

pTeX マニュアル

日本語 TeX 開発コミュニティ*

pTeX: version p4.1.2

upTeX: vesrion p4.1.2-u2.01

2025 年 5 月 10 日

本ドキュメントは、日本語 TeX 開発コミュニティ版の pTeX p4.1.2（以下、「コミュニティ版 pTeX」）および upTeX p4.1.2-u2.01 についてまとめたものである。

pTeX 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 pTeX」または「ASCII pTeX」と呼ばれる。コミュニティ版 pTeX は、日本語 TeX 開発コミュニティがアスキー pTeX を国際的なディストリビューション（かつての teTeX や現在の TeX 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。

upTeX は田中琢爾氏による pTeX の拡張版であり、 ϵ -pTeX および ϵ -upTeX は北川弘典氏によりそれぞれ pTeX と upTeX に ϵ -TeX がマージされた拡張版である。これらはいずれもコミュニティ版 pTeX をベースに開発されており、総称して「pTeX 系列」と呼ばれる。

本ドキュメント (ptex-manual.pdf) では、pTeX 系列における共通機能と upTeX による拡張を説明する。 ϵ -pTeX/ ϵ -upTeX による拡張については eptexdoc.pdf を参照されたい。

- コミュニティ版 pTeX 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/>

* <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	pTeX の日本語組版と追加プリミティブ	5
1	pTeX 系列で利用可能な文字	5
1.1	ファイルのエンコードと内部コード	5
1.2	有効な文字コードの範囲【pTeX の場合】	7
1.3	有効な文字コードの範囲【upTeX の場合】	8
1.4	文字コードの範囲の拡張【upTeX u1.35, u2.00】	10
1.5	文字コードの取得と指定	13
2	和文カテゴリコードと TeX の入力プロセッサ	14
2.1	和文カテゴリコード【pTeX の場合】	14
2.2	和文カテゴリコード, 和文文字と欧文文字の区別【upTeX の場合】	16
2.3	pTeX の入力プロセッサ	18
2.4	和文文字の文字列化の挙動	20
3	和文文字の出力	21
3.1	文字ノードの生成	21
3.2	和文フォント	22
4	pTeX 系列の組版処理	26
4.1	禁則	26
4.2	文字間のスペース	28
4.3	組方向	31
4.4	ベースライン補正	34
5	その他の補助機能	35
5.1	文字コード変換, 漢数字	35
5.2	長さ単位	38
5.3	バージョン番号	38
第 II 部	オリジナルの TeX 互換プリミティブの動作	40
6	和文に未対応のプリミティブ	40
7	和文に対応したプリミティブ	40
第 III 部	pTeX の出力する DVI フォーマット	42

表記について

pTeX 系列については、以下のようにまとめた表記もしばしば用いられる。

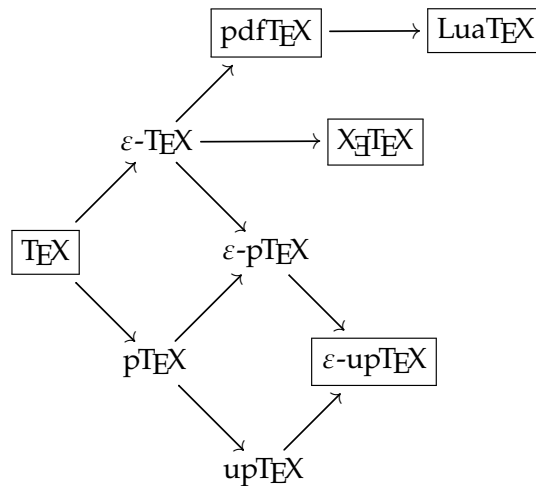
- pTeX, upTeX \rightarrow (u)pTeX
- pTeX, ϵ -pTeX \rightarrow (ϵ -)pTeX
- upTeX, ϵ -upTeX \rightarrow (ϵ -)upTeX
- ϵ -pTeX, ϵ -upTeX \rightarrow ϵ -(u)pTeX
- pTeX, upTeX, ϵ -pTeX, ϵ -upTeX \rightarrow (ϵ -)(u)pTeX

本ドキュメントでは、特に断らない限り単に pTeX と表記すれば (ϵ -)(u)pTeX の 4 種類全てを指すこととする。また upTeX と表記すれば (ϵ -)upTeX の 2 種類を指すこととする。

ただし、節タイトルで **【pTeX の場合】** **【upTeX の場合】** のように分割した箇所については、それぞれ (ϵ -)pTeX と (ϵ -)upTeX を指すこととする。また、(すぐ下のよう)に 列挙する形で記す場合はそれぞれ単独のエンジンを指す。

TeX Live における配布

現在、TeX Live で配布されている主要なエンジンと pTeX 系列の関係性は以下の図で表される。独立したプログラム (バイナリ) として配布されているものを 枠 で囲んである。



TeX Live 2022 までは pTeX, upTeX, ϵ -pTeX, ϵ -upTeX の 4 種類のエンジンが独立したプログラムとして配布されていたが、TeX Live 2023 では素の pTeX と upTeX の配布が停止され、それぞれ ϵ -pTeX と ϵ -upTeX の互換モードに置き換えられた。さらに TeX Live 2024 ではオリジナルの ϵ -pTeX の配布も停止され、4 種類のエンジンは全て ϵ -upTeX という単一のプログラムによるエミュレートへと整理された。表にまとめると次のようになる。

互換モード ϵ -TeX の拡張モード (extended mode) でない状態、すなわち `-etex` スイッチ無効状態。
例えば、2022 年まではコマンド `ptex` で `This is pTeX, Version ...` が起動していたが、2023 年は `This is e-pTeX, Version ...` が起動する。ここだけ見るとコマンド `eptex` と似ているが、`eptex` では直後に `entering extended mode` と拡張モードに入ると異なり、`ptex` では拡張

モードに入らないので、 ϵ -TeX 特有の機能およびプリミティブは無効化された状態となる。

内部レガシー (ϵ -upTeX が文字コードに関する内部処理を Unicode ではなく (ϵ -pTeX 同様のレガシーエンコーディング (EUC-JP や Shift-JIS) で行うことを指す。内部コードの差異だけであり、その他の (ϵ -upTeX 特有の機能およびプリミティブは利用可能な状態である。本文中 `-kanji-internal` の説明も参照。

例えば、2023 年のコマンド `eptex` が `This is e-pTeX, Version ... (utf8.euc)` であったならば、2024 年には `This is e-upTeX, Version ... (utf8.euc)` に変わっている。一方 `euptex` は `This is e-upTeX, Version ... (utf8.uptex)` であるから内部コードが異なる。

コマンド名	TeX Live 2022 まで	TeX Live 2023	TeX Live 2024 以降
<code>ptex</code>	pTeX	ϵ -pTeX の互換モード	ϵ -upTeX の内部レガシーの互換モード
<code>uptex</code>	upTeX	ϵ -upTeX の互換モード	→変更なし
<code>eptex</code>	ϵ -pTeX <small>拡張モード</small>	→変更なし	ϵ -upTeX の内部レガシー
<code>euptex</code>	ϵ -upTeX <small>拡張モード</small>	→変更なし	→変更なし

pTeX 系列における L^AT_EX

(ϵ -pTeX で動くように調整された L^AT_EX を pL^AT_EX, (ϵ -upTeX で動くように調整された L^AT_EX を upL^AT_EX と呼ぶ。

pL^AT_EX はもともとアスキー株式会社によって pTeX と一体で開発されていた^{*2}。2010 年にコミュニティ版 pTeX が TeX Live に収録されて以降、そこで独自の改良や仕様変更が加えられるようになり、またベースである L^AT_EX も更新が進んでいる。特に 2020 年以降は、L^AT_EX は素の TeX では動作せず、拡張モードの ϵ -TeX “+ α ” が動作条件となっており、したがって素の (u)pTeX では動作しない。そうした流れに合わせてアスキー版 pL^AT_EX から fork したのが「コミュニティ版 pL^AT_EX」である。upL^AT_EX はもともとアスキー版 pL^AT_EX を元に upTeX と一体で開発されていたが、現在はコミュニティ版 pL^AT_EX をベースにしている。

