pxrubrica パッケージ

八登 崇之　(Takayuki YATO; aka “ZR”)

v0.1　[2011/07/23]

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　基本機能

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。ここで、⟨オプション⟩ は以下の形式をもつ。
  ⟨前進入設定⟩⟨前空き設定⟩⟨モード⟩⟨前空き設定⟩⟨後設定⟩

⟨モード⟩ 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

⟨前空き設定⟩ は以下の値の何れか。

:	和欧文間空白

⟨モード⟩ は以下の値の何れか。

-	（無指定）		m	($<mono$)	モノルビ
h	($<head$)	肩付き	g	($<group$)	グループルビ
c	($<center$)	中付き	j	($<jukugo$)	熟語ルビ
P	($<primary$)	上側配置			
S	($<secondary$)	下側配置			

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。

⟨後空き設定⟩ は以下の値の何れか。

:	和欧文間空白

⟨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。⟨オプション⟩ の指定方法は \ruby と同じだが、欧文ルビは常に（単純）グループルビとなるので、m、g、j の指定は無視される。
- \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。
  両側ルビは常に（単純）グループルビとなるので、⟨オプション⟩ の中の m、g、j の指定は無視される。
- \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。

2.3 入力文字列のグループの指定

入力文字列（親文字列・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[m]{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（詳しくは後述）。

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。）なお、「ゴースト（ghost）」というのは Omega の用語で、「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ので、ルビ命令のオプションの ： が不要になる。

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。

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。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」である。［既定 = |cjP|］
- \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］

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。

- \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］

2.6 拡張機能

「行分割の有無により親文字とルビ文字の相対位置が変化する」ような処理は、TEX での実現は非常に難しい。これを ε-pTEX の拡張機能を用いて何とか実現したい。できたらいいな。

- 可動グループルビ機能： 例えば、
  `\ruby[g]{我思う|故に|我有り}{コギト・|エルゴ・|スム}`
  のようにグループルビで複数グループを指定すると、通常は「我思う故に我有り + コギト・エルゴ・スム」の 1 ブロックになるが、グループの区切りで行分割可能となり、例えば最初のグループの後で行分割された場合は、自動的に「我思う + コギト・」と「故に我有り + エルゴ・スム」の 2 ブロックでの組版に変化する。
- 行頭・行末での突出の自動補正： 行頭（行末）に配置されたルビ付き文字列では、自動的に前（後）突出を禁止する。
- 熟語ルビの途中での行分割の許可： 例えば、
  `\ruby[j]{熟語}{じゆく|ご}`
  の場合、結果はグループルビ処理の「熟語 + じゆくご」となるが、途中での行分割が可能で、その場合、「熟 + じゆく」「語 + ご」の 2 ブロックで出力される。

2.7 拡張機能設定の命令

- `\rubyuseextra{`⟨整数⟩`}`
  拡張機能の実装方法。[既定 = 0]
  − 0： 拡張機能を無効にする。
  − 1： まだよくわからないなにか（未実装）。
- `\rubyadjustatlineedge`／`\rubynoadjustatlineedge`
  行頭・行末での突出の自動補正を行う／行わない。[既定 = 行わない]
- `\rubybreakjukugo`／`\rubynobreakjukugo`
  モノルビ処理にならない熟語ルビで中間の行分割を許す／許さない。[既定 = 許さない]

3 実装

3.1 前提パッケージ

keyval を使う予定（まだ使っていない）。

```
1 \RequirePackage{keyval}
```

3.2 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@bad@athead 欧文ルビ、あるいは両側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@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 must have a single group%
  }\pxrr@eh@fatal
}
```

\pxrr@fatal@bad@movable 欧文ルビまたは両側ルビ(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@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
```

3.3 パラメタ

3.3.1 全般設定

\pxrr@big@intr
\pxrr@small@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）。

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

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

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

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

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

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

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

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

```
\def\pxrr@sprop@ex{1}
\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。

```
\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。

```
\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。

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

\pxrr@extra Extra 実現方法（\rubyuseextra）。整数定数。

```
\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。

```
\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。

```
\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。

```
\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（\[no]rubyadjustatlineedge）。スイッチ。

```
\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（\[no]rubyadjustatlineedge）。スイッチ。

```
\newif\ifpxrr@break@jukugo \pxrr@edge@adjustfalse
```

\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{}
```

\ifpxrr@d@athead 肩付きルビであるか否か（偽なら中付きルビ）。\rubysetup の h／c の設定。スイッチ。

```
\newif\ifpxrr@d@athead
```

\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
```

3.3.2 ルビ呼出時の設定

\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 空き補正量。\ruby の : 指定に由来する。暗黙の文字トークン（未定義値は \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@athead 肩付きルビであるか否か（偽なら中付きルビ）。\ruby の h／c の設定。スイッチ。

```
\newif\ifpxrr@athead
```

