（オプションで逆側にもできる）。

[*3] 安全モードでは、強制的にグループルビに切り替わる。また、親文字・ルビの両方の均等割り付け、および、小書き文字自動変換が無効になる。

⟨オプション⟩ は複数指定可能で、排他な指定が併存した場合は後のものが有効になる。また、省略された指定については \kentensetup で指定された既定値が用いられる。

オプションに指定できる値は以下の通り。

| | | | |
|---|---|---|---|
| p (< *primary*) | 主マーク | P (< *primary*) | 上側配置 |
| s (< *seconday*) | 副マーク | S (< *secondary*) | 下側配置 |
| f (< *full*) | 全文字付加有効 | | |
| F | 全文字付加無効 | | |

- p、s は付加する圏点の種類を表す。横組では主マーク（p）は黒中点、副マーク（s）は黒ゴマ点が用いられ、縦組では逆に主マークが黒ゴマ点、副マークが黒中点となる。ただし設定命令により圏点の種類は変更できる。
- P は親文字列の上側（横組）／右側（縦組）、S は親文字列の下側（横組）／左側（縦組）に圏点を付す指定。
- f 指定時は、親文字列に含まれる“通常文字”の全てに圏点を付加する。F 指定時は、約物である“通常文字”には圏点を付加しない。

## 3.2 圏点命令の親文字列

圏点付加の処理では親文字列を文字毎に分解する必要がある。このため、圏点命令の親文字列は一定の規則に従って書かれる必要がある。

圏点命令の親文字列には以下のものを含めることができる。

- 通常文字： LaTeX の命令や特殊文字や欧文空白でない、欧文または和文の文字を指す。通常文字には一つの圏点が付加される。
  - F オプションを指定した場合、約物（句読点等）の文字には圏点が付加されない。
  - 欧文文字に圏点を付けた場合、その文字は組版上“和文文字のように”振舞う。
- LaTeX の命令および欧文空白： これらには圏点が付加されない。
  - 主に \, や \quad のような空白用の命令の使用を意図している。
  - \hspace{1zw} のような引数を取る命令をそのまま書くことはできない。この場合は、以降に示す何れかの書式を利用する必要がある。[*4]
- グループ： すなわち、{ } に囲まれた任意のテキスト。ルビ命令のグループと同様に、一つの《文字》として扱われ、全体に対して一つの圏点が付加される。
  - japanese-otf パッケージの \CID 命令のような、「特殊な和文文字を出力する命令」の使用を意図している。
- \kspan{⟨テキスト⟩}： これは、出力されるテキストの幅に応じた個数の圏点が付加される。
  - 例えば、“くの字点”に圏点を付す場合に使える。

[*4] 全角空白（\hspace{1zw}）や和欧文間空白（\hspace{\kanjiskip}）を出力する専用のマクロを用意しておくと便利かもしれない。

  - あるいは、（少々手抜きであるが[*5]）\kenten{この\kspan{\textgt{文字}}だ}みたいな使い方も考えられる。
- \kspan*{⟨テキスト⟩}： これは圏点を付さずにテキストをそのまま出力する。
- ルビ命令（\ruby 等）: 例えば
  \kenten{これが\ruby[|j|]{圏点}{けん|てん}です}。
  のように、ルビ命令はそのまま書くことができる。
  - \kentenrubycombination の設定によっては、ルビと圏点の両方が付加される。
  - 実装上の制限[*6]のため、圏点命令の先頭にルビ命令がある場合、ルビの前側の進入が無効になる。同様に、圏点命令の末尾にルビ命令がある場合、ルビの後側の進入が無効になる。
  - 圏点命令中のルビの処理は通常の場合と比べて“複雑”であるため、自動的な禁則処理が働かない可能性が高い。従って、必要に応じて補助設定で分割禁止（*）を指定する必要がある。
  - 逆にルビ命令の入力に圏点命令をそのまま書くことはできない。
    \ruby[|j|]{\kenten{圏点}}{けん|てん}% 不可
    { } で囲った《文字》の中では使えるが、この場合は同時付加とは見なされず、独立に動作することになる。

## 3.3 ゴースト処理

圏点出力ではルビと異なり進入の処理が不要である。このため、現状では、圏点命令については**常に**和文ゴースト処理を適用する。
※ 非標準の和文メトリック（JFM）が使われている等の理由で、和文ゴースト処理が正常に機能しない場合が存在する。このため、将来的に、圏点命令についても和文ゴースト処理を行わない（ルビ命令と同様の補助設定を適用する）設定を用意する予定である。

## 3.4 パラメタ設定命令

- \kentensetup{⟨オプション⟩}
  オプションの既定値設定。［既定 = pPF］
- \kentenmarkinyoko{⟨名前またはテキスト⟩}
  横組時の主マーク（p 指定時）として使われる圏点を指定する。［既定 = bullet*］
  パッケージで予め用意されている圏点種別については名前で指定できる。

[*5] 本来は、\textgt の中で改めて \kenten を使うべきである。
[*6] 圏点命令は常にゴースト処理を伴うため、先述の「ゴースト処理と進入は共存しない」という制限に引っかかるのである。

| | | | | | |
|---|---|---|---|---|---|
| bullet* | ・（合成） | 黒中点 | triangle | ▲ 25B2 | 黒三角 |
| bullet | ・ 2022* | 黒中点 | Triangle | △ 25B3 | 白三角 |
| Bullet | ◦ 25E6* | 白中点 | circle | ● 25CF | 黒丸 |
| sesame* | 、（合成） | 黒ゴマ点 | Circle | ○ 25CB | 白丸 |
| sesame | 、 FE45* | 黒ゴマ点 | bullseye | ◎ 25CE | 二重丸 |
| Sesame | 〟 FE46* | 白ゴマ点 | fisheye | ◉ 25C9* | 蛇の目点 |

  - これらの圏点種別のうち、bullet* は中黒“・”（U+30FB）、sesame* は読点“、”（U+3001）の字形を加工したものを利用する。これらはどんな日本語フォントでもサポートされているので、確実に使用できる。
  - それ以外の圏点種別は、記載の文字コードをもつ Unicode 文字を出力する。使用するフォントによっては、字形を持っていないため何も出力されない、あるいは字形が全角幅でないため正常に出力されない、という可能性がある。
  - 文字コード値に * を付けたものは、その文字が JIS X 0208 にないことを表す。pLaTeX でこれらの圏点種別を利用するためには japanese-otf パッケージを読み込む必要がある。

  あるいは、名前の代わりに任意の LaTeX のテキストを書くことができる。[*7]
  `\kentenmarkinyoko{※}`
- `\kentensubmarkinyoko{`⟨名前またはテキスト⟩`}`
  横組時の副マーク（s 指定時）として使われる圏点を指定する。［既定 = sesame*］
- `\kentenmarkintate{`⟨名前またはテキスト⟩`}`
  縦組時の主マーク（p 指定時）として使われる圏点を指定する。［既定 = sesame*］
- `\kentensubmarkintate{`⟨名前またはテキスト⟩`}`
  縦組時の副マーク（s 指定時）として使われる圏点を指定する。［既定 = bullet*］
- `\kentenfontsetup{`⟨命令⟩`}`
  圏点用のフォント切替命令を設定する。
- `\kentenintergap{`⟨実数⟩`}`
  圏点と親文字の間の空き（親文字全角単位）。［既定 = 0］
- `\kentensizeratio{`⟨実数⟩`}`
  圏点サイズの親文字サイズに対する割合。［既定 = 0.5］

圏点とルビの同時付加に関する設定。

- `\kentenrubycombination{`⟨値⟩`}` 圏点命令の親文字中でルビ命令が使われた時の挙動を指定する。［既定 = both］
  - ruby： ルビのみを出力する。
  - both： ルビの外側に圏点を出力する。
- `\kentenrubyintergap{`⟨実数⟩`}`
  圏点とルビが同じ側に付いた時の間の空き（親文字全角単位）。［既定 = 0］

[*7] ただし、引数の先頭の文字が ASCII 英字である場合は名前の指定と見なされるため、テキストとして扱いたい場合は適宜 { } を補う等の措置が必要である。

# 4　実装（ルビ関連）

## 4.1　前提パッケージ

keyval を使う予定（まだ使っていない）。

```
\RequirePackage{keyval}
```

## 4.2　エラーメッセージ

\pxrr@error　エラー出力命令。

\pxrr@warn

```
\def\pxrr@pkgname{pxrubrica}
\def\pxrr@error{%
  \PackageError\pxrr@pkgname
}
\def\pxrr@warn{%
  \PackageWarning\pxrr@pkgname
}
```

\ifpxrr@fatal@error　致命的エラーが発生したか。スイッチ。

```
\newif\ifpxrr@fatal@error
```

\pxrr@fatal@error　致命的エラーのフラグを立てて、エラーを表示する。

```
\def\pxrr@fatal@error{%
  \pxrr@fatal@errortrue
  \pxrr@error
}
```

\pxrr@eh@fatal　致命的エラーのヘルプ。

```
\def\pxrr@eh@fatal{%
  The whole ruby input was ignored.\MessageBreak
  \@ehc
}
```

\pxrr@fatal@not@supported　未実装の機能を呼び出した場合。

```
\def\pxrr@fatal@not@supported#1{%
  \pxrr@fatal@error{Not yet supported: #1}%
  \pxrr@eh@fatal
}
```

\pxrr@err@inv@value　引数に無効な値が指定された場合。

```
\def\pxrr@err@inv@value#1{%
  \pxrr@error{Invalud value (#1)}%
  \@ehc
}
```

\pxrr@fatal@unx@letter　オプション中に不測の文字が現れた場合。

```
\def\pxrr@fatal@unx@letter#1{%
  \pxrr@fatal@error{Unexpected letter '#1' found}%
  \pxrr@eh@fatal
}
```

\pxrr@warn@bad@athead モノルビ以外、あるいは横組みで肩付き指定が行われた場合。強制的に中付きに変更される。

```
\def\pxrr@warn@bad@athead{%
  \pxrr@warn{Position 'h' not allowed here}%
}
```

\pxrr@warn@must@group 欧文ルビでグループルビ以外の指定が行われた場合。強制的にグループルビに変更される。

```
\def\pxrr@warn@must@group{%
  \pxrr@warn{Only group ruby is allowed here}%
}
```

\pxrr@warn@bad@jukugo 両側ルビで熟語ルビの指定が行われた場合。強制的に選択的モノルビ（M）に変更される。

```
\def\pxrr@warn@bad@jukugo{%
  \pxrr@warn{Jukugo ruby is not allowed here}%
}
```

\pxrr@fatal@bad@intr ゴースト処理が有効で進入有りを設定した場合。（致命的エラー）。

```
\def\pxrr@fatal@bad@intr{%
  \pxrr@fatal@error{%
    Intrusion disallowed when ghost is enabled%
  }\pxrr@eh@fatal
}
```

\pxrr@fatal@bad@no@protr 前と後の両方で突出禁止を設定した場合。（致命的エラー）。

```
\def\pxrr@fatal@bad@no@protr{%
  \pxrr@fatal@error{%
    Protrusion must be allowed for either end%
  }\pxrr@eh@fatal
}
```

\pxrr@fatal@bad@length 親文字列とルビ文字列でグループの個数が食い違う場合。（モノルビ・熟語ルビの場合、親文字のグループ数は実際には《文字》数のこと。）

```
\def\pxrr@fatal@bad@length#1#2{%
  \pxrr@fatal@error{%
    Group count mismatch between the ruby and\MessageBreak
    the body (#1 <> #2)%
  }\pxrr@eh@fatal
}
```

\pxrr@fatal@bad@mono モノルビ・熟語ルビの親文字列が 2 つ以上のグループを持つ場合。

```
\def\pxrr@fatal@bad@mono{%
  \pxrr@fatal@error{%
    Mono-ruby body must have a single group%
  }\pxrr@eh@fatal
}
```

\pxrr@fatal@bad@switching 選択的ルビの親文字列が 2 つ以上のグループを持つ場合。

```
\def\pxrr@fatal@bad@switching{%
  \pxrr@fatal@error{%
    The body of Switching-ruby (M/J) must\MessageBreak
    have a single group%
  }\pxrr@eh@fatal
}
```

\pxrr@fatal@bad@movable 欧文ルビ（必ずグループルビとなる）でルビ文字列が 2 つ以上のグループを持つ場合。

```
\def\pxrr@fatal@bad@movable{%
  \pxrr@fatal@error{%
    Novable group ruby is not allowed here%
  }\pxrr@eh@fatal
}
```

\pxrr@fatal@na@movable グループルビでルビ文字列が 2 つ以上のグループを持つ（つまり可動グループルビである）が、拡張機能が無効であるため実現できない場合。

```
\def\pxrr@fatal@na@movable{%
  \pxrr@fatal@error{%
    Feature of movable group ruby is disabled%
  }\pxrr@eh@fatal
}
```

\pxrr@warn@load@order Unicode TEX 用の日本語組版パッケージ（LuaTEX-ja 等）はこのパッケージより前に読み込むべきだが、後で読み込まれていることが判明した場合。

```
\def\pxrr@warn@load@order#1{%
  \pxrr@warn{%
    This package should be loaded after '#1'%
  }%
}
```

\pxrr@interror 内部エラー。これが出てはいけない。:-)

```
\def\pxrr@interror#1{%
  \pxrr@fatal@error{INTERNAL ERROR (#1)}%
  \pxrr@eh@fatal
}
```

\ifpxrrDebug デバッグモード指定。

```
\newif\ifpxrrDebug
```

## 4.3 パラメタ

### 4.3.1 全般設定

\pxrr@ruby@font ルビ用フォント切替命令。

```
\let\pxrr@ruby@font\@empty
```

\pxrr@big@intr 「大」と「小」の進入量（\rubybigintrusion／\rubysmallintrusion）。実数値マクロ（数字列に展開される）。
\pxrr@small@intr

```
87 \def\pxrr@big@intr{1}
88 \def\pxrr@small@intr{0.5}
```

`\pxrr@size@ratio` ルビ文字サイズ（`\rubysizeratio`）。実数値マクロ。

```
89 \def\pxrr@size@ratio{0.5}
```

`\pxrr@sprop@x` 伸縮配置比率（`\rubystretchprop`）。実数値マクロ。

`\pxrr@sprop@y`

`\pxrr@sprop@z`

```
90 \def\pxrr@sprop@x{1}
91 \def\pxrr@sprop@y{2}
92 \def\pxrr@sprop@z{1}
```

`\pxrr@sprop@hy` 伸縮配置比率（`\rubystretchprophead`）。実数値マクロ。

`\pxrr@sprop@hz`

```
93 \def\pxrr@sprop@hy{1}
94 \def\pxrr@sprop@hz{1}
```

`\pxrr@sprop@ex` 伸縮配置比率（`\rubystretchpropend`）。実数値マクロ。

`\pxrr@sprop@ey`

```
95 \def\pxrr@sprop@ex{1}
96 \def\pxrr@sprop@ey{1}
```

`\pxrr@maxmargin` ルビ文字列の最大マージン（`\rubymaxmargin`）。実数値マクロ。

```
97 \def\pxrr@maxmargin{0.75}
```

`\pxrr@yhtratio` 横組和文の高さの縦幅に対する割合（`\rubyyheightratio`）。実数値マクロ。

```
98 \def\pxrr@yhtratio{0.88}
```

`\pxrr@thtratio` 縦組和文の高さの縦幅に対する割合（`\rubytheightratio`）。実数値マクロ。

```
99 \def\pxrr@thtratio{0.5}
```

`\pxrr@extra` 拡張機能実装方法（`\rubyuseextra`）。整数定数。

```
100 \chardef\pxrr@extra=0
```

`\ifpxrr@jghost` 和文ゴースト処理を行うか（`\ruby[no]usejghost`）。スイッチ。

```
101 \newif\ifpxrr@jghost \pxrr@jghostfalse
```

`\ifpxrr@aghost` 欧文ゴースト処理を行うか（`\ruby[no]useaghost`）。スイッチ。

```
102 \newif\ifpxrr@aghost \pxrr@aghostfalse
```

`\pxrr@inter@gap` ルビと親文字の間の空き（`\rubyintergap`）。実数値マクロ。

```
103 \def\pxrr@inter@gap{0}
```

`\ifpxrr@edge@adjust` 行頭・行末での突出の自動補正を行うか（`\ruby[no]adjustatlineedge`）。スイッチ。

```
104 \newif\ifpxrr@edge@adjust \pxrr@edge@adjustfalse
```

`\ifpxrr@break@jukugo` 熟語ルビで中間の行分割を許すか（`\ruby[no]breakjukugo`）。スイッチ。

```
105 \newif\ifpxrr@break@jukugo \pxrr@break@jukugofalse
```

`\ifpxrr@safe@mode` 安全モードであるか。（`\ruby[no]safemode`）。スイッチ。

```
106 \newif\ifpxrr@safe@mode \pxrr@safe@modefalse
```

\ifpxrr@d@bprotr 突出を許すか否か。\rubysetup の ⟨前設定⟩／⟨後設定⟩ に由来する。スイッチ。

\ifpxrr@d@aprotr
```
\newif\ifpxrr@d@bprotr \pxrr@d@bprotrtrue
\newif\ifpxrr@d@aprotr \pxrr@d@aprotrtrue
```

\pxrr@d@bintr 進入量。\rubysetup の ⟨前設定⟩／⟨後設定⟩ に由来する。\pxrr@XXX@intr または空（進入無し）に展開されるマクロ。

\pxrr@d@aintr
```
\def\pxrr@d@bintr{}
\def\pxrr@d@aintr{}
```

\pxrr@d@athead 肩付き／中付きの設定。\rubysetup の c／h／H の設定。0 = 中付き（c）; 1 = 肩付き（h）; 2 = 拡張肩付き（H）。整数定数。

```
\chardef\pxrr@d@athead=0
```

\pxrr@d@mode モノルビ（m）・グループルビ（g）・熟語ルビ（j）のいずれか。\rubysetup の設定値。オプション文字への暗黙の（\let された）文字トークン。

```
\let\pxrr@d@mode=j
```

\pxrr@d@side ルビを親文字の上下のどちらに付すか。0 = 上側 ; 1 = 下側。\rubysetup の P／S の設定。整数定数。

```
\chardef\pxrr@d@side=0
```

\pxrr@d@evensp 親文字列均等割りの設定。0 = 無効 ; 1 = 有効。\rubysetup の e／E の設定。整数定数。

```
\chardef\pxrr@d@evensp=1
```

\pxrr@d@fullsize 小書き文字変換の設定。0 = 無効 ; 1 = 有効。\rubysetup の f／F の設定。整数定数。

```
\chardef\pxrr@d@fullsize=0
```

#### 4.3.2 呼出時パラメタ・変数

一般的に、特定のルビ・圏点命令の呼出に固有である（つまりその内側にネストされたルビ・圏点命令に継承すべきでない）パラメタは、呼出時の値を別に保持しておくべきである。

\ifpxrr@bprotr 突出を許すか否か。\ruby の ⟨前設定⟩／⟨後設定⟩ に由来する。スイッチ。

\ifpxrr@aprotr
```
\newif\ifpxrr@bprotr \pxrr@bprotrfalse
\newif\ifpxrr@aprotr \pxrr@aprotrfalse
```

\pxrr@bintr 進入量。\ruby の ⟨前設定⟩／⟨後設定⟩ に由来する。寸法値に展開されるマクロ。

\pxrr@aintr
```
\def\pxrr@bintr{}
\def\pxrr@aintr{}
```

\pxrr@bscomp 空き補正設定。\ruby の : 指定に由来する。暗黙の文字トークン（無指定は \relax）。

\pxrr@ascomp ※ 既定値設定（\rubysetup）でこれに対応するものはない。

```
\let\pxrr@bscomp\relax
\let\pxrr@ascomp\relax
```

\ifpxrr@bnobr ルビ付文字の直前／直後で行分割を許すか。\ruby の * 指定に由来する。スイッチ。

\ifpxrr@anobr ※ 既定値設定（\rubysetup）でこれに対応するものはない。

```
\newif\ifpxrr@bnobr \pxrr@bnobrfalse
\newif\ifpxrr@anobr \pxrr@anobrfalse
```

`\ifpxrr@bfintr` `\ifpxrr@afintr` 段落冒頭／末尾で進入を許可するか。`\ruby` の ! 指定に由来する。スイッチ。
※ 既定値設定（`\rubysetup`）でこれに対応するものはない。

```
\newif\ifpxrr@bfintr \pxrr@bfintrfalse
\newif\ifpxrr@afintr \pxrr@afintrfalse
```

`\pxrr@athead` 肩付き／中付きの設定。`\ruby` の c／h／H の設定。値の意味は `\pxrr@d@athead` と同じ。整数定数。

```
\chardef\pxrr@athead=0
```

`\ifpxrr@athead@iven` 肩付き／中付きの設定が明示的であるか。スイッチ。

```
\newif\ifpxrr@athead@given \pxrr@athead@givenfalse
```

`\pxrr@mode` モノルビ（m）・グループルビ（g）・熟語ルビ（j）のいずれか。`\ruby` のオプションの設定値。オプション文字への暗黙文字トークン。

```
\let\pxrr@mode=\@undefined
```

`\ifpxrr@mode@given` 基本モードの設定が明示的であるか。スイッチ。

```
\newif\ifpxrr@mode@given \pxrr@mode@givenfalse
\newif\ifpxrr@afintr \pxrr@afintrfalse
```

`\ifpxrr@abody` ルビが `\aruby`（欧文親文字用）であるか。スイッチ。

```
\newif\ifpxrr@abody
```

`\pxrr@side` ルビを親文字の上下のどちらに付すか。0 = 上側；1 = 下側；2 = 両側。`\ruby` の P／S が 0／1 に対応し、`\truby` では 2 が使用される。整数定数。

```
\chardef\pxrr@side=0
```

`\pxrr@evensp` 親文字列均等割りの設定。0 = 無効；1 = 有効。`\ruby` の e／E の設定。整数定数。

```
\chardef\pxrr@evensp=1
```

`\pxrr@revensp` ルビ文字列均等割りの設定。0 = 無効；1 = 有効。整数定数。
※ 通常は有効だが、安全モードでは無効になる。

```
\chardef\pxrr@revensp=1
```

`\pxrr@fullsize` 小書き文字変換の設定。0 = 無効；1 = 有効。`\ruby` の f／F の設定。整数定数。

```
\chardef\pxrr@fullsize=1
```

`\pxrr@c@ruby@font` `\pxrr@c@size@ratio` `\pxrr@c@inter@gap` 以下は“オプションで指定する”以外のパラメタに対応するもの。

```
\let\pxrr@c@ruby@font\@undefined
\let\pxrr@c@size@ratio\@undefined
\let\pxrr@c@inter@gap\@undefined
```

## 4.4 その他の変数

\pxrr@body@list 親文字列のために使うリスト。

```
\let\pxrr@body@list\@undefined
```

\pxrr@body@count \pxrr@body@list の長さ。整数値マクロ。

```
\let\pxrr@body@count\@undefined
```

\pxrr@ruby@list ルビ文字列のために使うリスト。

```
\let\pxrr@ruby@list\@undefined
```

\pxrr@ruby@count \pxrr@ruby@list の長さ。整数値マクロ。

```
\let\pxrr@ruby@count\@undefined
```

\pxrr@sruby@list 2 つ目のルビ文字列のために使うリスト。

```
\let\pxrr@sruby@list\@undefined
```

\pxrr@sruby@count \pxrr@sruby@list の長さ。整数値マクロ。

```
\let\pxrr@sruby@count\@undefined
```

\pxrr@whole@list 親文字とルビのリストを zip したリスト。

```
\let\pxrr@whole@list\@undefined
```

\pxrr@bspace ルビが親文字から前側にはみだす長さ。寸法値マクロ。

```
\let\pxrr@bspace\@undefined
```

\pxrr@aspace ルビが親文字から後側にはみだす長さ。寸法値マクロ。

```
\let\pxrr@aspace\@undefined
```

\pxrr@natwd \pxrr@evenspace@int のパラメタ。寸法値マクロ。

```
\let\pxrr@natwd\@undefined
```

\pxrr@all@input 両側ルビの処理で使われる一時変数。

```
\let\pxrr@all@input\@undefined
```

## 4.5 補助手続

### 4.5.1 雑多な定義

\ifpxrr@ok 汎用スイッチ。

```
\newif\ifpxrr@ok
```

\pxrr@cnta 汎用の整数レジスタ。

```
\newcount\pxrr@cnta
```

\pxrr@cntr 結果を格納する整数レジスタ。

```
\newcount\pxrr@cntr
```

\pxrr@dima 汎用の寸法レジスタ。

```
\newdimen\pxrr@dima
```

\pxrr@boxa 汎用のボックスレジスタ。

\pxrr@boxb

```
\newbox\pxrr@boxa
\newbox\pxrr@boxb
```

\pxrr@boxr 結果を格納するボックスレジスタ。

```
\newbox\pxrr@boxr
```

\pxrr@token \futurelet 用の一時変数。
※ if-トークンなどの“危険”なトークンになりうるので使い回さない。

```
\let\pxrr@token\relax
```

\pxrr@zero 整数定数のゼロ。\z@ と異なり、「単位付寸法」の係数として使用可能。

```
\chardef\pxrr@zero=0
```

\pxrr@zeropt 「0pt」という文字列。寸法値マクロへの代入に用いる。

```
\def\pxrr@zeropt{0pt}
```

\pxrr@hfilx \pxrr@hfilx{⟨実数⟩}： 「⟨実数⟩fil」のグルーを置く。

```
\def\pxrr@hfilx#1{%
  \hskip\z@\@plus #1fil\relax
}
```

\pxrr@res 結果を格納するマクロ。

```
\let\pxrr@res\@empty
```

\pxrr@ifx \pxrr@ifx{⟨引数⟩}⟨真⟩}{⟨偽⟩}： \ifx⟨引数⟩ を行うテスト。

```
\def\pxrr@ifx#1{%
  \ifx#1\expandafter\@firstoftwo
  \else\expandafter\@secondoftwo
  \fi
}
```