- コミュニティ版 pL^AT_EX の開発元：
<https://github.com/texjporg/platex/>
- コミュニティ版 upL^AT_EX の開発元：
<https://github.com/texjporg/uplatex/>

TeX Live でのエンジン利用状況は以下の通りである。ここで、2023-06-01 以降は `platex` で ϵ -pTeX ではなく ϵ -upTeX の内部レガシーが起動することに注意 (`eptex` より一足早く変更された)。

コマンド名	TeX Live 2012 から 2023 初期まで	TeX Live 2023 の 2023-06-01 以降
<code>platex</code>	ϵ -pTeX <small>拡張モード</small>	ϵ -upTeX <small>拡張モード</small> の内部レガシー
<code>uplatex</code>	ϵ -upTeX <small>拡張モード</small>	→変更なし

^{*2} 最新版は 2006/11/10。これを「アスキー版 pL^AT_EX」と呼ぶことにする。

第 I 部

pTeX の日本語組版と追加プリミティブ

ここでは、pTeX 系列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 系列で利用可能な文字

オリジナルの TeX で扱える文字コードの範囲は 0–255 であった^{*3}。pTeX 系列では、日本語の文字を (LaTeX の `inpuenc` のようなアクティブ化によらず) 直接扱えるように、文字コードとして有効な範囲を拡張している。以降では、従来の枠組みで扱える文字を **欧文文字**、それ以外を **和文文字** と呼ぶ。

なお、本節 (1 節) 全体を通して ϵ -pTeX は pTeX と同じ、 ϵ -upTeX は upTeX と同じである。

1.1 ファイルのエンコードと内部コード

pTeX 系列は、入出力ファイルのエンコードとして ISO-2022-JP (jis), EUC-JP (euc), Shift-JIS (sjis) または UTF-8 (utf8) に対応している。ただし、入力はそのままのエンコードで扱われるとは限らず、各行は内部コードに変換されてから内部処理に回され、ファイル出力時にもやはり内部コードからの変換が行われる。ファイルのエンコードと内部コードが異なる場合は `ptexenc` ライブラリに従ってエンコード変換が発生することとなる。TeX の入力プロセッサに実際に渡るのは入力各行の変換結果である。

pTeX の内部コードは Windows では Shift-JIS、それ以外では EUC-JP が既定であり、upTeX の内部コードは既定で Unicode であるが、起動時のオプション `-kanji-internal` (INI モード時のみ有効：後述) によって制御できる。

pTeX/upTeX の入出力ファイルのエンコードは、TeX Live 2018 以降は全てのプラットフォームで UTF-8 が既定である^{*4}が、個別の入力ファイルのエンコードは次の順番で決まる (`ptexenc` による)。この決定手続きはファイルの先頭行を読むときに行われる。

1. ファイル先頭に UTF-8 の BOM^{*5}を見つけたら UTF-8 (BOM 付き) とみなす。
2. エンコード推測機能が有効ならば推測を行い、成功すればそれを採用する^{*6}。
3. 起動時のオプション `-kanji` (後述) による指定があればそれを使う。
4. 環境変数 `PTEX_KANJI_ENC` が定義されていればそれを使う。
5. 以上で決まらない場合は既定のエンコードを使う。


^{*3} 当初の TeX82 では 7-bit だった：TeX90 で 8-bit になった。

^{*4} TeX Live 2017 以前では pTeX に限り、Windows では Shift-JIS、それ以外では UTF-8 であった。upTeX は当初から全てのプラットフォームで UTF-8 であった。

^{*5} 正確には、最初の 4byte が `0xEF 0xBB 0xBF 0x7E` 以下。

^{*6} TeX Live 2022 以前は Windows 専用の機能であったが、TeX Live 2023 で全てのプラットフォームで利用可能となった [18]。kpathsearch ライブラリ変数 `guess_input_kanji_encoding` によって制御でき、値が 1 ならば有効、0 ならば無効である。

なお、ISO-2022-JP でエンコードされた文字は現在のエンコードによらず正しく読まれる。また、 ε -(u)pTeX の `\epTeXinputencoding` を使用すればその次の行から新しいエンコードで読まれる (`./eptexdoc.pdf` を参照)。

 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 に至るエン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 バイナリではエンコードが 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 バイナリで複数のエンコードを選択処理できるようになったのは pTeX 3.0.1 と 3.0.4 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 TeX ディストリビューション pTeX^{*7} の開発過程で誕生した。2006 年頃からエンコードが UTF-8 のファイル入力への対応が進められ、2007 年にはエン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 TeX Live でも `ptexenc` は種々のプログラムで利用されている。

■ pTeX 系列の起動時のオプション

- `-kanji=<encoding>`
入出力テキストファイルのエンコードを指定する。
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^{*8})。
利用可能な `<encoding>` の値 : `euc, sjis` (upTeX のみ : `uptex` も利用可能)

pTeX で利用可能な内部コードは `euc` (EUC-JP) か `sjis` (Shift-JIS) のいずれかであるため、入力を仮に UTF-8 としても必ず内部コードへの変換が起こる。upTeX では既定状態の内部コードが `uptex` (Unicode) なので、入力を UTF-8 とすれば変換は“ほとんど”起こらない^{*9}。

■ pTeX 系列の起動時のバナー

入出力ファイルのエン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) エンコードは 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。入出力ファイルのエンコードと内部コードが同じ場合は、

^{*7} Thomas Esser さんによる `teTeX` をベースとして、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 TeX Live に取り込まれる際のベースになった。

^{*8} 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。

^{*9} 例外は `ptexenc` による基底文字と濁点に分解された形 (結合文字列) の合成など : 後述。

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 (入出力ファイルのエンコードと内部コードがともに Shift-JIS である例)。



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込が行われる。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 [12]。

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

1.2 有効な文字コードの範囲【pTeX の場合】

pTeX では、JIS X 0208 すなわち JIS 第 1, 2 水準の文字を利用可能である。入力は「7 ビット ASCII 文字集合」に「あるファイルのエンコードで表現された 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、以下の規則により和文文字と欧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。また、TeX90 互換の「^^ 記法」という間接的な入力法^{*10}も常に欧文扱いされる^{*11}。
- 最上位ビットが 1 の場合、そのバイトで始まる列についてファイルのエンコードから内部コード (EUC-JP または Shift-JIS) への変換を試みる (同一コードなら恒等変換)。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 $C_1 = C_2 = \{ "a1, \dots, "fe \}$ 。

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

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

である。この内部コードのパターンに合えば「変換後の文字コードを持つ一つの和文文字」として扱われ、合わなければ一旦 ^^ab 形式への変換 (後出の★) を経て「8 ビット欧文文字のバイト列」として扱われる。



一遍に書いているが、入力ファイルを一行読み込んで必要なエンコード変換を行った結果をバッファに格納するところまでが ptexenc による前処理である。そのバッファを TeX の入力プロセッサが読み、文字のカテゴリーコードに従って (和文と欧文の区別を付けながら) トークン列を生成する、という流れである。ptexenc による前処理には「内部コードに変換できない入力のバイト列を ^^ab 形式に変換」も含まれるが、この時点ではカテゴリーコードについて知らないため、文字 ^ の \catcode が 7 かどうかは関係なく ^^ が用いられる。