\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@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
```

3.4 補助手続

3.4.1 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@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@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@use@fontsize \pxrr@use@fontsize{⟨サイズ⟩}： フォントサイズを変更する。この命令の中ではマクロ展開のトレースを無効にする。

```
\def\pxrr@use@fontsize#1{%
  \chardef\pxrr@tracingmacros=\tracingmacros
  \tracingmacros\z@
  \fontsize{#1}{\z@}\selectfont
  \tracingmacros\pxrr@tracingmacros
}
```

3.4.2 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@gtmpa{\the\@tempdimb}%
  \endgroup
  #1=\pxrr@gtmpa\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
}
```

3.4.3 リスト分解

\pxrr@decompose

`\pxrr@decompose{`⟨要素 1⟩`}`$\cdots$`{`⟨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⟩`}`$\cdots$
`\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@tempa\pxrr@decompose@loopb
}
\def\pxrr@decompose@loopb{%
  \pxrr@ifx{\pxrr@tempa\pxrr@end}{%
    \pxrr@appto\pxrr@res{\pxrr@post}%
  }{%
    \pxrr@setok{\pxrr@ifx{\pxrr@tempa\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@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⟩}···\pxrr@inter{⟨Xn⟩}\pxrr@post

の時に、\pxrr@res を以下の内容に定義する。

⟨X1⟩⟨X2⟩···⟨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@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}%
}
```

3.5 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@
    \pxrr@csletcs{ifpxrr@d@athead}{ifpxrr@athead}%
    \let\pxrr@d@mode\pxrr@mode
    \let\pxrr@d@side\pxrr@side
  \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
}
```

\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
}
\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
}
```

\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
}
```

3.6 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、\ifpxrr@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
  \pxrr@csletcs{ifpxrr@athead}{ifpxrr@d@athead}%
  \let\pxrr@mode\pxrr@d@mode
  \let\pxrr@side\pxrr@d@side
```

次の 2 つの既定値は常に \relax（無効）である。

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

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
  \pxrr@letcs\pxrr@po@FS
   {pxrr@po@TR@\pxrr@po@FS @\@nameuse{pxrr@po@C@#1}}%
\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{%
```

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
```

既定値設定（\rubysetup）でない場合は整合性検査を行う。

```
  \ifpxrr@in@setup\else
    \pxrr@check@option
  \fi
```

\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}%
}
```

\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@<}{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@h{M}
\def\pxrr@po@C@c{M}
\def\pxrr@po@C@m{M}
\def\pxrr@po@C@g{M}
\def\pxrr@po@C@j{M}
\def\pxrr@po@C@P{M}
\def\pxrr@po@C@S{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@mi{%
  \def\pxrr@aintr@{}\pxrr@aprotrtrue
}
\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@ai{%
  \let\pxrr@ascomp=:\relax
}
\@namedef{pxrr@po@PR@*}{%
  \csname pxrr@po@PRstar@\pxrr@po@FS\endcsname
}
\def\pxrr@po@PRstar@bi{%
  \let\pxrr@bscomp=*\relax
}
\let\pxrr@po@PRstar@bb\pxrr@po@PRstar@bi
```

```
\let\pxrr@po@PRstar@bs\pxrr@po@PRstar@bi
\def\pxrr@po@PRstar@ai{%
  \let\pxrr@ascomp=*\relax
}
\@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@h{%
  \pxrr@atheadtrue
}
\def\pxrr@po@PR@c{%
  \pxrr@atheadfalse
}
\def\pxrr@po@PR@m{%
  \let\pxrr@mode=m%
}
\def\pxrr@po@PR@g{%
  \let\pxrr@mode=g%
}
\def\pxrr@po@PR@j{%
  \let\pxrr@mode=j%
}
\def\pxrr@po@PR@P{%
  \chardef\pxrr@side\z@
}
\def\pxrr@po@PR@S{%
  \chardef\pxrr@side\@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@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@ai@V{ab}
\def\pxrr@po@TR@ab@V{fi}
\def\pxrr@po@TR@bi@S{mi}
\def\pxrr@po@TR@bb@S{mi}
\def\pxrr@po@TR@bs@S{mi}
\def\pxrr@po@TR@mi@S{ai}
\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@ai@A{fi}
```

3.7 オプション整合性検査

\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
```

モノルビ（m）・熟語ルビ（j）に関する検査。

```
  \if g\pxrr@mode\else
```

欧文ルビでは不可なのでグループルビに変更する。

```
    \ifpxrr@abody
      \let\pxrr@mode=g\relax
    \fi
```

