

pT_EX マニュアル

日本語 T_EX 開発コミュニティ*

version p4.0.0, 2022 年 2 月 20 日

本ドキュメントは、日本語 T_EX 開発コミュニティ版の pT_EX p4.0.0（以下、「コミュニティ版 pT_EX」）についてまとめたものである。pT_EX はもともとアスキー株式会社によって開発された^{*1}ので、しばしば「アスキー pT_EX」または「ASCII pT_EX」と呼ばれる。コミュニティ版 pT_EX は、日本語 T_EX 開発コミュニティがアスキー pT_EX を国際的なディストリビューション（かつての teT_EX や現在の T_EX Live）へ導入するにあたって幾つかの改良を加えたものであり、オリジナルとは動作が異なる点もあるので注意されたい。

- コミュニティ版 pT_EX の開発元：

<https://github.com/texjporg/tex-jp-build/>

- 本ドキュメントの開発元：

<https://github.com/texjporg/ptex-manual/>

なお、本ドキュメントで述べる pT_EX のプリミティブや拡張機能は、一般に「pT_EX 系列」と呼ばれる ε -pT_EX, upT_EX, ε -upT_EX でもそのまま利用可能である。upT_EX, ε -upT_EX については文字コードや和文文字トークンの扱いが異なるので、簡単に注記してある。

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

^{*1} 最新版は p3.1.11 (2009/08/17). <https://asciidwango.github.io/ptex/>

目次

第 I 部	pT _E X の日本語組版と追加プリミティブ	3
1	pT _E X で利用可能な文字	3
1.1	ファイルの文字コードと内部コード	3
1.2	和文文字と欧文文字の区別	4
1.3	文字コードの取得と指定	5
2	和文文字と \kcatcode	6
2.1	pT _E X の入力プロセッサ【pT _E X 4.0.0 での改修を含む】	7
2.2	和文文字の文字列化の挙動【pT _E X 4.0.0 での改修を含む】	9
3	禁則	10
4	文字間のスペース	12
5	組方向	15
6	ベースライン補正	18
7	和文フォント	19
8	文字コード変換, 漢数字	21
9	長さ単位	23
10	バージョン番号	24
第 II 部	オリジナルの T _E X 互換プリミティブの動作	25
11	和文に未対応のプリミティブ	25
12	和文に対応したプリミティブ	25
第 III 部	pT _E X の出力する DVI フォーマット	27


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

T_EX82 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 の文字も利用可能になっており、入力ファイルの文字コードは ISO-2022-JP (jis), EUC-JP (euc), Shift-JIS (sjis) または UTF-8 (utf8) に対応している。

 ここでは、歴史に興味のある読者への情報として、アスキー pTeX から現在のコミュニティ版 pTeX に至る文字コードの扱いの変遷を述べる。

初期のアスキー pTeX は「EUC 版 pTeX」「SJIS 版 pTeX」「JIS 版 pTeX」の 3 つのプログラムを別々に用意しており、一つの pTeX バイナリでは文字コードが EUC-JP, Shift-JIS, ISO-2022-JP のどれか一つのファイルしか処理できなかった。-kanji オプションが追加され、一つの pTeX バイナリで複数の文字コードを選択処理できるようになったのは pTeX p3.0.1 と p3.0.4 の間 (2002 年 10 月頃) である。

コミュニティ版 pTeX は、UNIX 向け日本語 T_EX ディストリビューション ptetex^{*2}の開発過程で誕生した。2006 年頃から文字コードが UTF-8 のファイル入力への対応が進められ、2007 年には文字コード変換を担う関数群が ptexenc というライブラリに切り出された。現在の T_EX Live でも ptexenc は種々のプログラムで利用されている。

1.1 ファイルの文字コードと内部コード

pTeX への入力は、ファイルの文字コードそのまま扱われるとは限らず、内部コードに変換されて内部処理に回される。ファイルの文字コードと内部コードは、それぞれ起動時のオプションによって制御できる。

- -kanji=<encoding>
入出力テキストファイルの文字コードを指定する。
利用可能な <encoding> の値: euc, sjis, jis, utf8
- -kanji-internal=<encoding>
内部コードを指定する (INI モード専用^{*3})。
利用可能な <encoding> の値: euc, sjis

このとおり、pTeX への入力を仮に UTF-8 としても、必ず EUC-JP か Shift-JIS のいずれかの内部コードに変換されるため、pTeX は JIS X 0208 外の文字をサポートしない。

^{*2} 土村展之さんによって 2004 年から 2009 年まで開発。その後継の ptexlive は、2010 年に pTeX が T_EX Live に取り込まれる際のベースになった。

^{*3} フォーマットファイルは内部コードに依存するので、実際の処理と整合性をとるため virtual mode では読み込んだフォーマットファイルと同じ内部コードで動作するように固定している。pTeX p3.8.2 以降の仕様。

pTeX の入出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

- 入出力ファイルの（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。入出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、入出力ファイルの文字コードと内部コードがともに Shift-JIS であることを示す。



