

pT_EX マニュアル

日本語 T_EX 開発コミュニティ*

version p3.8.2, 2019 年 2 月 10 日

本ドキュメントはまだまだ未完成です.

目次

第 I 部	pT _E X の日本語組版と追加プリミティブ	2
1	pT _E X で利用可能な文字	2
2	文字コードの取得と指定	3
3	和文文字と <code>\kcatcode</code>	3
4	禁則	5
5	文字間のスペース	6
6	組方向	9
7	ベースライン補正	12
8	和文フォント	13
9	文字コード変換, 漢数字	14
10	長さ単位	15
11	バージョン番号	16
第 II 部	オリジナルの T _E X 互換プリミティブの動作	17
12	和文に未対応のプリミティブ	17
13	和文に対応したプリミティブ	17

* <https://texjp.org>, e-mail: [issue\(at\)texjp.org](mailto:issue(at)texjp.org)

第 I 部

pTeX の日本語組版と追加プリミティブ

ここでは、pTeX の日本語組版の概略と、それを実現するために導入されたプリミティブを説明する。

1 pTeX で利用可能な文字

TeX82 で扱える文字コードの範囲は 0–255 であった。pTeX ではこれに加えて JIS X 0208 の文字も利用可能であり、以上の文字集合が欧文文字と和文文字に区別されて扱われる。

[TODO] 欧文文字とは何か、和文文字とは何か

pTeX は、入力を EUC-JP または Shift-JIS のいずれか（後述）の内部コードに変換して取り扱う。和文文字の内部コードとして許される整数は $256c_1 + c_2$ 、但し $c_i \in C_i$ であり、ここで

内部コードが EUC-JP のとき $C_1 = C_2 = \{\text{"a1"}, \dots, \text{"fe"}\}$ 。

内部コードが Shift-JIS のとき $C_1 = \{\text{"81"}, \dots, \text{"9f"}\} \cup \{\text{"e0"}, \dots, \text{"fc"}\}$ 、

$C_2 = \{\text{"40"}, \dots, \text{"7e"}\} \cup \{\text{"80"}, \dots, \text{"fc"}\}$ 。

である。

入力ファイルの文字コードが UTF-8 の場合、JIS X 0208 にない文字は UTF-8 のバイト列として読み込まれる。

pTeX の入力ファイルの文字コードは、起動時のオプション `-kanji` によって変更できる（可能な値は `utf8`, `euc`, `sjis`, `jis`）。同様に、内部コードは `-kanji-internal` オプションによって変更できる（可能な値は `euc`, `sjis`）が、こちらは `ini mode` でのフォーマット作成時に限られ、`virtual mode` では不可である（フォーマットの内部コードと実際の処理の整合性をとるため、pTeX p3.8.2 以降の仕様）。

pTeX の入力ファイルの文字コードと内部コードは起動時のバナーから分かる。例えば

```
This is pTeX, Version 3.14159265-p3.8.0 (utf8.euc) (TeX Live 2018)
(preloaded format=ptex)
```

というバナーの場合は、`(utf8.euc)` から

- 入力ファイルの（既定）文字コードは UTF-8（但し、JIS X 0208 の範囲内）
- pTeX の内部コードは EUC-JP

という情報が見て取れる。入力ファイルの文字コードと内部コードが同じ場合は、

```
This is pTeX, Version 3.14159265-p3.8.0 (sjis) (TeX Live 2018)
(preloaded format=ptex)
```

のように表示される（この例は、入力ファイルの文字コードと内部コードがともに Shift-JIS の場合）。



上にはこのように書いたが、極めて細かい話をすれば、起動時のバナーは時にウソをつくので注意。ログファイルには記録されるバナーは常に正しい。これは以下の事情による。

`virtual mode` では、起動直後にバナーを表示してから、フォーマットファイル読み込みが行われる。この時点で初めて、起動時の内部コードとフォーマットの内部コードの整合性が確認される。ここでもし合致しなかった場合は、`pTeX` は警告を表示してフォーマットに合った内部コードを選択し、以降の処理を行う。ログファイルはこの後にオープンされるため、そこには正しい内部コード（フォーマットと同じ内部コード）が書き込まれる [6]。

このようなウソは、`pTeX` に限らず (`preloaded_format=***`) でも見られる。

[TODO] `pTeX` の内部コードは EUC-JP か Shift-JIS のいずれかであるが、DVI ファイル内の和文文字コードとしては JIS コードを出力する。

2 文字コードの取得と指定

`pTeX` でも `TeX82` と同様に、バッククォート (```) を使って「``あ`」のようにして和文文字の内部コードを内部整数として得ることができる。欧文文字については、1 文字の制御綴を代わりに指定することができた（例えば、「``b`」と「``\b`」は同じ意味だった）が、同じことを和文文字に対して「``\あ`」などを行うことはできない。

`pTeX` では「文字コードを引数にとるプリミティブ」といっても、状況によって

- 欧文文字の文字コード 0-255 をとる（例：`\catcode`）
- 和文文字の内部コードをとる（例：`\inhibitxspcode`）
- 上記 2 つのどちらでもとれる（例：`\prebreakpenalty`）

のいずれの場合もありうる。

本ドキュメントでは上のどれかを明示するために、以下のような記法を採用する。

`<8-bit number>` 0-255 の範囲内の整数

`<kanji code>` 和文文字の内部コード

`<character code>` 0-255 の範囲内の整数、および和文文字の内部コード

`<16-bit number>` 0-65535 の範囲内の整数

3 和文文字と `\kcatcode`

`TeX82` では、各文字に 0-15 のカテゴリーコードを割り当てており、`TeX82` の入力プロセッサは「どのカテゴリーコードの文字が来たか」で状態が遷移する有限オートマトンとして記述できる ([1])。 `pTeX` では、和文文字には 16 (*kanji*), 17 (*kana*), 18 (*other_kchar*) のカテゴリーコードの 3 つのいずれかが割り当てられており、`pTeX` の入力プロセッサの状態遷移図は図 1 のように `TeX82` のそれを拡張したものになっている。