\pxrr@cond \pxrr@cond\ifXXX...\fi{⟨真⟩}{⟨偽⟩}： 一般の TEX の if 文 \ifXXX... を行うテスト。
※ \fi を付けているのは、if-不均衡を避けるため。

```
\@gobbletwo\if\if \def\pxrr@cond#1\fi{%
  #1\expandafter\@firstoftwo
  \else\expandafter\@secondoftwo
  \fi
}
```

\pxrr@cslet \pxrr@cslet{NAMEa}\CSb： \NAMEa に \CSb を \let する。

\pxrr@letcs \pxrr@letcs\CSa{NAMEb}： \CSa に \NAMEb を \let する。

\pxrr@csletcs \pxrr@csletcs{NAMEa}{NAMEb}： \NAMEa に \NAMEb を \let する。

```
\def\pxrr@cslet#1{%
  \expandafter\let\csname#1\endcsname
```

```
}
\def\pxrr@letcs#1#2{%
  \expandafter\let\expandafter#1\csname#2\endcsname
}
\def\pxrr@csletcs#1#2{%
  \expandafter\let\csname#1\expandafter\endcsname
   \csname#2\endcsname
}
```

\pxrr@setok　\pxrr@setok{⟨テスト⟩}： テストの結果を \ifpxrr@ok に返す。

```
\def\pxrr@setok#1{%
  #1{\pxrr@oktrue}{\pxrr@okfalse}%
}
```

\pxrr@appto　\pxrr@appto\CS{⟨テキスト⟩}： 無引数マクロの置換テキストに追加する。

```
\def\pxrr@appto#1#2{%
  \expandafter\def\expandafter#1\expandafter{#1#2}%
}
```

\pxrr@nil　ユニークトークン。

\pxrr@end

```
\def\pxrr@nil{\noexpand\pxrr@nil}
\def\pxrr@end{\noexpand\pxrr@end}
```

\pxrr@without@macro@trace　\pxrr@without@macro@trace{⟨テキスト⟩}： マクロ展開のトレースを無効にした状態で ⟨テキスト⟩ を実行する。

```
\def\pxrr@without@macro@trace#1{%
  \chardef\pxrr@tracingmacros@save=\tracingmacros
  \tracingmacros\z@
  #1%
  \tracingmacros\pxrr@tracingmacros@save
}
\chardef\pxrr@tracingmacros@save=0
```

\pxrr@hbox　color パッケージ対応の \hbox と \hb@xt@（= \hbox to）。

\pxrr@hbox@to

```
\def\pxrr@hbox#1{%
  \hbox{%
    \color@begingroup
      #1%
    \color@endgroup
  }%
}
\def\pxrr@hbox@to#1#{%
  \pxrr@hbox@to@a{#1}%
}
\def\pxrr@hbox@to@a#1#2{%
  \hbox to#1{%
    \color@begingroup
      #2%
    \color@endgroup
```

```
  }%
}
```

color パッケージ不使用の場合は、本来の \hbox と \hb@xt@ に戻しておく。これと同期して \pxrr@takeout@any@protr の動作も変更する。

```
\AtBeginDocument{%
  \ifx\color@begingroup\relax
    \ifx\color@endgroup\relax
      \let\pxrr@hbox\hbox
      \let\pxrr@hbox@to\hb@xt@
      \let\pxrr@takeout@any@protr\pxrr@takeout@any@protr@nocolor
    \fi
  \fi
}
```

### 4.5.2 数値計算

\pxrr@invscale

\pxrr@invscale{⟨寸法レジスタ⟩}{⟨実数⟩}： 現在の ⟨寸法レジスタ⟩ の値を ⟨実数⟩ で除算した値に更新する。すなわち、⟨寸法レジスタ⟩=⟨実数⟩⟨寸法レジスタ⟩ の逆の演算を行う。

```
\mathchardef\pxrr@invscale@ca=259
\def\pxrr@invscale#1#2{%
  \begingroup
    \@tempdima=#1\relax
    \@tempdimb#2\p@\relax
    \@tempcnta\@tempdima
    \multiply\@tempcnta\@cclvi
    \divide\@tempcnta\@tempdimb
    \multiply\@tempcnta\@cclvi
    \@tempcntb\p@
    \divide\@tempcntb\@tempdimb
    \advance\@tempcnta-\@tempcntb
    \advance\@tempcnta-\tw@
    \@tempdimb\@tempcnta\@ne
    \advance\@tempcnta\@tempcntb
    \advance\@tempcnta\@tempcntb
    \advance\@tempcnta\pxrr@invscale@ca
    \@tempdimc\@tempcnta\@ne
    \@whiledim\@tempdimb<\@tempdimc\do{%
      \@tempcntb\@tempdimb
      \advance\@tempcntb\@tempdimc
      \advance\@tempcntb\@ne
      \divide\@tempcntb\tw@
      \ifdim #2\@tempcntb>\@tempdima
        \advance\@tempcntb\m@ne
        \@tempdimc=\@tempcntb\@ne
      \else
        \@tempdimb=\@tempcntb\@ne
      \fi}%
```

```
    \xdef\pxrr@gtempa{\the\@tempdimb}%
  \endgroup
  #1=\pxrr@gtempa\relax
}
```

\pxrr@interpolate `\pxrr@interpolate{⟨入力単位⟩}{⟨出力単位⟩}{⟨寸法レジスタ⟩}{(X1,Y1)(X2,Y2)⋯(Xn,Yn)}`： 線形補間を行う。すなわち、明示値

$$f(0\,\mathrm{pt}) = 0\,\mathrm{pt},\ f(\mathrm{X}_1\,\mathrm{iu}) = \mathrm{Y}_1\,\mathrm{ou},\ \ldots,\ f(\mathrm{X}_n\,\mathrm{iu}) = \mathrm{Y}_n\,\mathrm{ou}$$

（ただし $(0, \mathrm{pt} < \mathrm{X}_1\,\mathrm{iu} < \cdots < \mathrm{X}_n\,\mathrm{iu})$；ここで iu は ⟨入力単位⟩、ou は ⟨出力単位⟩ に指定されたもの）を線形補間して定義される関数 $f(\cdot)$ について、$f(\langle\text{寸法}\rangle)$ の値を ⟨寸法レジスタ⟩ に代入する。

※ $[0\,\mathrm{pt}, \mathrm{X}_n\,\mathrm{iu}]$ の範囲外では両端の 2 点による外挿を行う。

```
\def\pxrr@interpolate#1#2#3#4#5{%
  \edef\pxrr@tempa{#1}%
  \edef\pxrr@tempb{#2}%
  \def\pxrr@tempd{#3}%
  \setlength{\@tempdima}{#4}%
  \edef\pxrr@tempc{(0,0)#5(*,*)}%
  \expandafter\pxrr@interpolate@a\pxrr@tempc\@nil
}
\def\pxrr@interpolate@a(#1,#2)(#3,#4)(#5,#6){%
  \if*#5%
    \def\pxrr@tempc{\pxrr@interpolate@b{#1}{#2}{#3}{#4}}%
  \else\ifdim\@tempdima<#3\pxrr@tempa
    \def\pxrr@tempc{\pxrr@interpolate@b{#1}{#2}{#3}{#4}}%
  \else
    \def\pxrr@tempc{\pxrr@interpolate@a(#3,#4)(#5,#6)}%
  \fi\fi
  \pxrr@tempc
}
\def\pxrr@interpolate@b#1#2#3#4#5\@nil{%
  \@tempdimb=-#1\pxrr@tempa
  \advance\@tempdima\@tempdimb
  \advance\@tempdimb#3\pxrr@tempa
  \edef\pxrr@tempc{\strip@pt\@tempdimb}%
  \pxrr@invscale\@tempdima\pxrr@tempc
  \edef\pxrr@tempc{\strip@pt\@tempdima}%
  \@tempdima=#4\pxrr@tempb
  \@tempdimb=#2\pxrr@tempb
  \advance\@tempdima-\@tempdimb
  \@tempdima=\pxrr@tempc\@tempdima
  \advance\@tempdima\@tempdimb
  \pxrr@tempd=\@tempdima
}
```

### 4.5.3 リスト分解

\pxrr@decompose `\pxrr@decompose{`⟨要素 1⟩···⟨要素 n⟩`}`: ここで各 ⟨要素⟩ は単一トークンまたはグループ（`{...}` で囲まれたもの）とする。この場合、`\pxrr@res` を以下のトークン列に定義する。

`\pxrr@pre{`⟨要素 1⟩`}\pxrr@inter{`⟨要素 2⟩`}`···
`\pxrr@inter{`⟨要素 n⟩`}\pxrr@post`

そして、`\pxrr@cntr` を n に設定する。

※ ⟨要素⟩ に含まれるグルーピングは完全に保存される（最外の `{...}` が外れたりしない）。

```
\def\pxrr@decompose#1{%
  \let\pxrr@res\@empty
  \pxrr@cntr=\z@
  \pxrr@decompose@loopa#1\pxrr@end
}
\def\pxrr@decompose@loopa{%
  \futurelet\pxrr@token\pxrr@decompose@loopb
}
\def\pxrr@decompose@loopb{%
  \pxrr@ifx{\pxrr@token\pxrr@end}{%
    \pxrr@appto\pxrr@res{\pxrr@post}%
  }{%
    \pxrr@setok{\pxrr@ifx{\pxrr@token\bgroup}}%
    \pxrr@decompose@loopc
  }%
}
\def\pxrr@decompose@loopc#1{%
  \ifx\pxrr@res\@empty
    \def\pxrr@res{\pxrr@pre}%
  \else
    \pxrr@appto\pxrr@res{\pxrr@inter}%
  \fi
  \ifpxrr@ok
    \pxrr@appto\pxrr@res{{{#1}}}%
  \else
    \pxrr@appto\pxrr@res{{#1}}%
  \fi
  \advance\pxrr@cntr\@ne
  \pxrr@decompose@loopa
}
```

\pxrr@decompbar `\pxrr@decompbar{`⟨要素 1⟩`|`·····`|`⟨要素 n⟩`}`: ただし、各 ⟨要素⟩ はグルーピングの外の `|` を含まないとする。入力の形式と ⟨要素⟩ の構成条件が異なることを除いて、`\pxrr@decompose` と同じ動作をする。

```
\def\pxrr@decompbar#1{%
  \let\pxrr@res\@empty
  \pxrr@cntr=\z@
```

```
  \pxrr@decompbar@loopa\pxrr@nil#1|\pxrr@end|%
}
\def\pxrr@decompbar@loopa#1|{%
  \expandafter\pxrr@decompbar@loopb\expandafter{\@gobble#1}%
}
\def\pxrr@decompbar@loopb#1{%
  \pxrr@decompbar@loopc#1\relax\pxrr@nil{#1}%
}
\def\pxrr@decompbar@loopc#1#2\pxrr@nil#3{%
  \pxrr@ifx{#1\pxrr@end}{%
    \pxrr@appto\pxrr@res{\pxrr@post}%
  }{%
    \ifx\pxrr@res\@empty
      \def\pxrr@res{\pxrr@pre}%
    \else
      \pxrr@appto\pxrr@res{\pxrr@inter}%
    \fi
    \pxrr@appto\pxrr@res{{#3}}%
    \advance\pxrr@cntr\@ne
    \pxrr@decompbar@loopa\pxrr@nil
  }%
}
```

\pxrr@zip@list　\pxrr@zip@list\CSa\CSb： \CSa と \CSb が以下のように展開されるマクロとする：

\CSa = \pxrr@pre{⟨X1⟩}\pxrr@inter{⟨X2⟩}···\pxrr@inter{⟨Xn⟩}\pxrr@post
\CSb = \pxrr@pre{⟨Y1⟩}\pxrr@inter{⟨Y2⟩}···\pxrr@inter{⟨Yn⟩}\pxrr@post

この命令は \pxrr@res を以下の内容に定義する。

\pxrr@pre{⟨X1⟩}{⟨Y1⟩}\pxrr@inter{⟨X2⟩}{⟨Y2⟩}···
\pxrr@inter{⟨Xn⟩}{⟨Yn⟩}\pxrr@post

```
\def\pxrr@zip@list#1#2{%
  \let\pxrr@res\@empty
  \let\pxrr@post\relax
  \let\pxrr@tempa#1\pxrr@appto\pxrr@tempa{{}}%
  \let\pxrr@tempb#2\pxrr@appto\pxrr@tempb{{}}%
  \pxrr@zip@list@loopa
}
\def\pxrr@zip@list@loopa{%
  \expandafter\pxrr@zip@list@loopb\pxrr@tempa\pxrr@end
}
\def\pxrr@zip@list@loopb#1#2#3\pxrr@end{%
  \pxrr@ifx{#1\relax}{%
    \pxrr@zip@list@exit
  }{%
    \pxrr@appto\pxrr@res{#1{#2}}%
    \def\pxrr@tempa{#3}%
    \expandafter\pxrr@zip@list@loopc\pxrr@tempb\pxrr@end
```

```
  }%
}
\def\pxrr@zip@list@loopc#1#2#3\pxrr@end{%
  \pxrr@ifx{#1\relax}{%
    \pxrr@interror{zip}%
    \pxrr@appto\pxrr@res{{}}%
    \pxrr@zip@list@exit
  }{%
    \pxrr@appto\pxrr@res{{#2}}%
    \def\pxrr@tempb{#3}%
    \pxrr@zip@list@loopa
  }%
}
\def\pxrr@zip@list@exit{%
  \pxrr@appto\pxrr@res{\pxrr@post}%
}
```

\pxrr@tzip@list　\pxrr@tzip@list\CSa\CSb\CSc： \CSa、\CSb、\CSc が以下のように展開されるマクロとする：

\CSa = \pxrr@pre{⟨X1⟩}\pxrr@inter{⟨X2⟩}···\pxrr@inter{⟨Xn⟩}\pxrr@post
\CSb = \pxrr@pre{⟨Y1⟩}\pxrr@inter{⟨Y2⟩}···\pxrr@inter{⟨Yn⟩}\pxrr@post
\CSc = \pxrr@pre{⟨Z1⟩}\pxrr@inter{⟨Z2⟩}···\pxrr@inter{⟨Zn⟩}\pxrr@post

この命令は \pxrr@res を以下の内容に定義する。

\pxrr@pre{⟨X1⟩}{⟨Y1⟩}{⟨Z1⟩}\pxrr@inter{⟨X2⟩}{⟨Y2⟩}{⟨Z2⟩}···
\pxrr@inter{⟨Xn⟩}{⟨Yn⟩}{⟨Zn⟩}\pxrr@post

```
\def\pxrr@tzip@list#1#2#3{%
  \let\pxrr@res\@empty
  \let\pxrr@post\relax
  \let\pxrr@tempa#1\pxrr@appto\pxrr@tempa{{}}%
  \let\pxrr@tempb#2\pxrr@appto\pxrr@tempb{{}}%
  \let\pxrr@tempc#3\pxrr@appto\pxrr@tempc{{}}%
  \pxrr@tzip@list@loopa
}
\def\pxrr@tzip@list@loopa{%
  \expandafter\pxrr@tzip@list@loopb\pxrr@tempa\pxrr@end
}
\def\pxrr@tzip@list@loopb#1#2#3\pxrr@end{%
  \pxrr@ifx{#1\relax}{%
    \pxrr@tzip@list@exit
  }{%
    \pxrr@appto\pxrr@res{#1{#2}}%
    \def\pxrr@tempa{#3}%
    \expandafter\pxrr@tzip@list@loopc\pxrr@tempb\pxrr@end
  }%
}
\def\pxrr@tzip@list@loopc#1#2#3\pxrr@end{%
```

```
  \pxrr@ifx{#1\relax}{%
    \pxrr@interror{tzip}%
    \pxrr@appto\pxrr@res{{}}%
    \pxrr@tzip@list@exit
  }{%
    \pxrr@appto\pxrr@res{{#2}}%
    \def\pxrr@tempb{#3}%
    \expandafter\pxrr@tzip@list@loopd\pxrr@tempc\pxrr@end
  }%
}
\def\pxrr@tzip@list@loopd#1#2#3\pxrr@end{%
  \pxrr@ifx{#1\relax}{%
    \pxrr@interror{tzip}%
    \pxrr@appto\pxrr@res{{}}%
    \pxrr@tzip@list@exit
  }{%
    \pxrr@appto\pxrr@res{{#2}}%
    \def\pxrr@tempc{#3}%
    \pxrr@tzip@list@loopa
  }%
}
\def\pxrr@tzip@list@exit{%
  \pxrr@appto\pxrr@res{\pxrr@post}%
}
```

`\pxrr@concat@list` `\pxrr@concat@list\CS`： リストの要素を連結する。すなわち、`\CS` が

`\CSa` = `\pxrr@pre{`⟨X1⟩`}\pxrr@inter{`⟨X2⟩`}`$\cdots$`\pxrr@inter{`⟨Xn⟩`}\pxrr@post`

の時に、`\pxrr@res` を以下の内容に定義する。

⟨X1⟩⟨X2⟩$\cdots$⟨Xn⟩

```
\def\pxrr@concat@list#1{%
  \let\pxrr@res\@empty
  \def\pxrr@pre##1{%
    \pxrr@appto\pxrr@res{##1}%
  }%
  \let\pxrr@inter\pxrr@pre
  \let\pxrr@post\relax
  #1%
}
```

`\pxrr@unite@group` `\pxrr@unite@group\CS`： リストの要素を連結して 1 要素のリストに組み直す。すなわち、`\CS` が

`\CS` = `\pxrr@pre{`⟨X1⟩`}\pxrr@inter{`⟨X2⟩`}`$\cdots$`\pxrr@inter{`⟨Xn⟩`}\pxrr@post`

の時に、`\CS` を以下の内容で置き換える。

`\pxrr@pre{`⟨X1⟩⟨X2⟩$\cdots$⟨Xn⟩`}\pxrr@post`

```
\def\pxrr@unite@group#1{%
  \expandafter\pxrr@concat@list\expandafter{#1}%
  \expandafter\pxrr@unite@group@a\pxrr@res\pxrr@end#1%
}
\def\pxrr@unite@group@a#1\pxrr@end#2{%
  \def#2{\pxrr@pre{#1}\pxrr@post}%
}
```

\pxrr@zip@single　\pxrr@zip@single\CSa\CSb：

\CSa = ⟨X⟩; \CSb = ⟨Y⟩

の時に、\pxrr@res を以下の内容に定義する。

\pxrr@pre{⟨X⟩}{⟨Y⟩}\pxrr@post

```
\def\pxrr@zip@single#1#2{%
  \expandafter\pxrr@zip@single@a\expandafter#1#2\pxrr@end
}
\def\pxrr@zip@single@a#1{%
  \expandafter\pxrr@zip@single@b#1\pxrr@end
}
\def\pxrr@zip@single@b#1\pxrr@end#2\pxrr@end{%
  \def\pxrr@res{\pxrr@pre{#1}{#2}\pxrr@post}%
}
```

\pxrr@tzip@single　\pxrr@tzip@single\CSa\CSb\CSc：

\CSa = ⟨X⟩; \CSb = ⟨Y⟩; \CSc = ⟨Z⟩

の時に、\pxrr@res を以下の内容に定義する。

\pxrr@pre{⟨X⟩}{⟨Y⟩}{⟨Z⟩}\pxrr@post

```
\def\pxrr@tzip@single#1#2#3{%
  \expandafter\pxrr@tzip@single@a\expandafter#1\expandafter#2#3\pxrr@end
}
\def\pxrr@tzip@single@a#1#2{%
  \expandafter\pxrr@tzip@single@b\expandafter#1#2\pxrr@end
}
\def\pxrr@tzip@single@b#1{%
  \expandafter\pxrr@tzip@single@c#1\pxrr@end
}
\def\pxrr@tzip@single@c#1\pxrr@end#2\pxrr@end#3\pxrr@end{%
  \def\pxrr@res{\pxrr@pre{#1}{#2}{#3}\pxrr@post}%
}
```

## 4.6　エンジン依存処理

この小節のマクロ内で使われる変数。

```
\let\pxrr@x@tempa\@empty
```

```
\let\pxrr@x@tempb\@empty
\let\pxrr@x@gtempa\@empty
\newif\ifpxrr@x@swa
```

\pxrr@ifprimitive `\pxrr@ifprimitive\CS{⟨真⟩}{⟨偽⟩}`: `\CS` の現在の定義が同名のプリミティブであるかをテストする。

```
\def\pxrr@ifprimitive#1{%
  \edef\pxrr@x@tempa{\string#1}%
  \edef\pxrr@x@tempb{\meaning#1}%
  \ifx\pxrr@x@tempa\pxrr@x@tempb \expandafter\@firstoftwo
  \else \expandafter\@secondoftwo
  \fi
}
```

\ifpxrr@in@ptex エンジンが pTEX 系（upTEX 系を含む）であるか。`\kansuji` のプリミティブテストで判定する。

```
\pxrr@ifprimitive\kansuji{%
  \pxrr@csletcs{ifpxrr@in@ptex}{iftrue}%
}{%
  \pxrr@csletcs{ifpxrr@in@ptex}{iffalse}%
}
```

\ifpxrr@in@uptex エンジンが upTEX 系であるか。`\enablecjktoken` のプリミティブテストで判定する。

```
\pxrr@ifprimitive\enablecjktoken{%
  \pxrr@csletcs{ifpxrr@in@uptex}{iftrue}%
}{%
  \pxrr@csletcs{ifpxrr@in@uptex}{iffalse}%
}
```

\ifpxrr@in@xetex エンジンが XeTeX 系であるか。`\XeTeXrevision` のプリミティブテストで判定する。

```
\pxrr@ifprimitive\XeTeXrevision{%
  \pxrr@csletcs{ifpxrr@in@xetex}{iftrue}%
}{%
  \pxrr@csletcs{ifpxrr@in@xetex}{iffalse}%
}
```

\ifpxrr@in@xecjk xeCJK パッケージが使用されているか。

```
\@ifpackageloaded{xeCJK}{%
  \pxrr@csletcs{ifpxrr@in@xecjk}{iftrue}%
}{%
  \pxrr@csletcs{ifpxrr@in@xecjk}{iffalse}%
```

ここで未読込でかつプリアンブル末尾で読み込まれている場合は警告する。

```
  \AtBeginDocument{%
    \@ifpackageloaded{xeCJK}{%
      \pxrr@warn@load@order{xeCJK}%
    }{}%
  }%
}
```

\ifpxrr@in@luatex エンジンが LuaTEX 系であるか。\luatexrevision のプリミティブテストで判定する。

```
\pxrr@ifprimitive\luatexrevision{%
  \pxrr@csletcs{ifpxrr@in@luatex}{iftrue}%
}{%
  \pxrr@csletcs{ifpxrr@in@luatex}{iffalse}%
}
```

\ifpxrr@in@luatexja LuaTeX-ja パッケージが使用されているか。

```
\@ifpackageloaded{luatexja-core}{%
  \pxrr@csletcs{ifpxrr@in@luatexja}{iftrue}%
}{%
  \pxrr@csletcs{ifpxrr@in@luatexja}{iffalse}%
  \AtBeginDocument{%
    \@ifpackageloaded{luatexja-core}{%
      \pxrr@warn@load@order{LuaTeX-ja}%
    }{}%
  }%
}

\ifpxrr@in@xetex
\else\ifpxrr@in@luatex
\else\ifpxrr@in@ptex
\else
  \pxrr@ifprimitive\pdftexrevision{%
    \pxrr@warn{%
      The engine in use seems to be pdfTeX,\MessageBreak
      so safe mode is turned on%
    }%
    \AtEndOfPackage{%
      \rubysafemode
    }%
  }
\fi\fi\fi
```

\ifpxrr@in@unicode 「和文」内部コードが Unicode であるか。

```
\ifpxrr@in@xetex
  \pxrr@csletcs{ifpxrr@in@unicode}{iftrue}%
\else\ifpxrr@in@luatex
  \pxrr@csletcs{ifpxrr@in@unicode}{iftrue}%
\else\ifpxrr@in@uptex
  \pxrr@csletcs{ifpxrr@in@unicode}{iftrue}%
\else
  \pxrr@csletcs{ifpxrr@in@unicode}{iffalse}%
\fi\fi\fi
```

\pxrr@jc 和文の「複合コード」を内部コードに変換する（展開可能）。「複合コード」は「⟨JIS コード 16 進 4 桁⟩:⟨Unicode 16 進 4 桁⟩」の形式。

```
\def\pxrr@jc#1{%
  \pxrr@jc@a#1\pxrr@nil
```

```
}
\ifpxrr@in@unicode
  \def\pxrr@jc@a#1:#2\pxrr@nil{%
    "#2\space
  }
\else\ifpxrr@in@ptex
  \def\pxrr@jc@a#1:#2\pxrr@nil{%
    \jis"#1\space\space
  }
\else
  \def\pxrr@jc@a#1:#2\pxrr@nil{%
    `?\space
  }
\fi\fi
```

`\pxrr@jchardef` 和文用の `\chardef`。

```
\ifpxrr@in@uptex
  \let\pxrr@jchardef\kchardef
\else
  \let\pxrr@jchardef\chardef
\fi
```

`\pxrr@if@in@tate` `\pxrr@if@in@tate{⟨真⟩}{⟨偽⟩}`: 縦組であるか。

```
\ifpxrr@in@ptex
```

pTEX 系の場合、`\iftdir` プリミティブを利用する。

※ `\iftdir` が未定義のときに if が不均衡になるのを防ぐ。

```
  \begingroup \catcode`\|=0
    \gdef\pxrr@if@in@tate{%
      \pxrr@cond|iftdir|fi
    }
  \endgroup
\else\ifpxrr@in@luatexja
```

LuaTEX-ja 利用の場合、`direction` パラメタを利用する。

```
  \def\pxrr@if@in@tate{%
    \pxrr@cond\ifnum\ltjgetparameter{direction}=\thr@@\fi
  }
\else
```

それ以外は常に横組と見なす。

```
  \let\pxrr@if@in@tate\@secondoftwo
\fi\fi
```

`\pxrr@get@jchar@token` `\pxrr@get@jchar@token\CS{⟨整数⟩}`: 内部文字コードが ⟨整数⟩ である和文文字のトークンを得る。