両側ルビでは不可なのでグループルビに変更する。

```
    \ifnum\pxrr@side=\tw@
      \let\pxrr@mode=g\relax
    \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。

```
    \if g\pxrr@mode
      \if g\pxrr@d@mode
        \pxrr@warn@must@group
      \fi
    \fi
  \fi
```

肩付き指定（h）に関する検査。

```
  \ifpxrr@athead
```

横組みでは不可なので中付きに変更する。

```
    \ifydir
      \pxrr@atheadfalse
    \fi
```

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
      \pxrr@atheadfalse
    \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。

```
    \ifpxrr@athead\else
      \ifpxrr@d@athead\else
        \pxrr@warn@bad@athead
      \fi
    \fi
  \fi
}
```

3.8 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@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@size@ratio\@tempdima
  \edef\pxrr@ruby@fsize{\the\@tempdima}%
  \@tempdima=1zw\relax
  \edef\pxrr@body@zw{\the\@tempdima}%
  \begingroup
    \pxrr@use@fontsize{\pxrr@ruby@fsize}%
    \@tempdima=1zw\relax
    \xdef\pxrr@gtempa{\the\@tempdima}%
  \endgroup
  \let\pxrr@ruby@zw\pxrr@gtempa
```

`\pxrr@htratio` の値を設定する。

```
  \iftdir
    \let\pxrr@htratio\pxrr@thtratio
  \else
    \let\pxrr@htratio\pxrr@yhtratio
  \fi
```

`\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@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@inter@gap\@tempdimb
  \edef\pxrr@ruby@lower{\the\@tempdima}%
}
```

3.9 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@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\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=\hb@xt@#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%
}
```

\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
```

```
}
```

\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
}
```

3.10　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
}
```

3.11　ブロック毎の処理

\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@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⟩}{r ⟨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{%
  \setbox\pxrr@boxa\hbox{#2}%
  \setbox\pxrr@boxr\hbox{%
    \pxrr@use@fontsize{\pxrr@ruby@fsize}%
    #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。

```
    \pxrr@protrtrue
    \pxrr@decompose{#2}%
    \edef\pxrr@natwd{\the\wd\pxrr@boxa}%
    \pxrr@evenspace@int{#1}\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
    \ifpxrr@athead\else
      \pxrr@decompose{#3}%
      \edef\pxrr@natwd{\the\wd\pxrr@boxr}%
      \pxrr@evenspace@int{#1}\pxrr@boxr
       {\pxrr@use@fontsize{\pxrr@ruby@fsize}}{\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
  }%
  \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#1#2#3#4{%
  \setbox\pxrr@boxa\hbox{#2}%
  \setbox\pxrr@boxr\hbox{%
    \pxrr@use@fontsize{\pxrr@ruby@fsize}%
    #3%
  }%
  \setbox\pxrr@boxb\hbox{%
    \pxrr@use@fontsize{\pxrr@ruby@fsize}%
    #4%
  }%
```

3 つのボックスの最大の幅を求める。これが全体の幅となる。

```
  \@tempdima\wd\pxrr@boxa
  \ifdim\@tempdima<\wd\pxrr@boxr
    \@tempdima\wd\pxrr@boxr
  \fi
  \ifdim\@tempdima<\wd\pxrr@boxb
    \@tempdima\wd\pxrr@boxb
  \fi
  \edef\pxrr@maxwd{\the\@tempdima}%
  \advance\@tempdima-\pxrr@epsilon\relax
  \edef\pxrr@maxwdx{\the\@tempdima}%
```

全体の幅より短いボックスを均等割りで組み直す。

```
  \ifdim\pxrr@maxwdx>\wd\pxrr@boxa
    \pxrr@decompose{#2}%
    \edef\pxrr@natwd{\the\wd\pxrr@boxa}%
    \pxrr@evenspace@int{#1}\pxrr@boxa\relax{\pxrr@maxwd}%
  \fi
  \ifdim\pxrr@maxwdx>\wd\pxrr@boxr
    \pxrr@decompose{#3}%
```

```
    \edef\pxrr@natwd{\the\wd\pxrr@boxr}%
    \pxrr@evenspace@int{#1}\pxrr@boxr
     {\pxrr@use@fontsize{\pxrr@ruby@fsize}}{\pxrr@maxwd}%
    \pxrr@adjust@margin
  \fi
  \ifdim\pxrr@maxwdx>\wd\pxrr@boxb
    \pxrr@decompose{#4}%
    \edef\pxrr@natwd{\the\wd\pxrr@boxb}%
    \pxrr@evenspace@int{#1}\pxrr@boxb
     {\pxrr@use@fontsize{\pxrr@ruby@fsize}}{\pxrr@maxwd}%
    \pxrr@adjust@margin
  \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
  }%
  \ht\z@\z@ \dp\z@\z@
  \@tempdima\wd\z@
  \setbox\pxrr@boxr\hbox{%
    \box\z@
    \kern-\@tempdima
    \box\pxrr@boxa
  }%
}
```

3.12 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
}
```