^{*10} ^^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になされたのと同じ処理に回る。

^{*11} ^^ 記法の欧文扱いはアスキー pTeX 3.1.4 以降。pTeX 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 pTeX 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 2.4 節を参照。

pTeX における有効な文字コードの範囲は「欧文文字として有効な文字コード (0-255)」と「和文文字として有効な文字コード (内部コードとして許される整数値)」の和集合であり、前者は 1 バイト・後者は 2 バイトなので互いに重ならない。

最近 (2010 年代以降) では、pTeX でもファイルのエンコードを UTF-8 とする利用が増えてきた。コミュニティ版 pTeX は UTF-8 ファイル入力に対応しているが、先述の通り内部コードは EUC-JP または Shift-JIS のいずれかであるため、**pTeX は JIS X 0208 外の文字をサポートしない**。なお、ptexenc による前処理に以下の特殊な加工を含めてある。

- Unicode でのバラツキを同一視する多対一変換 (表 1)*¹²
- BOM の無視 (ファイル先頭に限らず)
- 結合濁点 (U+3099)・半濁点 (U+309A) の合字処理*¹³
- JIS 外のため変換できない文字を ^^ab 形式に変換*¹⁴...★

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まで出力される)。

このように **pTeX 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 TeX や pdfTeX などの欧文 TeX 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、**pTeX 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、

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

エンコードが JIS の場合 “! Missing \$ inserted.” というエラーが発生する。これは、pTeX 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。

エンコードが UTF-8 の場合 UTF-8 のバイト列 E6 93 84 として読み込まれる。

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 有効な文字コードの範囲【upTeX の場合】

upTeX は pTeX と事情が異なるので、ここで説明する。pTeX との違いを強調してある。

- 文字集合は内部コードにより範囲が異なる。
 - 内部レガシー (EUC-JP または Shift-JIS) の場合: 「7 ビット ASCII 文字集合」に「あるファイルのエンコードで表現された JIS X 0208 の文字集合」を加えたもの。つまり pTeX となら変わらない。

*¹² 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。

*¹³ ただし、Unicode で単一の文字コードで表せる場合に限る。

*¹⁴ 例えば、ソースファイル中に ç のような JIS X 0208 外の文字を直接書くと、これは和文文字の内部コードに変換できないため ^^c3^^a7 に変換されて欧文扱いされる。

- 内部 **Unicode** の場合：**Unicode 全体**.
- 和文文字と欧文文字の区別は以下の規則による：
 - 7ビット ASCII 文字集合は欧文文字として扱われる。また、TeX90 互換の「^^ 記法」という間接的な入力法も常に欧文扱いされる。(pTeX と同様)
 - 最上位ビットが 1 の場合、そのバイトで始まる列についてファイルのエンコードから内部コードへの変換を試みる(同一コードなら恒等変換)。内部コードのパターンに合えば「変換後の文字コードを持つ一つの文字」として扱われ、それが和文扱いか欧文扱いかは `\kcatcode` に依存する(詳細は 2.2 節)。合わなければ一旦 ^^ab 形式への変換(後出の★)を経て「8ビット欧文文字のバイト列」として扱われる。
- ptexenc ライブラリによる前処理も pTeX と類似だが、変換方向によって少々異なる。
 - Unicode 多対一変換：内部レガシーの場合は pTeX と同様。内部 **Unicode** の場合は、入出力 UTF-8 に対しては変換を行わず、入力 EUC-JP/Shift-JIS からの変換について一対多対応があれば **Unicode** への変換は表 1 に従う。
 - BOM の無視は行う。(pTeX と同様)
 - 結合濁点・半濁点の合字処理は行う (TeX Live 2016 以降^{*15})。 (pTeX と同様)。
 - 範囲外の文字の ^^ab 形式への変換：内部レガシーでは pTeX と同様。内部 **Unicode** では、文字集合が **Unicode 全体** なので範囲外の文字は存在せず、従って ^^ab 形式への変換は基本的に起こらない^{*16}。 …★

upTeX における有効な文字コードの範囲は「欧文文字として有効な文字コード (0-255)」と「和文文字として有効な文字コード」の和集合である。内部レガシーの場合は pTeX と全く同じであり、前者は 1 バイト・後者は 2 バイトなので互いに重ならない。内部 **Unicode** の場合は後者が **Unicode 全体** (0 以上 0x10FFFF 以下の整数値) となるので、**0-255 の範囲は重なっており和文・欧文どちらの文字コードとしても有効であることに注意**。

いずれの場合も、upTeX はオリジナルの TeX や pdfTeX などの欧文 TeX とは入力に関して


表 1 JIS ⇔ Unicode 一対多変換。Unicode → JIS 変換では複数の文字が同一視される。JIS → Unicode 変換では複数候補のうち太字が選択される。

区点	Character Name	文字	JIS X 0208	Unicode
1-17	OVERLINE	—	0x2131	U+203E, U+FFE3
1-29	EM DASH	—	0x213D	U+2014, U+2015
1-33	WAVE DASH	～	0x2141	U+301C , U+FF5E
1-34	DOUBLE VERTICAL LINE	∥	0x2142	U+2016 , U+2225
1-36	HORIZONTAL ELLIPSIS	…	0x2144	U+2026 , U+22EF
1-61	MINUS SIGN	−	0x215D	U+2212 , U+FF0D
1-79	YEN SIGN	¥	0x216F	U+00A5, U+FFE5
1-81	CENT SIGN	¢	0x2171	U+00A2, U+FFE0
1-82	POUND SIGN	£	0x2172	U+00A3, U+FFE1
2-44	NOT SIGN	¬	0x224C	U+00AC, U+FFE2

^{*15} 結合濁点・半濁点は TeX Live 2015 までは pTeX でのみ合字処理され、upTeX では行っていなかった。

^{*16} 不正な UTF-8 入力などに限り ^^ab 形式に変換される。

必ずしも互換でない（内部 Unicode への UTF-8 入力に限っても「BOM の無視」と「結合濁点・半濁点の合字処理」だけは起こる）が、pTeX と比較すれば幾分差異が軽減されている。

 内部コード Unicode (uptex) の場合の上限が 0x10FFFF というのは実は嘘である。24bit 整数値が有効であり、うち 0x110000 以上 0xFFFFFFFF 以下は特殊な用途で利用できるよう留保されている。

- Bad character code エラーが出ない範囲：0 以上 0xFFFFFFFF 以下^{*17}
- 和文文字トークンの文字コードとして可能な範囲：0 以上 0x10FFFF 以下
- 和文文字ノードの文字コードとして可能な範囲：0 以上 0xFFFFFFFF 以下

upTeX では和文文字トークンおよび和文文字ノードにおいて文字コードを UTF-32 の下位 24bit で格納する仕様となっている。Unicode 文字入力で 0x110000 以上 0xFFFFFFFF 以下の文字コードはありえないが、0x110000 以上 0x21FFFF 以下の文字コードを数値で指定する場合の扱いは以下としている：

- `\char`, `\chardef` 及び `\kchar`, `\kchardef` : 文字コードが 0x110000 以上でもそのままの値で和文文字ノードを生成する。DVI 出力時には「文字コード mod 0x110000」で書き込まれる。
- `\kansujichar` への文字コード代入：値の格納時に「文字コード mod 0x110000」を行うので、以降の `\kansuji` により生成する文字トークンは 0 以上 0x10FFFF 以下。
- ε -(u)pTeX の `\Uchar`, `\Ucharcat` : 文字トークン生成時に「文字コード mod 0x110000」を行うので、やはり 0 以上 0x10FFFF 以下になる。

特に `\kchar` の上記仕様は OTF パッケージで使われている。これは、`\CID (Adobe-Japan1)` のような Unicode でない文字コードを扱う内部処理において Unicode の正規の文字コード (0x10FFFF 以下) のカテゴリーコードや禁則ペナルティのしがらみの影響を受けない目的である。0x220000 以上の文字コードは上記の数値変換を行わない (1.4.1 節を参照)。