※ `.sty` ファイルは完全に ASCII 文字だけにする方針のため、和文文字が必要な場合はこの補助マクロや `\pxrr@jchardef` を利用して複合コード値から作り出すことになる。

pTEX 系の場合。`\kansuji` トリックを利用する。

```
\ifpxrr@in@ptex
  \def\pxrr@get@jchar@token#1#2{%
    \begingroup
      \kansujichar\@ne=#2\relax
      \xdef\pxrr@x@gtempa{\kansuji\@ne}%
    \endgroup
    \let#1\pxrr@x@gtempa
  }
```

Unicode 対応 TEX の場合。\lowercase トリックを利用する。

```
\else\ifpxrr@in@unicode
  \def\pxrr@get@jchar@token#1#2{%
    \begingroup
      \lccode`\?=#2\relax
      \lowercase{\xdef\pxrr@x@gtempa{?}}%
    \endgroup
    \let#1\pxrr@x@gtempa
  }
```

それ以外ではダミー定義。

```
\else
  \def\pxrr@get@jchar@token#1#2{%
    \def#1{?}%
  }
\fi\fi
```

\pxrr@zspace 全角空白文字。文字そのものをファイルに含ませたくないので chardef にする。

```
\pxrr@jchardef\pxrr@zspace=\pxrr@jc{2121:3000}
```

\pxrr@jghost@char 和文ゴースト処理に利用する文字。字形が空であり、かつ一般の漢字と同じ挙動を示す必要がある。実際のゴースト処理では字幅を相殺する処理を入れる為、字幅がゼロである必要はない。

ほとんどの場合、全角空白文字で構わないが、全角空白文字が文字タイプ 0 でない JFM が使われている場合は問題になる。

upTEX の場合、“拡張符号空間”の文字コードを使う。すなわち、文字コード "113000 の文字は DVI では文字コード "3000 と扱われるが、“BMP 外”にあるため必ず文字タイプ 0 になる。

```
\ifpxrr@in@uptex
  \kchardef\pxrr@jghost@char="113000
```

LuaTEX-ja の場合。文書先頭で“全角空白文字が使えるか”を検査して、失敗した場合は「和文の U+00A0」を代わりに利用することにする。

```
\else\ifpxrr@in@luatexja
  \let\pxrr@jghost@char\pxrr@zspace
  \def\pxrr@jghost@check{%
    \begingroup
%     \ltjsetparameter{jaxspmode={\pxrr@zspace,3}}%
%     \ltjsetparameter{xkanjiskip=\p@}%
```

```
%       \ltjsetparameter{autoxspacing=false}%
      \setbox\z@\hbox{\char"3001\char"3000}%
%       \ltjsetparameter{autoxspacing=true}%
      \setbox\tw@\hbox{\char"3001\inhibitglue\char"3000}%
      \ifdim\wd\tw@=\wd\z@
        \global\chardef\pxrr@jghost@char@="00A0
        \gdef\pxrr@jghost@char{\ltjjachar\pxrr@jghost@char@}%
      \fi
    \endgroup
  }
  \AtBeginDocument{%
    \pxrr@jghost@check
  }
```

それ以外の場合は（仕方が無いので）全角空白を用いる。

```
\else
  \let\pxrr@jghost@char\pxrr@zspace
\fi\fi
```

`\pxrr@x@K` 適当な漢字（実際は〈一〉）のトークン。

```
\pxrr@jchardef\pxrr@x@K=\pxrr@jc{306C:4E00}
```

`\pxrr@get@iiskip` `\pxrr@get@iiskip\CS`： 現在の実効の和文間空白の量を取得する。

pTEX 系の場合。

```
\ifpxrr@in@ptex
  \def\pxrr@get@iiskip#1{%
```

以下では `\kanjiskip` 挿入が有効であるかを検査している。

```
    \pxrr@x@swafalse
    \begingroup
      \inhibitxspcode\pxrr@x@K\thr@@
      \kanjiskip\p@
      \setbox\z@\hbox{\noautospacing\pxrr@x@K\pxrr@x@K}%
      \setbox\tw@\hbox{\pxrr@x@K\pxrr@x@K}%
      \ifdim\wd\tw@>\wd\z@
        \aftergroup\pxrr@x@swatrue
      \fi
    \endgroup
```

以下では `\kanjiskip` 挿入が有効ならば `\kanjiskip` の値、無効ならばゼロを返す。

```
    \edef#1{%
      \ifpxrr@x@swa \the\kanjiskip
      \else \pxrr@zeropt
      \fi
    }%
  }
```

LuaTeX-ja 使用の場合。

```
\else\ifpxrr@in@luatexja
  \def\pxrr@get@iiskip#1{%
```

```
    \edef#1{%
      \ifnum\ltjgetparameter{autospacing}=\@ne
        \ltjgetparameter{kanjiskip}%
      \else \pxrr@zeropt
      \fi
    }%
  }
```

それ以外の場合はゼロとする。

```
\else
  \def\pxrr@get@iiskip#1{%
    \let#1\pxrr@zeropt
  }
\fi\fi
```

`\pxrr@get@iaiskip` `\pxrr@get@iaiskip\CS`： 現在の実効の和欧文間空白の量を取得する。

pTEX 系の場合。

```
\ifpxrr@in@ptex
  \def\pxrr@get@iaiskip#1{%
    \pxrr@x@swafalse
    \begingroup
      \inhibitxspcode\pxrr@x@K\thr@@ \xspcode`X=\thr@@
      \xkanjiskip\p@
      \setbox\z@\hbox{\noautoxspacing\pxrr@x@K X}%
      \setbox\tw@\hbox{\pxrr@x@K X}%
      \ifdim\wd\tw@>\wd\z@
        \aftergroup\pxrr@x@swatrue
      \fi
    \endgroup
    \edef#1{%
      \ifpxrr@x@swa \the\xkanjiskip
      \else \pxrr@zeropt
      \fi
    }%
  }
```

LuaTeX-ja 使用の場合。

```
\else\ifpxrr@in@luatexja
  \def\pxrr@get@iaiskip#1{%
    \edef#1{%
      \ifnum\ltjgetparameter{autoxspacing}=\@ne
        \ltjgetparameter{xkanjiskip}%
      \else \pxrr@zeropt
      \fi
    }%
  }
```

それ以外の場合は実際の組版結果から判断する。

```
\else
  \def\pxrr@get@iaiskip#1{%
```

```
    \begingroup
      \setbox\z@\hbox{M\pxrr@x@K}%
      \setbox\tw@\hbox{M\vrule\@width\z@\relax\pxrr@x@K}%
      \@tempdima\wd\z@ \advance\@tempdima-\wd\tw@
      \@tempdimb\@tempdima \divide\@tempdimb\thr@@
      \xdef\pxrr@x@gtempa{\the\@tempdima\space minus \the\@tempdimb}%
    \endgroup
    \let#1=\pxrr@x@gtempa
  }%
\fi\fi
```

\pxrr@get@zwidth `\pxrr@get@zwidth\CS`: 現在の和文フォントの全角幅を取得する。

pTEX の場合、`1zw` でよい。

```
\ifpxrr@in@ptex
  \def\pxrr@get@zwidth#1{%
    \@tempdima=1zw\relax
    \edef#1{\the\@tempdima}%
  }
```

`\zw` が定義されている場合は `1\zw` とする。

```
\else\if\ifx\zw\@undefined T\else F\fi F% if defined
  \def\pxrr@get@zwidth#1{%
    \@tempdima=1\zw\relax
    \edef#1{\the\@tempdima}%
  }
```

`\jsZw` が定義されている場合は `1\jsZw` とする。

```
\else\if\ifx\jsZw\@undefined T\else F\fi F% if defined
  \def\pxrr@get@zwidth#1{%
    \@tempdima=1\jsZw\relax
    \edef#1{\the\@tempdima}%
  }
```

それ以外で、`\pxrr@x@K` が有効な場合は実際の組版結果から判断する。

```
\else\ifnum\pxrr@x@K>\@cclv
  \def\pxrr@get@zwidth#1{%
    \setbox\tw@\hbox{\pxrr@x@K}%
    \@tempdima\wd\tw@
    \ifdim\@tempdima>\z@\else \@tempdima\f@size\p@ \fi
    \edef#1{\the\@tempdima}%
  }
```

それ以外の場合は要求サイズと等しいとする。

```
\else
  \def\pxrr@get@zwidth#1{%
    \@tempdima\f@size\p@\relax
    \edef#1{\the\@tempdima}%
  }
\fi\fi\fi\fi
```

\pxrr@get@prebreakpenalty `\pxrr@get@prebreakpenalty\CS{⟨文字コード⟩}`: 文字の後禁則ペナルティ値を整数レジスタに代入する。

pTEX の場合、\prebreakpenalty を使う。

```
\ifpxrr@in@ptex
  \def\pxrr@get@prebreakpenalty#1#2{%
    #1=\prebreakpenalty#2\relax
  }
```

LuaTeX-ja 使用時は、prebreakpenalty プロパティを読み出す。

```
\else\ifpxrr@in@luatexja
  \def\pxrr@get@prebreakpenalty#1#2{%
    #1=\ltjgetparameter{prebreakpenalty}{#2}\relax
  }
```

それ以外の場合はゼロとして扱う。

```
\else
  \def\pxrr@get@prebreakpenalty#1#2{%
    #1=\z@
  }
\fi\fi
```

\pxrr@get@postbreakpenalty `\pxrr@get@postbreakpenalty\CS{⟨文字コード⟩}`: 文字の前禁則ペナルティ値を整数レジスタに代入する。

pTEX の場合、\postbreakpenalty を使う。

```
\ifpxrr@in@ptex
  \def\pxrr@get@postbreakpenalty#1#2{%
    #1=\postbreakpenalty#2\relax
  }
```

LuaTeX-ja 使用時は、postbreakpenalty プロパティを読み出す。

```
\else\ifpxrr@in@luatexja
  \def\pxrr@get@postbreakpenalty#1#2{%
    #1=\ltjgetparameter{postbreakpenalty}{#2}\relax
  }
```

それ以外の場合はゼロとして扱う。

```
\else
  \def\pxrr@get@postbreakpenalty#1#2{%
    #1=\z@
  }
\fi\fi
```

\pxrr@check@punct@char `\pxrr@is@punct@char{⟨文字コード⟩}{⟨和文フラグ⟩}`: 指定の文字コードの文字が“約物であるか”を調べて、結果を \ifpxrr@ok に返す。⟨和文フラグ⟩ は“対象が pTEX の和文である”場合に 1、それ以外は 0。

pTEX の場合、欧文なら \xspcode、和文なら \inhibitxspcode の値を見て、それが 3 以外なら約物と見なす。

```
\ifpxrr@in@ptex
```

```
\def\pxrr@check@punct@char#1#2{%
  \pxrr@okfalse
  \ifcase#2\relax
    \ifnum\xspcode#1=\thr@@\else
      \pxrr@oktrue
    \fi
  \else
    \ifnum\inhibitxspcode#1=\thr@@\else
      \pxrr@oktrue
    \fi
  \fi
}
```

LuaTeX-ja 使用時も基本的に pTEX と同じロジックを使う。ただし LuaTeX-ja では「文字トークンの和文と欧文の区別」という概念が存在しないため、⟨和文フラグ⟩ は必ず 0 となる。そして、\xspcode／\inhibitxspcode に相当するパラメタとしては、欧文用の alxspmode と和文用の jaxspmode が一応あるが、実際には和文と欧文の区別はなくこの両者は同義になっている。従って、「jaxspmode が 3 以外か」を調べることにする。

```
\else\ifpxrr@in@luatexja
  \def\pxrr@check@punct@char#1#2{%
    \ifnum\ltjgetparameter{jaxspmode}{#1}=\thr@@
      \pxrr@okfalse
    \else
      \pxrr@oktrue
    \fi
  }
```

それ以外の場合はゼロとして扱う。

```
\else
  \def\pxrr@check@punct@char#1#2{%
    \pxrr@okfalse
  }
\fi\fi
```

\pxrr@inhibitglue　\inhibitglue が定義されているなら実行する。

```
\ifx\inhibitglue\@undefined
  \let\pxrr@inhibitglue\relax
\else
  \let\pxrr@inhibitglue\inhibitglue
\fi
```

## 4.7　パラメタ設定公開命令

\ifpxrr@in@setup　\pxrr@parse@option が \rubysetup の中で呼ばれたか。真の場合は警告処理を行わない。

```
\newif\ifpxrr@in@setup \pxrr@in@setupfalse
```

\rubysetup　\pxrr@parse@option で解析した後、設定値を全般設定にコピーする。

```
\newcommand*\rubysetup[1]{%
  \pxrr@in@setuptrue
  \pxrr@fatal@errorfalse
  \pxrr@parse@option{#1}%
  \ifpxrr@fatal@error\else
    \pxrr@csletcs{ifpxrr@d@bprotr}{ifpxrr@bprotr}%
    \pxrr@csletcs{ifpxrr@d@aprotr}{ifpxrr@aprotr}%
    \let\pxrr@d@bintr\pxrr@bintr@
    \let\pxrr@d@aintr\pxrr@aintr@
    \let\pxrr@d@athead\pxrr@athead
    \let\pxrr@d@mode\pxrr@mode
    \let\pxrr@d@side\pxrr@side
    \let\pxrr@d@evensp\pxrr@evensp
    \let\pxrr@d@fullsize\pxrr@fullsize
  \fi
```

\ifpxrr@in@setup を偽に戻す。ただし \ifpxrr@fatal@error は書き換えられたままであることに注意。

```
  \pxrr@in@setupfalse
}
```

\rubyfontsetup 対応するパラメタを設定する。

```
\newcommand*\rubyfontsetup{}
\def\rubyfontsetup#{%
  \def\pxrr@ruby@font
}
```

\rubybigintrusion 対応するパラメタを設定する。
\rubysmallintrusion
\rubymaxmargin
\rubyintergap
\rubysizeratio

```
\newcommand*\rubybigintrusion[1]{%
  \edef\pxrr@big@intr{#1}%
}
\newcommand*\rubysmallintrusion[1]{%
  \edef\pxrr@small@intr{#1}%
}
\newcommand*\rubymaxmargin[1]{%
  \edef\pxrr@maxmargin{#1}%
}
\newcommand*\rubyintergap[1]{%
  \edef\pxrr@inter@gap{#1}%
}
\newcommand*\rubysizeratio[1]{%
  \edef\pxrr@size@ratio{#1}%
}
```

\rubyusejghost 対応するスイッチを設定する。
\rubynousejghost

```
\newcommand*\rubyusejghost{%
  \pxrr@jghosttrue
}
\newcommand*\rubynousejghost{%
```

```
  \pxrr@jghostfalse
}
```

\rubyuseaghost
\rubynouseaghost

対応するスイッチを設定する。

```
\newcommand*\rubyuseaghost{%
  \pxrr@aghosttrue
  \pxrr@setup@aghost
}
\newcommand*\rubynouseaghost{%
  \pxrr@aghostfalse
}
```

\rubyadjustatlineedge
\rubynoadjustatlineedge

対応するスイッチを設定する。

```
\newcommand*\rubyadjustatlineedge{%
  \pxrr@edge@adjusttrue
}
\newcommand*\rubynoadjustatlineedge{%
  \pxrr@edge@adjustfalse
}
```

\rubybreakjukugo
\rubynobreakjukugo

対応するスイッチを設定する。

```
\newcommand*\rubybreakjukugo{%
  \pxrr@break@jukugotrue
}
\newcommand*\rubynobreakjukugo{%
  \pxrr@break@jukugofalse
}
```

\rubysafemode
\rubynosafemode

対応するスイッチを設定する。

```
\newcommand*\rubysafemode{%
  \pxrr@safe@modetrue
}
\newcommand*\rubynosafemode{%
  \pxrr@safe@modefalse
}
```

\rubystretchprop
\rubystretchprophead
\rubystretchpropend

対応するパラメタを設定する。

```
\newcommand*\rubystretchprop[3]{%
  \edef\pxrr@sprop@x{#1}%
  \edef\pxrr@sprop@y{#2}%
  \edef\pxrr@sprop@z{#3}%
}
\newcommand*\rubystretchprophead[2]{%
  \edef\pxrr@sprop@hy{#1}%
  \edef\pxrr@sprop@hz{#2}%
}
\newcommand*\rubystretchpropend[2]{%
  \edef\pxrr@sprop@ex{#1}%
  \edef\pxrr@sprop@ey{#2}%
```

```
}
```

\rubyuseextra 残念ながら今のところは使用不可。

```
\newcommand*\rubyuseextra[1]{%
  \pxrr@cnta=#1\relax
  \ifnum\pxrr@cnta=\z@
    \chardef\pxrr@extra\pxrr@cnta
  \else
    \pxrr@err@inv@value{\the\pxrr@cnta}%
  \fi
}
```

## 4.8 ルビオプション解析

\pxrr@bintr@ \pxrr@aintr@ オプション解析中にのみ使われ、進入の値を \pxrr@d@?intr と同じ形式で保持する。（\pxrr@?intr は形式が異なることに注意。）

```
\let\pxrr@bintr@\@empty
\let\pxrr@aintr@\@empty
```

\pxrr@doublebar \pxrr@parse@option 中で使用される。

```
\def\pxrr@doublebar{||}
```

\pxrr@parse@option \pxrr@parse@option{⟨オプション⟩}: ⟨オプション⟩ を解析し、\pxrr@athead や \pxrr@mode 等のパラメタを設定する。

```
\def\pxrr@parse@option#1{%
```

入力が「||」の場合は、「|-|」に置き換える。

```
  \edef\pxrr@tempa{#1}%
  \ifx\pxrr@tempa\pxrr@doublebar
    \def\pxrr@tempa{|-|}%
  \fi
```

各パラメタの値を全般設定のもので初期化する。

```
  \pxrr@csletcs{ifpxrr@bprotr}{ifpxrr@d@bprotr}%
  \pxrr@csletcs{ifpxrr@aprotr}{ifpxrr@d@aprotr}%
  \let\pxrr@bintr@\pxrr@d@bintr
  \let\pxrr@aintr@\pxrr@d@aintr
  \let\pxrr@athead\pxrr@d@athead
  \let\pxrr@mode\pxrr@d@mode
  \let\pxrr@side\pxrr@d@side
  \let\pxrr@evensp\pxrr@d@evensp
  \let\pxrr@fullsize\pxrr@d@fullsize
```

以下のパラメタの既定値は固定されている。

```
  \let\pxrr@bscomp\relax
  \let\pxrr@ascomp\relax
  \pxrr@bnobrfalse
  \pxrr@anobrfalse
```

```
  \pxrr@bfintrfalse
  \pxrr@afintrfalse
```
明示フラグを偽にする。
```
  \pxrr@mode@givenfalse
  \pxrr@athead@givenfalse
```
両側ルビの場合、基本モード既定値が M に固定される。
```
  \ifpxrr@truby
    \let\pxrr@mode=M%
  \fi
```
有限状態機械を開始させる。入力の末尾に @ を加えている。\pxrr@end はエラー時の脱出に用いる。
```
  \def\pxrr@po@FS{bi}%
  \expandafter\pxrr@parse@option@loop\pxrr@tempa @\pxrr@end
}
```
有限状態機械のループ。
```
\def\pxrr@parse@option@loop#1{%
\ifpxrrDebug
\typeout{\pxrr@po@FS/#1[\@nameuse{pxrr@po@C@#1}]}%
\fi
  \csname pxrr@po@PR@#1\endcsname
  \expandafter\ifx\csname pxrr@po@C@#1\endcsname\relax
    \let\pxrr@po@FS\relax
  \else
    \pxrr@letcs\pxrr@po@FS
     {pxrr@po@TR@\pxrr@po@FS @\@nameuse{pxrr@po@C@#1}}%
  \fi
\ifpxrrDebug
\typeout{->\pxrr@po@FS}%
\fi
  \pxrr@ifx{\pxrr@po@FS\relax}{%
    \pxrr@fatal@unx@letter{#1}%
    \pxrr@parse@option@exit
  }{%
    \pxrr@parse@option@loop
  }%
}
```
後処理。
```
\def\pxrr@parse@option@exit#1\pxrr@end{%
```
既定値設定（\rubysetup）である場合何もしない。
```
  \ifpxrr@in@setup\else
```
両側ルビ命令の場合は、\pxrr@side の値を変更する。
```
    \ifpxrr@truby
      \chardef\pxrr@side\tw@
    \fi
```

整合性検査を行う。

```
    \pxrr@check@option
```

\pxrr@?intr の値を設定する。

```
    \@tempdima=\pxrr@ruby@zw\relax
    \@tempdimb=\pxrr@or@zero\pxrr@bintr@\@tempdima
    \edef\pxrr@bintr{\the\@tempdimb}%
    \@tempdimb=\pxrr@or@zero\pxrr@aintr@\@tempdima
    \edef\pxrr@aintr{\the\@tempdimb}%
  \fi
}
```

\pxrr@or@zero　\pxrr@or@zero\pxrr@?intr@ とすると、\pxrr@?intr@ が空の時に代わりにゼロと扱う。

```
\def\pxrr@or@zero#1{%
  \ifx#1\@empty \pxrr@zero
  \else #1%
  \fi
}
```

以下はオプション解析の有限状態機械の定義。

記号のクラスの設定。

```
\def\pxrr@po@C@@{F}
\@namedef{pxrr@po@C@|}{V}
\@namedef{pxrr@po@C@:}{S}
\@namedef{pxrr@po@C@.}{S}
\@namedef{pxrr@po@C@*}{S}
\@namedef{pxrr@po@C@!}{S}
\@namedef{pxrr@po@C@<}{B}
\@namedef{pxrr@po@C@(}{B}
\@namedef{pxrr@po@C@>}{A}
\@namedef{pxrr@po@C@)}{A}
\@namedef{pxrr@po@C@-}{M}
\def\pxrr@po@C@c{M}
\def\pxrr@po@C@h{M}
\def\pxrr@po@C@H{M}
\def\pxrr@po@C@m{M}
\def\pxrr@po@C@g{M}
\def\pxrr@po@C@j{M}
\def\pxrr@po@C@M{M}
\def\pxrr@po@C@J{M}
\def\pxrr@po@C@P{M}
\def\pxrr@po@C@S{M}
\def\pxrr@po@C@e{M}
\def\pxrr@po@C@E{M}
\def\pxrr@po@C@f{M}
\def\pxrr@po@C@F{M}
```

機能プロセス。

```
\def\pxrr@po@PR@@{%
```

```
  \pxrr@parse@option@exit
}
\@namedef{pxrr@po@PR@|}{%
  \csname pxrr@po@PRbar@\pxrr@po@FS\endcsname
}
\def\pxrr@po@PRbar@bi{%
  \def\pxrr@bintr@{}\pxrr@bprotrtrue
}
\def\pxrr@po@PRbar@bb{%
  \pxrr@bprotrfalse
}
\def\pxrr@po@PRbar@bs{%
  \def\pxrr@aintr@{}\pxrr@aprotrtrue
}
\let\pxrr@po@PRbar@mi\pxrr@po@PRbar@bs
\let\pxrr@po@PRbar@as\pxrr@po@PRbar@bs
\let\pxrr@po@PRbar@ai\pxrr@po@PRbar@bs
\def\pxrr@po@PRbar@ab{%
  \pxrr@aprotrfalse
}
\@namedef{pxrr@po@PR@:}{%
  \csname pxrr@po@PRcolon@\pxrr@po@FS\endcsname
}
\def\pxrr@po@PRcolon@bi{%
  \let\pxrr@bscomp=:\relax
}
\let\pxrr@po@PRcolon@bb\pxrr@po@PRcolon@bi
\let\pxrr@po@PRcolon@bs\pxrr@po@PRcolon@bi
\def\pxrr@po@PRcolon@mi{%
  \let\pxrr@ascomp=:\relax
}
\let\pxrr@po@PRcolon@as\pxrr@po@PRcolon@mi
\@namedef{pxrr@po@PR@.}{%
  \csname pxrr@po@PRdot@\pxrr@po@FS\endcsname
}
\def\pxrr@po@PRdot@bi{%
  \let\pxrr@bscomp=.\relax
}
\let\pxrr@po@PRdot@bb\pxrr@po@PRdot@bi
\let\pxrr@po@PRdot@bs\pxrr@po@PRdot@bi
\def\pxrr@po@PRdot@mi{%
  \let\pxrr@ascomp=.\relax
}
\let\pxrr@po@PRdot@as\pxrr@po@PRdot@mi
\@namedef{pxrr@po@PR@*}{%
  \csname pxrr@po@PRstar@\pxrr@po@FS\endcsname
}
\def\pxrr@po@PRstar@bi{%
  \pxrr@bnobrtrue
```

```
}
\let\pxrr@po@PRstar@bb\pxrr@po@PRstar@bi
\let\pxrr@po@PRstar@bs\pxrr@po@PRstar@bi
\def\pxrr@po@PRstar@mi{%
  \pxrr@anobrtrue
}
\let\pxrr@po@PRstar@as\pxrr@po@PRstar@mi
\@namedef{pxrr@po@PR@!}{%
  \csname pxrr@po@PRbang@\pxrr@po@FS\endcsname
}
\def\pxrr@po@PRbang@bi{%
  \pxrr@bfintrtrue
}
\let\pxrr@po@PRbang@bb\pxrr@po@PRbang@bi
\let\pxrr@po@PRbang@bs\pxrr@po@PRbang@bi
\def\pxrr@po@PRbang@mi{%
  \pxrr@afintrtrue
}
\let\pxrr@po@PRbang@as\pxrr@po@PRbang@mi
\@namedef{pxrr@po@PR@<}{%
  \def\pxrr@bintr@{\pxrr@big@intr}\pxrr@bprotrtrue
}
\@namedef{pxrr@po@PR@(}{%
  \def\pxrr@bintr@{\pxrr@small@intr}\pxrr@bprotrtrue
}
\@namedef{pxrr@po@PR@>}{%
  \def\pxrr@aintr@{\pxrr@big@intr}\pxrr@aprotrtrue
}
\@namedef{pxrr@po@PR@)}{%
  \def\pxrr@aintr@{\pxrr@small@intr}\pxrr@aprotrtrue
}
\def\pxrr@po@PR@c{%
  \chardef\pxrr@athead\z@
  \pxrr@athead@giventrue
}
\def\pxrr@po@PR@h{%
  \chardef\pxrr@athead\@ne
  \pxrr@athead@giventrue
}
\def\pxrr@po@PR@H{%
  \chardef\pxrr@athead\tw@
  \pxrr@athead@giventrue
}
\def\pxrr@po@PR@m{%
  \let\pxrr@mode=m%
  \pxrr@mode@giventrue
}
\def\pxrr@po@PR@g{%
  \let\pxrr@mode=g%
```

```
  \pxrr@mode@giventrue
}
\def\pxrr@po@PR@j{%
  \let\pxrr@mode=j%
  \pxrr@mode@giventrue
}
\def\pxrr@po@PR@M{%
  \let\pxrr@mode=M%
  \pxrr@mode@giventrue
}
\def\pxrr@po@PR@J{%
  \let\pxrr@mode=J%
  \pxrr@mode@giventrue
}
\def\pxrr@po@PR@P{%
  \chardef\pxrr@side\z@
}
\def\pxrr@po@PR@S{%
  \chardef\pxrr@side\@ne
}
\def\pxrr@po@PR@E{%
  \chardef\pxrr@evensp\z@
}
\def\pxrr@po@PR@e{%
  \chardef\pxrr@evensp\@ne
}
\def\pxrr@po@PR@F{%
  \chardef\pxrr@fullsize\z@
}
\def\pxrr@po@PR@f{%
  \chardef\pxrr@fullsize\@ne
}
```