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 [11]。

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

1.2 和文文字と欧文文字の区別

pTeX への入力は「7 ビット ASCII 文字集合」に「あるファイルの文字コードでエンコードされた JIS X 0208 の文字集合」を加えたものとして解釈される。そして、以下の規則により欧文文字と和文文字に区別して取り扱われる。

- 7 ビット ASCII 文字集合は欧文文字として扱われる。
- 最上位ビットが 1 の場合、pTeX はそのバイトで始まる列についてファイルの文字コードから EUC-JP または Shift-JIS のいずれかの内部コードへの変換を試みる。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。この内部コードのパターンに合えば和文文字として扱われ、合わなければ 8 ビット欧文文字のバイト列として扱われる^{*4}。

^{*4} 例えば、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。

なお、 $\text{T}_{\text{E}}\text{X}82$ には「 $\wedge\wedge$ 記法」という間接的な入力法があり、 $\wedge\wedge\text{ab}$ のようにカテゴリーコード 7 の文字 2 つに続いて 0–9, a–f のいずれかが 2 つ続くとそれを 16 進文字コードとする文字入力がなされたのと同じ処理に回るが、 $\text{pT}_{\text{E}}\text{X}$ 系列ではこれは常に欧文扱いされる（アスキー $\text{pT}_{\text{E}}\text{X}$ 3.1.4 以降^{*5}）。

最近（2010 年代以降）では、 $\text{pT}_{\text{E}}\text{X}$ でもファイルの文字コードを UTF-8 とする利用が増えてきた。コミュニティ版 $\text{pT}_{\text{E}}\text{X}$ では、`ptexenc` ライブラリによって UTF-8 ファイル入力を内部コード EUC-JP または Shift-JIS に変換する際に以下の特殊な加工を行っている。

- Unicode でのバラツキを同一視する多対一変換^{*6}
- BOM の無視（ファイル先頭に限らず）
- 濁音・半濁音の合字処理
- JIS に変換できない文字を $\wedge\wedge\text{ab}$ 形式に変換^{*7}

なお、UTF-8 ファイル出力時にはこのような加工の逆変換は行わない（入力時に加工されたままで出力される）。

このように $\text{pT}_{\text{E}}\text{X}$ は入力を内部コードに変換する処理を含むため、オリジナルの $\text{T}_{\text{E}}\text{X}$ や $\text{pdfT}_{\text{E}}\text{X}$ などの欧文 $\text{T}_{\text{E}}\text{X}$ とは入力に関して必ずしも互換でないことに注意が必要である。



以上で述べた「内部コードの範囲」は JIS X 0213 の漢字集合 1 面（Shift-JIS の場合は 2 面も）をまるまる含んでいるが、 $\text{pT}_{\text{E}}\text{X}$ は JIS X 0213 には対応していない。

JIS X 0213 で規定された「85 区 1 点」の位置の文字を入力ファイル中に書いた場合、

文字コードが EUC, Shift-JIS の場合 DVI には 29985 ("7521) 番の文字として出力される。

文字コードが JIS の場合「! Missing \$ inserted.」というエラーが発生する。これは、 $\text{pT}_{\text{E}}\text{X}$ が JIS X 0213 の 1 面を指示するエスケープシーケンス 1B 24 28 4F (JIS2000), 1B 24 28 51 (JIS2004) を認識せず、24 (\$) を数式モード区切りと解釈してしまうためである。

文字コードが UTF-8 の場合 UTF-8 のバイト列 E6 93 84 と読み込まれる。

なお、`\char` により「`\char\kuten"5521`」のようにして区点コードを指定した場合は、DVI には 29985 ("7521) 番の文字として出力される。また、DVI に文字として出力されたからといって、それを PostScript や PDF に変換したときに意図通りに出力されるかは全くの別問題である。

1.3 文字コードの取得と指定

$\text{pT}_{\text{E}}\text{X}$ でも $\text{T}_{\text{E}}\text{X}82$ と同様に、バッククオート (```) 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`」などで行うことはできない。

$\text{pT}_{\text{E}}\text{X}$ 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

^{*5} $\text{pT}_{\text{E}}\text{X}$ 3.1.8 で制御綴内での扱いが改善された。また、コミュニティ版 $\text{pT}_{\text{E}}\text{X}$ 4.0.0 で文字列化においても常に欧文扱いするように改修した。これについては 2.2 節を参照。

^{*6} 例えば、ソースファイル中の全角ダッシュ (U+2015) と EM ダッシュ (U+2014) は同一視され、内部的には JIS コード "213D として扱われ、ファイルに書き出される時は U+2015 になる。

^{*7} 例えば、ソースファイル中に `ç` のような欧文文字を直接書くと、これは和文文字の内部コードに変換できないため $\wedge\wedge\text{c3}\wedge\wedge\text{a7}$ に変換される。

- 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

2 和文文字と `\kcatcode`

$\mathrm{T}_{\mathrm{E}}\mathrm{X}82$ 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、 $\mathrm{T}_{\mathrm{E}}\mathrm{X}82$ 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]). $\mathrm{pT}_{\mathrm{E}}\mathrm{X}$ 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いずれかが割り当てられている。