1.4 文字コードの範囲の拡張【upTeX u1.35, u2.00】

upTeX では従来の文字コードの範囲からの拡張を u1.35 および u2.00 (TeX Live 2025) にて行った。従来は和文文字を Unicode 1 文字のコードポイントで「1 文字 1 トークン」として取り扱うのに対し、欧文文字は 8bit の範囲内にとどまり「1 バイト 1 トークン」で扱っていたが、それを超えて扱えるようにするものである。やや特殊な扱いを要する上、新規の要素も多いためここに節を立てて説明する。

1.4.1 Unicode 合成文字と異体字セレクタ (IVS) (upTeX u1.35)

Unicode では、一つの文字を表すために複数のコードポイントの連なり (シーケンス) で示すことが規定されている文字がある。その例が異体字セレクタ (IVS, Ideographic Variation Sequence/Selector)[23] や絵文字シーケンス [24] などである。upTeX ではそれら合成文字を従来扱うことができなかったが、upTeX u1.35 で対応を進めた [11]。仕様は以下の通りである。

- 想定した範囲のシーケンスの集合^{*18}を定め、upTeX 独自の内部用の文字コード (合成文字用拡張文字コード) を定義する (後述)。合成文字用拡張文字コードにはコード値 0x220000 以降を用い、和文文字として扱う。
- `ptexenc`, upTeX エンジン拡張する。ptexenc ライブラリで文字コードの合成・分解

^{*17} Invalid KANSUJI char エラーが出ない範囲も同じ。

^{*18} 言い換えれば網羅的ではなく想定外のシーケンスには対応していない。

の API を用意する。upTeX と dviware の間の文字データ渡しには合成文字用拡張文字コードを用いる。

- upTeX に `\kcatcode 20 (modifier)` を新設する。これは UTF-8 の入力バッファの文字列で現れる異体字セレクタや合成用濁点を想定しているが、**基底文字とは独立した 1 つの和文文字トークン**として扱われる。トークンとしての扱いは `\kcatcode 16, 17` と同様である（直後の改行は空白とみなさない、コントロールワードに使用可能）。
- upTeX がトークン列から文字ノードを生成する段階で、基底文字と *modifier* が連続してノード化されようとする場合^{*19}には、独立した 2 つのノードを生成する代わりに、基底文字の文字コードを合成文字用拡張文字コードに置き換えた**単一の合成文字ノード**を生成する^{*20}。合成に失敗した *modifier* は普通の文字（CJK 記号）としてふるまい、`\kcatcode 18 (other_kchar)` の文字ノードを生成する。
- upTeX エンジン内部では合成文字ノードを単一のノードとして扱うが、処理中に合成文字のデータを文字バッファに書き戻す際には、常に UTF-8 の複数コードポイントのシーケンスに戻す^{*21}。それゆえ、テキスト形式の入出力では特殊な内部コードが現れることはなく必ず正規の Unicode になる。
- upTeX の合成文字用拡張文字コード値を `\char`, `\chardef` などの文字ノード生成プリミティブに直接与えることもできる。また、合成文字用拡張文字コード値を `\kansujichar` や (ϵ -upTeX の) `\Uchar` などの文字トークン生成プリミティブに与えると、展開結果は基底文字 + *modifier* の 2 トークンとなる。
- upTeX が合成文字ノードを DVI に書き出す際には、その合成文字用拡張文字コード値で DVI 命令 `set3` で書き込む。
- `dvips`, `dvipdfmx` など Adobe-Japan1 (AJ1) の CID が扱える dviware では、AJ1 の文字に対して DVI 命令 `set3` から AJ1 の CID コードへ変換する *virtual font* を経由して出力することが可能である。
- dviware 側の実装として、合成文字用拡張文字コードを直接扱い DVI 命令 `set3` に対し適切なグリフを選択するようにしてもよい。

upTeX u1.35 で対応した合成文字用拡張文字コードの概略を表 2 にまとめた。各文字コードの具体的な変換式は `uptex-base` で配布している `01uptex_doc_utf8.txt` を参照されたい。異体字セレクタ (VS, Variation Selector) は前述の IVS の他、SVS (Standardized Variation Sequence) にも対応している。IVS の最大 VS256 まで拡張可能になるよう合成文字用拡張文字コードでは考慮してあるが、おおもとの Unicode Ideographic Variation Database[23] では今のところ VS49~VS256 が未定義である。仮名の濁点・半濁点は基底文字が平仮名や片仮名の場合に加え変体仮名にも対応している。絵文字はかつての日本の携帯電話由来の絵文字の大部分に対応しているが、最近よく追加で規定される長くて複雑なシーケンスには対応していない。絵文字の国旗 (RGI Emoji Flag Sequence) は、*modifier* と *modifier* の連続が規定

^{*19} ファイル中に直接文字を入力した場合に限らず、基底文字の直接入力 + `\char` などによる *modifier* の文字コード指定入力、またはその逆でも良い。

^{*20} 文字トークンの段階で 2 文字だったものが 1 つの文字ノードに変換されるという点では欧文の合字処理 (リガチャ) に似ているが、基底文字と *modifier* の合成は展開不能トークンが来ても中断されない。

^{*21} 文字ノードのデータを `\showlists` で表示する際などが該当。文字トークンは元々合成されていない。

通りに組み合わせると合成文字の扱いになる。国旗および肌色 (Emoji Modifier Fitzpatrick Type-1..6) など絵文字全般について、DVI に書かれたデータをどう出力するかは dviware にお任せする。今後、さらに合成文字を追加対応するかどうかは未検討であり計画もない。

表中で modifier に相当する文字コードは upTeX u1.35 では既定で `\kcatcode=20(modifier)` に設定してある。合成文字用拡張文字コードのメトリックとしては通常 JFM の既定値の全角幅を想定しているが、文字コード 0xFFFFFFFF 以内なので個別に設定することもできる。

virtual font の仕様としてフォーマットの面では今回新規な部分はないものの、最大で文字コード 0xFFFFFFFF となる点は従来の TeX ファミリーで例がない。このため周辺ソフトウェアではこれに準拠した拡張が必要になる場合が多い。omfonts (ovp2ovf, ovf2ovp), dvips, dvipdfmx は対応を済ませてあり TeX Live 2025 では利用可能である。



紛らわしいが、1.3 節で述べたように、結合濁点・半濁点の合字処理のうち、

- 入力ファイルを一行読み込んでバッファに格納する段階で、
- かつ Unicode で単一の文字コードで表せるシーケンスがあった場合

には、従来から ptexenc ライブラリによる前処理によりトークン化以前の段階で合字処理が行われている (例えば、`ピ (U+30D2 U+309A)` は `ピ (U+30D4)` と同一の文字トークンとみなされる)。

一方、

- Unicode では文字合成でないと表せない仮名の濁点・半濁点^{*22}
- トークンを処理した結果として連続したシーケンスが現れてノード化される場合^{*23}

については、upTeX u1.35 で導入したノード化時の合成で対応する。

1.4.2 欧文 8bit 超 (upTeX u2.00, 実験的)

upTeX では従来、欧文文字として扱える文字コードの範囲は 8bit (TeX90 互換の 0-255) であったが、upTeX u2.00 では 0x2E7F 以下まで拡張した [22]。動作実績がまだ少なく関連フォントの整備等にも時間を要するため、今のところ実験的機能と位置づけている。仕様は以下の通りである。

- upTeX エンジンが扱える欧文文字の範囲は 0~0x2E7F である。文字コード 256 以上を含む文字フォントは Unicode U+0000..2E7F であることを想定する。
- `\kcatcode 14 (latin_ucs)` を新設する。UTF-8 の入力バッファで 8bit 文字コード列が