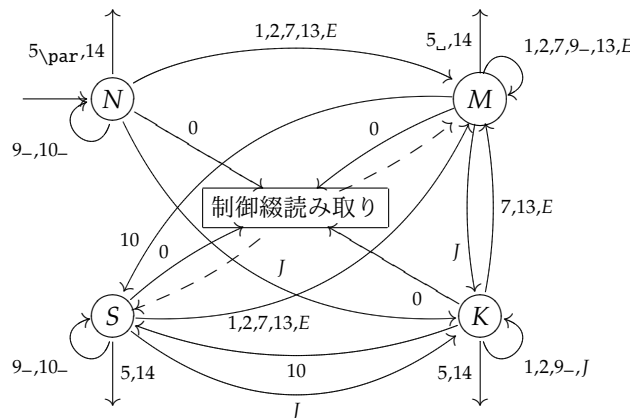
図 1 を見れば分かるように、（欧文文字直後の改行は空白文字扱いされるのと対照的に）和文文字直後の改行は何も発生しない。和文文字の直後にグループ開始・終了が来て行が終わった場合も同様である。

和文文字のカテゴリーコードの値による動作の違いは次のようになる：

- $\text{T}_{\text{E}}\text{X}82$ では、「複数文字からなる命令」（コントロールワード）にはカテゴリーコードが 11 (*letter*) の文字しか使用できないようになっていたが、 $\text{p}_{\text{T}}\text{E}\text{X}$ では 16, 17 の和文文字も合わせて使用することができる。
- 一方、カテゴリーコードが 18 の和文文字はコントロールワード中には使用できない。「\」のように一文字命令（コントロールシンボル）に使用することはできる^{*1}。
- 後で説明する `\jcharwidowpenalty` は、カテゴリーコードが 16, 17 の和文文字の前のみ挿入されうるもので、カテゴリーコードが 18 の和文文字の前には挿入されない。
- 欧文文字は $\text{T}_{\text{E}}\text{X}82$ と同様に 1 つの文字トークンはカテゴリーコード c と文字コード s の組み合わせ ($256c + s$) で表現されていた。 $\text{p}_{\text{T}}\text{E}\text{X}$ では和文文字トークンは文字コードのみで表現され、そのカテゴリーコードは随時算出されるようになっている^{*2}。

► `\kcatcode <character code>=<16-18>`

`\kcatcode` は DVI 中の上位バイト（すなわち、JIS コードでいう区ごと）に値が設定可



- 矢印の上下に書いた数字はカテゴリーコードを表す。
- “E” はカテゴリーコード 3, 4, 6, 8, 11, 12 の文字達を，“J” は和文文字を表す。
- “5\par” のような下付き添字は「挿入するトークン」を表している。但し，“9_”、“10_” はその文字を無視することを示している。
- 図中央の「制御綴読み取り」の後で S, M どちらの状態になるかは、次のように決まっている：
 - 制御綴が「_」のようなカテゴリーコード 10 の文字からなるコントロールシンボルのときは状態 S に遷移する。
 - 制御綴が「\#」「\」のようなその他のコントロールシンボルのときは状態 M に遷移する。
 - 制御綴がコントロールワードのときは状態 S に遷移する。

図 1 $\text{p}_{\text{T}}\text{E}\text{X}$ の入力プロセッサの状態遷移図

^{*1} 「\」のように和文文字からなるコントロールシンボルを文字列化する際に、バージョン $\text{p}_{\text{T}}\text{E}\text{X}$ 3.7.2 以前の $\text{p}_{\text{T}}\text{E}\text{X}$ では「\」と後ろに余計な空白文字を補ってしまうという問題があった。 $\text{T}_{\text{E}}\text{X}$ Live 2018 の $\text{p}_{\text{T}}\text{E}\text{X}$ 3.8.1 でこの問題は修正された ([5])。

また、バージョン $\text{p}_{\text{T}}\text{E}\text{X}$ 3.8.1 以前では「\」のような和文のコントロールシンボルで行が終わった場合、「\!」のような欧文コントロールシンボルと同様に改行由来の空白が追加されてしまい、和文文字直後の改行は何も発生しないという原則に反していたが、こちらは $\text{T}_{\text{E}}\text{X}$ Live 2019 の $\text{p}_{\text{T}}\text{E}\text{X}$ 3.8.2 で修正された ([5])。

^{*2} $\text{u}_{\text{T}}\text{E}\text{X}$ では和文文字トークンについてもそのカテゴリーコードの情報が含まれるようになった。

能である。初期状態では 1, 2, 7–15, 85–94 区の文字が 18, 3–6 区の文字が 17, 16–84 区の文字が 16 である。



`\kcatcode` では欧文文字の文字コード (0–255) も指定することができるが、その場合「0 区扱い」として扱われる^{*3}。

なお、`upTeX` においては、`\kcatcode` は大きく仕様変更されている。

4 禁則

欧文と和文の処理で見かけ上最も大きな違いは、行分割処理であろう。

- 欧文中での行分割は、ハイフネーション処理等の特別な場合を除いて、単語中すなわち連続する文字列中はブレイクポイントとして選択されない。
- 和文中では禁則（行頭禁則と行末禁則）の例外を除いて、全ての文字間がブレイクポイントになり得る。