また、 $\mathrm{pT}_{\mathrm{E}}\mathrm{X}$ p4.0.0 での入力プロセッサは 2.1 節で述べるように $\mathrm{T}_{\mathrm{E}}\mathrm{X}82$ のそれを拡張したものになっている。後の図 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**。これは、日本語の原稿内では自由な箇所で行が行えたほうが便利なためである。

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：

- $\mathrm{T}_{\mathrm{E}}\mathrm{X}82$ 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ことになっていたが、 $\mathrm{pT}_{\mathrm{E}}\mathrm{X}$ ではカテゴリーコードが 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^{*8}。
- 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 $\mathrm{T}_{\mathrm{E}}\mathrm{X}82$ 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。 $\mathrm{pT}_{\mathrm{E}}\mathrm{X}$ 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^{*9}。

► `\kcatcode <character code>={16–18}`

コミュニティ版 $\mathrm{pT}_{\mathrm{E}}\mathrm{X}$ では、和文文字のカテゴリーコード (`\kcatcode`) は DVI 中の上

^{*8} 「`\]`」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、これは $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ Live 2019 の $\mathrm{pT}_{\mathrm{E}}\mathrm{X}$ p3.8.2 で修正された ([10])。

^{*9} $\mathrm{upT}_{\mathrm{E}}\mathrm{X}$ 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。

位バイトごと（すなわち、JIS コードでいう区ごと）に値が設定可能である^{*10}。初期状態では、1, 2, 7–15, 85–94 区の文字の `\kcatcode` は 18, 3–6 区の文字は 17, 16–84 区の文字は 16 に設定されている。

`pTeX` においては、`\kcatcode` を上記の初期状態から変更することは想定されていない。また、変更したとしても、（先に述べた通り）和文文字トークンにカテゴリーコードの情報は保存されず、和文文字が処理対象となるたびにカテゴリーコードの値が再取得される。ただし、`\let\CS=あ` などとして和文文字トークンを `\let` すると、`\CS` にはその時のカテゴリーコード (`\kcatcode`) が保存される^{*11}。



例えば、以下のコードで定義される `\X`, `\Y` で引数終端を示す「あ」にはカテゴリーコードの情報は格納されない。そのため、`pTeX` では `\X`, `\Y` は全く同じ動作となる。

```
\kcatcode`あ=16
\def\X#1あ{\message{X: #1}}
\kcatcode`あ=17
\def\Y#1あ{\message{X: #1}}
```

なお、`upTeX` では `\kcatcode` は大きく仕様変更されており、`\X` と `\Y` は異なる動作となる。



`\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。`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。

2.1 `pTeX` の入力プロセッサ【`pTeX 4.0.0` での改修を含む】

`TeX82` の入力プロセッサは有限オートマトンとして記述できる。`pTeX` においてもその点是不変変わらないが、和文文字絡みで拡張されたものになっている。

図 1 は **`pTeX p4.0.0` 既定時**における入力プロセッサの状態遷移図である。`TeX82` から拡張された点、および説明が必要な点を以下に述べる。

内部状態の追加 `TeX82` では状態 *N* (new line), 状態 *M* (middle of line), 状態 *S* (skipping spaces) という 3 状態であったが、`pTeX` では次の 2 状態が追加された：

状態 *K* 和文文字の直後。状態 *M* との差異は改行でスペースを出力しないこと。

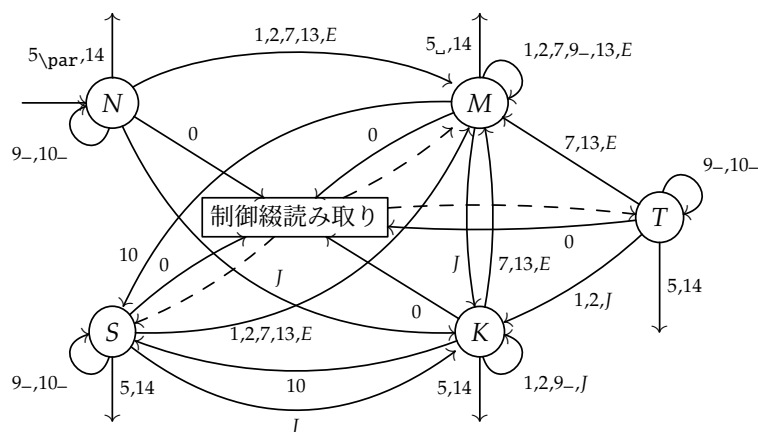
状態 *T* 和文文字で終わるコントロールワードの直後。状態 *S* との差異はグループ開始・終了 (`{, }`) での遷移先が状態 *M* でなく状態 *K* であること。

「制御綴読み取り」後の状態 以下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

^{*10} オリジナルのアスキー `pTeX` では、内部コードの上位バイトごとに値が設定可能であった。すなわち、内部コードが EUC-JP のときは区ごとに設定可能であったが、内部コードが Shift-JIS のときは $2n - 1$ 区・ $2n$ 区 ($1 \leq n \leq 47$) は同一のカテゴリーコードを持つことになる。

^{*11} コミュニティ版 `pTeX` では、一時的に「和文文字トークンを `\let` した `\CS` においても、それが処理対象となるたびにカテゴリーコードの値を再取得する」という挙動に変更しようとした (`r51021`)。しかし、この変更が不完全で「`\ifcat` では再取得するが、`\ifx` では再取得しない」という不統一な状態となってしまったため、`r59699` で従来の挙動に戻した（アスキー版と同じ）。結果的に、`TeX Live 2019–2021` では「ただし、…」が当てはまらない ([14])。



“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」を表す.
 但し, “9₋”, “10₋” はその文字を無視することを示している.