表 2 合成文字用拡張文字コード。upTeX u1.35 での対応状況。

種類	基底文字	modifier	合成文字用拡張文字コード	パラメータ
仮名 + 濁点	仮名	U+3099	0x22,0000~0x23,FFFF	
仮名 + 半濁点	仮名	U+309A	0x24,0000~0x25,E6E5	
絵文字 (国旗)	U+1F1E6..1F1FF	U+1F1E6..1F1FF	0x25,E6E6~0x25,FFFF	26 種 × 26 種
絵文字 (肌色)	絵文字	U+1F3FB..1F3FF	0x26,0000~0x2F,FFFF	Type-1-2~6
SVS	漢字, 記号など	U+FE00..FE0F	0x40,0000~0x7F,FFFF	VS1~VS16
IVS	漢字	U+E0100..E011F	0x80,0080~0xFF,FFFF	VS17~VS48

^{*22} 想定は、JIS X 0213 に含まれる「カ」などの半濁点や、変体仮名の濁点・半濁点である。

^{*23} `\kansuji`, `\Uchar` 等による文字トークン生成や、基底文字と結合濁点・半濁点を連続して `\char` で文字ノード生成する場合など。

正規の UTF-8 として U+2E7F 以下と読み取れて、かつそのコード値を含むブロックの `\kcatcode` が 14 となっているとき、Unicode の欧文であると解釈して符号位置 0～0x2E7F の欧文文字ノードを生成する。同時にその文字コードに相当する欧文文字の `\catcode` が付与される。このとき「欧文 Unicode 1 文字に対し 1 トークン」となる。

- 欧文用 8bit TFM に加えて欧文用 16bit OFM Level-0 が読み込める。符号位置 0～0x2E7F の欧文文字ノードはこれで扱う。和文文字は従来同様 JFM で扱う。
- 256～0x2E7F の欧文文字コードは、DVI ファイルにはそのままの文字コード (UCS-2 相当) で DVI 命令 `set2` で書き込む。
- U+2E80 以上の Unicode 文字は `upTeX` では欧文文字として扱わない。必要な場合は `virtual font` と `dviware` で処置^{*24}をする。
- `\catcode`, `\lccode`, `\uccode`, `\sfcode` の扱う欧文文字コードの最大値は 0x2E7F とする (`\kcatcode=14,15` 両方)。
- `\kcatcode=14` のとき、`\char`, `\chardef` は 0～0x2E7F の欧文文字が扱える。
- `ε-upTeX` では `\kcatcode=14` のとき、`\Uchar`, `\Ucharcat`, `\iffontchar` は 0～0x2E7F の欧文文字が扱える。
- `^^^abcd` の書式は、その 16bit コード (0xABCD) の欧文文字トークンを生成する (`\kcatcode` の設定に依らず常に欧文扱い。^^ab 書式と同様)。

関連ソフトウェアでは、`updvi` も欧文用 16bit OFM Level-0 への対応を済ませた。pTeX 系列に対応した `dviware` ではすでに OFM への対応が施されていることが多く、`dvipdfmx`, `dvips` のような主要な `dviware` でも `upTeX` の欧文文字コードの範囲拡張はすでにカバーされている。現時点で `upTeX` は欧文用 OFM Level-1 には未対応である。

1.5 文字コードの取得と指定

(u)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

和文文字の文字コードを数値で指定するには、内部コードで表現する必要がある。この方法

^{*24} 頻出する例に U+FB00..FB06 のリガチャ (fi, fi, fl 等) が挙げられる。TeX90 ではそれらを 8bit (0–255) の内部コードに割り当てて使用しているが、それと同様の取り扱いを維持する必要がある。

として、(u)pTeX では以下が利用できる。

- TeX82 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る（例えば、「 ` b 」と「 ` \b 」は同じ意味である）。従来 pTeX 系列では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かったが、TeX Live 2025 pTeX 4.1.2 以降可能になった [21]。
- 数値で直接表現するために、異なるエンコードから内部コードへの文字コード変換を行うプリミティブが用意されている。5.1 節を参照。

2 和文カテゴリーコードと TeX の入力プロセッサ

TeX の処理は以下のような段階を踏み、pTeX 系列でもこれは共通である。

- 入力を一行単位で読み込む（改行文字は CR, LF いずれも可）。
- (ptexenc によるエンコード変換)
- 入力プロセッサがカテゴリーコードに従って「順に」トークン化しつつ、その過程で
 - 展開プロセッサ：マクロやアクティブ文字など展開可能トークンの展開
 - 実行プロセッサ：代入やマクロ定義、出力リスト構築など展開不能トークンの実行も介入する^{*25}。
- ビジュアルプロセッサが受け取った出力リストを元に行分割・ページ分割などを行い、ページを DVI に出力する。

TeX82 では各文字に 0–15 のカテゴリーコード (`\catcode`) を割り当てており、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 系列においても同様であり、和文文字には欧文文字とは異なる専用のカテゴリーコード (`\kcatcode`) を割り当てることで拡張している。`\kcatcode` の仕様は pTeX と upTeX で異なるので、別々に説明する。

2.1 和文カテゴリーコード【pTeX の場合】

pTeX が欧文 TeX と大きく異なるのは以下の点である。

- 文字コードによって和文文字（2 バイト）と欧文文字（1 バイト）を区別する。
- 和文文字直後の改行は（欧文文字直後の改行と異なり）空白とみなさない。
- 和文文字には役割に応じて 16 (*kanji*), 17 (*kana*), 18 (*other_kchar*) の和文カテゴリーコードのいずれかを割り当てる。初期状態では、JIS の 1, 2, 7–15, 85–94 区の文字は 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。
- 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

^{*25} 例えば、実行プロセッサの結果によりカテゴリーコードが変更されれば、行内でも入力プロセッサの動作に影響を及ぼし、以降の展開・実行結果も変化する。ただし、順に一度トークン化されたもののカテゴリーコードは固定である。

ようになっている。欧文文字トークンが TeX82 と同様にカテゴリーコード c と文字コード s の組み合わせ ($256c + s$) で表現されているのとは対照的である。

和文カテゴリーコードの値は一応、16が「漢字」、17が「かな」、18が「その他の和文記号」を意図している。ただし区の中身を見れば分かる通り、pTeX では特に全角数字・アルファベット (3区) とギリシャ文字 (6区) も17であり、キリル文字 (7区) は18となっている。

和文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 (16と17はpTeXの内部処理においては全く等価で、18だけが異なる)。


- TeX82 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、pTeX 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^{*26}。
- 後で説明する `\jcharwidowpenalty` は、カテゴリーコードの値が 16, 17 の和文文字の前にはのみ挿入されるもので、値が 18 の和文文字の前には挿入されない。
- いずれにせよ、和文文字は決して“アクティブ”(欧文文字におけるカテゴリーコード 13 のような状態) にはならない。

従って、pTeX の既定ではコントロールワードに全角数字・アルファベット・ギリシャ文字を含めることができるが、キリル文字は不可 (コントロールシンボルのみ) であることがわかる (upTeX の既定では全て不可)。

和文カテゴリーコードを取得・設定するプリミティブが `\kcatcode` である。

▶ `\kcatcode <character code>={16-18}`

コミュニティ版 pTeX では、和文カテゴリーコード (`\kcatcode`) は DVI 中の上位バイトごと (すなわち、JIS コードでいう区ごと) に値が設定可能である^{*27}。

 `\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。

しかし、pTeX においては、`\kcatcode` を文書の処理途中で変更することは想定されていない。というのも、pTeX では (upTeX と異なり) 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随時算出されるためである。ただし、pTeX でも `\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^{*28}。

^{*26} 「\】」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 TeX Live 2019 の pTeX 3.8.2 で修正された [10]。

^{*27} 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。

^{*28} 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



例えば、pTeX では以下のコードで定義される `\X`、`\Y` で引数終端を示す「あ」にはカテゴリコードの情報は格納されないため、`\X` と `\Y` は全く同じ動作となる。一方、同じコードを `upTeX` で実行すると、`\X` と `\Y` は異なる動作となる。