しかし、`pTeX` の行分割は `TeX` 内部の処理からさほど大きな変更を加えられてはいない。というのも、`TeX` のペナルティ^{*4}という概念を和文の禁則処理にも適用しているためである。

行頭禁則と行末禁則を実現する方法として、`pTeX` では「禁則テーブル」が用意されている。このテーブルには

- 禁則文字
- その文字に対応するペナルティ値
- ペナルティの挿入位置（その文字の前に挿入するか後に挿入するかの別）

を登録でき、`pTeX` は文章を読み込むたびにその文字がテーブルに登録されているかどうかを調べ、登録されていればそのペナルティを文字の前後適切な位置に挿入する。

禁則テーブルに情報を登録する手段として、以下のプリミティブが追加されている。

▶ `\prebreakpenalty <character code>=<number>`

指定した文字の前方にペナルティを挿入する。正の値を与えると行頭禁則の指定にあたる。例えば `\prebreakpenalty`.=10000` とすれば、句点の直前に 10000 のペナルティが付けられ、行頭禁則文字の対象となる。

▶ `\postbreakpenalty <character code>=<number>`

指定した文字の後方にペナルティを挿入する。正の値を与えると行末禁則の指定にあたる。例えば `\postbreakpenalty` (=10000` とすれば、始め丸括弧の直後に 10000 のペナルティが付けられ、行末禁則文字の対象となる。

`\prebreakpenalty`, `\postbreakpenalty` は和文文字、欧文文字の区別無しに指定できる

^{*3} `pTeX` の処理でこの「0 区」の `\kcatcode` が使われることはないので、事実上は「16–18 のどれかを格納可能な追加レジスタ」程度の使い方しかない。

^{*4} ペナルティとは、行分割時やページ分割時に「その箇所がブレイクポイントとしてどの程度適切であるか」を示す一般的な評価値である（適切であれば負値、不適切であれば正值とする）。絶対値が 10000 以上のペナルティは無限大として扱われる。

が、同一の文字に対して両方のペナルティを同時に与えるような指定はできない（もし両方指定された場合、後から指定されたものに置き換えられる）。禁則テーブルには 256 文字分の領域しかないので、禁則ペナルティを指定できる文字数は最大で 256 文字までである。

禁則テーブルからの登録の削除はペナルティ値 0 をグローバルに（つまり、`\global` を用いて）設定することによって行われる^{*5}。

▶ `\jcharwidowpenalty=<number>`

パラグラフの最終行が「す。」のように孤立するのを防ぐためのペナルティを設定する。

5 文字間のスペース

欧文では単語毎にスペースが挿入され、その量を調整することにより行長が調整される。一方、和文ではそのような調整ができる箇所がほぼ存在しない代わりに、ほとんどの場合は行長を全角幅の整数倍に取ることで、（和文文字だけの行では）綺麗に行末が揃うようにして組まれる。

ただし、禁則処理が生じたり、途中で欧文が挿入されたりした場合はやはり調整が必要となる。そこで一般に行われるのが、追い込み、欧文間や和欧文間のスペース量の調整、追い出しなどの処理である。これらを自動で行うため、`pTeX` には下記の機能が備わっている。

1. 連続する和文文字間に自動的にグルー（伸縮する空白）を挿入
2. 和欧文間に（ソース中に空白文字を含めずとも）自動的にグルーを挿入
3. 和文の約物類にグルーを設定

上記のうち、1 と 2 についてはそれぞれ `\kanjiskip`, `\xkanjiskip` というパラメータを設けている。3 については、`TeX` が使うフォントメトリック (TFM) を拡張した `pTeX` 専用の形式、JFM フォーマットによって実現している。

[TODO] JFM グルーの挿入規則について

- メトリック由来空白の挿入処理は展開不能トークンが来たら中断
- 展開不能トークンや欧文文字は「文字クラス 0」扱い。「」`\relax`（」の例

▶ `\kanjiskip=<skip>`

連続する和文文字間に標準で入るグルーを設定する。段落途中でこの値を変えても影響はなく、段落終了時の値が段落全体にわたって用いられる。



和文文字を表すノードが連続した場合、その間に `\kanjiskip` があるものとして行分割やボックスの寸法計算が行われる。`\kanjiskip` の大部分はこのような暗黙のうちに挿入されるものであるので、`\lastskip` などで取得することはできないし、`\showlists` や `\showbox` でも表示されない。

その一方で、ノードの形で明示的に挿入される `\kanjiskip` も存在する。このようになるのは次の場合である：

^{*5} 0 をローカルに設定した場合でも、その設定が最も外側のグループ（`ε-TeX` 拡張でいう `\currentgrouplevel` が 0 のとき）には削除される。「ローカルで削除された場所に別の禁則ペナルティ設定がグローバルで行われる」ことがおこらないようにするため、最外グループ以外での 0 設定ではテーブルから削除されない。

- 水平ボックス (`\hbox`) が和文文字で開始しており、そのボックスの直前が和文文字であった場合、ボックスの直前に `\kanjiskip` が挿入される。
- 水平ボックス (`\hbox`) が和文文字で終了しており、そのボックスの直後が和文文字であった場合、ボックスの直後に `\kanjiskip` が挿入される。
- 連続した和文文字の間にペナルティがあった場合、暗黙の `\kanjiskip` が挿入されないので明示的にノードが作られる。

なお、水平ボックスであっても `\raise`、`\lower` で上下位置をシフトさせた場合は上記で述べた `\kanjiskip` を前後に挿入処理の対象にはならない。



しばしば

```
\hbox to 15zw{%
  \kanjiskip=0pt plus 1fil
  あ「い」うえ、お}
```

のように `\kanjiskip` に無限の伸長度を持たせることで均等割付を行おうとするコードを見かけますが、連続する和文文字の間にはメトリック由来の空白と `\kanjiskip` は同時には入らないので、上に書いたコードは不適切である^{*6}。