図1 pTeX 4.0.0 (既定) の入力プロセッサの状態遷移図

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, その名称の末尾の文字が欧文文字のときは状態 S に, 和文文字のときは状態 T に遷移する.

実際には以下に説明する `\ptexlineendmode` プリミティブによって, pTeX の入力プロセッサの挙動はある程度ユーザが制御できる.

▶ `\ptexlineendmode=<0-7>`

pTeX の入力プロセッサの挙動を制御する. この値を 2 進法で zyx と表したとき^{*12}, x, y, z の値は, それぞれ以下のような状態で行が終わった場合, 改行が空白文字を発生させるかを制御する. いずれも 0 では「空白文字を発生させない», 1 では「空白文字を発生させる」.

- x 和文文字で終わるコントロールワードの直後にグループ開始・終了が 1 つ以上^{*13}
- y 和文文字からなるコントロールシンボルの直後
- z (コントロールワード・コントロールシンボルの名称の一部でない) 和文文字の直後にグループ開始・終了が 1 つ以上

このプリミティブは pTeX 4.0.0 で追加された. 既定値は 0 (つまり $x = y = z = 0$).



「和文文字からなるコントロールシンボルの直後にグループ開始・終了が 1 つ以上ある」状態で行が終了した場合, 改行が空白文字を発生させるのは y, z のうち少なくとも 1 つが 1 のときである.

^{*12} つまり x は「一の位」, y は「二の位」, z は「四の位」である. 8 以上の値を指定した場合は, 下位 3 bit の値が使われる. 負数を指定してもエラーは発生しないが, その場合の動作は未定義である.

^{*13} pTeX では, このとき「改行が何も発生させない」挙動を伝統的にとってきた (TeX82 からの改造量を減らしたかったのであろう). この挙動の是非が `\ptexlineendmode` 実装の一要因となった.



従って、和文文字と行末の関係为例で示すと、以下のようになる：

1. コントロールワード・コントロールシンボルの一部でない和文文字で行が終わった場合、改行は何も発生しない。

あいあ → あいあい
い

2. コントロールワード・コントロールシンボルの一部でない和文文字の直後にグループ開始・終了 (`{`, `}`) が 1 つ以上ある状態で行が終わった場合、`\ptexlineendmode` の値が 0–3 のとき、改行は何も発生しない。4–7 のとき、改行は空白文字を発生する。

<code>\ptexlineendmode=0</code> → あいあい <code>{あいあ}</code> い	<code>\ptexlineendmode=4</code> → あいあい <code>{あいあ}</code> い
--	--

3. 和文文字で終わるコントロールワードの直後にグループ開始・終了が 1 つ以上ある状態で行が終わった場合、`\ptexlineendmode` の値が偶数のとき、改行は何も発生しない。奇数のとき、改行は空白文字を発生する。

<code>\ptexlineendmode=0</code> → あい <code>\def\漢{あ}{\漢}</code> い	<code>\ptexlineendmode=1</code> → あい <code>\def\漢{あ}{\漢}</code> い
--	--

4. 和文文字からなるコントロールシンボルで行が終わった場合、`\ptexlineendmode` の値が 0, 1, 4, 5 のとき、改行は何も発生しない。2, 3, 6, 7 のとき、改行は空白文字を発生する。

<code>\ptexlineendmode=0</code> → あい <code>\def\】{あ}{\】}</code> い	<code>\ptexlineendmode=2</code> → あい <code>\def\】{あ}{\】}</code> い
--	--

5. 和文文字からなるコントロールシンボルの直後にグループ開始・終了が 1 つ以上ある状態で行が終わった場合、`\ptexlineendmode` の値が 0, 1 のとき、改行は何も発生しない。その他のとき、改行は空白文字を発生する。

<code>\ptexlineendmode=0</code> → あい <code>\def\】{あ}{\】}</code> い	<code>\ptexlineendmode=1</code> → あい <code>\def\】{あ}{\】}</code> い
<code>\ptexlineendmode=2</code> → あい <code>\def\】{あ}{\】}</code> い	<code>\ptexlineendmode=4</code> → あい <code>\def\】{あ}{\】}</code> い

なお、以前の $\mathrm{pT\!E\!X}$ の入力プロセッサの挙動は

- $\mathrm{pT\!E\!X}$ p3.8.1 以前（アスキー版も同じ）の挙動は `\ptexlineendmode=3` ($(x, y, z) = (1, 1, 0)$)
- $\mathrm{pT\!E\!X}$ p3.8.2 以降 p3.10.0 まで挙動は `\ptexlineendmode=1` ($(x, y, z) = (1, 0, 0)$)

としてそれぞれ再現できる ([15]).