遷移表。

```
\def\pxrr@po@TR@bi@F{fi}
\def\pxrr@po@TR@bb@F{fi}
\def\pxrr@po@TR@bs@F{fi}
\def\pxrr@po@TR@mi@F{fi}
\def\pxrr@po@TR@as@F{fi}
\def\pxrr@po@TR@ai@F{fi}
\def\pxrr@po@TR@ab@F{fi}
\def\pxrr@po@TR@fi@F{fi}
\def\pxrr@po@TR@bi@V{bb}
\def\pxrr@po@TR@bb@V{bs}
\def\pxrr@po@TR@bs@V{ab}
\def\pxrr@po@TR@mi@V{ab}
\def\pxrr@po@TR@as@V{ab}
\def\pxrr@po@TR@ai@V{ab}
\def\pxrr@po@TR@ab@V{fi}
```

```
\def\pxrr@po@TR@bi@S{bs}
\def\pxrr@po@TR@bb@S{bs}
\def\pxrr@po@TR@bs@S{bs}
\def\pxrr@po@TR@mi@S{as}
\def\pxrr@po@TR@as@S{as}
\def\pxrr@po@TR@bi@B{bs}
\def\pxrr@po@TR@bi@M{mi}
\def\pxrr@po@TR@bb@M{mi}
\def\pxrr@po@TR@bs@M{mi}
\def\pxrr@po@TR@mi@M{mi}
\def\pxrr@po@TR@bi@A{fi}
\def\pxrr@po@TR@bb@A{fi}
\def\pxrr@po@TR@bs@A{fi}
\def\pxrr@po@TR@mi@A{fi}
\def\pxrr@po@TR@as@A{fi}
\def\pxrr@po@TR@ai@A{fi}
```

## 4.9 オプション整合性検査

`\pxrr@mode@grand` 基本モードの“大分類”。モノ（m）・熟語（j）・グループ（g）の何れか。つまり“選択的”設定の M・J を m・j に寄せる。

※ 完全展開可能であるが、“先頭完全展開可能”でないことに注意。

```
\def\pxrr@mode@grand{%
  \if      m\pxrr@mode m%
  \else\if M\pxrr@mode m%
  \else\if j\pxrr@mode j%
  \else\if J\pxrr@mode j%
  \else\if g\pxrr@mode g%
  \else ?%
  \fi\fi\fi\fi\fi
}
```

`\pxrr@check@option` `\pxrr@parse@option` の結果であるオプション設定値の整合性を検査し、必要に応じて、致命的エラーを出したり、警告を出して適切な値に変更したりする。

```
\def\pxrr@check@option{%
```

前と後の両方で突出が禁止された場合は致命的エラーとする。

```
  \ifpxrr@bprotr\else
    \ifpxrr@aprotr\else
      \pxrr@fatal@bad@no@protr
    \fi
  \fi
```

ゴースト処理有効で進入有りの場合は致命的エラーとする。

```
  \pxrr@oktrue
  \ifx\pxrr@bintr@\@empty\else
    \pxrr@okfalse
  \fi
```

```
\ifx\pxrr@aintr@\@empty\else
  \pxrr@okfalse
\fi
\ifpxrr@ghost\else
  \pxrr@oktrue
\fi
\ifpxrr@ok\else
  \pxrr@fatal@bad@intr
\fi
```

欧文ルビではモノルビ（m）・熟語ルビ（j）は指定不可なので、グループルビに変更する。この時に明示指定である場合は警告を出す。

```
\if g\pxrr@mode\else
  \ifpxrr@abody
    \let\pxrr@mode=g\relax
    \ifpxrr@mode@given
      \pxrr@warn@must@group
    \fi
  \fi
\fi
```

両側ルビでは熟語ルビ（j）は指定不可なので、グループルビに変更する。この時に明示指定である場合は警告を出す。

```
\if \pxrr@mode@grand j%
  \ifnum\pxrr@side=\tw@
    \let\pxrr@mode=g\relax
    \ifpxrr@mode@given
      \pxrr@warn@bad@jukugo
    \fi
  \fi
\fi
```

肩付き指定（h）に関する検査。

```
\ifnum\pxrr@athead>\z@
```

横組みでは不可なので中付きに変更する。

```
  \pxrr@if@in@tate{}{%else
    \chardef\pxrr@athead\z@
  }%
```

グループルビでは不可なので中付きに変更する。

```
  \if g\pxrr@mode
    \chardef\pxrr@athead\z@
  \fi
```

以上の 2 つの場合について、明示指定であれば警告を出す。

```
  \ifnum\pxrr@athead=\z@
    \ifpxrr@athead@given
      \pxrr@warn@bad@athead
    \fi
```

```
    \fi
  \fi
```

親文字列均等割り抑止（E）の再設定（エラー・警告なし）。
欧文ルビの場合は、均等割りを常に無効にする。

```
  \ifpxrr@abody
    \chardef\pxrr@evensp\z@
  \fi
```

グループルビ以外では、均等割りを有効にする。（この場合、親文字列は一文字毎に分解されるので、意味はもたない。均等割り抑止の方が特殊な処理なので、通常の処理に合わせる。）

```
  \if g\pxrr@mode\else
    \chardef\pxrr@evensp\@ne
  \fi
```

圏点ルビ同時付加の場合の調整。

```
    \ifpxrr@combo
      \pxrr@ck@check@option
    \fi
}
```

## 4.10 フォントサイズ

`\pxrr@ruby@fsize` ルビ文字の公称サイズ。寸法値マクロ。ルビ命令呼出時に `\f@size`（親文字の公称サイズ）の `\pxrr@size@ratio` 倍に設定される。

```
\let\pxrr@ruby@fsize\pxrr@zeropt
```

`\pxrr@body@zw` `\pxrr@ruby@zw` それぞれ、親文字とルビ文字の全角幅（実際の 1 zw の寸法）。寸法値マクロ。pTEX では和文と欧文のバランスを整えるために和文を縮小することが多く、その場合「全角幅」は「公称サイズ」より小さくなる。なお、このパッケージでは漢字の幅が 1 zw であることを想定する。これらもルビ命令呼出時に正しい値に設定される。

```
\let\pxrr@body@zw\pxrr@zeropt
\let\pxrr@ruby@zw\pxrr@zeropt
```

`\pxrr@ruby@raise` ルビ文字に対する垂直方向の移動量。

```
\let\pxrr@ruby@raise\pxrr@zeropt
```

`\pxrr@ruby@lower` ルビ文字に対する垂直方向の移動量（下側ルビ）。

```
\let\pxrr@ruby@lower\pxrr@zeropt
```

`\pxrr@htratio` 現在の組方向により、`\pxrr@yhtratio` と `\pxrr@thtratio` のいずれか一方に設定される。

```
\def\pxrr@htratio{0}
```

`\pxrr@iiskip` `\pxrr@iaiskip` 和文間空白および和欧文間空白の量。

```
\let\pxrr@iiskip\pxrr@zeropt
\let\pxrr@iaiskip\pxrr@zeropt
```

\pxrr@assign@fsize　上記の変数（マクロ）を設定する。

```
\def\pxrr@assign@fsize{%
  \@tempdima=\f@size\p@
  \@tempdima\pxrr@c@size@ratio\@tempdima
  \edef\pxrr@ruby@fsize{\the\@tempdima}%
  \pxrr@get@zwidth\pxrr@body@zw
  \begingroup
    \pxrr@use@ruby@font
    \pxrr@get@zwidth\pxrr@ruby@zw
    \global\let\pxrr@gtempa\pxrr@ruby@zw
  \endgroup
  \let\pxrr@ruby@zw\pxrr@gtempa
  \pxrr@get@iiskip\pxrr@iiskip
  \pxrr@get@iaiskip\pxrr@iaiskip
```

\pxrr@htratio の値を設定する。

```
  \pxrr@if@in@tate{%
    \let\pxrr@htratio\pxrr@thtratio
  }{%
    \let\pxrr@htratio\pxrr@yhtratio
  }%
```

\pxrr@ruby@raise の値を計算する。

```
  \@tempdima\pxrr@body@zw\relax
  \@tempdima\pxrr@htratio\@tempdima
  \@tempdimb\pxrr@ruby@zw\relax
  \advance\@tempdimb-\pxrr@htratio\@tempdimb
  \advance\@tempdima\@tempdimb
  \@tempdimb\pxrr@body@zw\relax
  \advance\@tempdima\pxrr@c@inter@gap\@tempdimb
  \edef\pxrr@ruby@raise{\the\@tempdima}%
```

\pxrr@ruby@lower の値を計算する。

```
  \@tempdima\pxrr@body@zw\relax
  \advance\@tempdima-\pxrr@htratio\@tempdima
  \@tempdimb\pxrr@ruby@zw\relax
  \@tempdimb\pxrr@htratio\@tempdimb
  \advance\@tempdima\@tempdimb
  \@tempdimb\pxrr@body@zw\relax
  \advance\@tempdima\pxrr@c@inter@gap\@tempdimb
  \edef\pxrr@ruby@lower{\the\@tempdima}%
```

圏点ルビ同時付加の設定。

```
  \ifpxrr@combo
    \pxrr@ck@assign@fsize
  \fi
}
```

\pxrr@use@ruby@font　ルビ用のフォントに切り替える。

```
\def\pxrr@use@ruby@font{%
```

```
  \pxrr@without@macro@trace{%
    \let\rubyfontsize\pxrr@ruby@fsize
    \fontsize{\pxrr@ruby@fsize}{\z@}\selectfont
    \pxrr@c@ruby@font
  }%
}
```

## 4.11 ルビ用均等割り

\pxrr@locate@inner
\pxrr@locate@head
\pxrr@locate@end

ルビ配置パターン（行頭／行中／行末）を表す定数。

```
\chardef\pxrr@locate@inner=1
\chardef\pxrr@locate@head=0
\chardef\pxrr@locate@end=2
```

\pxrr@evenspace
\pxrr@evenspace@int
\pxrr@makebox@res

\pxrr@evenspace{⟨パターン⟩}\CS{⟨フォント⟩}{⟨幅⟩}{⟨テキスト⟩}：⟨テキスト⟩ を指定の ⟨幅⟩ に対する ⟨パターン⟩（行頭／行中／行末）の「行中ルビ用均等割り」で配置し、結果をボックスレジスタ \CS に代入する。均等割りの要素分割は \pxrr@decompose を用いて行われるので、要素数が \pxrr@cntr に返る。また、先頭と末尾の空きの量をそれぞれ \pxrr@bspace と \pxrr@aspace に代入する。

\pxrr@evenspace@int{⟨パターン⟩}\CS{⟨フォント⟩}{⟨幅⟩}： \pxrr@evenspace の実行を、

> \pxrr@res と \pxrr@cntr にテキストの \pxrr@decompose の結果が入っていて、テキストの自然長がマクロ \pxrr@natwd に入っている

という状態で、途中から開始する。

```
\def\pxrr@evenspace#1#2#3#4#5{%
```

⟨テキスト⟩ の自然長を計測し、\pxrr@natwd に格納する。

```
  \setbox#2\pxrr@hbox{#5}\@tempdima\wd#2%
  \edef\pxrr@natwd{\the\@tempdima}%
```

⟨テキスト⟩ をリスト解析する（\pxrr@cntr に要素数が入る）。\pxrr@evenspace@int に引き継ぐ。

```
  \pxrr@decompose{#5}%
  \pxrr@evenspace@int{#1}{#2}{#3}{#4}%
}
```

ここから実行を開始することもある。

```
\def\pxrr@evenspace@int#1#2#3#4{%
```

比率パラメタの設定。

```
  \pxrr@save@listproc
  \ifcase#1%
    \pxrr@evenspace@param\pxrr@zero\pxrr@sprop@hy\pxrr@sprop@hz
  \or
    \pxrr@evenspace@param\pxrr@sprop@x\pxrr@sprop@y\pxrr@sprop@z
  \or
```

```
    \pxrr@evenspace@param\pxrr@sprop@ex\pxrr@sprop@ey\pxrr@zero
  \fi
```

挿入される fil の係数を求め、これがゼロの場合（この時 X = Z = 0 である）は、アンダーフル防止のため、X = Z = 1 に変更する。

```
  \pxrr@dima=\pxrr@cntr\p@
  \advance\pxrr@dima-\p@
  \pxrr@dima=\pxrr@sprop@y@\pxrr@dima
  \advance\pxrr@dima\pxrr@sprop@x@\p@
  \advance\pxrr@dima\pxrr@sprop@z@\p@
  \ifdim\pxrr@dima>\z@\else
    \ifnum#1>\z@
      \let\pxrr@sprop@x@\@ne
      \advance\pxrr@dima\p@
    \fi
    \ifnum#1<\tw@
      \let\pxrr@sprop@z@\@ne
      \advance\pxrr@dima\p@
    \fi
  \fi
  \edef\pxrr@tempa{\strip@pt\pxrr@dima}%
\ifpxrrDebug
\typeout{\number\pxrr@sprop@x@:\number\pxrr@sprop@z@:\pxrr@tempa}%
\fi
```

\pxrr@pre/inter/post にグルーを設定して、\pxrr@res を組版する。なお、\setbox... を一旦マクロ \pxrr@makebox@res に定義しているのは、後で \pxrr@adjust@margin で再度呼び出せるようにするため。

```
  \def\pxrr@pre##1{\pxrr@hfilx\pxrr@sprop@x@ ##1}%
  \def\pxrr@inter##1{\pxrr@hfilx\pxrr@sprop@y@ ##1}%
  \def\pxrr@post{\pxrr@hfilx\pxrr@sprop@z@}%
  \def\pxrr@makebox@res{%
    \setbox#2=\pxrr@hbox@to#4{#3\pxrr@res}%
  }%
  \pxrr@makebox@res
```

前後の空白の量を求める。

```
  \pxrr@dima\wd#2%
  \advance\pxrr@dima-\pxrr@natwd\relax
  \pxrr@invscale\pxrr@dima\pxrr@tempa
  \@tempdima\pxrr@sprop@x@\pxrr@dima
  \edef\pxrr@bspace{\the\@tempdima}%
  \@tempdima\pxrr@sprop@z@\pxrr@dima
  \edef\pxrr@aspace{\the\@tempdima}%
  \pxrr@restore@listproc
\ifpxrrDebug
\typeout{\pxrr@bspace:\pxrr@aspace}%
\fi
}
```

```
\def\pxrr@evenspace@param#1#2#3{%
  \let\pxrr@sprop@x@#1%
  \let\pxrr@sprop@y@#2%
  \let\pxrr@sprop@z@#3%
}
\let\pxrr@makebox@res\@undefined
```

\pxrr@adjust@margin　\pxrr@adjust@margin： \pxrr@evenspace(@int) を呼び出した直後に呼ぶ必要がある。先頭と末尾の各々について、空きの量が \pxrr@maxmargin により決まる上限値を超える場合に、空きを上限値に抑えるように再調整する。

```
\def\pxrr@adjust@margin{%
  \pxrr@save@listproc
  \@tempdima\pxrr@body@zw\relax
  \@tempdima\pxrr@maxmargin\@tempdima
```

再調整が必要かを \if@tempswa に記録する。1 文字しかない場合は調整不能だから検査を飛ばす。

```
  \@tempswafalse
  \def\pxrr@pre##1{\pxrr@hfilx\pxrr@sprop@x@ ##1}%
  \def\pxrr@inter##1{\pxrr@hfilx\pxrr@sprop@y@ ##1}%
  \def\pxrr@post{\pxrr@hfilx\pxrr@sprop@z@}%
  \ifnum\pxrr@cntr>\@ne
    \ifdim\pxrr@bspace>\@tempdima
      \edef\pxrr@bspace{\the\@tempdima}%
      \def\pxrr@pre##1{\hskip\pxrr@bspace\relax ##1}%
      \@tempswatrue
    \fi
    \ifdim\pxrr@aspace>\@tempdima
      \edef\pxrr@aspace{\the\@tempdima}%
      \def\pxrr@post{\hskip\pxrr@aspace\relax}%
      \@tempswatrue
    \fi
  \fi
```

必要に応じて再調整を行う。

```
  \if@tempswa
    \pxrr@makebox@res
  \fi
  \pxrr@restore@listproc
\ifpxrrDebug
\typeout{\pxrr@bspace:\pxrr@aspace}%
\fi
}
```

\pxrr@save@listproc　\pxrr@pre/inter/post の定義を退避する。

※ 退避のネストはできない。

```
\def\pxrr@save@listproc{%
  \let\pxrr@pre@save\pxrr@pre
  \let\pxrr@inter@save\pxrr@inter
```

```
  \let\pxrr@post@save\pxrr@post
}
\let\pxrr@pre@save\@undefined
\let\pxrr@inter@save\@undefined
\let\pxrr@post@save\@undefined
```

\pxrr@restore@listproc　\pxrr@pre/inter/post の定義を復帰する。

```
\def\pxrr@restore@listproc{%
  \let\pxrr@pre\pxrr@pre@save
  \let\pxrr@inter\pxrr@inter@save
  \let\pxrr@post\pxrr@post@save
}
```

## 4.12　小書き仮名の変換

\pxrr@trans@res　\pxrr@transform@kana 内で変換結果を保持するマクロ。

```
\let\pxrr@trans@res\@empty
```

\pxrr@transform@kana　\pxrr@transform@kana\CS： マクロ \CS の展開テキストの中でグループに含まれない小書き仮名を対応する非小書き仮名に変換し、\CS を上書きする。

```
\def\pxrr@transform@kana#1{%
  \let\pxrr@trans@res\@empty
  \def\pxrr@transform@kana@end\pxrr@end{%
    \let#1\pxrr@trans@res
  }%
  \expandafter\pxrr@transform@kana@loop@a#1\pxrr@end
}
\def\pxrr@transform@kana@loop@a{%
  \futurelet\pxrr@token\pxrr@transform@kana@loop@b
}
\def\pxrr@transform@kana@loop@b{%
  \ifx\pxrr@token\pxrr@end
    \let\pxrr@tempb\pxrr@transform@kana@end
  \else\ifx\pxrr@token\bgroup
    \let\pxrr@tempb\pxrr@transform@kana@loop@c
  \else\ifx\pxrr@token\@sptoken
    \let\pxrr@tempb\pxrr@transform@kana@loop@d
  \else
    \let\pxrr@tempb\pxrr@transform@kana@loop@e
  \fi\fi\fi
  \pxrr@tempb
}
\def\pxrr@transform@kana@loop@c#1{%
  \pxrr@appto\pxrr@trans@res{{#1}}%
  \pxrr@transform@kana@loop@a
}
\expandafter\def\expandafter\pxrr@transform@kana@loop@d\space{%
  \pxrr@appto\pxrr@trans@res{ }%
```

```
  \pxrr@transform@kana@loop@a
}
\def\pxrr@transform@kana@loop@e#1{%
  \expandafter\pxrr@transform@kana@loop@f\string#1\pxrr@nil#1%
}
\def\pxrr@transform@kana@loop@f#1#2\pxrr@nil#3{%
  \@tempswafalse
  \ifnum`#1>\@cclv
    \begingroup\expandafter\expandafter\expandafter\endgroup
    \expandafter\ifx\csname pxrr@nonsmall/#3\endcsname\relax\else
      \@tempswatrue
    \fi
  \fi
  \if@tempswa
    \edef\pxrr@tempa{%
      \noexpand\pxrr@appto\noexpand\pxrr@trans@res
       {\csname pxrr@nonsmall/#3\endcsname}%
    }%
    \pxrr@tempa
  \else
    \pxrr@appto\pxrr@trans@res{#3}%
  \fi
  \pxrr@transform@kana@loop@a
}
\def\pxrr@assign@nonsmall#1/#2\pxrr@nil{%
  \pxrr@get@jchar@token\pxrr@tempa{\pxrr@jc{#1}}%
  \pxrr@get@jchar@token\pxrr@tempb{\pxrr@jc{#2}}%
  \expandafter\edef\csname pxrr@nonsmall/\pxrr@tempa\endcsname
   {\pxrr@tempb}%
}
\@tfor\pxrr@tempc:=%
    {2421:3041/2422:3042}{2423:3043/2424:3044}%
    {2425:3045/2426:3046}{2427:3047/2428:3048}%
    {2429:3049/242A:304A}{2443:3063/2444:3064}%
    {2463:3083/2464:3084}{2465:3085/2466:3086}%
    {2467:3087/2468:3088}{246E:308E/246F:308F}%
    {2521:30A1/2522:30A2}{2523:30A3/2524:30A4}%
    {2525:30A5/2526:30A6}{2527:30A7/2528:30A8}%
    {2529:30A9/252A:30AA}{2543:30C3/2544:30C4}%
    {2563:30E3/2564:30E4}{2565:30E5/2566:30E6}%
    {2567:30E7/2568:30E8}{256E:30EE/256F:30EF}%
  \do{%
  \expandafter\pxrr@assign@nonsmall\pxrr@tempc\pxrr@nil
}
```

## 4.13 ブロック毎の組版

\ifpxrr@protr ルビ文字列の突出があるか。スイッチ。

```
\newif\ifpxrr@protr
```

\ifpxrr@any@protr 複数ブロックの処理で、いずれかのブロックにルビ文字列の突出があるか。スイッチ。

```
\newif\ifpxrr@any@protr
```

\pxrr@locate@temp \pxrr@compose@*side@block@do で使われる一時変数。整数定数。

```
\let\pxrr@locate@temp\relax
```

\pxrr@epsilon ルビ文字列と親文字列の自然長の差がこの値以下の場合は、差はないものとみなす（演算誤差対策）。

```
\def\pxrr@epsilon{0.01pt}
```

\pxrr@compose@block \pxrr@compose@block{⟨パターン⟩}{⟨親文字ブロック⟩}{⟨ルビ文字ブロック⟩}： 1 つのブロックの組版処理。⟨パターン⟩ は \pxrr@evenspace と同じ意味。突出があるかを \ifpxrr@protr に返し、前と後の突出の量をそれぞれ \pxrr@bspace と \pxrr@aspace に返す。

```
\def\pxrr@compose@block#1#2#3{%
```

本体の前に加工処理を介入させる。

※ \pxrr@compose@block@pre は 2 つのルビ引数を取る。\pxrr@compose@block@do に本体マクロを \let する。

```
  \let\pxrr@compose@block@do\pxrr@compose@oneside@block@do
  \pxrr@compose@block@pre{#1}{#2}{#3}{}%
}
```

こちらが本体。

```
% #4 は空
\def\pxrr@compose@oneside@block@do#1#2#3#4{%
  \setbox\pxrr@boxa\pxrr@hbox{#2}%
  \edef\pxrr@ck@body@natwd{\the\wd\pxrr@boxa}%
  \let\pxrr@ck@locate\pxrr@locate@inner
  \setbox\pxrr@boxr\pxrr@hbox{%
    \pxrr@use@ruby@font
    #3%
  }%
  \@tempdima\wd\pxrr@boxr
  \advance\@tempdima-\wd\pxrr@boxa
  \ifdim\pxrr@epsilon<\@tempdima
```

ルビ文字列の方が長い場合。親文字列をルビ文字列の長さに合わせて均等割りで組み直す。\pxrr@?space は \pxrr@evenspace@int が返す値のままでよい。「拡張肩付き」指定の場合、前側の突出を抑止する。

```
    \pxrr@protrtrue
    \let\pxrr@locate@temp#1%
```

```
    \ifnum\pxrr@athead>\@ne
      \ifnum\pxrr@locate@temp=\pxrr@locate@inner
        \let\pxrr@locate@temp\pxrr@locate@head
      \fi
    \fi
    \let\pxrr@ck@locate\pxrr@locate@temp
    \pxrr@decompose{#2}%
    \edef\pxrr@natwd{\the\wd\pxrr@boxa}%
    \pxrr@evenspace@int\pxrr@locate@temp\pxrr@boxa\relax
     {\wd\pxrr@boxr}%
  \else\ifdim-\pxrr@epsilon>\@tempdima
```

ルビ文字列の方が短い場合。ルビ文字列を親文字列の長さに合わせて均等割りで組み直す。この場合、\pxrr@maxmargin を考慮する必要がある。ただし肩付きルビの場合は組み直しを行わない。\pxrr@?space はゼロに設定する。

```
    \pxrr@protrfalse
    \ifnum\pxrr@athead=\z@
      \pxrr@decompose{#3}%
      \edef\pxrr@natwd{\the\wd\pxrr@boxr}%
      \pxrr@evenspace@int{#1}\pxrr@boxr
       \pxrr@use@ruby@font{\wd\pxrr@boxa}%
      \pxrr@adjust@margin
    \fi
    \let\pxrr@bspace\pxrr@zeropt
    \let\pxrr@aspace\pxrr@zeropt
  \else
```