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

2.2 和文カテゴリコード，和文文字と欧文文字の区別【upTeX の場合】

upTeX においては `\kcatcode` が pTeX から大きく仕様変更されている^{*29}：

- `\kcatcode` プリミティブに和文文字の役割の分類だけでなく、トークン列生成における和文文字と欧文文字の区別という機能も付与する。具体的には、`\kcatcode` に特別な値として `15 (not_cjk)` を設定すると、それは 8 ビット欧文文字のバイト列として扱われ、和文扱いされなくなる。
- 和文文字に可能なカテゴリコードとして `19 (hangul)` を追加する。カテゴリコード `19` の和文文字は、直後の改行を欧文同様に空白とみなすという点が特別であり、それ以外の点ではカテゴリコード `16, 17` の和文文字と同様である（例えば、コントロールワード中にも使用できる）。
- 和文文字トークンについてもカテゴリコード (`\kcatcode`) の情報を含む。欧文トークンを `catcode 4bit + charcode 8bit` で、和文トークンを `kcatcode 5bit + charcode 24bit` で表す。
- `\kcatcode` に `14 (latin_ucs)` と `20 (modifier)` が TeX Live 2025 で新設された。内容は 1.4 節で詳説した。

▶ `\kcatcode <character code>=(14–20)`

内部レガシーの upTeX では、pTeX と同様に JIS コードでいう区ごとに値が設定可能。内部 Unicode の upTeX では、和文カテゴリコード (`\kcatcode`) は概ね Unicode のブロックごと（※一部のブロックは分割してある）に値が設定可能である。

このように内部 Unicode の upTeX では `\kcatcode` の設定が Unicode のブロックごとであるので、pTeX の JIS コードの区ごとに設定したものと分類が必ずしも一致しない。各ブロックの初期値については `uptex-base` で配布している `01uptex_doc_utf8.txt` を参照のこと。



例えば長音記号「ー」は JIS コードで `"213C` であり、記号として 1 区に含まれるから pTeX での `\kcatcode` は `18` である（内部レガシーの upTeX でも同様）。一方 Unicode では `"30FC` であり、

るたびにカテゴリ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 [15]。


^{*29} upTeX の正式用語としては、和文文字トークン（和文トークン）は CJK 文字トークン（CJK トークン）と呼ぶべきだが、本文書では単純化のため pTeX と同じ用語を用いる。

カタカナと同じブロックに含まれるから内部 Unicode の `upTeX` での `\kcatcode` は 17 である。すなわち、`pTeX` では「ー」をコントロールワード中に含まれないが、`upTeX` では許される。

```
\def\黄マーカー{...}
% => pTeX では「\黄マ」という命令と「-カー」という終端
% => upTeX では「\黄マーカー」という命令が定義される
\newcommand{\黄マーカー}[1]{...}% => pTeX ではエラー, upTeX では通る
```

もちろんこの逆の例もあり、`pTeX` では全角数字とアルファベット (Halfwidth and Fullwidth Forms)、ギリシャ文字が 17 であったが、`upTeX` では (キリル文字と同じく) 18 としている。

なお 1.3 節で述べた通り、`upTeX` では内部コードによらず 0-127 の 7 ビット ASCII 文字集合は常に欧文文字トークンとして扱うこととしている。裏を返せば和文文字トークンで文字コードが 0-127 という状況は起こりえない^{*30}し、このブロックの `\kcatcode` を 15 以外に設定しても和文扱いはされない。

 例えば `upTeX` の既定 (内部 Unicode) から `\kcatcode`あ=15` を実行すると、以降の「あ」は `^^e3^^81^^82` を入力したとみなされる^{*31}。この状況で続けて `\kcatcode`あ=17` を実行しても、それは `\kcatcode`^^e3^^81^^82` の入力とみなされるので元には戻らない: 代わりに `\kcatcode\ucs"3042=17` と実行する必要がある。

既定ではこのように和文と欧文が `\kcatcode` の値 (16-20 か, 14-15 か) に従って区別されるが、以下のプリミティブによって変更可能である。

▶ `\enablecjktoken`

和文文字と欧文文字の区別について `\kcatcode` の設定に従い、15 以下なら欧文扱い、16 以上なら和文扱いにする。`upTeX` の既定はこの状態である。

▶ `\disablecjktoken`

全ての Unicode 文字を (`\kcatcode` にかかわらず) 欧文扱いにする (単なるバイト列として扱う)。入力に関して (`ptexenc` ライブラリによる前処理を除き) 8-bit 欧文 `TeX` と互換になる。

▶ `\forcecjktoken`

ASCII 文字以外の Unicode 文字を (`\kcatcode` にかかわらず) 強制的に和文扱いにする。このとき、`\kcatcode` が 15 の文字は「その他和文」(18) 扱いになる。

```
\forcecjktoken\kcatcode`西=15
\message{\the\kcatcode`@ % ==> 18
\ifcat 西@ T\else F\fi}% ==> T
```

2.3 pTeX の入力プロセッサ

先に述べた通りであるが、pTeX の入力プロセッサは TeX82 のそれを拡張したものになっている。図 1 は **pTeX 4.0.0 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：

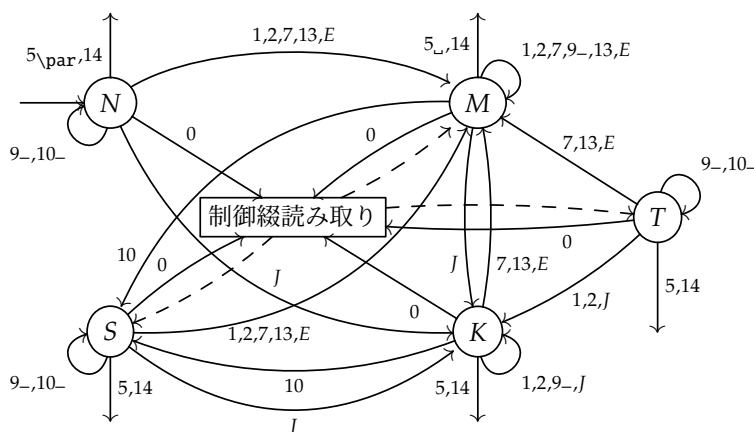
状態 *K* 和文文字の直後。状態 *M* との差異は改行でスペースを出力しないこと。

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。

図を見れば分かる通り、(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の改行が行えたほうが便利なためである。



“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

^{*30} 和文文字ノードとしては文字コード 0-127 も可能：3.1 節で出てくる `\kchar` で生成される。

^{*31} 内部 EUC-JP では `^^a4^^a2`, 内部 Shift-JIS では `^^82^^a0` になる。

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^{*32}、 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^{*33}
- 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。



従って、和文文字と行末の関係を例で示すと、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。

<p style="padding-left: 40px;"><code>\ptexlineendmode=0</code> → あいあい <code>{あいあ}</code> い</p>	<p style="padding-left: 40px;"><code>\ptexlineendmode=4</code> → あいあい <code>{あいあ}</code> い</p>
--	--

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。

<p style="padding-left: 40px;"><code>\ptexlineendmode=0</code> → あい <code>\def\漢{あ}{\漢}</code> い</p>	<p style="padding-left: 40px;"><code>\ptexlineendmode=1</code> → あい <code>\def\漢{あ}{\漢}</code> い</p>
--	--

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。

<p style="padding-left: 40px;"><code>\ptexlineendmode=0</code> → あい <code>\def\】{あ}\】</code> い</p>	<p style="padding-left: 40px;"><code>\ptexlineendmode=2</code> → あい <code>\def\】{あ}\】</code> い</p>
--	--

^{*32} 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。

^{*33} `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。

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。

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

なお、以前の pTeX の入力プロセッサの挙動は


- pTeX 3.8.1 以前 (アスキー版も同じ) の挙動は `\ptexlineendmode=3` ($(x, y, z) = (1, 1, 0)$)
 - pTeX 3.8.2 以降 3.10.0 までの挙動は `\ptexlineendmode=1` ($(x, y, z) = (1, 0, 0)$)
- としてそれぞれ再現できる [16].