2.2 和文文字の文字列化の挙動【 $\mathrm{pT\!E\!X}$ 4.0.0 での改修を含む】

`\string` や `\meaning` などの文字トークン列生成については以下の通りである。

- 欧文文字は $\mathrm{T\!E\!X}82$ と同様で、（文字コード 32 の空白を除き）すべてカテゴリーコード 12 の文字トークンになるが、和文文字は 16–18 のいずれかのカテゴリーコードを持つ。
- 「コントロールワードの文字列化では後ろに空白文字を補い、コントロールシンボルの文


字列化では空白文字を補わない」という点は、和文文字を含む場合も同様である^{*14}。

- 和文文字はそのカテゴリコードによらず、`\meaning` すると

- 16: `kanji character` 漢
- 17: `kanji character` あ
- 18: `kanji character`)

となる。

さて、`pTeX` version 3 系列 (`TeX` Live 2021 まで) では `\meaning`, `\string` 等の文字列化や制御綴名において和文文字と欧文文字の区別が失われてしまうケースがあった。`pTeX` 4.0.0 以降 (`TeX` Live 2022) では、この状況でも和欧文の区別が維持される [16]。

 `TeX` Live 2021 では、ファイル文字コード UTF-8、内部コード EUC-JP の場合に例えば

```
\def\fuga{^^c3^^bf 耽}\fuga
\meaning\fuga
```

と入力すると、一行目は「`y 耽`」であるのに対し二行目は「`macro:->耽 耽`」となっていた (耽は EUC-JP で `"C3BF` である。`^^c3^^bf` の代わりに直接 `y` と入力しても同様)。また

```
\catcode"C3=11 \catcode"BF=11
\def\耽{P} \def^^c3^^bf{Q}
```

としたときの `\meaning\耽` は「`macro:->Q`」であった。改修後の `TeX` Live 2022 では、一つ目の例は「`macro:->Ãf 耽`」であるし、二つ目の例は「`macro:->P`」となる。

3 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^{*15}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

^{*14} 「`\`」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 p3.7.2 以前の `pTeX` 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。`TeX` Live 2018 の `pTeX` p3.8.1 でこの問題は修正された ([10])。

^{*15} 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。

- 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。ただし、欧文文字に設定されたこれらのペナルティが実際に挿入されるのは以下の場合に限られる（つまり、欧文組版だけの範囲では挿入されない）。

- 当該の欧文文字の直後が和文文字（前方に禁則ペナルティを伴っても構わない）の場合に限り、その欧文文字に設定された `\postbreakpenalty` を挿入する。
- 当該の欧文文字の直前が和文文字（後方に禁則ペナルティを伴っても構わない）の場合に限り、その欧文文字に設定された `\prebreakpenalty` を挿入する。



禁則ペナルティはリスト構築中に自動的に挿入されるので、`\showlists` で

```
\penalty 10000(for kinsoku)
```

のように表示される。

また、和文文字の後方に挿入された `\postbreakpenalty` は `\lastpenalty` で取得できるし、`\unpenalty` で取り除くこともできる^{*16}。しかし、上述のとおり、欧文文字に設定された `\postbreakpenalty` は後に和文文字が連続して初めて挿入されるため、`\lastpenalty` で取得できないし、`\unpenalty` で取り除くこともできない。

`\prebreakpenalty` は和文・欧文によらず、その直後に文字ノードを伴うため、原理的に `\lastpenalty` で取得できないし、`\unpenalty` で取り除くこともできない。

同一の文字に対して `\prebreakpenalty` と `\postbreakpenalty` の両方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以下の時に行われる [4, 5]：

^{*16} 以前の pTeX では、`\unpenalty` したはずの和文文字の後方の `\postbreakpenalty` が復活してしまう場合があったが、2017-04-06 のコミット (r43707) で修正された [6]。

- ペナルティ値 0 をグローバルに（つまり、`\global` を用いて）設定した場合.
- ペナルティ値 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）の場合^{*17}.

► `\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.

4 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もしくはカーンを設定

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
- 禁則ペナルティ (`\prebreakpenalty` や `\postbreakpenalty`) とが同じ箇所に発行される場合は「禁則ペナルティ → JFM 由来空白」の順に発行される.
- もし水平ボックス (`\hbox`) や `\noindent` で開始された段落が JFM 由来グルーで始まった場合は、そのグルーは取り除かれる (カーンは除かれない). また水平ボックスが JFM 由来グルーで終了した場合は、そのグルーは自然長・伸び量・縮み量のすべてが 0 となる.

► `\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に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.

^{*17} 禁則テーブルのある場所がローカルで 0 に設定されても、その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のないように、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.