▶ `\xkanjiskip=<skip>`

和文文字と欧文文字の間に標準で入るグルーを設定する。段落途中でこの値を変えても影響はなく、段落終了時の値が段落全体にわたって用いられる。



`\kanjiskip` と異なり、`\xkanjiskip` はノードの形で挿入される。この挿入処理は段落の行分割処理の直前や、`\hbox` を閉じるときに行われるので、「どこに `\xkanjiskip` が入っているか」を知るためには現在の段落や `\hbox` を終了させる必要がある。

▶ `\xspcode <8-bit number>=<0-3>`

コード番号が `<8-bit number>` の欧文文字の周囲に `\xkanjiskip` が挿入可能が否かを 0-3 の値で指定する。それぞれの意味は次の通り：

- 0 欧文文字の前側、後側ともに `\xkanjiskip` の挿入を禁止する。
- 1 欧文文字の前側にのみ `\xkanjiskip` の挿入を許可する。後側は禁止。
- 2 欧文文字の後側にのみ `\xkanjiskip` の挿入を許可する。前側は禁止。
- 3 欧文文字の前側、後側ともに `\xkanjiskip` の挿入を許可する。

pTeX の標準値は、数字 0-9 と英文字 A-Z, a-z に対する値は 3 (両側許可)、その他の文字に対しては 0 (両側禁止)。

▶ `\inhibitxspcode <kanji code>=<0-3>`

コード番号が `<kanji code>` の和文文字の周囲に `\xkanjiskip` が挿入可能が否かを 0-3 の値で指定する。それぞれの意味は次の通り：

- 0 和文文字の前側、後側ともに `\xkanjiskip` の挿入を禁止する。
- 1 和文文字の後側にのみ `\xkanjiskip` の挿入を許可する。前側は禁止。
- 2 和文文字の前側にのみ `\xkanjiskip` の挿入を許可する。後側は禁止。
- 3 和文文字の前側、後側ともに `\xkanjiskip` の挿入を許可する。

この `\inhibitxspcode` の設定値の情報は 256 文字分のテーブルに格納されている、未登

^{*6} 実際、開き括弧の前・閉じ括弧（全角コンマを含む）の後には JFM グルーが入っているので半角しかない。

録時は 3 (両側許可) であるとみなされ、またグローバルに 3 を代入するか、あるいは最も外側のグループで 3 を代入するとテーブルからの削除が行われる。



`\xspmode` と `\inhibitxspmode` では、一見すると設定値 1 と 2 の意味が反対のように感じるかもしれない。しかし、実は両者とも

1: 「和文文字→欧文文字」の場合のみ許可。「欧文文字→和文文字」の場合は禁止。

2: 「欧文文字→和文文字」の場合のみ許可。「和文文字→欧文文字」の場合は禁止。

となっている。

▶ `\autospacing`, `\noautospacing`

連続する和文文字間に、標準で `\kanjiskip` で指定されただけのグルーを挿入する (`\autospacing`) か挿入しない (`\noautospacing`) を設定する。段落途中でこの値を変えても影響はなく、段落終了時の値が段落全体にわたって用いられる。

▶ `\autoxspacing`, `\noautoxspacing`

和文文字と欧文文字の間に、標準で `\xkanjiskip` で指定されただけのグルーを挿入する (`\autoxspacing`) か挿入しない (`\noautoxspacing`) を設定する。段落途中でこの値を変えても影響はなく、段落終了時の値が段落全体にわたって用いられる。

どちらの設定も標準では有効 (`\autospacing`, `\autoxspacing`) である。



すでに述べたように、`\kanjiskip` の一部と `\xkanjiskip` はノードの形で挿入される。`\noautospacing` や `\noautoxspacing` を指定しても、このノードの形で挿入自体は行われる (ただノードが `\kanjiskip` や `\xkanjiskip` の代わりに長さ 0 のグルーを表すだけ)。

これにより、例えば `\noautoxspacing` 状況下で「あa」と入力しても、間に長さ 0 のグルーがあるため「あ」と「a」の間で改行可能となることに注意。

▶ `\showmode`

`\kanjiskip` の挿入や `\xkanjiskip` の挿入が有効になっているか否かを

> auto spacing mode;

> no auto xspacing mode.

という形式 (上の例では `\autospacing` かつ `\noautoxspacing` の状況) で端末やログに表示する。

▶ `\inhibitglue`

この命令が実行された位置において、メトリック由来の空白の挿入を禁止する。以下の点に注意。

- メトリック由来の空白が挿入されないだけであり、その代わりに `\kanjiskip` や `\xkanjiskip` が挿入されることは禁止していない。
- 本命令は現在のモードが (非限定, 限定問わず) 水平モードのときしか効力を発揮しない (数式モードでも効かない)。段落が和文文字「【」で始まり、その文字の直前にメトリック由来の空白が入ることを抑止したい場合は、次のように一旦段落を開始してから `\inhibitglue` を実行する必要がある。

`\leavevmode\inhibitglue` 【

以前の pTeX では「この命令が実行された位置」が何を指すのか大雑把でわかりにくかったが、TeX Live 2019 の pTeX p3.8.2 以降では、明確に新たなノードが追加されない限り、と定めた ([3, 7]). すなわち、

1. `\inhibitglue` は、ノード挿入処理を行う命令 (`\null`, `\hskip`, `\kern`, `\vrule`, ...) が後ろに来た場合は無効化される。
2. 一方、`\relax` やレジスタへの代入などのノードを作らない処理では無効化されない。
3. `\inhibitglue` の効果は別レベルのリストには波及しない。