2.4 和文文字の文字列化の挙動

`\meaning` や `\string` などの文字トークン列生成 (いわゆる “`\the-` 文字列化”) については以下の通りである。

- 欧文文字は TeX82 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 の欧文文字トークンになるが、和文文字は和文文字トークンになる。その和文カテゴリーコードは pTeX では随時算出により、upTeX では「その時の `\kcatcode` の値」(ただしそれが 15 の場合は 18) になる。
- 制御綴名の文字列化において「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*34。
- 和文文字はそのカテゴリーコードによらず、`\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 [17].

 ϵ -TeX 由来の `\scantokens` (ϵ -(u)pTeX で利用可能) は和文文字と欧文文字の区別が維持されない。例えば `\forcecjktoken` の状態で欧文バイト列を `\scantokens` すると和文になる*35。

*34 「`\]`」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 3.8.1 でこの問題は修正された [10].

*35 `\scantokens` は、引数のトークン列を脱トークン化して、改めて字句解析対象となる文字列として読み込む展開可能プリミティブである。基本的にはトークン列を一旦 (仮想的な) ファイルに書き出してそれをすぐに読み込んだのと同じになるが、「引数を文字列化→buffer に書き戻し→それを入力として扱う」という動作手順



TeX Live 2021 では、ファイルのエンコード 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 和文文字の出力

TeX の実行プロセッサは展開不能トークンを実行し、組版結果を出力リストとして構築していく。このとき、文字トークンは**文字ノード**に変換される。また、文字コードの数値から文字ノードを生成できる `\char <character code>` や、その短縮形 *<control sequence>* を定義する `\chardef <control sequence>=<character code>` も同様に文字ノードを生成する。この仕組みは pTeX 系列でも同じであり、ただし日本語の文字を出力するために和文文字ノードを使用する。

欧文文字ノードは 1 つの *char_node* にフォントと文字コードの情報を格納している。和文文字ノードは内部的には連続する 2 つの *char_node* を用いて

- 1 つ目にフォントと文字タイプの情報
- 2 つ目に文字コードの情報（upTeX ではさらに `\kcatcode` の情報も）

を格納している^{*36}。

3.1 文字ノードの生成

実際に文字ノードが生成する状況は以下である。

- 欧文文字トークンは欧文フォントを使用した**欧文文字ノード**になる。
- 和文文字トークンは和文フォントを使用した**和文文字ノード**になる。
- `\char`, `\chardef` は文字コード 0–255 からは**欧文文字ノード**を、それ以外の文字コードからは**和文文字ノード**を生成する^{*37}。

ここで、upTeX では `\char`, `\chardef` の挙動が問題になる。これらは文字コードの数値か

のうち `buffer` に書き戻す時点で和欧文の区別が失われる。実際に一旦ファイルに書いて読み込む場合は、欧文バイトは ^^ 形式で書き出されて欧文のままになるので、それとは異なる。

^{*36} ここで、pTeX では（和文文字トークンと同様）和文文字ノードに `\kcatcode` の情報が保存されず、必要に応じて（=後述の `\jcharwidowpenalty` 挿入時）再計算される。upTeX では（こちらも和文文字トークンと同様）和文文字ノードに `\kcatcode` の情報が保存される（=後述の `\jcharwidowpenalty` 挿入のため）。

^{*37} `\char`/`\chardef` は文字トークンを経由しないので、upTeX においても `\kcatcode` は基本的に関係ない。ただし upTeX u2.00 以降では、文字コード 256–0x2E7F についても、`\kcatcode=14` に設定されている場合に限って**欧文文字ノード**を生成する。1.4 節を参照のこと

ら直接文字ノードを生成するが、pTeX および upTeX の内部レガシーでは欧文文字と和文文字の有効な文字コードが重ならないので、生成すべき文字ノードの種類が文字コードによって一意に定まっていた。ところが upTeX の内部 Unicode では、0–255 の値が欧文文字と和文文字のいずれの文字コードとしても有効であり、`\char`、`\chardef` はそのうち欧文文字ノードを選択していることを意味する。文字トークンに限っても 128–255 の値はいずれも有効であるのに、このままでは、和文文字トークンから出力できる文字コード 128–255 の和文文字ノードを、文字コードの数値からは直接出力できないことになる。

そこで、upTeX では新たに「必ず和文文字ノードを生成するプリミティブ」を追加している。必要に駆られた 128–255 のみならず、便宜のため文字コード 0–127 に対しても機能する。また内部コードによらず利用可能である。

▶ `\kchar` *<character code>* (upTeX のみ)

`\char` の和文固定版で、文字コード *<character code>* から和文文字ノードを生成する。

▶ `\kchardef` *<control sequence>=**<character code>* (upTeX のみ)

`\chardef` の和文固定版で、`\kchar` *<character code>* の短縮形として *<control sequence>* を定義する。

3.2 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 系列も同様で、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 `\tfont`, `\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）。書式については後述。
- `\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、pTeX 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、pTeX 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^{*38}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。

なお、pTeX 4.1.0 以降では `\font`, `\jfont`, `\tfont` のいずれもフォントを定義する際の書式が拡張されている。

- TeX82 書式：`\font<control sequence>⟨equals⟩⟨file name⟩⟨at clause⟩`
- (u)pTeX 書式：`\font [(in spec)]⟨control sequence⟩⟨equals⟩⟨file name⟩⟨at clause⟩`

なお、ここで

- `⟨in spec⟩` → `in⟨encoding⟩`
- `⟨at clause⟩` → `at⟨dimen⟩ | scaled⟨number⟩ | ⟨optional spaces⟩`

であり、追加された `⟨in spec⟩` は「和文フォントを定義した JFM が JIS エンコードであるか Unicode エンコードであるか」を明示的に指定する場合に用いる。指定可能な `⟨encoding⟩` は `jis`, `ucs` に限られる。この書式を用いることで、例えば

- pTeX（内部コード `euc` または `sjis`）でも `\font in ucs \myfontA=upjisr-h` により upTeX 用 JFM である `upjisr-h.tfm` (UCS-encoded) を、
- upTeX（内部コード `uptex` すなわち Unicode）でも `\font in jis \myfontB=min10` により pTeX 用 JFM である `min10.tfm` (JIS-encoded) を、

それぞれ（エンジン内部で JIS ⇔ Unicode 変換を行いながら）読み込んで正しくグルー・カーンを挿入できるし、DVI への出力時にも（逆変換により）指定されたエンコードで文字を出力する^{*39}。なお、`⟨in spec⟩` のスキャン時には展開を行わない。これは後続の `⟨control sequence⟩` が未定義のときに先にエラーを発生のを防ぐためである。



pTeX で `\font in ucs ...` により upTeX 用 JFM を読み込んでも、JIS X 0208 外の文字が出力できるわけではない。また、upTeX で

^{*38} 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。

^{*39} この新書式を導入した目的は、将来 upTeX の内部コードが Unicode 固定になった場合に字幅やグルー・カーンが定義された pTeX 用 JFM/VF セットを upTeX 用に用意しな小さなくても済むようにするためである [19]。

```
\kcatcode"D8=16\relax % JIS範囲外の文字(Latin-1 Supplement)を和文扱いに
\font in jis\x=jis \x <文字> % 例えば U+00D8 など
```