両者の長さが等しい（とみなす）場合。突出フラグは常に偽にする（実際にはルビの方が僅かだけ長いかも知れないが）。

```
    \pxrr@protrfalse
    \let\pxrr@bspace\pxrr@zeropt
    \let\pxrr@aspace\pxrr@zeropt
  \fi\fi
```

実際に組版を行う。

```
  \setbox\z@\hbox{%
    \ifnum\pxrr@side=\z@
      \raise\pxrr@ruby@raise\box\pxrr@boxr
    \else
      \lower\pxrr@ruby@lower\box\pxrr@boxr
    \fi
  }%
  \ifnum \ifpxrr@combo\pxrr@ck@ruby@combo\else\z@\fi >\z@
    \pxrr@ck@compose{#2}%
  \fi
  \ht\z@\z@ \dp\z@\z@
  \@tempdima\wd\z@
  \setbox\pxrr@boxr\hbox{%
    \box\z@
```

```
    \kern-\@tempdima
    \box\pxrr@boxa
  }%
```

\ifpxrr@any@protr を設定する。

```
  \ifpxrr@protr
    \pxrr@any@protrtrue
  \fi
}
```

\pxrr@compose@twoside@block　両側ルビ用のブロック構成。

```
\def\pxrr@compose@twoside@block{%
  \let\pxrr@compose@block@do\pxrr@compose@twoside@block@do
  \pxrr@compose@block@pre
}
\def\pxrr@compose@twoside@block@do#1#2#3#4{%
```

\pxrr@boxa に親文字、\pxrr@boxr に上側ルビ、\pxrr@boxb に下側ルビの出力を保持する。

```
  \setbox\pxrr@boxa\pxrr@hbox{#2}%
  \edef\pxrr@ck@body@natwd{\the\wd\pxrr@boxa}%
  \let\pxrr@ck@locate\pxrr@locate@inner
  \setbox\pxrr@boxr\pxrr@hbox{%
    \pxrr@use@ruby@font
    #3%
  }%
  \setbox\pxrr@boxb\pxrr@hbox{%
    \pxrr@use@ruby@font
    #4%
  }%
```

「何れかのルビが親文字列より長いか」を検査する。

```
  \@tempswafalse
  \@tempdima\wd\pxrr@boxr
  \advance\@tempdima-\wd\pxrr@boxa
  \ifdim\pxrr@epsilon<\@tempdima \@tempswatrue \fi
  \@tempdima\wd\pxrr@boxb
  \advance\@tempdima-\wd\pxrr@boxa
  \ifdim\pxrr@epsilon<\@tempdima \@tempswatrue \fi
```

親文字より長いルビが存在する場合。長い方のルビ文字列の長さに合わせて、親文字列と他方のルビ文字列を組み直す。（実際の処理は \pxrr@compose@twoside@block@sub で行う。）

```
  \if@tempswa
    \pxrr@protrtrue
```

「拡張肩付き」指定の場合、前側の突出を抑止する。

```
    \let\pxrr@locate@temp#1%
    \ifnum\pxrr@athead>\@ne
      \ifnum\pxrr@locate@temp=\pxrr@locate@inner
```

```
        \let\pxrr@locate@temp\pxrr@locate@head
      \fi
    \fi
    \let\pxrr@ck@locate\pxrr@locate@temp
```

上側と下側のどちらのルビが長いかに応じて引数を変えて、\pxrr@compose@twoside@block@sub を呼び出す。

```
    \ifdim\wd\pxrr@boxr<\wd\pxrr@boxb
      \pxrr@compose@twoside@block@sub{#2}{#3}%
       \pxrr@boxr\pxrr@boxb
    \else
      \pxrr@compose@twoside@block@sub{#2}{#4}%
       \pxrr@boxb\pxrr@boxr
    \fi
```

親文字の方が長い場合。親文字列の長さに合わせて、両方のルビを（片側の場合と同様の）均等割りで組み直す。

```
  \else
    \pxrr@protrfalse
```

肩付きルビの場合は組み直しを行わない。

```
    \ifnum\pxrr@athead=\z@
      \@tempdima\wd\pxrr@boxa
      \advance\@tempdima-\wd\pxrr@boxr
      \ifdim\pxrr@epsilon<\@tempdima
        \pxrr@decompose{#3}%
        \edef\pxrr@natwd{\the\wd\pxrr@boxr}%
        \pxrr@evenspace@int{#1}\pxrr@boxr
         \pxrr@use@ruby@font{\wd\pxrr@boxa}%
        \pxrr@adjust@margin
      \fi
      \@tempdima\wd\pxrr@boxa
      \advance\@tempdima-\wd\pxrr@boxb
      \ifdim\pxrr@epsilon<\@tempdima
        \pxrr@decompose{#4}%
        \edef\pxrr@natwd{\the\wd\pxrr@boxb}%
        \pxrr@evenspace@int{#1}\pxrr@boxb
         \pxrr@use@ruby@font{\wd\pxrr@boxa}%
        \pxrr@adjust@margin
      \fi
    \fi
```

\pxrr@?space はゼロに設定する。

```
    \let\pxrr@bspace\pxrr@zeropt
    \let\pxrr@aspace\pxrr@zeropt
  \fi
```

実際に組版を行う。

```
  \setbox\z@\hbox{%
    \@tempdima\wd\pxrr@boxr
```

```
    \raise\pxrr@ruby@raise\box\pxrr@boxr
    \kern-\@tempdima
    \lower\pxrr@ruby@lower\box\pxrr@boxb
  }%
  \ifnum \ifpxrr@combo\pxrr@ck@ruby@combo\else\z@\fi >\z@
    \pxrr@ck@compose{#2}%
  \fi
  \ht\z@\z@ \dp\z@\z@
  \@tempdima\wd\z@
  \setbox\pxrr@boxr\hbox{%
    \box\z@
    \kern-\@tempdima
    \box\pxrr@boxa
  }%
}
```

\pxrr@body@wd　\pxrr@compose@twoside@block@sub の内部で用いられる変数で、“親文字列の実際の長さ”（均等割りで入った中間の空きを入れるが両端の空きを入れない）を表す。寸法値マクロ。

```
\let\pxrr@body@wd\relax
```

xrr@compose@twoside@block@sub　\pxrr@compose@twoside@block@sub の内部で用いられるマクロ。

```
\let\pxrr@restore@margin@values\relax
```

xrr@compose@twoside@block@sub　\pxrr@compose@twoside@block@sub{⟨親文字⟩}{⟨短い方のルビ文字⟩}\CSa\CSb：両側ルビで親文字列より長いルビ文字列が存在する場合の組み直しの処理を行う。このマクロの呼出時、上側ルビの出力結果が \pxrr@boxr、下側ルビの出力結果が \pxrr@boxb に入っているが、この 2 つのボックスのうち、短いルビの方が \CSa、長いルビの方が \CSb として渡されている。

```
\def\pxrr@compose@twoside@block@sub#1#2#3#4{%
  \pxrr@decompose{#1}%
  \edef\pxrr@natwd{\the\wd\pxrr@boxa}%
  \pxrr@evenspace@int\pxrr@locate@temp\pxrr@boxa\relax{\wd#4}%
  \@tempdima\wd#4%
  \advance\@tempdima-\pxrr@bspace\relax
  \advance\@tempdima-\pxrr@aspace\relax
  \edef\pxrr@body@wd{\the\@tempdima}%
  \advance\@tempdima-\wd#3%
  \ifdim\pxrr@epsilon<\@tempdima
    \edef\pxrr@restore@margin@values{%
      \edef\noexpand\pxrr@bspace{\pxrr@bspace}%
      \edef\noexpand\pxrr@aspace{\pxrr@aspace}%
    }%
    \pxrr@decompose{#2}%
    \edef\pxrr@natwd{\the\wd#3}%
    \pxrr@evenspace@int\pxrr@locate@temp#3%
     \pxrr@use@ruby@font{\pxrr@body@wd}%
    \pxrr@adjust@margin
```

```
    \pxrr@restore@margin@values
    \setbox#3\hbox{%
      \kern\pxrr@bspace\relax
      \box#3%
    }%
  \else
    \ifnum\pxrr@locate@temp=\pxrr@locate@head
      \@tempdima\z@
    \else\ifnum\pxrr@locate@temp=\pxrr@locate@inner
      \@tempdima.5\@tempdima
    \fi\fi
    \advance\@tempdima\pxrr@bspace\relax
    \setbox#3\hbox{%
      \kern\@tempdima
      \box#3%
    }%
  \fi
}
%     \end{macrocode}
% \end{macro}
%
% \begin{macro}{\pxrr@compose@block@pre}
% |\pxrr@compose@block@pre{|\jmeta{パターン}|}{|^^A
%r \jmeta{親文字}|}{|\jmeta{ルビ 1}|}{|\jmeta{ルビ 2}|}|\Means
% 親文字列・ルビ文字列の加工を行う。
% \Note 両側ルビ対応のため、ルビ用引数が 2 つある。
%    \begin{macrocode}
\def\pxrr@compose@block@pre{%
```

f 指定時は小書き仮名の変換を施す。

```
  \pxrr@cond\ifnum\pxrr@fullsize>\z@\fi{%
    \pxrr@compose@block@pre@a
  }{%
    \pxrr@compose@block@pre@d
  }%
}
% {パターン}{親文字}{ルビ 1}{ルビ 2}
\def\pxrr@compose@block@pre@a#1#2#3#4{%
  \def\pxrr@compose@block@tempa{#4}%
  \pxrr@transform@kana\pxrr@compose@block@tempa
  \expandafter\pxrr@compose@block@pre@b
   \expandafter{\pxrr@compose@block@tempa}{#1}{#2}{#3}%
}
% {ルビ 2}{パターン}{親文字}{ルビ 1}
\def\pxrr@compose@block@pre@b#1#2#3#4{%
  \def\pxrr@compose@block@tempa{#4}%
  \pxrr@transform@kana\pxrr@compose@block@tempa
  \expandafter\pxrr@compose@block@pre@c
   \expandafter{\pxrr@compose@block@tempa}{#1}{#2}{#3}%
```

```
}
% {ルビ 1}{ルビ 2}{パターン}{親文字}
\def\pxrr@compose@block@pre@c#1#2#3#4{%
  \pxrr@compose@block@pre@d{#3}{#4}{#1}{#2}%
}
\def\pxrr@compose@block@pre@d{%
  \pxrr@cond\ifnum\pxrr@evensp=\z@\fi{%
    \pxrr@compose@block@pre@e
  }{%
    \pxrr@compose@block@pre@f
  }%
}
% {パターン}{親文字}
\def\pxrr@compose@block@pre@e#1#2{%
  \pxrr@compose@block@pre@f{#1}{{#2}}%
}
\def\pxrr@compose@block@pre@f{%
  \pxrr@cond\ifnum\pxrr@revensp=\z@\fi{%
    \pxrr@compose@block@pre@g
  }{%
    \pxrr@compose@block@do
  }%
}
% {パターン}{親文字}{ルビ 1}{ルビ 2}
\def\pxrr@compose@block@pre@g#1#2#3#4{%
  \pxrr@compose@block@do{#1}{#2}{{#3}}{{#4}}%
}
\let\pxrr@compose@block@tempa\@undefined
```

## 4.14 命令の頑強化

\pxrr@add@protect

\pxrr@add@protect\CS： 命令 \CS に \protect を施して頑強なものに変える。\CS は最初から \DeclareRobustCommand で定義された頑強な命令とほぼ同じように振舞う——例えば、\CS の定義の本体は \CS␣ という制御綴に移される。唯一の相違点は、「組版中」（すなわち \protect = \@typeset@protect）の場合は、\CS は \protect\CS␣ ではなく、単なる \CS␣ に展開されることである。組版中は \protect は結局 \relax であるので、\DeclareRobustCommand 定義の命令の場合、\relax が「実行」されることになるが、pTEX ではこれがメトリックグルーの挿入に干渉するので、このパッケージの目的に沿わないのである。

※ \CS は「制御語」（制御記号でなく）である必要がある。

```
\def\pxrr@add@protect#1{%
  \expandafter\pxrr@add@protect@a
    \csname\expandafter\@gobble\string#1\space\endcsname#1%
}
\def\pxrr@add@protect@a#1#2{%
  \let#1=#2%
```

```
  \def#2{\pxrr@check@protect\protect#1}%
}
\def\pxrr@check@protect{%
  \ifx\protect\@typeset@protect
    \expandafter\@gobble
  \fi
}
```

## 4.15 致命的エラー対策

致命的エラーが起こった場合は、ルビ入力を放棄して単に親文字列を出力することにする。

`\pxrr@body@input` 入力された親文字列。

```
\let\pxrr@body@input\@empty
```

`\pxrr@prepare@fallback` `\pxrr@prepare@fallback{`⟨親文字列⟩`}`：

```
\def\pxrr@prepare@fallback#1{%
  \pxrr@fatal@errorfalse
  \def\pxrr@body@input{#1}%
}
```

`\pxrr@fallback` 致命的エラー時に出力となるもの。単に親文字列を出力することにする。

```
\def\pxrr@fallback{%
  \pxrr@body@input
}
```

`\pxrr@if@alive` `\pxrr@if@alive{`⟨コード⟩`}`： 致命的エラーが未発生の場合に限り、⟨コード⟩ に展開する。

```
\def\pxrr@if@alive{%
  \ifpxrr@fatal@error \expandafter\@gobble
  \else \expandafter\@firstofone
  \fi
}
```

## 4.16 先読み処理

ゴースト処理が無効の場合に後ろ側の禁則処理を行うため、ルビ命令の直後に続くトークンを取得して、その前禁則ペナルティ（`\prebreakpenalty`）の値を保存する。信頼性の低い方法なので、ゴースト処理が可能な場合はそちらを利用するべきである。

`\pxrr@end@kinsoku` ルビ命令直後の文字の前禁則ペナルティ値とみなす値。

```
\def\pxrr@end@kinsoku{0}
```

`\pxrr@ruby@scan` 片側ルビ用の先読み処理。

```
\def\pxrr@ruby@scan#1#2{%
```

`\pxrr@check@kinsoku` の続きの処理。`\pxrr@cntr` の値を `\pxrr@end@kinsoku` に保存して、ルビ処理本体を呼び出す。

```
  \def\pxrr@tempc{%
    \edef\pxrr@end@kinsoku{\the\pxrr@cntr}%
    \pxrr@do@proc{#1}{#2}%
  }%
  \pxrr@check@kinsoku\pxrr@tempc
}
```

\pxrr@truby@scan　両側ルビ用の先読み処理。

```
\def\pxrr@truby@scan#1#2#3{%
  \def\pxrr@tempc{%
    \edef\pxrr@end@kinsoku{\the\pxrr@cntr}%
    \pxrr@do@proc{#1}{#2}{#3}%
  }%
  \pxrr@check@kinsoku\pxrr@tempc
}
```

\pxrr@check@kinsoku　\pxrr@check@kinsoku\CS： \CS の直後に続くトークンについて、それが「通常文字」（和文文字トークンまたはカテゴリコード 11、12 の欧文文字トークン）である場合にはその前禁則ペナルティ（\prebreakpenalty）の値を、そうでない場合はゼロを \pxrr@cntr に代入する。その後、\CS を実行（展開）する。

※ ただし、欧文ルビの場合、欧文文字の前禁則ペナルティは 20000 として扱う。

```
\def\pxrr@check@kinsoku#1{%
  \let\pxrr@tempb#1%
  \futurelet\pxrr@token\pxrr@check@kinsoku@a
}
\def\pxrr@check@kinsoku@a{%
  \pxrr@check@char\pxrr@token
```

和文ルビの場合は、欧文通常文字も和文通常文字と同じ扱いにする。

```
  \ifpxrr@abody\else
    \ifnum\pxrr@cntr=\@ne
      \pxrr@cntr\tw@
    \fi
  \fi
  \ifcase\pxrr@cntr
    \pxrr@cntr\z@
    \expandafter\pxrr@tempb
  \or
    \pxrr@cntr\@MM
    \expandafter\pxrr@tempb
  \else
    \expandafter\pxrr@check@kinsoku@b
  \fi
}
```

\let されたトークンのままでは符号位置を得ることができないため、改めてマクロの引数として受け取り、複製した上で片方を後の処理に使う。既に後続トークンは「通常文字」である（つまり空白や { ではない）ことが判明していることに注意。

```
\def\pxrr@check@kinsoku@b#1{%
  \pxrr@check@kinsoku@c#1#1%
}
\def\pxrr@check@kinsoku@c#1{%
  \pxrr@get@prebreakpenalty\pxrr@cntr{`#1}%
  \pxrr@tempb
}
```

\pxrr@check@char　\pxrr@check@char\CS： トークン \CS が「通常文字」であるかを調べ、以下の値を \pxrr@cntr に返す： 0 = 通常文字でない；1 = 欧文通常文字；2 = 和文通常文字。

定義本体の中でカテゴリコード 12 の kanji というトークン列が必要なので、少々特殊な処置をしている。まず \pxrr@check@char を定義するためのマクロを用意する。

```
\def\pxrr@tempa#1#2\pxrr@nil{%
```

実際に呼び出される時には #2 はカテゴリコード 12 の kanji に置き換わる。（不要な \ を #1 に受け取らせている。）

```
  \def\pxrr@check@char##1{%
```

まず制御綴とカテゴリコード 11、12、13 を手早く \ifcat で判定する。

```
    \ifcat\noexpand##1\relax
      \pxrr@cntr\z@
    \else\ifcat\noexpand##1\noexpand~%
      \pxrr@cntr\z@
    \else\ifcat\noexpand##1A%
      \pxrr@cntr\@ne
    \else\ifcat\noexpand##10%
      \pxrr@cntr\@ne
    \else
```

それ以外の場合。和文文字トークンであるかを \meaning テストで調べる。（和文文字の \ifcat 判定は色々と面倒な点があるので避ける。）

```
      \pxrr@cntr\z@
      \expandafter\pxrr@check@char@a\meaning##1#2\pxrr@nil
    \fi\fi\fi\fi
  }%
  \def\pxrr@check@char@a##1#2##2\pxrr@nil{%
    \ifcat @##1@%
      \pxrr@cntr\tw@
    \fi
  }%
}
```

規定の引数を用意して「定義マクロ」を呼ぶ。

```
\expandafter\pxrr@tempa\string\kanji\pxrr@nil
```

## 4.17　進入処理

\pxrr@auto@penalty　自動挿入されるペナルティ。（整数定数への \let。）

```
\let\pxrr@auto@penalty\z@
```

\pxrr@auto@icspace　文字間の空き。寸法値マクロ。

```
\let\pxrr@auto@icspace\pxrr@zeropt
```

\pxrr@intr@amount　進入の幅。寸法値マクロ。

```
\let\pxrr@intr@amount\pxrr@zeropt
```

\pxrr@intrude@setauto@j　和文の場合の \pxrr@auto@* の設定。

```
\def\pxrr@intrude@setauto@j{%
```

行分割禁止（*）の場合、ペナルティを 20000 とし、字間空きはゼロにする。

```
  \ifpxrr@bnobr
    \let\pxrr@auto@penalty\@MM
    \let\pxrr@auto@icspace\pxrr@zeropt
```

それ以外の場合は、ペナルティはゼロで、\pxrr@bspace の設定を活かす。

```
  \else
    \let\pxrr@auto@penalty\z@
    \if :\pxrr@bscomp
      \let\pxrr@auto@icspace\pxrr@iaiskip
    \else\if .\pxrr@bscomp
      \let\pxrr@auto@icspace\pxrr@zeropt
    \else
      \let\pxrr@auto@icspace\pxrr@iiskip
    \fi\fi
  \fi
}
```

\pxrr@intrude@setauto@a　欧文の場合の \pxrr@auto@* の設定。

```
\def\pxrr@intrude@setauto@a{%
```

欧文の場合、和欧文間空白挿入指定（:）でない場合は、（欧文同士と見做して）行分割禁止にする。

```
  \if :\pxrr@bscomp\else
    \pxrr@bnobrtrue
  \fi
  \ifpxrr@bnobr
    \let\pxrr@auto@penalty\@MM
    \let\pxrr@auto@icspace\pxrr@zeropt
  \else
```

この分岐は和欧文間空白挿入指定（:）に限る。

```
    \let\pxrr@auto@penalty\z@
    \let\pxrr@auto@icspace\pxrr@iaiskip
  \fi
}
```

### 4.17.1 前側進入処理

\pxrr@intrude@head 前側の進入処理。

```
\def\pxrr@intrude@head{%
```

ゴースト処理が有効な場合は進入処理を行わない。（だから進入が扱えない。）

```
  \ifpxrr@ghost\else
```

実効の進入幅は \pxrr@bintr と \pxrr@bspace の小さい方。

```
    \let\pxrr@intr@amount\pxrr@bspace
    \ifdim\pxrr@bintr<\pxrr@intr@amount\relax
      \let\pxrr@intr@amount\pxrr@bintr
    \fi
```

\pxrr@auto@* の設定法は和文ルビと欧文ルビで処理が異なる。

```
    \ifpxrr@abody
      \pxrr@intrude@setauto@a
    \else
      \pxrr@intrude@setauto@j
    \fi
```

実際に項目の出力を行う。

段落冒頭の場合、! 指定（pxrr@bfintr が真）ならば進入のための負のグルーを入れる（他の項目は入れない）。

```
    \ifpxrr@par@head
      \ifpxrr@bfintr
        \hskip-\pxrr@intr@amount\relax
      \fi
```

段落冒頭でない場合、字間空きのグルー、進入用のグルーを順番に入れる。

※ ペナルティは \pxrr@put@head@penalty で既に入れている。

```
    \else
%     \penalty\pxrr@auto@penalty\relax
      \hskip-\pxrr@intr@amount\relax
      \hskip\pxrr@auto@icspace\relax
    \fi
  \fi
}
```

\pxrr@put@head@penalty 前側に補助指定で定められた値のペナルティを置く。現在位置に既にペナルティがある場合は合算する。

```
\def\pxrr@put@head@penalty{%
  \ifpxrr@ghost\else \ifpxrr@par@head\else
    \ifpxrr@abody
      \pxrr@intrude@setauto@a
    \else
      \pxrr@intrude@setauto@j
    \fi
    \ifnum\pxrr@auto@penalty=\z@\else
```

```
      \pxrr@cnta\lastpenalty \unpenalty
      \advance\pxrr@cnta\pxrr@auto@penalty\relax
      \penalty\pxrr@cnta
    \fi
  \fi\fi
}
```

### 4.17.2 後側進入処理

\pxrr@intrude@end 末尾での進入処理。

```
\def\pxrr@intrude@end{%
  \ifpxrr@ghost\else
```

実効の進入幅は \pxrr@aintr と \pxrr@aspace の小さい方。

```
    \let\pxrr@intr@amount\pxrr@aspace
    \ifdim\pxrr@aintr<\pxrr@intr@amount\relax
      \let\pxrr@intr@amount\pxrr@aintr
    \fi
```

\pxrr@auto@* の設定法は和文ルビと欧文ルビで処理が異なる。

```
    \pxrr@csletcs{ifpxrr@bnobr}{ifpxrr@anobr}%
    \let\pxrr@bscomp\pxrr@ascomp
    \ifpxrr@abody
      \pxrr@intrude@setauto@a
    \else
      \pxrr@intrude@setauto@j
    \fi
```

直後の文字の前禁則ペナルティが、挿入されるグルーの前に入るようにする。

```
    \ifnum\pxrr@auto@penalty=\z@
      \let\pxrr@auto@penalty\pxrr@end@kinsoku
    \fi
    \ifpxrr@afintr
```

段落末尾での進入を許す場合。

```
      \ifnum\pxrr@auto@penalty=\z@\else
        \penalty\pxrr@auto@penalty\relax
      \fi
      \kern-\pxrr@intr@amount\relax
```

段落末尾では次のグルーを消滅させる（前のカーンは残る）。そのため、禁則ペナルティがある（段落末尾ではあり得ない）場合にのみその次のペナルティ 20000 を置く。本物の禁則ペナルティはこれに加算されるが、合計値は 10000 以上になるのでこの位置での行分割が禁止される。

```
      \hskip\pxrr@auto@icspace\relax
      \ifnum\pxrr@auto@penalty=\z@\else
        \penalty\@MM
      \fi
    \else
```

段落末尾での進入を許さない場合。

```
      \@tempskipa-\pxrr@intr@amount\relax
      \advance\@tempskipa\pxrr@auto@icspace\relax
      \ifnum\pxrr@auto@penalty=\z@\else
        \penalty\pxrr@auto@penalty\relax
      \fi
      \hskip\@tempskipa
      \ifnum\pxrr@auto@penalty=\z@\else
        \penalty\@MM
      \fi
    \fi
  \fi
}
```

## 4.18 メインです

### 4.18.1 エントリーポイント

\ruby \jruby 和文ルビの公開命令。\jruby を頑強な命令として定義した上で、\ruby はそれに展開されるマクロに（未定義ならば）定義する。

```
\AtBeginDocument{%
  \providecommand*{\ruby}{\jruby}%
}
\newcommand*{\jruby}{%
  \pxrr@jprologue
  \pxrr@trubyfalse
  \pxrr@ruby
}
```

頑強にするために、先に定義した \pxrr@add@protect を用いる。

```
\pxrr@add@protect\jruby
```

\aruby 欧文ルビの公開命令。こちらも頑強な命令にする。

```
\newcommand*{\aruby}{%
  \pxrr@aprologue
  \pxrr@trubyfalse
  \pxrr@ruby
}
\pxrr@add@protect\aruby
```

\truby 和文両側ルビの公開命令。

```
\newcommand*{\truby}{%
  \pxrr@jprologue
  \pxrr@trubytrue
  \pxrr@ruby
}
\pxrr@add@protect\truby
```

\atruby 欧文両側ルビの公開命令。

```
\newcommand*{\atruby}{%
  \pxrr@aprologue
  \pxrr@trubytrue
  \pxrr@ruby
}
\pxrr@add@protect\atruby
```

`\ifpxrr@truby` 両側ルビであるか。スイッチ。`\pxrr@parse@option` で `\pxrr@side` を適切に設定するために使われる。

```
\newif\ifpxrr@truby
```