以上の説明の具体例を以下に示す：

```

) \vrule (\
) \vrule\inhibitglue (\
) \inhibitglue\vrule (\
) \inhibitglue\relax (\
) \relax\inhibitglue (\ % 「」」「\relax」間で二分空きが入る
あ\setbox0=\hbox{\inhibitglue} (

```



pLaTeX 2017-10-28 以降では、`\inhibitglue` の短縮として `\<` が次のように定義されている (`\protected` は ϵ -TeX 拡張の機能だが、現在では LaTeX 自体が ϵ -TeX 拡張を要求している)。

```
\protected\def\<{\ifvmode\leavevmode\fi\inhibitglue}
```

▶ `\disinhibitglue`

`\inhibitglue` の効果は無効化 (つまり、メトリック由来の空白の挿入を許可) する。pTeX p3.8.2 で新しく追加された。

6 組方向



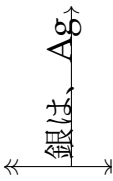

従来の TeX では、字送り方向が水平右向き (\rightarrow)、行送り方向が垂直下向き (\downarrow) に固定されていた。pTeX では、TeX の状態として“組方向 (ディレクション)”を考え、ディレクションによって字送り方向と行送り方向を変えることにしてある。なお、行は水平ボックス (horizontal box)、ページは垂直ボックス (vertical box) であるという点は、pTeX でも従来の TeX と同様である。

pTeX のサポートする組方向は横組、縦組、そして **DtoU** 方向^{*7}の 3 つである。また数式モード中で作られたボックスは数式ディレクションというまた別の状態になる。横組での数式ディレクション (横数式ディレクション) と DtoU 方向での数式ディレクションはそれぞれ非数式の場合と区別はないが、縦組での数式ディレクション (縦数式ディレクション) では横組を時計回りに 90 度回転させたような状態となる。従って、pTeX では縦数式ディレクションまで含めると合計 4 種類の組方向がサポートされているといえる (表 1)。

以下が、組方向の変更や現在の組方向判定に関わるプリミティブの一覧である。

^{*7} 下から上 (Down to Up) であろう。

表 1 pTeX のサポートする組方向

	横組	縦組	DtoU 方向	縦数式ディレクション
命令	<code>\yoko</code>	<code>\tate</code>	<code>\dtou</code>	—
字送り方向	水平右向き (→)	垂直下向き (↓)	垂直上向き (↑)	垂直下向き (↓)
行送り方向	垂直下向き (↓)	水平左向き (←)	水平右向き (→)	水平左向き (←)
使用する和文 フォント	横組用 (<code>\jfont</code>)	縦組用 (<code>\tfont</code>)	横組用 (<code>\jfont</code>) の 90° 回転	
組版例				

▶ `\tate`, `\yoko`, `\dtou`

組方向をそれぞれ縦組，横組，DtoU 方向に変更する．カレントの和文フォントは縦組では縦組用フォント，横組および DtoU 方向では横組用フォントになる．

組方向の変更は，原則として作成中のリストやボックスに何のノードも作られていない状態でのみ許される．より詳細には，

- 制限水平モード (`\hbox`)，内部垂直モード (`\vbox`) では，上記に述べた原則通り．

```
\hbox{\hsize=20em\tate ……}
```

のように，ノードが作られない代入文などは組方向変更前に実行しても良い．

違反すると次のようなエラーが出る．

! Use `\tate` at top of list.

- 非制限水平モード（行分割される段落）や，数式モード（文中数式，ディスプレイ数式問わず）での実行は禁止．
- 外部垂直モードの場合は，次の 2 点が同時に満たされる場合のみ実行可能である^{*8}.
 - `current page` の中身にはボックス，罫線 (`\hrule`)，insertion (`\insert`) はない．言い換えれば，`current page` の中身はあってもマーク (`\mark`) か `whatsit` のみ^{*9}.
 - `recent contributions` の中身にもボックス，罫線，insertion はない．

違反すると次のようなエラーが出る．

! Use `\tate` at top of the page.

^{*8} TeX は，外部垂直モードでボックスその他のノードを追加する際に，まずそのノードを `recent contributions` というリストの末尾に追加し，その後 `recent contributions` の中身が徐々に `current page` という別のリストに移されるという処理を行っている．

そのため，単純に `current page` または `recent contributions` という 1 つのリストを調べるだけでは「ページが空」か正しく判断できない．

^{*9} `current page` の中身にグルー，カーン，ペナルティがあるのは，それらの前にボックス，罫線，insertion が存在する場合にのみである．

また、ボックスの中身を `\unhbox`, `\unvbox` 等で取り出す場合は、同じ組方向のボックス内でなければならない^{*10}。違反した場合、

`! Incompatible direction list can't be unboxed.`

なるエラーが出る。



`\discretionary` 命令では `\discretionary{<pre>}{<post>}{<nobreak>}` と 3 つの引数を指定するが、この 3 引数の中で組方向を変更することはできない（常に周囲の組方向が使われる）。

► `\iftdir`, `\ifydir`, `\ifddir`, `\ifmdir`

現在の組方向を判定する。 `\iftdir`, `\ifydir`, `\ifddir` はそれぞれ縦組, 横組, DtoU 方向であるかどうかを判定する（数式ディレクションであるかは問わない）。一方, `\ifmdir` は数式ディレクションであるかどうかを判定する。

従って、表 1 に示した 4 つの状況のどれに属するかは以下のようにして判定できることになる。

```
\iftdir
  \ifmdir
    (縦数式ディレクション)
  \else
    (通常の縦組)
  \fi
\else\ifydir
  (横組)
\else
  (DtoU方向)
\fi
```