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：

- 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^{*18}。

▶ `\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。

^{*18} 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。

- 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 (禁則テーブルからの削除と同様の規則)。



`\xspcode` と `\inhibitxspcode` 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 ([7, 12])。すなわち，

1. `\inhibitglue` は，ノード挿入処理を行う命令 (`\null`, `\hskip`, `\kern`, `\vrule`, ...) が後ろにき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：

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



pTeX 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` の効果を無効化（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。

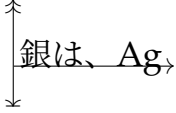

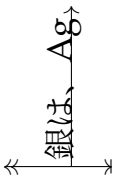

5 組方向

従来の TeX では，字送り方向が水平右向き（→），行送り方向が垂直下向き（↓）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**^{*19}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

^{*19} 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				

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。

► `\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^{*20}。
 - `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^{*21}。

^{*20} 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。

^{*21} `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

– 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.

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^{*22}。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 ([3])。この問題を解決する副産物として、

在する場合にのみである。

^{*22} 数式ディレクションか否かは異なっても良い。

バージョン p3.7.1 で `\ifmbox` プリミティブが実装された。

6 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、 $\text{p}\text{T}\text{E}\text{X}$ 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、 $\text{p}\text{T}\text{E}\text{X}$ p3.7^{*23}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。

^{*23} TEX Live 2016, 厳密には 2016-03-05 のコミット (r39938)。



`\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：

```
\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^{*24}。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。

7 和文フォント

\TeX は組版を行うとき、文字の情報を TFM ファイルから参照する。 \pTeX も同様で、欧文文字については TFM ファイルを参照するが、和文文字については JFM ファイルを参照する。ファイル名はどちらも **フォント名.tfm** である。

TFM ファイルには、大別すると

- それぞれの文字に関する事柄（各文字の幅、高さ、深さなど）
- その TFM ファイルに収められている文字に共通する事柄（文字の傾き、デザインサイズ、そのフォントの基準値 (em, ex) など）
- 特定の文字の組み合わせのときの事柄（合字やカーニングなど）

の 3 種類の情報が設定されている。JFM ファイルも TFM ファイルと同様の情報を持つが、

- 日本語の文字は数が多いので、文字を単位とするのではなく、共通する性質を持つ文字を一つにまとめてタイプ別の設定を行う
- 横組み用と縦組み用の区別がある

という点が異なる。JFM フォーマットの詳細は、この文書と同じディレクトリにある [2] を参照されたい。

► `\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したりする `\font` の和文版で

^{*24} `\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)。

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（要求された実際の TFM/JFM に応じて、自動的にアサインされる）。
- `\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。
- `\nullfont` は全ての文字が未定義な「空フォント」を指すが、これは欧文フォントであり、和文版 `\nullfont` という概念は存在しない。これは、 $\text{p}\text{T}\text{E}\text{X}$ では「全ての和文フォントには、和文文字コードとして有効な全ての文字が存在する」という扱いになっているためである。



細かい話をすれば、 $\text{p}\text{T}\text{E}\text{X}$ の ini mode でのフォーマット作成時に和文フォントを何も選択しなければ、`\fontname\jfont` が `\nullfont` となり、また和文文字を入力してもノードは作られない^{*25}ので、「和文版 `\nullfont` が選択されている」と言えなくもない。ただ、いったん実際の和文フォントを選択した後に「和文版 `\nullfont` を選択する」という制御綴は作れないと思われる。

► `\ifjfont `, `\iftfont `

`\ifjfont` は `` が和文の横組用フォントかどうか、`\iftfont` は和文の縦組用フォントかどうかを判定する。2020-02-05 のコミット (r53681) で追加され、 TEX Live 2020 の $\text{p}\text{T}\text{E}\text{X}$ (p3.8.3) で利用可能である。

これにより、例えば `\HOG` が和文フォントか否かは

```
\ifjfont\HOG
  (和文の横組用フォント)
\else\iftfont\HOG
  (和文の縦組用フォント)
\else
  (欧文フォント)
\fi
```

として判定できる。



上述のとおり、欧文フォント・和文の横組用フォント・和文の縦組用フォントのいずれも `\font` 一つで定義可能だが、定義したフォントが実際にどの種類だったかを知る手段はバージョン p3.8.2 までの $\text{p}\text{T}\text{E}\text{X}$ には存在しなかった。



ちなみに、 $\varepsilon\text{-p}\text{T}\text{E}\text{X}$, $\varepsilon\text{-up}\text{T}\text{E}\text{X}$ には `\iffontchar` プリミティブが存在する。しかし、`` が和文フォントか否かを判定するために `\iffontchar 256` などと第二引数に 256 以上の値を指定することはできない。なぜなら、欧文フォントに対し第二引数に 0–255 以外の値を指定すると「! Bad character code (...)」エラーが発生するからである。

^{*25} ただし、`\tracinglostchars > 0` でも Missing character: There is no あ in font nullfont! のような警告は出ない。

► `\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^{*26}。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。

8 文字コード変換，漢数字

pT_EX 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^{*27}。

► `\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。

► `\ucs <number>`

Unicode から内部コードへの変換を行う。もともと upT_EX で実装されていたが、pT_EX 3.10.0 で取り入れた。

► `\toucs <number>`

内部コードから Unicode への変換を行う。pT_EX 3.10.0 で追加した。



1 節でも述べたように、pT_EX は JIS X 0213 には対応せず、JIS X 0208 の範囲のみ扱える。なお、pT_EX 3.9.1 以前では、不正な文字コードを与えたときの挙動が不統一で、特に以下の値を返すケースもあった：

^{*26} pT_EX, upT_EX では 0–15 の範囲が許される。ε-pT_EX, ε-upT_EX では欧文の `\fam` と共に 0–255 に範囲が拡張されている。

^{*27} 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。

- 区点コード表の JIS X 0208 における最初の未定義位置 (JIS 0x222F, EUC 0xA2AF, SJIS 0x81AD)…和文文字コードとして有効で, JIS X 0213 では定義されている.
- 区点コード表の 1 区 0 点 (JIS 0x2120, EUC 0xA1A0, SJIS 0x813F)…文字コードとして無効.
- -1…文字コードとして無効.

pTeX 3.10.0 では `\ucs` と `\toucs` を追加し, さらに不正な文字コードを容易に判別できるように以下の仕様にした:

- 文字コード変換が**不要**なケース^{*28}…恒等変換となる. 不正な文字コードを与えてもそのまま通る. (これは従来どおりの挙動)
- 文字コード変換が**必要**なケース…不正な文字コードを与えると -1 を返す. (返り値を統一)

► `\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「〇一二三四五六七八九」). たとえば