`\pxrr@option` オプションおよび第 2 オプションを格納するマクロ。

`\pxrr@exoption`

```
\let\pxrr@option\@empty
\let\pxrr@exoption\@empty
```

`\pxrr@do@proc` `\pxrr@ruby` の処理中に使われる。

`\pxrr@do@scan`

```
\let\pxrr@do@proc\@empty
\let\pxrr@do@scan\@empty
```

`\pxrr@ruby` `\ruby` および `\aruby` の共通の下請け。オプションの処理を行う。
オプションを読みマクロに格納する。

```
\def\pxrr@ruby{%
  \@testopt\pxrr@ruby@a{}%
}
\def\pxrr@ruby@a[#1]{%
  \def\pxrr@option{#1}%
  \@testopt\pxrr@ruby@b{}%
}
\def\pxrr@ruby@b[#1]{%
  \def\pxrr@exoption{#1}%
  \ifpxrr@truby
    \let\pxrr@do@proc\pxrr@truby@proc
    \let\pxrr@do@scan\pxrr@truby@scan
  \else
    \let\pxrr@do@proc\pxrr@ruby@proc
    \let\pxrr@do@scan\pxrr@ruby@scan
  \fi
  \pxrr@ruby@c
}
\def\pxrr@ruby@c{%
  \ifpxrr@ghost
    \expandafter\pxrr@do@proc
  \else
    \expandafter\pxrr@do@scan
  \fi
}
```

`\pxrr@mode@is@switching` `\if\pxrr@mode@is@switching{`⟨基本モード⟩`}` の形の if 文として使う。モードが“選択的”（M・J）であるか。

```
\def\pxrr@mode@is@switching{%
  \if      M\pxrr@mode T%
  \else\if J\pxrr@mode T%
  \else F%
  \fi\fi T%
}
```

\pxrr@bind@param “呼出時変数”へのコピーを行う。

```
\def\pxrr@bind@param{%
```

圏点ルビ同時付加フラグの処理。圏点側が指定した apply@combo の値を“呼出時パラメタ”の pxrr@combo に移動させる。

```
  \ifpxrr@apply@combo
    \pxrr@apply@combofalse
    \pxrr@combotrue
    \pxrr@ck@bind@param
  \else
    \pxrr@combofalse
  \fi
  \let\pxrr@c@ruby@font\pxrr@ruby@font
  \let\pxrr@c@size@ratio\pxrr@size@ratio
  \let\pxrr@c@inter@gap\pxrr@inter@gap
}
```

\pxrr@ruby@proc \pxrr@ruby@proc{⟨親文字列⟩}{⟨ルビ文字列⟩}： これが手続の本体となる。

```
\def\pxrr@ruby@proc#1#2{%
  \pxrr@prepare@fallback{#1}%
```

フォントサイズの変数を設定して、

```
  \pxrr@bind@param
  \pxrr@assign@fsize
```

オプションを解析する。

```
  \pxrr@parse@option\pxrr@option
```

ルビ文字入力をグループ列に分解する。

```
  \pxrr@decompbar{#2}%
  \let\pxrr@ruby@list\pxrr@res
  \edef\pxrr@ruby@count{\the\pxrr@cntr}%
  \let\pxrr@sruby@list\relax
```

親文字入力をグループ列に分解する。

```
  \pxrr@decompbar{#1}%
  \let\pxrr@body@list\pxrr@res
  \edef\pxrr@body@count{\the\pxrr@cntr}%
```

安全モードに関する処理を行う。

```
  \ifpxrr@safe@mode
    \pxrr@setup@safe@mode
  \fi
```

モードが“選択的”である場合、“普通の”モード（m・j・g）に帰着させる。

```
  \if\pxrr@mode@is@switching
    \pxrr@resolve@mode
  \fi
\ifpxrrDebug
  \pxrr@debug@show@input
\fi
```

入力検査を行い、パスした場合は組版処理に進む。

```
  \pxrr@if@alive{%
    \if g\pxrr@mode
      \pxrr@ruby@check@g
      \pxrr@if@alive{%
        \ifnum\pxrr@body@count>\@ne
          \pxrr@ruby@main@mg
        \else
          \pxrr@ruby@main@g
        \fi
      }%
    \else
      \pxrr@ruby@check@m
      \pxrr@if@alive{\pxrr@ruby@main@m}%
    \fi
  }%
```

後処理を行う。

```
  \pxrr@ruby@exit
}
```

\pxrr@truby@proc `\pxrr@ruby@proc{`⟨親文字列⟩`}{`⟨上側ルビ文字列⟩`}{`⟨下側ルビ文字列⟩`}`：両側ルビの場合の手続の本体。

```
\def\pxrr@truby@proc#1#2#3{%
  \pxrr@prepare@fallback{#1}%
```

フォントサイズの変数を設定して、

```
  \pxrr@bind@param
  \pxrr@assign@fsize
```

オプションを解析する。

```
  \pxrr@parse@option\pxrr@option
```

両側のグループルビでは pxrr@all@input を利用するので、入力文字列を設定する。

```
  \def\pxrr@all@input{{#1}{#2}{#3}}%
```

入力文字列のグループ分解を行う。

```
  \pxrr@decompbar{#3}%
  \let\pxrr@sruby@list\pxrr@res
  \edef\pxrr@sruby@count{\the\pxrr@cntr}%
  \pxrr@decompbar{#2}%
  \let\pxrr@ruby@list\pxrr@res
```

```
  \edef\pxrr@ruby@count{\the\pxrr@cntr}%
  \pxrr@decompbar{#1}%
  \let\pxrr@body@list\pxrr@res
  \edef\pxrr@body@count{\the\pxrr@cntr}%
```

安全モードに関する処理を行う。

```
  \ifpxrr@safe@mode
    \pxrr@setup@safe@mode
  \fi
  \if\pxrr@mode@is@switching
    \pxrr@resolve@mode
  \fi
\ifpxrrDebug
  \pxrr@debug@show@input
\fi
```

入力検査を行い、パスした場合は組版処理に進む。

```
  \pxrr@if@alive{%
    \if g\pxrr@mode
      \pxrr@ruby@check@tg
      \pxrr@if@alive{\pxrr@ruby@main@tg}%
    \else
      \pxrr@ruby@check@tm
      \pxrr@if@alive{\pxrr@ruby@main@tm}%
    \fi
  }%
```

後処理を行う。

```
  \pxrr@ruby@exit
}
```

\pxrr@setup@safe@mode　安全モード用の設定。

```
\def\pxrr@setup@safe@mode{%
```

単純グループルビに強制的に変更する。これに応じて、親文字列とルビ文字列のグループを 1 つに集成する。

```
  \let\pxrr@mode=g\relax
  \pxrr@unite@group\pxrr@body@list
  \def\pxrr@body@count{1}%
  \pxrr@unite@group\pxrr@ruby@list
  \def\pxrr@ruby@count{1}%
  \ifx\pxrr@sruby@list\relax\else
    \pxrr@unite@group\pxrr@sruby@list
    \def\pxrr@sruby@count{1}%
  \fi
```

“文字単位のスキャン”が必要な機能を無効にする。

```
  \chardef\pxrr@evensp\z@
  \chardef\pxrr@revensp\z@
  \chardef\pxrr@fullsize\z@
}
```

\pxrr@resolve@mode 基本モードが“選択的”（M・J）である場合に、状況に応じて適切な通常のモードに切り替える。

```
\def\pxrr@resolve@mode{%
  \ifnum\pxrr@body@count=\@ne
```

ルビグループが 1 つで親文字が複数ある場合にはグループルビを選択し、

```
    \ifnum\pxrr@ruby@count=\@ne
      \let\pxrr@pre\pxrr@decompose
      \let\pxrr@post\relax
      \pxrr@body@list
      \ifnum\pxrr@cntr=\@ne\else
        \let\pxrr@mode=g%
      \fi
    \fi
```

それ以外はモノルビ・熟語ルビを選択する。

```
    \if M\pxrr@mode \let\pxrr@mode=m\fi
    \if J\pxrr@mode \let\pxrr@mode=j\fi
\ifpxrrDebug
  \pxrr@debug@show@resolve@mode
\fi
```

\pxrr@check@option で行っている調整をやり直す。

```
    \if g\pxrr@mode
      \chardef\pxrr@athead\z@
    \fi
    \if g\pxrr@mode\else
      \chardef\pxrr@evensp\@ne
    \fi
  \else
    \pxrr@fatal@bad@switching
  \fi
}
```

### 4.18.2 入力検査

グループ・文字の個数の検査を行う手続。

\pxrr@ruby@check@g グループルビの場合、ルビ文字グループと親文字グループの個数が一致する必要がある。さらに、グループが複数（可動グループルビ）にできるのは、和文ルビであり、しかも拡張機能が有効である場合に限られる。

```
\def\pxrr@ruby@check@g{%
  \ifnum\pxrr@body@count=\pxrr@ruby@count\relax
    \ifnum\pxrr@body@count=\@ne\else
      \ifpxrr@abody
        \pxrr@fatal@bad@movable
      \else\ifnum\pxrr@extra=\z@
        \pxrr@fatal@na@movable
      \fi\fi
```

```
    \fi
  \else
    \pxrr@fatal@bad@length\pxrr@body@count\pxrr@ruby@count
  \fi
}
```

\pxrr@ruby@check@m　モノルビ・熟語ルビの場合、親文字列は単一のグループからなる必要がある。さらに、親文字列の《文字》の個数とルビ文字列のグループの個数が一致する必要がある。

```
\def\pxrr@ruby@check@m{%
  \ifnum\pxrr@body@count=\@ne
```

ここで \pxrr@body@list／count を文字ごとの分解に置き換える。

```
    \let\pxrr@pre\pxrr@decompose
    \let\pxrr@post\relax
    \pxrr@body@list
    \let\pxrr@body@list\pxrr@res
    \edef\pxrr@body@count{\the\pxrr@cntr}%
    \ifnum\pxrr@body@count=\pxrr@ruby@count\relax\else
      \pxrr@fatal@bad@length\pxrr@body@count\pxrr@ruby@count
    \fi
  \else
    \pxrr@fatal@bad@mono
  \fi
}
```

\pxrr@ruby@check@tg　両側のグループルビの場合。ルビが 2 つあることを除き、片側の場合と同じ。

```
\def\pxrr@ruby@check@tg{%
  \ifnum\pxrr@body@count=\pxrr@ruby@count\relax\else
    \pxrr@fatal@bad@length\pxrr@body@count\pxrr@ruby@count
  \fi
  \ifnum\pxrr@body@count=\pxrr@sruby@count\relax\else
    \pxrr@fatal@bad@length\pxrr@body@count\pxrr@sruby@count
  \fi
  \pxrr@if@alive{%
    \ifnum\pxrr@body@count=\@ne\else
      \ifpxrr@abody
        \pxrr@fatal@bad@movable
      \else\ifnum\pxrr@extra=\z@
        \pxrr@fatal@na@movable
      \fi\fi
    \fi
  }%
}
```

\pxrr@ruby@check@tm　両側のモノルビの場合。ルビが 2 つあることを除き、片側の場合と同じ。

```
\def\pxrr@ruby@check@tm{%
  \ifnum\pxrr@body@count=\@ne
    \let\pxrr@pre\pxrr@decompose
    \let\pxrr@post\relax
```

```
    \pxrr@body@list
    \let\pxrr@body@list\pxrr@res
    \edef\pxrr@body@count{\the\pxrr@cntr}%
    \ifnum\pxrr@body@count=\pxrr@ruby@count\relax\else
      \pxrr@fatal@bad@length\pxrr@body@count\pxrr@ruby@count
    \fi
    \ifnum\pxrr@body@count=\pxrr@sruby@count\relax\else
      \pxrr@fatal@bad@length\pxrr@body@count\pxrr@sruby@count
    \fi
  \else
    \pxrr@fatal@bad@mono
  \fi
}
```

#### 4.18.3 ルビ組版処理

`\ifpxrr@par@head` ルビ付文字列の出力位置が段落の先頭であるか。

```
\newif\ifpxrr@par@head
```

`\pxrr@check@par@head` 現在の位置に基づいて `\ifpxrr@par@head` の値を設定する。当然、何らかの出力を行う前に呼ぶ必要がある。

```
\def\pxrr@check@par@head{%
  \ifvmode
    \pxrr@par@headtrue
  \else
    \pxrr@par@headfalse
  \fi
}
```

`\pxrr@if@last` `\pxrr@if@last{⟨真⟩}{⟨偽⟩}`： `\pxrr@pre/inter` の本体として使い、それが最後の `\pxrr@pre/inter` である（`\pxrr@post` の直前にある）場合に ⟨真⟩、ない場合に ⟨偽⟩ に展開される。このマクロの呼出は `\pxrr@preinterpre` の本体の末尾でなければならない。

```
\def\pxrr@if@last#1#2#3{%
  \ifx#3\pxrr@post #1%
  \else #2%
  \fi
  #3%
}
```

`\pxrr@inter@mono` モノルビのブロック間に挿入される空き。和文間空白とする。

```
\def\pxrr@inter@mono{%
  \hskip\pxrr@iiskip\relax
}
```

`\pxrr@takeout@any@protr` `\ifpxrr@any@protr` の値を `\pxrr@hbox` の外に出す。

※ color 不使用時は `\hbox` による 1 段のグループだけ処理すればよいが、color 使用時は `\color@begingroup`〜`\color@endgroup` によるグループが生じるので、2 段分の処理が必要。

color 不使用時の定義。

```
\def\pxrr@takeout@any@protr@nocolor{%
  \ifpxrr@any@protr
    \aftergroup\pxrr@any@protrtrue
  \fi
}
```

color 使用時の定義。

```
\def\pxrr@takeout@any@protr{%
  \ifpxrr@any@protr
    \aftergroup\pxrr@takeout@any@protr@a
  \fi
}
\def\pxrr@takeout@any@protr@a{%
  \aftergroup\pxrr@any@protrtrue
}
```

`\pxrr@ruby@main@m` モノルビ。

```
\def\pxrr@ruby@main@m{%
  \pxrr@zip@list\pxrr@body@list\pxrr@ruby@list
  \let\pxrr@whole@list\pxrr@res
  \pxrr@check@par@head
  \pxrr@put@head@penalty
  \pxrr@any@protrfalse
\ifpxrrDebug
\pxrr@debug@show@recomp
\fi
```

`\ifpxrr@?intr` の値に応じて `\pxrr@locate@*@` の値を決定する。なお、両側で突出を禁止するのは不可であることに注意。

```
  \let\pxrr@locate@head@\pxrr@locate@inner
  \let\pxrr@locate@end@\pxrr@locate@inner
  \let\pxrr@locate@sing@\pxrr@locate@inner
  \ifpxrr@aprotr\else
    \let\pxrr@locate@end@\pxrr@locate@end
    \let\pxrr@locate@sing@\pxrr@locate@end
  \fi
  \ifpxrr@bprotr\else
    \let\pxrr@locate@head@\pxrr@locate@head
    \let\pxrr@locate@sing@\pxrr@locate@head
  \fi
  \def\pxrr@pre##1##2{%
    \pxrr@if@last{%
```

単独ブロックの場合。

```
      \pxrr@compose@block\pxrr@locate@sing@{##1}{##2}%
      \pxrr@intrude@head
      \unhbox\pxrr@boxr
      \pxrr@intrude@end
```

```
      \pxrr@takeout@any@protr
    }{%
```

先頭ブロックの場合。

```
      \pxrr@compose@block\pxrr@locate@head@{##1}{##2}%
      \pxrr@intrude@head
      \unhbox\pxrr@boxr
    }%
  }%
  \def\pxrr@inter##1##2{%
    \pxrr@if@last{%
```

末尾ブロックの場合。

```
      \pxrr@compose@block\pxrr@locate@end@{##1}{##2}%
      \pxrr@inter@mono
      \unhbox\pxrr@boxr
      \pxrr@intrude@end
      \pxrr@takeout@any@protr
    }{%
```

中間ブロックの場合。

```
      \pxrr@compose@block\pxrr@locate@inner{##1}{##2}%
      \pxrr@inter@mono
      \unhbox\pxrr@boxr
    }%
  }%
  \let\pxrr@post\@empty
  \setbox\pxrr@boxr\pxrr@hbox{\pxrr@whole@list}%
```

熟語ルビ指定の場合、\ifpxrr@any@protr が真である場合は再調整する。

```
  \if j\pxrr@mode
    \ifpxrr@any@protr
      \pxrr@ruby@redo@j
    \fi
  \fi
  \unhbox\pxrr@boxr
}
```

\pxrr@ruby@redo@j　モノルビ処理できない（ルビが長くなるブロックがある）熟語ルビを適切に組みなおす。現状では、単純にグループルビの組み方にする。

```
\def\pxrr@ruby@redo@j{%
  \pxrr@concat@list\pxrr@body@list
  \let\pxrr@body@list\pxrr@res
  \pxrr@concat@list\pxrr@ruby@list
  \let\pxrr@ruby@list\pxrr@res
  \pxrr@zip@single\pxrr@body@list\pxrr@ruby@list
  \let\pxrr@whole@list\pxrr@res
\ifpxrrDebug
\pxrr@debug@show@concat
\fi
```

```
  \let\pxrr@locate@sing@\pxrr@locate@inner
  \ifpxrr@aprotr\else
    \let\pxrr@locate@sing@\pxrr@locate@end
  \fi
  \ifpxrr@bprotr\else
    \let\pxrr@locate@sing@\pxrr@locate@head
  \fi
  \def\pxrr@pre##1##2{%
    \pxrr@compose@block\pxrr@locate@sing@{##1}{##2}%
    \pxrr@intrude@head
    \unhbox\pxrr@boxr
    \pxrr@intrude@end
  }%
  \let\pxrr@inter\@undefined
  \let\pxrr@post\@empty
  \setbox\pxrr@boxr\pxrr@hbox{\pxrr@whole@list}%
}
```

\pxrr@ruby@main@g　単純グループルビの場合。

グループが 1 つしかない前提なので多少冗長となるが、基本的に \pxrr@ruby@main@m の処理を踏襲する。

```
\def\pxrr@ruby@main@g{%
  \pxrr@zip@list\pxrr@body@list\pxrr@ruby@list
  \let\pxrr@whole@list\pxrr@res
  \pxrr@check@par@head
  \pxrr@put@head@penalty
\ifpxrrDebug
\pxrr@debug@show@recomp
\fi
  \let\pxrr@locate@sing@\pxrr@locate@inner
  \ifpxrr@aprotr\else
    \let\pxrr@locate@sing@\pxrr@locate@end
  \fi
  \ifpxrr@bprotr\else
    \let\pxrr@locate@sing@\pxrr@locate@head
  \fi
  \def\pxrr@pre##1##2{%
    \pxrr@compose@block\pxrr@locate@sing@{##1}{##2}%
    \pxrr@intrude@head
    \unhbox\pxrr@boxr
    \pxrr@intrude@end
  }%
  \let\pxrr@inter\@undefined
  \let\pxrr@post\@empty
```

グループルビは \ifpxrr@any@protr の判定が不要なので直接出力する。

```
  \pxrr@whole@list
}
```

\pxrr@ruby@main@tm　両側のモノルビの場合。

```
\def\pxrr@ruby@main@tm{%
  \pxrr@tzip@list\pxrr@body@list\pxrr@ruby@list\pxrr@sruby@list
  \let\pxrr@whole@list\pxrr@res
  \pxrr@check@par@head
  \pxrr@any@protrfalse
\ifpxrrDebug
\pxrr@debug@show@recomp
\fi
  \let\pxrr@locate@head@\pxrr@locate@inner
  \let\pxrr@locate@end@\pxrr@locate@inner
  \let\pxrr@locate@sing@\pxrr@locate@inner
  \ifpxrr@aprotr\else
    \let\pxrr@locate@end@\pxrr@locate@end
    \let\pxrr@locate@sing@\pxrr@locate@end
  \fi
  \ifpxrr@bprotr\else
    \let\pxrr@locate@head@\pxrr@locate@head
    \let\pxrr@locate@sing@\pxrr@locate@head
  \fi
  \def\pxrr@pre##1##2##3{%
    \pxrr@if@last{%
      \pxrr@compose@twoside@block\pxrr@locate@sing@
       {##1}{##2}{##3}%
      \pxrr@intrude@head
      \unhbox\pxrr@boxr
      \pxrr@intrude@end
      \pxrr@takeout@any@protr
    }{%
      \pxrr@compose@twoside@block\pxrr@locate@head@
       {##1}{##2}{##3}%
      \pxrr@intrude@head
      \unhbox\pxrr@boxr
    }%
  }%
  \def\pxrr@inter##1##2##3{%
    \pxrr@if@last{%
      \pxrr@compose@twoside@block\pxrr@locate@end@
       {##1}{##2}{##3}%
      \pxrr@inter@mono
      \unhbox\pxrr@boxr
      \pxrr@intrude@end
      \pxrr@takeout@any@protr
    }{%
      \pxrr@compose@twoside@block\pxrr@locate@inner
       {##1}{##2}{##3}%
      \pxrr@inter@mono
      \unhbox\pxrr@boxr
```

```
    }%
  }%
  \let\pxrr@post\@empty
  \setbox\pxrr@boxr\pxrr@hbox{\pxrr@whole@list}%
  \unhbox\pxrr@boxr
}
```

\pxrr@ruby@main@tg　両側の単純グループルビの場合。

```
\def\pxrr@ruby@main@tg{%
  \pxrr@check@par@head
  \pxrr@put@head@penalty
  \let\pxrr@locate@sing@\pxrr@locate@inner
  \ifpxrr@aprotr\else
    \let\pxrr@locate@sing@\pxrr@locate@end
  \fi
  \ifpxrr@bprotr\else
    \let\pxrr@locate@sing@\pxrr@locate@head
  \fi
  \expandafter\pxrr@compose@twoside@block\expandafter\pxrr@locate@sing@
   \pxrr@all@input
  \pxrr@intrude@head
  \unhbox\pxrr@boxr
  \pxrr@intrude@end
}
```

\pxrr@ruby@main@mg　未実装（呼出もない）。

```
\let\pxrr@ruby@main@mg\@undefined
```

#### 4.18.4　前処理

ゴースト処理する。そのため、展開不能命令が…。

\ifpxrr@ghost　実行中のルビ命令でゴースト処理が有効か。

```
\newif\ifpxrr@ghost
```

\pxrr@jprologue　和文ルビ用の開始処理。

```
\def\pxrr@jprologue{%
```

ゴースト処理を行う場合、一番最初に現れる展開不能トークンがゴースト文字（全角空白）であることが肝要である。

```
  \ifpxrr@jghost
    \pxrr@jghost@char
    \pxrr@inhibitglue
  \fi
```

ルビの処理の本体は全てこのグループの中で行われる。

```
  \begingroup
    \pxrr@abodyfalse
    \pxrr@csletcs{ifpxrr@ghost}{ifpxrr@jghost}%
```

出力した全角空白の幅だけ戻しておく。

```
    \ifpxrr@jghost
      \setbox\pxrr@boxa\hbox{\pxrr@jghost@char}%
      \kern-\wd\pxrr@boxa
    \fi
}
```

\pxrr@aghost 欧文用のゴースト文字の定義。合成語記号は T1 エンコーディングの位置 23 にある。従って、T1 のフォントが必要になるが、ここでは Latin Modern Roman を 2.5 pt のサイズで用いる。極小のサイズにしているのは、合成語記号の高さが影響する可能性を避けるためである。LM フォントの TEX フォント名は版により異なるようなので、NFSS を通して目的のフォントの fontdef を得ている。（グループ内で `\usefont{T1}{lmr}{m}{n}` を呼んでおくと、大域的に `\T1/lmr/m/n/2.5` が定義される。）

```
\chardef\pxrr@aghostchar=23 % compwordmark
\let\pxrr@aghost\relax
\let\pxrr@aghostfont\relax
\def\pxrr@setup@aghost{%
  \global\let\pxrr@setup@aghost\relax
  \IfFileExists{t1lmr.fd}{%
    \begingroup
      \fontsize{2.5}{0}\usefont{T1}{lmr}{m}{n}%
    \endgroup
    \global\pxrr@letcs\pxrr@aghostfont{T1/lmr/m/n/2.5}%
    \gdef\pxrr@aghost{{\pxrr@aghostfont\pxrr@aghostchar}}%
    \global\xspcode\pxrr@aghostchar=3 %
  }{%else
    \pxrr@warn{Ghost embedding for \string\aruby\space
      is disabled,\MessageBreak
      since package lmodern is missing}%
    \global\pxrr@aghostfalse
    \global\let\pxrr@aghosttrue\relax
  }%
}
```

\pxrr@aprologue 欧文ルビ用の開始処理。

```
\def\pxrr@aprologue{%
  \ifpxrr@aghost
    \pxrr@aghost
  \fi
  \begingroup
    \pxrr@abodytrue
    \pxrr@csletcs{ifpxrr@ghost}{ifpxrr@aghost}%
}
```

#### 4.18.5 後処理

ゴースト処理する。

\pxrr@ruby@exit　出力を終えて、最後に呼ばれるマクロ。致命的エラーが起こった場合はフォールバック処理を行う。その後は、和文ルビと欧文ルビで処理が異なる。

```
\def\pxrr@ruby@exit{%
  \ifpxrr@fatal@error
    \pxrr@fallback
  \fi
  \ifpxrr@abody
    \expandafter\pxrr@aepilogue
  \else
    \expandafter\pxrr@jepilogue
  \fi
}
```

\pxrr@jepilogue　和文の場合の終了処理。開始処理と同様、全角空白をゴースト文字に用いる。

```
\def\pxrr@jepilogue{%
    \ifpxrr@jghost
      \setbox\pxrr@boxa\hbox{\pxrr@jghost@char}%
      \kern-\wd\pxrr@boxa
    \fi
```

\pxrr@?prologue の中の \begingroup で始まるグループを閉じる。

```
  \endgroup
  \ifpxrr@jghost
    \pxrr@inhibitglue
    \pxrr@jghost@char
  \fi
}
```

\pxrr@aepilogue　欧文の場合の終了処理。合成語記号をゴースト文字に用いる。

```
\def\pxrr@aepilogue{%
  \endgroup
  \ifpxrr@aghost
    \pxrr@aghost
  \fi
}
```