► `\iftbox <number>`, `\ifybox <number>`, `\ifdbbox <number>`, `\ifmbox <number>`

`<number>` 番のボックスの組方向を判定する。 `<number>` は有効な `box` レジスタでなければならない。

バージョン p3.7 までの `pTeX` では、ボックスが一旦ノードとして組まれてしまうと、通常の縦組で組まれているのか、それとも縦数式ディレクションで組まれているのかという情報が失われていた。しかし、それでは後述の「ベースライン補正の戻し量」を誤り、欧文の垂直位置が揃わないという問題が生じた ([2])。この問題を解決する副産物として、バージョン p3.7.1 で `\ifmbox` プリミティブが実装された。

^{*10} 数式ディレクションか否かは異なっても良い。

7 ベースライン補正

和文文字のベースラインと欧文文字のベースラインが一致した状態で組むと、行がずれて見えてしまう場合がある。特に縦組の状況が典型的である（表 1 の「組版例」参照）。

この状況を解決するため、pTeX では欧文文字のベースラインを行送り方向に移動させることができる：

▶ `\tbaselineshift=<dimen>`, `\ybaselineshift=<dimen>`

指定した箇所以降の欧文文字のベースラインシフト量を格納する。両者の使い分けは
`\tbaselineshift` 縦組用和文フォントが使われるとき（つまり組方向が縦組のとき）、
`\ybaselineshift` 横組用和文フォントが使われるとき（横組，DtoU 方向，縦数式
ディレクション）

となっている。どちらの命令においても、正の値を指定すると行送り方向（横組ならば下，縦組ならば左）にずらすことになる。



`disp_node`

欧文文字だけでなく、文中数式 (\dots) もベースライン補正の対象である。文中数式は全体に `\tbaselineshift`（もしくは `\ybaselineshift`）だけのベースライン補正がかかるが、それだけでは

数式中のボックスの欧文は（文中数式全体にかかる分も合わせて）二重にベースライン補正がされる

という問題が起きてしまう。この問題を解決するための命令が以下の 3 つの命令であり、pTeX p3.7^{*11}で追加された。

▶ `\textbaselineshiftfactor=<number>`, `\scriptbaselineshiftfactor=<number>`

▶ `\scriptscriptbaselineshiftfactor=<number>`

文中数式全体にかかるベースライン補正量に対し、文中数式内の明示的なボックスを逆方向に移動させる割合を指定する。1000 が 1 倍（ベースライン補正をちょうど打ち消す）に相当する。

プリミティブが 3 つあるのは、数式のスタイルが `\textstyle` 以上（`\displaystyle` 含む）、`\scriptstyle`, `\scriptscriptstyle` のときにそれぞれ適用される値を変えられるようにするためである。既定値はそれぞれ 1000（1 倍）、700（0.7 倍）、500（0.5 倍）である。

TeX Live 2015 以前の動作に戻すには、上記の 3 プリミティブに全て 0 を指定すれば良い。



`\scriptbaselineshiftfactor` を設定するときには、`\scriptstyle` 下で追加するボックス内のベースライン補正量をどうするかを常に気にしないといけない。例えば次のコードを考える：

^{*11} TeX Live 2016, 厳密には 2016-03-05 のコミット (r39938).

```
\ybaselineshift=10pt\scriptbaselineshiftfactor=700
漢字pqr$a\hbox{xあ}^{\%
b\hbox{\scriptsize yう}
\hbox{\scriptsize\ybaselineshift=7pt zえ}
}$か%$
```

組版結果は以下のようになる：

漢字 あ^{うえ}_y z か
pqrax

この例で、ボックス `\hbox{\scriptsize yう}` 内ではベースライン補正量は 10pt のままである^{*12}。それが添字内に配置された場合、このボックスは

$$(\text{文中数式全体のシフト量}) \times \frac{\text{\scriptbaselineshiftfactor}}{1000} = 7\text{pt}$$

だけ上に移動するので、結果として「y」は添字内に直書きした「b」と上下位置が $10\text{pt} - 7\text{pt} = 3\text{pt}$ だけ上に配置されてしまっている。

なお、`\scriptscriptbaselineshiftfactor` についても全く同様の注意が当てはまる。

8 和文フォント

► `\jfam=<number>`

現在の和文数式フォントファミリの番号を格納する^{*13}。現在の欧文数式ファミリの番号を格納する `\fam` と原理的に同じ番号を指定することは原理的には可能だが、数式ファミリは和文・欧文共用であるので実際には異なる値を指定することになる。

欧文フォントが設定されている数式ファミリの番号を `\jfam` に指定し、数式中で和文文字を記述すると

! Not two-byte family.

というエラーが発生する。逆に、和文フォントが設定されているファミリの番号を `\fam` に指定し、数式中に欧文文字を記述すると

! Not one-byte family.

というエラーが発生する。

► `\jfont, \tfont`

欧文フォントを定義したり、現在の欧文フォントを取得する `\font` の和文版である。一応 `\jfont` が「和文の横組用フォント」の、`\tfont` が「和文の縦組用フォント」のために用いる命令である。

- フォントを定義する際は、欧文フォント・和文の横組用フォント・和文の縦組用のフォントのいずれも `\font, \jfont, \tfont` のどれを用いても定義できる。

^{*12} `\scriptsize` などのフォントサイズ変更命令では、標準では `\ybaselineshift` の値は変更しない (`\tbaselineshift` の値はその都度変更する)。

^{*13} `pTeX, upTeX` では 0-15 の範囲が許される。`ε-pTeX, ε-upTeX` では欧文の `\fam` と共に 0-255 に範囲が拡張されている。