```
\kansuji 1978年
```

は「一九七八年」と出力され,

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

は「壺式参四」と出力される. なお, `\kansuji` に続く数値 `<number>` が負の場合は, 空文字列になる (ちょうど `\romannumeral` にゼロまたは負の値を与えた場合と同様).

`\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, 例えば pTeX で `\kansujichar1=`A` のように無効な値 (pTeX において ``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^{*29}. また, `\kansujichar` の引数に許される値は 0-9 に限られ, 例えば `\kansujichar10=`拾` とすると

```
! Invalid KANSUJI number (10).
```

というエラーが発生する.



`\kansujichar` は整数値パラメータであるが, p3.8.2 までは「代入できるが取得はできない」という挙動であった (例えば `\count255=\kansujichar1` はエラー). pTeX p3.8.3 で取得もできるように修正された ([9]) が, 以前の pTeX も考慮すると, 値の取得は以下のようにするのが安全である: `\count255=\expandafter`\kansuji1`

^{*28} 内部 euc における `\euc`, 内部 sjis における `\sjis`, および upTeX で内部 `uptex` における `\ucs` と `\toucs` がこれに該当する.

^{*29} 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.



以上に挙げたプリミティブ (`\kuten`, `\jis`, `\euc`, `\sjis`, `\ucs`, `\toucs`, `\kansuji`) は展開可能 (expandable) 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 (T_EX82 の `\number`, `\romannumeral` と同様)。

```
\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^{*30}、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。

9 長さ単位

pT_EX では T_EX82 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、高さや深さの和を表す。

ただ、pT_EX 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。

^{*30} ただし、upT_EX 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カテゴリーコード 12 の欧文文字トークンになる [8]。

10 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。

11 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, \lccode, \uccode
```

違反すると

```
! Bad character code (42146).
```

というエラーが発生する。



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。

また、下記のプリミティブは名称が `\...char` であるが、値は 0-255 の範囲のみ有効であり、256 以上あるいは負の値を指定すると無効である（オリジナルの T_EX 同様、エラーにはならない）。