## 4.19　デバッグ用出力

```
\def\pxrr@debug@show@input{%
  \typeout{----\pxrr@pkgname\space input:^^J%
    ifpxrr@abody = \meaning\ifpxrr@abody^^J%
    ifpxrr@truby = \meaning\ifpxrr@truby^^J%
    pxrr@ruby@fsize = \pxrr@ruby@fsize^^J%
    pxrr@body@zw = \pxrr@body@zw^^J%
    pxrr@ruby@zw = \pxrr@ruby@zw^^J%
    pxrr@iiskip = \pxrr@iiskip^^J%
    pxrr@iaiskip = \pxrr@iaiskip^^J%
    pxrr@htratio = \pxrr@htratio^^J%
```

```
    pxrr@ruby@raise = \pxrr@ruby@raise^^J%
    pxrr@ruby@lower = \pxrr@ruby@lower^^J%
    ifpxrr@bprotr = \meaning\ifpxrr@bprotr^^J%
    ifpxrr@aprotr = \meaning\ifpxrr@aprotr^^J%
    pxrr@side = \the\pxrr@side^^J%
    pxrr@evensp = \the\pxrr@evensp^^J%
    pxrr@fullsize = \the\pxrr@fullsize^^J%
    pxrr@bscomp = \meaning\pxrr@bscomp^^J%
    pxrr@ascomp = \meaning\pxrr@ascomp^^J%
    ifpxrr@bnobr = \meaning\ifpxrr@bnobr^^J%
    ifpxrr@anobr = \meaning\ifpxrr@anobr^^J%
    ifpxrr@bfintr = \meaning\ifpxrr@bfintr^^J%
    ifpxrr@afintr = \meaning\ifpxrr@afintr^^J%
    pxrr@bintr = \pxrr@bintr^^J%
    pxrr@aintr = \pxrr@aintr^^J%
    pxrr@athead = \the\pxrr@athead^^J%
    pxrr@mode = \meaning\pxrr@mode^^J%
    ifpxrr@athead@given = \meaning\ifpxrr@athead@given^^J%
    ifpxrr@mode@given = \meaning\ifpxrr@mode@given^^J%
    pxrr@body@list = \meaning\pxrr@body@list^^J%
    pxrr@body@count = \@nameuse{pxrr@body@count}^^J%
    pxrr@ruby@list = \meaning\pxrr@ruby@list^^J%
    pxrr@ruby@count = \@nameuse{pxrr@ruby@count}^^J%
    pxrr@end@kinsoku = \pxrr@end@kinsoku^^J%
    ----
  }%
}
\def\pxrr@debug@show@recomp{%
  \typeout{----\pxrr@pkgname\space recomp:^^J%
    pxrr@body@list = \meaning\pxrr@body@list^^J%
    pxrr@body@count = \pxrr@body@count^^J%
    pxrr@ruby@list = \meaning\pxrr@ruby@list^^J%
    pxrr@ruby@count = \pxrr@ruby@count^^J%
    pxrr@res = \meaning\pxrr@res^^J%
    ----
  }%
}
\def\pxrr@debug@show@concat{%
  \typeout{----\pxrr@pkgname\space concat:^^J%
    pxrr@body@list = \meaning\pxrr@body@list^^J%
    pxrr@ruby@list = \meaning\pxrr@ruby@list^^J%
    pxrr@whole@list = \meaning\pxrr@whole@list^^J%
    ----
  }%
}
\def\pxrr@debug@show@resolve@mode{%
  \typeout{----\pxrr@pkgname\space resolve-mode:
    \meaning\pxrr@mode}%
}
```

# 5 実装（圏点関連）

## 5.1 エラーメッセージ

指定の名前の圏点文字が未登録の場合。

```
\def\pxrr@warn@na@kmark#1{%
  \pxrr@warn{Unavailable kenten mark '#1'}%
}
```

パラメタ設定命令で無効な値が指定された場合。

```
\def\pxrr@err@invalid@value#1{%
  \pxrr@error{Invalid value '#1'}%
      {\@eha}%
}
```

## 5.2 パラメタ

### 5.2.1 全般設定

`\pxrr@k@ymark` 横組の主の圏点マークのコード。

```
\let\pxrr@k@ymark\@undefined
```

`\pxrr@k@ysmark` 横組の副の圏点マークのコード。

```
\let\pxrr@k@ysmark\@undefined
```

`\pxrr@k@tmark` 縦組の主の圏点マークのコード。

```
\let\pxrr@k@tmark\@undefined
```

`\pxrr@k@tsmark` 縦組の服の圏点マークのコード。

```
\let\pxrr@k@tsmark\@undefined
```

圏点マークの初期値の設定。

```
\AtEndOfPackage{%
  \pxrr@k@get@mark\pxrr@k@ymark{bullet*}%
  \pxrr@k@get@mark\pxrr@k@ysmark{sesame*}%
  \pxrr@k@get@mark\pxrr@k@tmark{sesame*}%
  \pxrr@k@get@mark\pxrr@k@tsmark{bullet*}%
}
```

`\pxrr@k@ruby@font` 圏点用フォント切替命令。

```
\let\pxrr@k@ruby@font\@empty
```

`\pxrr@k@size@ratio` 圏点文字サイズ。(`\kentensizeratio`)。実数値マクロ。

```
\def\pxrr@k@size@ratio{0.5}
```

`\ifpxrr@k@ghost` ゴースト処理を行うか。スイッチ。

※ 圏点では和文ゴースト処理を必ず行う。

```
\newif\ifpxrr@k@ghost \pxrr@k@ghosttrue
```

\pxrr@k@inter@gap 圏点と親文字の間の空き（\kentenintergap）。実数値マクロ。

```
\def\pxrr@k@inter@gap{0}
```

\pxrr@k@ruby@inter@gap 圏点とルビの間の空き（\kentenrubyintergap）。実数値マクロ。

```
\def\pxrr@k@ruby@inter@gap{0}
```

\pxrr@k@d@side 圏点を親文字の上下のどちらに付すか。0 = 上側；1 = 下側。\kentensetup の P／S の設定。整数定数。

```
\chardef\pxrr@k@d@side=0
```

\pxrr@k@d@mark 圏点マークの種類。0 = 主；1 = 副。\kentensetup の p／s の設定。整数定数。

```
\chardef\pxrr@k@d@mark=0
```

\pxrr@k@ruby@combo ルビと圏点が同時に適用された場合の挙動。0 = ルビだけ出力；1 = ルビの上に圏点（同時付加）。\kentenrubycombination の設定値に対応する。整数定数。

```
\chardef\pxrr@k@ruby@combo=1
```

\pxrr@k@d@full 約物にも圏点を付加するか。0 = 無効；1 = 有効。\kentensetup の f／F の設定。整数定数。

```
\chardef\pxrr@k@d@full=0
```

#### 5.2.2 呼出時の設定

\kenten の P／S の設定は、\pxrr@side をルビと共用する。

\pxrr@k@mark 圏点マークの種類。0 = 主；1 = 副。\kenten の p／s の設定。整数定数。

```
\chardef\pxrr@k@mark=0
```

\pxrr@k@full 約物にも圏点を付加するか。0 = 無効；1 = 有効。\kenten の f／F の設定。整数定数。

```
\chardef\pxrr@k@full=0
```

\pxrr@k@the@mark 適用される圏点マークの命令。

```
\let\pxrr@k@the@mark\relax
```

### 5.3 補助手続

#### 5.3.1 \UTF 命令対応

\ifpxrr@avail@UTF \UTF 命令が利用できるか。スイッチ。

```
\newif\ifpxrr@avail@UTF
```

\pxrr@decide@avail@UTF \ifpxrr@avail@UTF の値を確定させる。

```
\def\pxrr@decide@avail@UTF{%
  \global\let\pxrr@decide@avail@UTF\relax
  \ifx\UTF\@undefined \global\pxrr@avail@UTFfalse
  \else \global\pxrr@avail@UTFtrue
  \fi
}
```

### 5.3.2 リスト分解

\pxrr@k@decompose

\pxrr@k@decompose{⟨テキスト⟩}： テキスト（圏点命令の引数）を分解した結果の圏点項目リストを \pxrr@res に返す。

※ 圏点項目リストの形式：

\pxrr@entry[@XXX]{⟨引数⟩}……\pxrr@entry[@XXX]{⟨引数⟩}\pxrr@post

```
\def\pxrr@k@decompose#1{%
  \let\pxrr@res\@empty
  \pxrr@cntr=\z@
  \pxrr@k@decompose@loopa#1\pxrr@end
}
\def\pxrr@k@decompose@loopa{%
  \futurelet\pxrr@token\pxrr@k@decompose@loopb
}
\def\pxrr@k@decompose@loopb{%
  \pxrr@cond\ifx\pxrr@token\pxrr@end\fi{%
    \pxrr@appto\pxrr@res{\pxrr@post}%
  }{\pxrr@if@kspan@cmd\pxrr@token{%
    \pxrr@k@decompose@special\pxrr@k@decompose@kspan
  }{\pxrr@if@ruby@cmd\pxrr@token{%
    \pxrr@k@decompose@special\pxrr@k@decompose@ruby
  }{\pxrr@if@truby@cmd\pxrr@token{%
    \pxrr@k@decompose@special\pxrr@k@decompose@truby
  }{\pxrr@if@kenten@cmd\pxrr@token{%
    \pxrr@k@decompose@special\pxrr@k@decompose@kenten
  }{\pxrr@cond\ifx\pxrr@token\@sptoken\fi{%
    \pxrr@k@decompose@loope
  }{%
    \pxrr@setok{\pxrr@ifx{\pxrr@token\bgroup}}%
    \pxrr@k@decompose@loopc
  }}}}}%
}
\def\pxrr@k@decompose@loopc#1{%
  \pxrr@appto\pxrr@res{\pxrr@entry}%
  \ifpxrr@ok
    \pxrr@appto\pxrr@res{{{#1}}}%
  \else
    \pxrr@appto\pxrr@res{{#1}}%
  \fi
  \pxrr@k@decompose@loopd
}
\def\pxrr@k@decompose@loopd{%
  \advance\pxrr@cntr\@ne
  \pxrr@k@decompose@loopa
}
\expandafter\def\expandafter\pxrr@k@decompose@loope\space{%
  \pxrr@okfalse
```

```
  \pxrr@k@decompose@loopc{ }%
}
\def\pxrr@k@decompose@special#1#2#{%
  #1{#2}%
}
\def\pxrr@k@decompose@kspan#1#2{%
  \pxrr@appto\pxrr@res{\pxrr@entry@kspan{#1{#2}}}%
  \pxrr@k@decompose@loopd
}
\def\pxrr@k@decompose@ruby#1#2#3{%
  \pxrr@appto\pxrr@res{\pxrr@entry@ruby{#1{#2}{#3}}}%
  \pxrr@k@decompose@loopd
}
\def\pxrr@k@decompose@truby#1#2#3#4{%
  \pxrr@appto\pxrr@res{\pxrr@entry@ruby{#1{#2}{#3}{#4}}}%
  \pxrr@k@decompose@loopd
}
\def\pxrr@k@decompose@kenten#1#2{%
  \pxrr@appto\pxrr@res{\pxrr@entry@kenten{#1{#2}}}%
  \pxrr@k@decompose@loopd
}
\def\pxrr@cmd@ruby{\jruby}
\def\pxrr@cmd@kenten{\jkenten}
\def\pxrr@if@ruby@cmd#1{%
  \if \ifcat\noexpand#1\relax
        \ifx#1\pxrr@cmd@ruby T%
        \else\ifx#1\jruby T%
        \else\ifx#1\aruby T%
        \else F%
        \fi\fi\fi
      \else F%
      \fi T\expandafter\@firstoftwo
  \else \expandafter\@secondoftwo
  \fi
}
\def\pxrr@if@truby@cmd#1{%
  \if \ifcat\noexpand#1\relax
        \ifx#1\truby T%
        \else\ifx#1\atruby T%
        \else F%
        \fi\fi
      \else F%
      \fi T\expandafter\@firstoftwo
  \else \expandafter\@secondoftwo
  \fi
}
\def\pxrr@if@kspan@cmd#1{%
  \pxrr@cond\ifx#1\kspan\fi
}
```

```
\def\pxrr@if@kenten@cmd#1{%
  \if \ifcat\noexpand#1\relax
        \ifx#1\pxrr@cmd@kenten T%
        \else\ifx#1\jkenten T%
        \else F%
        \fi\fi
      \else F%
      \fi T\expandafter\@firstoftwo
  \else \expandafter\@secondoftwo
  \fi
}
```

## 5.4 パラメタ設定公開命令

\kentensetup \pxrr@k@parse@option で解析した後、設定値を全般設定にコピーする。

```
\newcommand*\kentensetup[1]{%
  \pxrr@in@setuptrue
  \pxrr@fatal@errorfalse
  \pxrr@k@parse@option{#1}%
  \ifpxrr@fatal@error\else
    \let\pxrr@k@d@side\pxrr@side
    \let\pxrr@k@d@mark\pxrr@k@mark
    \let\pxrr@k@d@full\pxrr@k@full
  \fi
```

\ifpxrr@in@setup を偽に戻す。ただし \ifpxrr@fatal@error は書き換えられたままであることに注意。

```
  \pxrr@in@setupfalse
}
```

\kentenfontsetup 対応するパラメタを設定する。

```
\newcommand*\kentenfontsetup{}
\def\kentenfontsetup#{%
  \def\pxrr@k@ruby@font
}
```

\kentensizeratio 対応するパラメタを設定する。

```
\newcommand*\kentensizeratio[1]{%
  \edef\pxrr@k@size@ratio{#1}%
}
```

\kentenintergap 対応するパラメタを設定する。

```
\newcommand*\kentenintergap[1]{%
  \edef\pxrr@k@inter@gap{#1}%
}
```

\kentenrubyintergap 対応するパラメタを設定する。

```
\newcommand*\kentenrubyintergap[1]{%
```

```
  \edef\pxrr@k@ruby@inter@gap{#1}%
}
```

\kentenmarkinyoko 対応するパラメタを設定する。

\kentensubmarkinyoko
\kentenmarkintate
\kentensubmarkintate

```
\newcommand*\kentenmarkinyoko[1]{%
  \pxrr@k@get@mark\pxrr@k@ymark{#1}%
}
\newcommand*\kentensubmarkinyoko[1]{%
  \pxrr@k@get@mark\pxrr@k@ysmark{#1}%
}
\newcommand*\kentenmarkintate[1]{%
  \pxrr@k@get@mark\pxrr@k@tmark{#1}%
}
\newcommand*\kentensubmarkintate[1]{%
  \pxrr@k@get@mark\pxrr@k@tsmark{#1}%
}
```

\kentenrubycombination 対応するパラメタを設定する。

```
\chardef\pxrr@k@ruby@combo@ruby=0
\chardef\pxrr@k@ruby@combo@both=1
\newcommand*\kentenrubycombination[1]{%
  \pxrr@letcs\pxrr@tempa{pxrr@k@ruby@combo@#1}%
  \ifx\pxrr@tempa\relax
    \pxrr@err@invalid@value{#1}%
  \else
    \let\pxrr@k@ruby@combo\pxrr@tempa
  \fi
}
```

## 5.5 圏点文字

\pxrr@k@declare@mark \pxrr@k@declare@mark{⟨名前⟩}{⟨本体⟩}： 圏点マーク命令を定義する。

```
\def\pxrr@k@declare@mark#1{%
  \global\@namedef{pxrr@k@mark@@#1}%
}
```

\pxrr@k@let@mark \pxrr@k@declare@mark{⟨名前⟩}\CS： 圏点マーク命令を \let で定義する。

```
\def\pxrr@k@let@mark#1{%
  \global\pxrr@cslet{pxrr@k@mark@@#1}%
}
```

\pxrr@k@get@mark \pxrr@k@get@mark\CS{⟨名前または定義本体⟩}： 指定の圏点マーク命令を \CS に代入する。第 2 引数の先頭トークンが ASCII 英字の場合は名前と見なし、それ以外は定義本体のコードと見なす。

```
\def\pxrr@k@get@mark#1#2{%
  \futurelet\pxrr@token\pxrr@k@get@mark@a#2\pxrr@nil#1%
}
```

```
\def\pxrr@k@get@mark@a{%
  \pxrr@cond\ifcat A\noexpand\pxrr@token\fi{%
    \pxrr@k@get@mark@c
  }{%else
    \pxrr@k@get@mark@b
  }%
}
\def\pxrr@k@get@mark@b#1\pxrr@nil#2{%
  \def#2{#1}%
}
\def\pxrr@k@get@mark@c#1#2\pxrr@nil#3{%
  \ifnum`#1<128
    \pxrr@letcs\pxrr@tempa{pxrr@k@mark@@#1#2}%
    \ifx\pxrr@tempa\relax
      \pxrr@warn@na@kmark{#1#2}%
    \else
      \let#3\pxrr@tempa
    \fi
  \else
    \pxrr@k@get@mark@b#1#2\pxrr@nil#3%
  \fi
}
```

\pxrr@k@declare@mark@char　\pxrr@k@declare@mark@char\CS{⟨二重コード⟩}: 指定のコード値の文字の（和文）chardef を \CS に代入する。ただし pTEX で JIS に無い文字（便宜的に和文空白の JIS コード値 2121 で表す）の場合は代わりに \pxrr@k@char@UTF を利用する。

```
\def\pxrr@k@declare@mark@char#1#2{%
  \pxrr@k@declare@mark@char@a{#1}#2\pxrr@end
}
\def\pxrr@k@declare@mark@char@a#1#2:#3\pxrr@end{%
  \pxrr@jchardef\pxrr@tempa\pxrr@jc{#2:#3}%
  \ifnum\pxrr@tempa=\pxrr@zspace
```

エンジンが pTEX でかつ JIS に無い文字である場合。

```
    \pxrr@k@declare@mark{#1}{\pxrr@k@char@UTF{#1}{#3}}%
  \else
    \pxrr@k@let@mark{#1}\pxrr@tempa
  \fi
}
```

\pxrr@k@char@UTF　\pxrr@k@char@UTF{⟨名前⟩}{⟨Unicode 値⟩}: \UTF{⟨Unicode 値⟩} を実行するが、\UTF が利用不可の場合は、（最初の 1 回だけ）警告した上で何も出力しない。

```
\def\pxrr@k@char@UTF#1#2{%
  \pxrr@decide@avail@UTF
  \ifpxrr@avail@UTF
    \pxrr@k@declare@mark{#1}{\UTF{#2}}%
    \UTF{#2}%
  \else
    \pxrr@k@let@mark{#1}\@empty
```

```
    \pxrr@warn@na@kmark{#1}%
  \fi
}
```

標準サポートの圏点マークの定義。

```
\pxrr@k@declare@mark@char{bullet}  {2121:2022}
\pxrr@k@declare@mark@char{triangle}{2225:25B2}
\pxrr@k@declare@mark@char{Triangle}{2224:25B3}
\pxrr@k@declare@mark@char{fisheye} {2121:25C9}
\pxrr@k@declare@mark@char{Circle}  {217B:25CB}
\pxrr@k@declare@mark@char{bullseye}{217D:25CE}
\pxrr@k@declare@mark@char{circle}  {217C:25CF}
\pxrr@k@declare@mark@char{Bullet}  {2121:25E6}
\pxrr@k@declare@mark@char{sesame}  {2121:FE45}
\pxrr@k@declare@mark@char{Sesame}  {2121:FE46}
\pxrr@jchardef\pxrr@ja@dot=\pxrr@jc{2126:30FB}
\pxrr@jchardef\pxrr@ja@comma=\pxrr@jc{2122:3001}
\pxrr@k@declare@mark{bullet*}{%
  \pxrr@dima=\pxrr@ruby@zw\relax
  \hb@xt@\pxrr@dima{%
    \kern-.5\pxrr@dima
    \pxrr@if@in@tate{}{\lower.38\pxrr@dima}%
    \hbox{%
      \pxrr@dima=\f@size\p@
      \fontsize{2\pxrr@dima}{\z@}\selectfont
      \pxrr@ja@dot
    }%
    \hss
  }%
}
\pxrr@k@declare@mark{sesame*}{%
  \pxrr@dima=\pxrr@ruby@zw\relax
  \hb@xt@\pxrr@dima{%
    \pxrr@if@in@tate{\kern.1\pxrr@dima}{\kern.05\pxrr@dima}%
    \pxrr@if@in@tate{\lower.85\pxrr@dima}{\raise.3\pxrr@dima}%
    \hbox{%
      \pxrr@dima=\f@size\p@
      \fontsize{2.4\pxrr@dima}{\z@}\selectfont
      \pxrr@ja@comma
    }%
    \hss
  }%
}
```

## 5.6 圏点オプション解析

`\pxrr@k@parse@option` `\pxrr@k@parse@option{`⟨オプション⟩`}`：⟨オプション⟩を解析し、`\pxrr@side` や `\pxrr@k@mark` 等のパラメタを設定する。

```
\def\pxrr@k@parse@option#1{%
  \edef\pxrr@tempa{#1}%
  \let\pxrr@side\pxrr@k@d@side
  \let\pxrr@k@mark\pxrr@k@d@mark
  \let\pxrr@k@full\pxrr@k@d@full
  \expandafter\pxrr@k@parse@option@loop\pxrr@tempa @\pxrr@end
}
\def\pxrr@k@parse@option@loop#1{%
```

圏点オプションの解析器は“有限状態”を持たないので非常に単純である。

```
  \pxrr@letcs\pxrr@tempa{pxrr@k@po@PR@#1}%
  \pxrr@cond\ifx\pxrr@tempa\relax\fi{%
    \pxrr@fatal@knx@letter{#1}%
    \pxrr@k@parse@option@exit
  }{%
    \pxrr@tempa
    \pxrr@k@parse@option@loop
  }%
}
\def\pxrr@k@parse@option@exit#1\pxrr@end{%
  \ifpxrr@in@setup\else
    \pxrr@k@check@option
```

ここで \pxrr@k@the@mark を適切に定義する。

```
    \pxrr@if@in@tate{%
      \ifcase\pxrr@k@mark \let\pxrr@k@the@mark\pxrr@k@tmark
      \or \let\pxrr@k@the@mark\pxrr@k@tsmark
      \fi
    }{%
      \ifcase\pxrr@k@mark \let\pxrr@k@the@mark\pxrr@k@ymark
      \or \let\pxrr@k@the@mark\pxrr@k@ysmark
      \fi
    }%
  \fi
}
\def\pxrr@k@po@PR@@{%
  \pxrr@k@parse@option@exit
}
\def\pxrr@k@po@PR@P{%
  \chardef\pxrr@side\z@
}
\def\pxrr@k@po@PR@S{%
  \chardef\pxrr@side\@ne
}
\def\pxrr@k@po@PR@p{%
  \chardef\pxrr@k@mark\z@
}
\def\pxrr@k@po@PR@s{%
  \chardef\pxrr@k@mark\@ne
```

```
}
\def\pxrr@k@po@PR@F{%
  \chardef\pxrr@k@full\z@
}
\def\pxrr@k@po@PR@f{%
  \chardef\pxrr@k@full\@ne
}
```

## 5.7 オプション整合性検査

今のところ検査すべき点がない。

```
\def\pxrr@k@check@option{%
}
```

## 5.8 ブロック毎の組版

\pxrr@k@compose@block

\pxrr@k@compose@block{⟨親文字ブロック⟩}{⟨圏点の個数⟩}: 1 つのブロックの組版処理。ボックス \pxrr@boxb に圏点 1 つを組版したものが入っている必要がある。なお、圏点はゼロ幅に潰した形で扱う前提のため、\pxrr@boxb の幅はゼロでないといけない。

基本的に、ルビ用の \pxrr@compose@oneside@block を非常に簡略化した処理になっている。

```
\def\pxrr@k@compose@block#1#2{%
  \setbox\pxrr@boxa\pxrr@hbox{#1}%
```

\pxrr@evenspace@int を使うために辻褄を合わせる。すなわち、\copy\pxrr@boxb を圏点個数分だけ反復したリストを \pxrr@res に入れて、“圏点の自然長”に当たる \pxrr@natwd をゼロとする。

```
  \pxrr@k@make@rep@list{\copy\pxrr@boxb}{#2}%
  \let\pxrr@natwd\pxrr@zeropt
  \pxrr@evenspace@int\pxrr@locate@inner\pxrr@boxr
       \relax{\wd\pxrr@boxa}%
  \setbox\z@\hbox{%
    \ifnum\pxrr@side=\z@
      \raise\pxrr@ruby@raise\box\pxrr@boxr
    \else
      \lower\pxrr@ruby@lower\box\pxrr@boxr
    \fi
  }%
  \ht\z@\z@ \dp\z@\z@
  \@tempdima\wd\z@
  \setbox\pxrr@boxr\hbox{%
    \box\z@
    \kern-\@tempdima
    \box\pxrr@boxa
  }%
}
```

\pxrr@k@make@rep@list　\pxrr@k@make@rep@list{⟨要素⟩}{⟨回数⟩}： 要素を指定の回数だけ反復したリストを \pxrr@res に代入する。

```
\def\pxrr@k@make@rep@list#1#2{%
  \def\pxrr@res{\pxrr@pre{#1}}%
  \pxrr@cntr=#2\relax
  \ifnum\pxrr@cntr>\@ne
    \@tempcnta\pxrr@cntr \advance\@tempcnta\m@ne
    \@whilenum{\@tempcnta>\z@}\do{%
      \pxrr@appto\pxrr@res{\pxrr@inter{#1}}%
      \advance\@tempcnta\m@ne
    }%
  \fi
  \pxrr@appto\pxrr@res{\pxrr@post}%
}
```

## 5.9 圏点項目

- 圏点項目リスト： テキストを \pxrr@k@decompose で分解した結果のリスト。
- 圏点項目： 圏点リストに含まれる \pxrr@entry[@XXX]{...} という形式のこと。圏点項目は直接に実行する（出力する）ことができる。
- 圏点ブロック： 一つの《文字》に圏点を付加して出力したもの。
- 参照文字コード： 圏点項目の出力の前後の禁則ペナルティの扱いにおいて、「ある文字と同等」と扱う場合の、その文字の文字コード。