- `\the` 等で「現在のフォント」を取得する際には、`\jfont` で「和文の横組用フォント」を、`\tfont` で「和文の縦組用フォント」を返す。

9 文字コード変換，漢数字

pTeX の内部コードは環境によって異なる。そこで，異なる文字コード間で同じ文字を表現するため，文字コードから内部コードへの変換プリミティブが用意されている。また，漢数字を出力するプリミティブも用意されている^{*14}。

▶ `\kuten` *<16-bit number>*

区点コードから内部コードへの変換を行う。16 進 4 桁の上 2 桁が区，下 2 桁が点であると解釈する。たとえば，`\char\kuten"253C` は，「怒」（37 区 60 点）である。

▶ `\jis` *<16-bit number>*，`\euc` *<16-bit number>*，`\sjis` *<16-bit number>*

それぞれ JIS コード，EUC コード，Shift_JIS コードから内部コードへの変換を行う。たとえば，`\char\jis"346E`，`\char\euc"B0A5`，`\char\sjis"8A79` は，それぞれ「喜」，「哀」，「楽」である。

▶ `\kansuji` *<number>*，`\kansujichar` *<0-9>=<kanji code>*

`\kansuji` は，続く数値 *<number>* を漢数字の文字列で出力する。出力される文字は `\kansujichar` で指定できる（デフォルトは「〇一三四五六七八九」）。たとえば

```
\kansuji 1978年
```

は「一九七八年」と出力され，

```
\kansujichar1=`壹
\kansujichar2=\euc"C6F5\relax
\kansujichar3=\jis"3B32\relax
\kansuji 1234
```

は「壹貳參四」と出力される。

`\kansujichar` で指定できるのは「和文文字の内部コードとして有効な値」であり，無効な値（例えば pTeX では，``A` は欧文文字コードであり，和文文字コードではない）を指定すると

```
! Invalid KANSUJI char ("41)
```

というエラーが発生する^{*15}。



以上に挙げたプリミティブ (`\kuten`，`\jis`，`\euc`，`\sjis`，`\kansuji`) は展開可能 (expandable)

^{*14} 実は `\kansuji`，`\kansujichar` プリミティブは p3.1.1 でいったん削除され，p3.1.2 で復活したという経緯がある。

^{*15} upTeX では 0-127 も含め，Unicode の文字コードすべてが和文文字コードとして有効であり (`\kchar` で任意の文字コードを和文文字コードに変換して出力できる)，基本的にこのエラーは発生しない。

であり、内部整数を引数にとるが、実行結果は文字列であることに注意 (TeX82 の `\number`, `\romannumeral` と同様)。また、pTeX は JIS X 0213 には対応せず、JIS X 0208 の範囲のみ扱える。

```
\newcount\hoge
\hoge="2423          9251, 九二五一
\the\hoge, \kansuji\hoge\
42147, い
\jis\hoge, \char\jis\hoge\
一七〇一
\kansuji1701
```

以上の挙動から、`\kansuji` を「整数値をその符号値をもつ和文文字トークンに変換する」という目的に用いることもでき^{*16}、これは時に“`\kansuji` トリック”と呼ばれる。例えば

```
\kansujichar1=\jis"2422 \edef\X{\kansuji1}
```

としておけば、`\expandafter\meaning\X` は「kanji character あ」であるし、

```
\begingroup \kansujichar5=\jis"467C\relax \kansujichar6=\jis"4B5C\relax
\expandafter\gdef\csname\kansuji56\endcsname{test}
\endgroup
```

とすれば、`\日本` という和文の制御綴を ASCII 文字だけで定義できる。

10 長さ単位

pTeX では TeX82 に加えて以下の単位が使用可能である：

▶ zw

現在の和文フォント（通常の縦組のときは縦組用フォント、それ以外のときは横組用フォント）における「全角幅」。例えばこの文書の本文では $1\text{ zw} = 10.12534\text{ pt}$ である。

▶ zh

現在の和文フォント（通常の縦組のときは縦組用フォント、それ以外のときは横組用フォント）における「全角高さ」。例えばこの文書の本文では $1\text{ zh} = 9.64365\text{ pt}$ である。



より正確に言えば、`zw`, `zh` はそれぞれ標準の文字クラス（文字クラス 0）に属する和文文字の幅、高さと深さの和を表す。

ただ、pTeX の標準和文フォントメトリックの一つである `min10.tfm` では、 $1\text{ zw} = 9.62216\text{ pt}$, $1\text{ zh} = 9.16443\text{ pt}$ となっており、両者の値は一致していない。JIS フォントメトリックでも同様の寸法となっている。一方、実際の表示に使われる和文フォントの多くは、 $1\text{ zw} = 1\text{ zh}$ 、すなわち正方形のボディに収まるようにデザインされているから、これと合致しない。したがって、単位 `zh` はあまり意味のある値とはいえない。

なお、`japanese-otf` (OTF パッケージ) が用いているフォントメトリックは $1\text{ zw} = 1\text{ zh}$ である。

▶ Q, H

両者とも 0.25 mm ($7227/10160\text{ sp}$) を意味する。写植機における文字の大きさの単位である `Q` 数（級数）と、字送り量や行送り量の単位である歯数に由来する。

^{*16} ただし、upTeX で 0-127 の文字コードを `\kansujichar` で指定した場合のみ、`\kansuji` で生成されるトークンは欧文文字トークンになる [4]。

11 バージョン番号

► `\ptexversion`, `\ptexminorversion`, `\ptexrevision`

pTeX のバージョン番号は `px.y.z` の形式となっており，それらを取得するための命令である．`\ptexversion`, `\ptexminorversion` はそれぞれ x , y の値を内部整数で返し，`\ptexrevision` はその後ろの「.z」を文字列で返す．従って，全部合わせた pTeX のバージョン番号は

```
\number\ptexversion.\number\ptexminorversion\ptexrevision
```