```
\endlinechar, \newlinechar, \escapechar,  
\defaultthyphenchar, \defaultskewchar
```

12 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。

► `\font, \fontname, \fontdimen`

`\font` については 7 節を参照。 `\fontname` は和文フォントからもフォント名を取得でき、 `\fontdimen` は和文フォントのパラメータ表 (JFM で定義される *param* テーブル) から値を取得できる。

► `\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^{*31}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` は挿入されうる）。
 - 後側には JFM グルーは挿入されない（ただし `\kanjiskip`, `\xkanjiskip` は挿入されうる）。

► `\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 $\mathrm{T}_{\mathrm{E}}\mathrm{X}82$ では 0–255 のみが許されるが、 $\mathrm{pT}_{\mathrm{E}}\mathrm{X}$ では和文文字トークンも指定することができる。

`\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。



$\mathrm{T}_{\mathrm{E}}\mathrm{X}book$ には、オリジナルの $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ における `\if` と `\ifcat` の説明として

If either token is a control sequence, $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ 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、 $\mathrm{pT}_{\mathrm{E}}\mathrm{X}$ 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。

一方、文字コードについては、確かに `tex.web` は `\if` 判定においてコントロールシーケンスを 256 とみなしている。しかし、 $\mathrm{upT}_{\mathrm{E}}\mathrm{X}$ では文字コード 256 の和文文字と衝突するので、2019-05-06 のコミット (r51021) で「原理的に文字コードが取り得ない値」に変更した ([13])。

^{*31} 欧文フォントでは `x-height` である。

第 III 部

pTeX の出力する DVI フォーマット

pTeX が出力する DVI ファイルは、欧文の横組のみを行って行けばオリジナルの TeX が出力する DVI ファイルと全く同様に解釈できる。一方、pTeX で和文文字を出力する場合、および組方向変更を行う場合は以下の DVI 命令が使用される。set2, set3 は [17] で定義されているが、オリジナルの TeX では使われていない。dir は [17] で定義されておらず、pTeX の独自拡張である。

- set2 (129) c[2]
コード番号が c ($0x100 \leq c < 0x10000$) の文字を印字し、参照点を移動する。pTeX では JIS コード、upTeX では UCS-2 が用いられる。
- set3 (130) c[3]
コード番号が c ($0x10000 \leq c < 0x1000000$) の文字を印字し、参照点を移動する。upTeX では UCS-4 の下位 3 バイトが用いられる (pTeX では現れない)。
- dir (255) d[1]
組方向を変更する。d[1] = 0 が横組、d[1] = 1 が縦組、d[1] = 3 が DtoU 組を示す。

pTeX が出力する DVI ファイルのプリアンブル部のフォーマット ID は、オリジナルの TeX と同じく常に 2 である。一方、ポストアンブル部の post_post 命令に続くフォーマット ID は pTeX でも通常 2 であるが、pTeX の拡張 DVI 命令である dir が使用されている場合のみ 3 にセットされる。

\special 命令の文字列は内部コードで符号化されたバイト列として書き出される。

参考文献

- [1] Victor Eijkhout, *TeX by Topic, A TeXnician's Reference*, Addison-Wesley, 1992.
<https://www.eijkhout.net/texbytopic/texbytopic.html>
- [2] ASCII Corporation & Japanese TeX Development Community, 「JFM ファイルフォーマット」, ./jfm.pdf
- [3] 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>
- [4] h-kitagawa, 「禁則テーブル, \inhibitxspcode 情報テーブルからのエントリ削除」,
2017/09/10,
<https://github.com/texjporg/tex-jp-build/pull/26>
- [5] Man-Ting-Fang, *[upTeX] Unexpected behaviour in kinsoku processing*, 2018/04/13,
<https://github.com/texjporg/tex-jp-build/issues/57>
- [6] aminophen, 「pTeX の後禁則ペナルティ」, 2017/04/05,
<https://github.com/texjporg/tex-jp-build/issues/11>
- [7] h-kitagawa, 「[ptex] \inhibitglue の効力」, 2017/09/20,

- <https://github.com/texjporg/tex-jp-build/issues/28>
- [8] aminophen, 「欧文文字の `\kansujichar`, `\inhibitxspcode`」, 2017/11/26,
<https://github.com/texjporg/tex-jp-build/issues/36>
- [9] aminophen, 「`[ptex]` reading `\kansujichar`」, 2019/10/14,
<https://github.com/texjporg/tex-jp-build/issues/93>
- [10] 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>
- [11] 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>
- [12] 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>
- [13] aminophen, 「`upTeX` の `\if` と `\ifcat`」, 2019/01/17,
<https://github.com/texjporg/tex-jp-build/issues/68>
- [14] aminophen, 「`pTeX` の和文文字トークンのカテゴリーコード」, 2019/04/22,
<https://github.com/texjporg/ptex-manual/issues/4>
- [15] h-kitagawa, 「`[ptex]` `[和字]` + `[ブレース]` で終わっている行の行端の扱い」, 2019/08/05,
<https://github.com/texjporg/tex-jp-build/issues/87>
- [16] h-kitagawa, 「バイト列と和文文字トークンの区別」, 2019/06/08,
<https://github.com/texjporg/tex-jp-build/issues/81>
- [17] TUG DVI Standards Working Group, *The DVI Driver Standard, Level 0*.
<https://ctan.org/pkg/dvistd>

索引

Symbols	
\accent	26
\autospacing	14
\autoxspacing	14
\char	25
\chardef	25
\disinhibitglue	15
\dtou	16
\euc	21
\font	25
\fontdimen	25
\fontname	25
\if	26
\ifcat	26
\ifdbox	17
\ifddir	17
\ifjfont	20
\ifmbox	17
\ifmdir	17
\iftbox	17
\iftdir	17
\iftfont	20
\ifybox	17
\ifydir	17
\inhibitglue	14
\inhibitxspcode	13
\jcharwidowpenalty	12
\jfam	21
\jfont	19
\jis	21
\kanjiskip	12
\kansuji	22
\kansujichar	22
\kcatcode	6
\kuten	21
\noautospacing	14
\noautoxspacing	14
\postbreakpenalty	11
\prebreakpenalty	11
\ptexlineendmode	8
\ptexminorversion	24
\ptexrevision	24
\ptexversion	24
\scriptbaselineshiftfactor	18
\scriptscriptbaselineshiftfactor	18
\showmode	14
\sjis	21
\tate	16
\tbaselineshift	18
\textbaselineshiftfactor	18
\tfont	19
\toucs	21
\ucs	21
\xkanjiskip	13
\xspcode	13
\ybaselineshift	18
\yoko	16
H	
H	23
Q	
Q	23
Z	
zh	23
zw	23