3.13 メインです

3.13.1 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@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@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}%
```

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}%
\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@assign@fsize
```

オプションを解析する。

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

両側ルビの場合、入力文字列をグループ分解せずに、そのままの引数列の形でマクロに記憶する。

```
  \def\pxrr@all@input{{#1}{#2}{#3}}%
\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{%
    \pxrr@ruby@check@tg
    \pxrr@if@alive{\pxrr@ruby@main@tg}%
  }%
```

後処理を行う。

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

3.13.2 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、\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@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 \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。

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

```
    \expandafter\pxrr@check@kinsoku@b
  \else
    \pxrr@cntr\z@
    \expandafter\pxrr@tempb
  \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@cntr\prebreakpenalty`#1\relax
  \pxrr@tempb
}
```

\pxrr@check@char　\pxrr@check@char\CS： トークン \CS が「通常文字」であるかを \if@tempswa に返す。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
      \@tempswafalse
    \else\ifcat\noexpand##1\noexpand~%
      \@tempswafalse
    \else\ifcat\noexpand##1A%
      \@tempswatrue
    \else\ifcat\noexpand##10%
      \@tempswatrue
    \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。）

```
      \@tempswafalse
      \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@%
      \@tempswatrue
    \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
```

3.13.3 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　両側ルビの場合、ここで検査する内容はない。（両側ルビの入力文字列はグループ分割されず、常に単一グループとして扱われる。）

```
\def\pxrr@ruby@check@tg{%
}
```

3.13.4 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。

```
\def\pxrr@inter@mono{%
  \hskip\kanjiskip
}
```

\pxrr@intrude@head　先頭での進入処理。

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

ゴースト処理が行われている場合は、こちらの処理は行わない。（だから進入が扱えない。）

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

段落冒頭では処理なし。

```
    \ifpxrr@par@head\else
```

前空き補正の処理。

```
      \if *\pxrr@bscomp
        \penalty\@M
      \else\if :\pxrr@bscomp
        \hskip\xkanjiskip
      \else
        \hskip\kanjiskip
      \fi\fi
```

実際の進入の量を求め、その量の負のグルーを入れる。

```
      \pxrr@dima\pxrr@bspace\relax
      \ifdim\pxrr@bintr<\pxrr@dima
        \pxrr@dima\pxrr@bintr\relax
      \fi
      \hskip-\pxrr@dima
    \fi
  \fi
}
```

`\pxrr@intrude@end` 末尾での進入処理。

```
\def\pxrr@intrude@end{%
  \ifpxrr@ghost\else
    \pxrr@dima\pxrr@aspace\relax
    \ifdim\pxrr@aintr<\pxrr@dima
      \pxrr@dima\pxrr@aintr\relax
    \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。

```
    \if *\pxrr@bscomp
      \penalty\@M
      \hskip-\pxrr@dima
    \else\if :\pxrr@bscomp
      \penalty\pxrr@end@kinsoku
      \hskip-\pxrr@dima
      \hskip\xkanjiskip
    \else
      \penalty\pxrr@end@kinsoku
      \hskip-\pxrr@dima
      \hskip\kanjiskip
    \fi\fi
```

本物の前禁則ペナルティ（負かも知れない）はここに加算される。ここで行分割してはいけないので大きな値にする。

```
    \penalty\@MM
  \fi
}
```

`\pxrr@takeout@any@protr` `\ifpxrr@any@protr` の値をグループの外に出す。

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

`\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@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\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\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
\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@tg　両側ルビ（必ず単純グループルビである）の場合。

```
\def\pxrr@ruby@main@tg{%
  \pxrr@check@par@head
  \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
}

\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@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@bscomp = \meaning\pxrr@bscomp^^J%
    pxrr@ascomp = \meaning\pxrr@ascomp^^J%
    pxrr@bintr = \pxrr@bintr^^J%
    pxrr@aintr = \pxrr@aintr^^J%
    ifpxrr@athead = \meaning\ifpxrr@athead^^J%
    pxrr@mode = \meaning\pxrr@mode^^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%
    ----
  }%
}
\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%
    ----
  }%
}
```

3.13.5 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@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。

```
\chardef\pxrr@zspace=\jis"2121\relax
```

\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@zspace
  \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@zspace}%
      \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。）

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

`\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}%
}
```

3.13.6 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@zspace}%
      \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@zspace
  \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
}
```