で取得できる．pTeX p3.8.0 で導入された．

第 II 部

オリジナルの T_EX 互換プリミティブの動作

オリジナルの T_EX に存在したプリミティブの 2 バイト以上のコードへの対応状況を説明する。

12 和文に未対応のプリミティブ

以下のプリミティブでは、文字コードに指定可能な値は 0-255 の範囲に限られている：

`\catcode`, `\sfcode`, `\mathcode`, `\delcode`, `\mathchardef`, `\lccode`, `\uccode`



以前の pT_EX では、これらの命令の文字コード部分に和文文字の内部コードを指定することもでき、その場合は「引数の上位バイトの値に対する操作」として扱われていた：

```
\catcode"E0=1 \message{\the\catcode"E0E1}% ==> 1
```

しかしこの挙動は 2016-09-06 のコミット (r41998) により禁止され、T_EX Live 2017 の pT_EX (p3.7.1) で反映されている。

13 和文に対応したプリミティブ

▶ `\char <character code>`, `\chardef <control sequence>=<character code>`

引数として 0-255 に加えて和文文字の内部コードも指定できる。和文文字の内部コードを指定した場合は和文文字を出力する。

▶ `\accent <character code>=<character>`

`\accent` プリミティブにおいても、アクセントの部分に和文文字の内部コードを指定できるほか、アクセントのつく親文字を和文文字にすることもできる。

- 和文文字をアクセントにした場合、その上下位置が期待されない結果になる可能性が大きい。これは、アクセントの上下位置補正で用いる `\fontdimen5` の値が和文フォントでは特に意味を持たない^{*17}ためである。
- 和文文字にアクセントをつけた場合、
 - 前側には JFM グルーや `\kanjiskip` は挿入されない（ただし `\xkanjiskip` は挿入される）。
 - 後側には JFM グルーは挿入されない（ただし `\kanjiskip`, `\xkanjiskip` は挿入される）。

▶ `\if <token1> <token2>`, `\ifcat <token1> <token2>`

文字トークンを指定する場合、その文字コードは T_EX82 では 0-255 のみが許されるが、pT_EX では和文文字トークンも指定することができる。

^{*17} 欧文フォントでは x-height である。

`\if` による判定では、欧文文字トークン・和文文字トークンともにその文字コードが比較される。`\ifcat` による判定では、欧文文字トークンについては `\catcode`、和文文字トークンについては `\kcatcode` が比較される。



TeXbook には、オリジナルの TeX における `\if` と `\ifcat` の説明として

If either token is a control sequence, TeX considers it to have character code 256 and category code 16, unless the current equivalent of that control sequence has been `\let` equal to a non-active character token.

とある。すなわち

`\if` や `\ifcat` の判定では（実装の便宜上）コントロールシーケンスは文字コード 256、カテゴリーコード 16 を持つとみなされる

というのである。ところが、`tex.web` の実装はこの通りでなく、コントロールシーケンスをカテゴリーコード 0 とみなしている。そのため、pTeX 系列において和文文字トークンの `\kcatcode` の値が 16 である場合も、`\ifcat` 判定でコントロールシーケンスと混同されることはない。

参考文献

- [1] Victor Eijkhout, *TeX by Topic, A T_EXnician's Reference*, Addison-Wesley, 1992.
<https://www.eijkhout.net/texbytopic/texbytopic.html>
- [2] aminophen, 「縦数式ディレクションとベースライン補正」, 2016/09/05,
<https://github.com/texjporg/platex/issues/22>
- [3] h-kitagawa, 「[ptex] `\inhibitglue` の効力」, 2017/09/20,
<https://github.com/texjporg/tex-jp-build/issues/28>
- [4] aminophen, 「欧文文字の `\kansujichar`, `\inhibitxspcode`」, 2017/11/26,
<https://github.com/texjporg/tex-jp-build/issues/36>
- [5] aminophen, 「和文のコントロールシンボル」, 2017/11/29,
<https://github.com/texjporg/tex-jp-build/issues/37>
- [6] aminophen, 「[(u)pTeX] 内部コードの `-kanji-internal` オプション」, 2018/04/03,
<https://github.com/texjporg/tex-jp-build/issues/55>
- [7] aminophen, 「TeX Live 2019 での `\inhibitglue` の挙動変更【予定】」, 2019/02/06,
<https://oku.edu.mie-u.ac.jp/tex/mod/forum/discuss.php?d=2566>

索引

Symbols	
\accent	17
\autospaceing	8
\autoxspacing	8
\char	17
\chardef	17
\disinhibitglue	9
\dtou	10
\euc	14
\if	17
\ifcat	17
\ifdbox	11
\ifddir	11
\ifmbox	11
\ifmdir	11
\iftbox	11
\iftdir	11
\ifybox	11
\ifydir	11
\inhibitglue	8
\inhibitxspcode	7
\jcharwidowpenalty	6
\jfam	13
\jfont	13
\jis	14
\kanjiskip	6
\kansuji	14
\kansujichar	14
\kcatcode	4
\kuten	14
\noautospaceing	8
\noautoxspacing	8
\postbreakpenalty	5
\prebreakpenalty	5
\ptexminorversion	16
\ptexrevision	16
\ptexversion	16
\scriptbaselineshiftfactor	12
\scriptscriptbaselineshiftfactor	12
\showmode	8
\sjis	14
\tate	10
\tbaselineshift	12
\textbaselineshiftfactor	12
\tfont	13
\xkanjiskip	7
\xspcode	7
\ybaselineshift	12
\yoko	10
H	
H	15
Q	
Q	15
Z	
zh	15
zw	15