※現状では、まず \pxrr@kenten@entry@XXX というマクロを定義して圏点命令の実行時にそれを \pxrr@entry@XXX にコピーする、という手続きを採っている。（ただそうする意味が全く無い気がする。）

\ifpxrr@k@first@entry　先頭の項目であるか。

```
\newif\ifpxrr@k@first@entry
```

\ifpxrr@k@last@entry　末尾の項目であるか。

```
\newif\ifpxrr@k@last@entry
```

\ifpxrr@k@prev@is@block　直前の項目の結果が圏点ブロックであったか。

```
\newif\ifpxrr@k@prev@is@block
```

\pxrr@k@accum@res　累積の直接出力。

```
\let\pxrr@k@accum@res\relax
```

以下の 3 つの変数は“項目の下請けマクロ”が値を返すべきもの。これらに加えて、\pxrr@res と \pxrr@boxr の一方に（組版の）結果を返す必要がある。

\pxrr@k@prebreakpenalty　圏点項目の前禁則ペナルティ。

```
\mathchardef\pxrr@k@prebreakpenalty\z@
```

\pxrr@k@postbreakpenalty　圏点項目の後禁則ペナルティ。

```
\mathchardef\pxrr@k@postbreakpenalty\z@
```

\pxrr@k@entry@res@type　項目の出力のタイプ。0=直接出力；1=ボックス出力；2=圏点ブロック。0 の場合、出力は \pxrr@res にあり、それ以外は、出力は \pxrr@boxr にある。

```
\chardef\pxrr@k@entry@res@type\z@
```

\pxrr@k@list@pre　圏点項目リストの出力の開始時に行う処理。

```
\def\pxrr@k@list@pre{%
  \pxrr@k@first@entrytrue
  \pxrr@k@last@entryfalse
  \pxrr@k@prev@is@blockfalse
  \let\pxrr@k@accum@res\@empty
  \chardef\pxrr@k@block@seq@state\z@
}
```

\pxrr@k@entry@with　補助マクロ。各種圏点項目の共通の処理を行う。

※ #1 は各圏点項目命令の下請けのマクロで、#2 は圏点項目の引数。

```
\def\pxrr@k@entry@with#1#2{%
  \pxrr@if@last{%
    \pxrr@k@last@entrytrue
    \pxrr@k@entry@with@a#1{#2}%
  }{%
    \pxrr@k@entry@with@a#1{#2}%
  }%
}
\def\pxrr@k@entry@with@a#1#2{%
  \mathchardef\pxrr@k@prebreakpenalty\z@
  \mathchardef\pxrr@k@postbreakpenalty\z@
```

下請けマクロを実行して結果を得る。

```
  #1{#2}%
%\typeout{%
%first=\meaning\ifpxrr@k@first@entry^^J%
%last=\meaning\ifpxrr@k@last@entry^^J%
%prev=\meaning\ifpxrr@k@prev@is@block^^J%
%res=\meaning\pxrr@res^^J%
%type=\meaning\pxrr@k@entry@res@type^^J%
%prepen=\the\pxrr@k@prebreakpenalty^^J%
%postpen=\the\pxrr@k@postbreakpenalty}%
```

累積直接出力の処理。

```
  \ifnum\pxrr@k@entry@res@type=\z@
    \expandafter\pxrr@appto\expandafter\pxrr@k@accum@res
        \expandafter{\pxrr@res}%
  \else
    \pxrr@k@accum@res
    \let\pxrr@k@accum@res\@empty
  \fi
```

前禁則ペナルティを入れる。

```
  \ifnum\pxrr@k@prebreakpenalty>\z@
    \@tempcntb\lastpenalty \unpenalty
    \advance\@tempcntb\pxrr@k@prebreakpenalty
    \penalty\@tempcntb
  \fi
```

圏点ブロックが連続する場合は和文間空白を入れる。

```
  \ifnum\pxrr@k@entry@res@type=\tw@
    \ifpxrr@k@prev@is@block
      \pxrr@inter@mono
    \fi
    \pxrr@k@prev@is@blocktrue
  \else
    \pxrr@k@prev@is@blockfalse
  \fi
```

ボックスの結果を実際に出力する。

```
  \ifnum\pxrr@k@entry@res@type>\z@
    \unhbox\pxrr@boxr
  \fi
```

後禁則ペナルティを入れる。

```
  \ifnum\pxrr@k@postbreakpenalty>\z@
    \penalty\pxrr@k@postbreakpenalty
  \fi
```

次の項目に進む。

```
  \pxrr@k@first@entryfalse
}
```

`\pxrr@k@list@post` 圏点項目リストの出力の最後に行う処理。

```
\def\pxrr@k@list@post{%
  \pxrr@k@accum@res
  \let\pxrr@k@accum@res\@empty
}
```

`\pxrr@kenten@entry` 一般の《文字》を表す圏点項目 `\pxrr@entry{`⟨文字⟩`}` の処理。圏点を 1 つ付けて出力する。

```
\def\pxrr@kenten@entry{%
  \pxrr@k@entry@with\pxrr@kenten@entry@
}
\def\pxrr@kenten@entry@#1{%
  \pxrr@k@check@char{#1}%
  \ifpxrr@ok
    \pxrr@k@compose@block{#1}\@ne
    \chardef\pxrr@k@entry@res@type=\tw@
  \else
    \def\pxrr@res{#1}%
    \chardef\pxrr@k@entry@res@type=\z@
  \fi
```

```
}
```

\pxrr@kenten@entry@kspan　\kspan 命令を表す圏点項目 \pxrr@entry@kspan{\kspan{⟨テキスト⟩}} の処理。テキストの幅が“およそ $n$ 全角”である場合に、$n$ 個の圏点をルビ均等割りで配置して出力する。

```
\def\pxrr@kenten@entry@kspan{%
  \pxrr@k@entry@with\pxrr@kenten@entry@kspan@
}
\def\pxrr@kenten@entry@kspan@#1{%
  \pxrr@kenten@entry@kspan@a#1%
}
\def\pxrr@kenten@entry@kspan@a#1{%
```

\kspan（＝#1）が * 付かを調べる。

```
  \@ifstar{%
    \@testopt\pxrr@kenten@entry@kspan@c{}%
  }{%
    \@testopt\pxrr@kenten@entry@kspan@b{}%
  }%
}
\def\pxrr@kenten@entry@kspan@b[#1]#2{%
```

$(n-1/4)$zw 以上 $(n+3/4)$zw 未満の時に“およそ $n$ 全角”と見なす。

```
  \setbox\z@\pxrr@hbox{#2}%
  \@tempdima\pxrr@body@zw\relax
  \@tempdimb\wd\z@ \advance\@tempdimb.25\@tempdima
  \divide\@tempdimb\@tempdima
  \edef\pxrr@kenten@entry@tempa{\number\@tempdimb}%
  \pxrr@k@compose@block{#2}\pxrr@kenten@entry@tempa
  \chardef\pxrr@k@entry@res@type=\tw@
}
\def\pxrr@kenten@entry@kspan@c[#1]#2{%
```

\kspan* となっている場合。この時は圏点を付加せず直接出力する。

```
  \def\pxrr@res{#2}%
  \chardef\pxrr@k@entry@res@type=\z@
}
```

\pxrr@kenten@entry@kenten　ネストした \kenten 命令の圏点項目。単純にその \kenten を実行したものを出力とする。すなわち、内側の圏点の設定のみが生きる。

```
\def\pxrr@kenten@entry@kenten{%
  \pxrr@k@entry@with\pxrr@kenten@entry@kenten@
}
\def\pxrr@kenten@entry@kenten@#1{%
```

この場合は圏点ブロックとは見なさないことに注意。

```
  \setbox\pxrr@boxr\hbox{#1}%
  \chardef\pxrr@k@entry@res@type=\@ne
}
```

\pxrr@kenten@entry@ruby　ルビ命令の圏点項目。

```
\def\pxrr@kenten@entry@ruby{%
  \pxrr@k@entry@with\pxrr@kenten@entry@ruby@
}
\def\pxrr@kenten@entry@ruby@#1{%
  \pxrr@apply@combotrue
  \setbox\pxrr@boxr\hbox{#1}%
  \chardef\pxrr@k@entry@res@type=\@ne
}
```

### 5.9.1 \kspan 命令

\kspan テキストの幅に相応した個数の圏点を付ける命令。\kenten の引数のテキストの中で使う。\kenten の外で使われた場合は単純に引数を出力するだけ。

※ 処理の都合上、オプション引数を持たせているが、実際には（現在は）これは使われない。

```
\newcommand*\kspan{%
  \@ifstar{%
    \@testopt\pxrr@kspan@a{}%
  }{%
    \@testopt\pxrr@kspan@a{}%
  }%
}
\pxrr@add@protect\kspan
\def\pxrr@kspan@a[#1]#2{%
  \begingroup
    #2%
  \endgroup
}
```

## 5.10 自動抑止の検査

\pxrr@k@check@char 通常項目（\pxrr@entry）の引数を検査して、圏点を付加すべきか否かをスイッチ pxrr@ok に返す。また、項目の前禁則・後禁則ペナルティを設定する。

引数が（単一の）通常文字である時はその文字、引数がグループの場合は和文空白の内部文字コードを \pxrr@cntr に返す（禁則ペナルティを後で見られるように）。

```
\def\pxrr@k@check@char#1{%
  \futurelet\pxrr@token\pxrr@k@check@char@a#1\pxrr@end
}
\def\pxrr@k@check@char@a#1\pxrr@end{%
  \pxrr@cond\ifx\pxrr@token\bgroup\fi{%
```

グループには圏点を付ける。

```
    \pxrr@oktrue
  }{\pxrr@cond\ifx\pxrr@token\@sptoken\fi{%
```

欧文空白には圏点を付けない。

```
    \pxrr@okfalse
  }{%
```

```
    \pxrr@check@char\pxrr@token
    \ifcase\pxrr@cntr
```
通常文字でないので圏点を付けない。
```
      \pxrr@okfalse
    \or
```
欧文の通常文字。圏点を付ける。
```
      \pxrr@oktrue
      \chardef\pxrr@check@char@temp\z@
    \or
```
和文の通常文字。圏点を付ける。
```
      \pxrr@oktrue
      \chardef\pxrr@check@char@temp\@ne
    \fi
```
約物の圏点付加が無効の場合は、引数の文字が約物であるか検査し、そうである場合は圏点を付けない。
```
    \ifnum\pxrr@k@full=\z@\ifpxrr@ok
      \pxrr@check@punct@char{`#1}\pxrr@check@char@temp
      \ifpxrr@ok \pxrr@okfalse
      \else \pxrr@oktrue
      \fi
    \fi\fi
    \ifpxrr@ok
      \pxrr@get@prebreakpenalty\@tempcnta{`#1}%
      \mathchardef\pxrr@k@prebreakpenalty\@tempcnta
      \pxrr@get@postbreakpenalty\@tempcnta{`#1}%
      \mathchardef\pxrr@k@postbreakpenalty\@tempcnta
    \fi
  }}%
}
```

## 5.11 メインです

### 5.11.1 エントリーポイント

\kenten
\jkenten

圏点の公開命令。\jkenten を頑強な命令として定義した上で、\kenten はそれに展開されるマクロに（未定義ならば）定義する。

```
\AtBeginDocument{%
  \providecommand*{\kenten}{\jkenten}%
}
\newcommand*{\jkenten}{%
  \pxrr@k@prologue
  \pxrr@kenten
}
\pxrr@add@protect\jkenten
```

\pxrr@kenten

オプションの処理を行う。

```
\def\pxrr@kenten{%
  \@testopt\pxrr@kenten@a{}%
}
\def\pxrr@kenten@a[#1]{%
  \def\pxrr@option{#1}%
  \ifpxrr@safe@mode
```

安全モードでは圏点機能は無効なので、フォールバックとして引数のテキストをそのまま出力する。

```
    \expandafter\@firstofone
  \else
    \expandafter\pxrr@kenten@proc
  \fi
}
```

`\pxrr@k@bind@param` “呼出時変数”へのコピーを行う。

```
\def\pxrr@k@bind@param{%
  \let\pxrr@c@ruby@font\pxrr@k@ruby@font
  \let\pxrr@c@size@ratio\pxrr@k@size@ratio
  \let\pxrr@c@inter@gap\pxrr@k@inter@gap
}
```

`\pxrr@kenten@proc` `\pxrr@kenten@proc{`⟨親文字列⟩`}`： これが手続の本体となる。

```
\def\pxrr@kenten@proc#1{%
  \pxrr@prepare@fallback{#1}%
  \pxrr@k@bind@param
  \pxrr@assign@fsize
  \pxrr@k@parse@option\pxrr@option
  \pxrr@if@alive{%
    \pxrr@k@decompose{#1}%
    \let\pxrr@body@list\pxrr@res
    \pxrr@kenten@main
  }%
  \pxrr@kenten@exit
}
```

### 5.11.2 組版処理

`\pxrr@kenten@main` 圏点の組版処理。

```
\def\pxrr@kenten@main{%
  \setbox\pxrr@boxb\pxrr@hbox@to\z@{%
    \pxrr@use@ruby@font
    \hss\pxrr@k@the@mark\hss
  }%
  \let\pxrr@entry\pxrr@kenten@entry
  \let\pxrr@entry@kspan\pxrr@kenten@entry@kspan
  \let\pxrr@entry@ruby\pxrr@kenten@entry@ruby
  \let\pxrr@entry@kenten\pxrr@kenten@entry@kenten
  \let\pxrr@post\pxrr@k@list@post
```

```
  \pxrr@k@list@pre
  \pxrr@body@list
}
```

### 5.11.3 前処理

\pxrr@jprologue 圏点用の開始処理。

```
\def\pxrr@k@prologue{%
  \ifpxrr@k@ghost
    \pxrr@jghost@char
    \pxrr@inhibitglue
  \fi
  \begingroup
    \ifpxrr@k@ghost
      \setbox\pxrr@boxa\hbox{\pxrr@jghost@char}%
      \kern-\wd\pxrr@boxa
    \fi
}
```

### 5.11.4 後処理

\pxrr@kenten@exit 出力を終えて、最後に呼ばれるマクロ。

```
\def\pxrr@kenten@exit{%
  \ifpxrr@fatal@error
    \pxrr@fallback
  \fi
  \pxrr@k@epilogue
}
```

\pxrr@jepilogue 終了処理。

```
\def\pxrr@k@epilogue{%
    \ifpxrr@k@ghost
      \setbox\pxrr@boxa\hbox{\pxrr@jghost@char}%
      \kern-\wd\pxrr@boxa
    \fi
  \endgroup
  \ifpxrr@k@ghost
    \pxrr@inhibitglue
    \pxrr@jghost@char
  \fi
}
```

## 5.12 デバッグ用出力

```
\def\pxrr@debug@show@kenten@input{%
  \typeout{%
pxrr@k@the@mark=\meaning\pxrr@k@the@mark^^J%
```

```
pxrr@side=\meaning\pxrr@side^^J%
pxrr@body@list=\meaning\pxrr@body@list^^J%
  }%
}
```

# 6 実装（圏点ルビ同時付加）

コンボ！

## 6.1 呼出時パラメタ

\ifpxrr@apply@combo 直後に実行するルビ命令について同時付加を行うか。スイッチ。

```
\newif\ifpxrr@apply@combo
```

\ifpxrr@combo 現在実行中のルビ命令について同時付加を行うか。スイッチ。

```
\newif\ifpxrr@combo
```

\pxrr@ck@ruby@font 同時付加時の圏点側の呼出時パラメタの値。

\pxrr@ck@size@ratio
\pxrr@ck@inter@gap
\pxrr@ck@ruby@inter@gap
\pxrr@ck@side
\pxrr@ck@the@mark
\pxrr@ck@ruby@combo

```
\let\pxrr@ck@ruby@font\relax
\let\pxrr@ck@size@ratio\relax
\let\pxrr@ck@inter@gap\relax
\let\pxrr@ck@ruby@inter@gap\relax
\let\pxrr@ck@side\relax
\let\pxrr@ck@the@mark\relax
\let\pxrr@ck@ruby@combo\relax
```

\ifpxrr@ck@kenten@head 当該のルビ命令が、圏点命令の引数の先頭にあるか。

```
\newif\ifpxrr@ck@kenten@head
```

\ifpxrr@ck@kenten@end 当該のルビ命令が、圏点命令の引数の先頭にあるか。

```
\newif\ifpxrr@ck@kenten@end
```

\pxrr@ck@bind@param “呼出時変数”へのコピーを行う。

```
\def\pxrr@ck@bind@param{%
  \let\pxrr@ck@ruby@font\pxrr@c@ruby@font
  \let\pxrr@ck@size@ratio\pxrr@c@size@ratio
  \let\pxrr@ck@inter@gap\pxrr@c@inter@gap
  \let\pxrr@ck@ruby@inter@gap\pxrr@k@ruby@inter@gap
  \let\pxrr@ck@side\pxrr@side
  \let\pxrr@ck@the@mark\pxrr@k@the@mark
  \let\pxrr@ck@ruby@combo\pxrr@k@ruby@combo
  \pxrr@csletcs{ifpxrr@ck@kenten@head}{ifpxrr@k@first@entry}%
  \pxrr@csletcs{ifpxrr@ck@kenten@end}{ifpxrr@k@last@entry}%
}
```

## 6.2 その他の変数

\pxrr@ck@zw 圏点の全角幅。

```
\let\pxrr@ck@zw\relax
```

\pxrr@ck@raise@P　ルビ側が P である場合の、圏点の垂直方向の移動量。
※ 圏点側が S である場合は負値になる。

```
\let\pxrr@ck@raise@P\relax
```

\pxrr@ck@raise@S　ルビ側が S である場合の、圏点の垂直方向の移動量。

```
\let\pxrr@ck@raise@S\relax
```

\pxrr@ck@raise@t　ルビ側が両側ルビである場合の、圏点の垂直方向の移動量。

```
\let\pxrr@ck@raise@t\relax
```

## 6.3　オプション整合性検査

\pxrr@ck@check@option　同時付加のための呼出時パラメタの調整。

```
\def\pxrr@ck@check@option{%
  \ifpxrr@ck@kenten@head
    \let\pxrr@bintr@\@empty
    \let\pxrr@bscomp=.\relax
    \pxrr@bnobrtrue
  \fi
  \ifpxrr@ck@kenten@end
    \let\pxrr@aintr@\@empty
    \let\pxrr@ascomp=.\relax
    \pxrr@anobrtrue
  \fi
}
```

## 6.4　フォントサイズ

\pxrr@ck@assign@fsize　フォントに関連する設定。

```
\def\pxrr@ck@assign@fsize{%
```

\pxrr@ck@zw の値を求める。

```
  \begingroup
    \@tempdima=\f@size\p@
    \@tempdima\pxrr@ck@size@ratio\@tempdima
    \edef\pxrr@ruby@fsize{\the\@tempdima}%
    \let\pxrr@c@ruby@font\pxrr@ck@ruby@font
    \pxrr@use@ruby@font
    \pxrr@get@zwidth\pxrr@ck@zw
    \global\let\pxrr@gtempa\pxrr@ck@zw
  \endgroup
  \let\pxrr@ck@zw\pxrr@gtempa
```

\pxrr@ck@raise@P、\pxrr@ck@raise@S の値を計算する。

```
  \ifcase\pxrr@ck@side
```

圏点側が P の場合。

```
    \@tempdimc\pxrr@ck@zw
    \advance\@tempdimc-\pxrr@htratio\@tempdimc
    \@tempdima\pxrr@ruby@raise\relax
    \@tempdimb\pxrr@ruby@zw\relax
    \advance\@tempdima\pxrr@htratio\@tempdimb
    \@tempdimb\pxrr@body@zw\relax
    \advance\@tempdima\pxrr@ck@ruby@inter@gap\@tempdimb
    \advance\@tempdima\@tempdimc
    \edef\pxrr@ck@raise@P{\the\@tempdima}%
    \@tempdima\pxrr@body@zw\relax
    \@tempdima\pxrr@htratio\@tempdima
    \@tempdimb\pxrr@body@zw\relax
    \advance\@tempdima\pxrr@ck@inter@gap\@tempdimb
    \advance\@tempdima\@tempdimc
    \edef\pxrr@ck@raise@S{\the\@tempdima}%
    \let\pxrr@ck@raise@t\pxrr@ck@raise@P
  \or
```

圏点側が S の場合。

```
    \@tempdimc\pxrr@ck@zw
    \@tempdimc\pxrr@htratio\@tempdimc
    \@tempdima-\pxrr@ruby@lower\relax
    \@tempdimb\pxrr@ruby@zw\relax
    \advance\@tempdimb-\pxrr@htratio\@tempdimb
    \advance\@tempdima-\@tempdimb
    \@tempdimb\pxrr@body@zw\relax
    \advance\@tempdima-\pxrr@ck@ruby@inter@gap\@tempdimb
    \advance\@tempdima-\@tempdimc
    \edef\pxrr@ck@raise@S{\the\@tempdima}%
    \@tempdima-\pxrr@body@zw\relax
    \advance\@tempdima-\pxrr@htratio\@tempdima
    \@tempdimb\pxrr@body@zw\relax
    \advance\@tempdima-\pxrr@ck@inter@gap\@tempdimb
    \advance\@tempdima-\@tempdimc
    \edef\pxrr@ck@raise@P{\the\@tempdima}%
    \let\pxrr@ck@raise@t\pxrr@ck@raise@S
  \fi
}
```

## 6.5 ブロック毎の組版

\pxrr@ck@body@natwd 親文字列の自然長。

```
\let\pxrr@ck@body@natwd\relax
```

\pxrr@ck@locate 圏点列のパターン指定。

```
\let\pxrr@ck@locate\relax
```

\pxrr@ck@kenten@list　圏点列のリスト。

```
\let\pxrr@ck@kenten@list\relax
```

\pxrr@ck@compose　#1 に親文字テキスト、\pxrr@ck@body@natwd に親文字の自然長、ボックス 0 にルビ出力、\pxrr@boxa に親文字出力、\pxrr@ck@locate にパターンが入っている前提で、ボックス 0 に圏点を追加する。

```
\def\pxrr@ck@compose#1{%
```

圏点を組んだボックスを作る。

```
  \setbox\tw@\pxrr@hbox@to\z@{%
    \@tempdima=\f@size\p@
    \@tempdima\pxrr@ck@size@ratio\@tempdima
    \edef\pxrr@ruby@fsize{\the\@tempdima}%
    \let\pxrr@c@ruby@font\pxrr@ck@ruby@font
    \pxrr@use@ruby@font
    \hss\pxrr@ck@the@mark\hss
  }%
```

親文字テキストを分解した後、リスト \pxrr@res を圏点のリストに置き換える。

```
  \pxrr@save@listproc
  \pxrr@decompose{#1}%
  \def\pxrr@pre{%
    \let\pxrr@res\@empty
    \pxrr@ck@compose@entry\pxrr@pre
  }%
  \def\pxrr@inter{%
    \pxrr@ck@compose@entry\pxrr@inter
  }%
  \def\pxrr@post{%
    \pxrr@appto\pxrr@res{\pxrr@post}%
  }%
  \pxrr@res
  \pxrr@restore@listproc
  \let\pxrr@natwd\pxrr@ck@body@natwd
```

圏点リストを均等配置する。

```
  \pxrr@evenspace@int\pxrr@ck@locate\pxrr@boxb\relax
      {\wd\pxrr@boxa}%
```

合成処理。

```
  \setbox\z@\hbox{%
    \unhcopy\z@
    \kern-\wd\z@
    \ifcase\pxrr@side
      \raise\pxrr@ck@raise@P
    \or
      \raise\pxrr@ck@raise@S
    \or
      \raise\pxrr@ck@raise@t
    \fi
```

```
    \hb@xt@\wd\pxrr@boxa{\hss\copy\pxrr@boxb\hss}%
  }%
}
\def\pxrr@ck@compose@entry#1#2{%
  \setbox\pxrr@boxb\pxrr@hbox{#2}%
  \edef\pxrr@tempa{%
    \noexpand\pxrr@appto\noexpand\pxrr@res{\noexpand#1{%
        \hb@xt@\the\wd\pxrr@boxb{\hss\copy\tw@\hss}}}%
  }\pxrr@tempa
}
```

# 7 実装：hyperref 対策

PDF 文字列中ではルビ命令や圏点命令が“無難な出力”をするようにする。現状では、ルビ・圏点ともに親文字のみを出力することにする。

`\pxrr@dumb@sub` オプション部分を読み飛ばす補助マクロ。

```
\def\pxrr@dumb@sub#1#2#{#1}
```

`\pxrr@dumb@ruby` 無難なルビ命令。

```
\def\pxrr@dumb@ruby{%
  \pxrr@dumb@sub\pxrr@dumb@ruby@
}
\def\pxrr@dumb@ruby@#1#2{#1}
```

`\pxrr@dumb@truby` 無難な両側ルビ命令。

```
\def\pxrr@dumb@truby{%
  \pxrr@dumb@sub\pxrr@dumb@truby@
}
\def\pxrr@dumb@truby@#1#2#3{#1}
```

`\pxrr@dumb@tkenten` 無難な圏点命令。

※ `\kspan` もこの定義を利用する。

```
\def\pxrr@dumb@kenten{%
  \pxrr@dumb@sub\pxrr@dumb@kenten@
}
\def\pxrr@dumb@kenten@#1{#1}
```

hyperref の `\pdfstringdef` 用のフック `\pdfstringdefPreHook` に上書き処理を追記する。

```
\providecommand*\pdfstringdefPreHook{}
\g@addto@macro\pdfstringdefPreHook{%
```

`\ruby` と `\kenten` は「本パッケージの命令であるか」の検査が必要。

```
  \ifx\pxrr@cmd@ruby\ruby
    \let\ruby\pxrr@dumb@ruby
  \fi
  \let\jruby\pxrr@dumb@ruby
  \let\aruby\pxrr@dumb@ruby
```

```
  \let\truby\pxrr@dumb@truby
  \let\atruby\pxrr@dumb@truby
  \ifx\pxrr@cmd@kenten\kenten
    \let\kenten\pxrr@dumb@kenten
  \fi
  \let\kspan\pxrr@dumb@kenten
}
```