のように JIS エンコードのフォント指定下で範囲外の文字を使うと、DVI 出力中 (`\shipout` 時) に

```
Character <文字> ("D8) cannot be typeset in JIS-encoded JFM jis,
so I use .notdef glyph instead.
```

という警告^{*40}が発生し、DVI には豆腐 (`set2 0`) が書かれる。



なお、TeX82 における読込済フォントの判定 (「同じ TFM ファイル名」かつ「同じサイズ」) は (u)pTeX でも変更していないため、エンコード無指定・`in jis` 指定・`in ucs` 指定だけを変えて複数回読み込みもうとしても、新しいフォント識別子 (font identifier) は発行されないし、最初のエンコードが常に使われる。

```
#!/ptex
\font\xA=min10
\font in jis\xB=min10 % => min10 は無指定時のエンコードのまま
\font in ucs\xC=min10 % => min10 は無指定時のエンコードのまま

#!/ptex
\font in ucs\yA=umin10
\font\yB=umin10 % => umin10 は Unicode のまま
\font in jis\yC=umin10 % => umin10 は Unicode のまま
```



欧文フォントに対する `<in spec>` は意味を持たないので、単に無視される。また `\font`、`\jfont`、`\tfont` の直後が `in` でないか `<encoding>` が `jis`、`ucs` 以外の場合は

```
! Missing control sequence inserted.
```

というエラーが発生する (TeX82 と同じ挙動)。upTeX u2.00 では、欧文文字を Unicode 1 文字に対し 1 トークンとして扱う機能が追加された (1.4.2 節) が、それに伴い `<in spec>` の書式も拡張された。指定可能な `<encoding>` は `t1`、`ts1`、`ly1`、`t5`、`l7x`、`t2a`、`t2b`、`t2c`、`lgr` である。レガシーエンコーディングの欧文 TFM に対してこの指定をすると、内部コードが Unicode である欧文文字トークンとレガシーエンコーディングの TFM の情報との相互変換が機能する。

▶ `\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 pTeX (p3.8.3) で利用可能である。

これにより、例えば `\HOGE` が和文フォントか否かは

```
\ifjfont\HOGE
  (和文の横組用フォント)
\else\iftfont\HOGE
  (和文の縦組用フォント)
\else
```

^{*40} この警告も `\tracinglostchars` に従う。Missing character 警告と違ってノードは破棄されないなので `lost` という名称は微妙だが…。また、JIS 範囲外の警告が発生するタイミングは Missing character 警告 (ノード生成失敗時に発生) とは異なるので、`\shipout` 時の `\tracinglostchars` の値に依ることになる。

(欧文フォント)

`\fi`

として判定できる.



上述の通り, 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 pTeX には存在しなかった.



ちなみに, ϵ -pTeX, ϵ -upTeX には `\iffontchar` プリミティブが存在する. しかし, $\langle font \rangle$ が和文フォントか否かを判定するために `\iffontchar $\langle font \rangle$ 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.

▶ `\jfam= $\langle number \rangle$`

現在の和文数式フォントファミリの番号を格納する^{*41}. 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.

▶ `\ptextracingfonts (integer)`

`\tracingoutput=1` のときに意味を持つパラメータで, これは元来 `\shipout` 時にログに出るものであるから, 「`\shipout` 時点での値」によって以下の情報を表示する. pTeX 4.1.0 で導入された.

- 値を 1 以上に設定すると, pdfTeX の `\pdftracingfonts` と同じ書式でフォント名とサイズを表示する (font expansion には非対応).
- 値を 2 以上に設定すると, 追加で (u)pTeX 特有の以下の情報を表示する.
 - 和文フォント (JFM) の横組 (/YOKO)・縦組 (/TATE) の区別
 - 明示的なエンコード指定 (in jis \rightarrow +JIS / in ucs \rightarrow +Unicode)

書式は以下のようになる.

- 欧文フォントの場合^{*42} \rightarrow ファイル名@サイズ
- 和文フォントの場合^{*43} \rightarrow ファイル名@サイズ/組方向+エンコード

^{*41} 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.

^{*42} サイズが TFM デザインサイズと同じ場合は @サイズ が省略される.

^{*43} サイズについては欧文フォントと同様. 和文フォントでは /YOKO 又は /TATE のいずれか一方が常に表示され

▶ \ptexfontname

pTeX 4.1.0 で導入された。欧文フォントに対しては `\fontname` と類似だが、フォントサイズの表示が `..._at_...pt` ではなく `...@...pt` の書式となる。また、和文フォントに対しては追加情報として以下も表示する。

- 和文フォント (JFM) の場合は横組・縦組の情報を表示
- 明示的に `in jis/in ucs` が指定された場合に限りエンコードを表示

書式は `\ptextracingfonts` と同じであるが、その値は `\ptexfontname` の出力に影響しない。ここでは例を示そう。

- `\font\x=cmr10 at 7pt → cmr10@7.0pt`
- `\font\x=nmin10 → nmin10/YOKO`
- `\font\x=min10 at 8pt → min10@8.0pt/YOKO`
- `\font in jis \x=ngoth10 at 6pt → ngoth10@6.0pt/YOKO+JIS`
- `\font in ucs \x=utgoth10 → utgoth10/TATE+Unicode`

これにより「そのフォントが JIS コードと Unicode のどちらで DVI 出力されるか」を知ることができるし、TFM ファイル名には現れることがない / が含まれるかどうかで「和文フォントかどうか」を判定することもできる。

4 pTeX 系列の組版処理

和文文字を単に出力できるだけでは、日本語の組版としてまともにならない。pTeX 系列は、禁則を考慮した行分割処理、和文文字と欧文文字のスペーシングといった日本語特有の組版機能を有している。

4.1 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^{*44}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

る。またエンコード無指定の場合は +エンコード が省略される。

^{*44} 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^{*45}。しかし、上述の通り、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 1,024 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 1,024 文字までである^{*46}。

^{*45} 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]。

^{*46} 最大 1,024 文字となったのは TeX Live 2025 (r73803) 以降。なお、TeX Live 2022 までは 256 文字までの領域しかなく、TeX Live 2023 (r65246-65248) で 1,024 文字に拡張しようとしたが不完全で動作していなかったの

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] :

- 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^{*47}.

▶ `\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.

4.2 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 (欧文 TFM のカーニングや合字処理と同様)
- 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.

で, TeX Live 2025 (r73803) で再修正した.

^{*47} 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.



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：

- 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るが、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^{*48}。

▶ `\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：

^{*48} 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。

- 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 1,024 文字分のテーブルに格納されている^{*49}。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，このノードの形での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。

^{*49} 最大 1,024 文字となったのは TeX Live 2025 (r73803) 以降。なお，TeX Live 2022 までは 256 文字分のテーブルしかなく，TeX Live 2023 (r65246–65248) で 1,024 文字に拡張しようとしたが不完全で動作していなかったため，TeX Live 2025 (r73803) で再修正した。

▶ `\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 3.8.2 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, 13]。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} ( あ (
```



pL^AT_EX 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 L^AT_EX 自体が ϵ -TeX 拡張を要求している)。

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

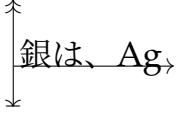

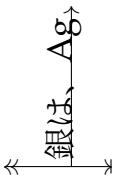

▶ `\disinhibitglue`

`\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 3.8.2 で新しく追加された。

4.3 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

表3 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				

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**^{*50}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3）。

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^{*51}。

^{*50} 下から上 (Down to Up) であろう。

^{*51} 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

- 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^{*52}.
 - 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^{*53}。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 3 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>`, `\ifdbox <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

が空」か正しく判断できない。

^{*52} 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。

^{*53} 数式ディレクションか否かは異なっても良い。

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

4.4 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 3 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 3.7^{*54}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`

^{*54} TeX Live 2016, 厳密には 2016-03-05 のコミット (r39938)。

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:

```
\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^{*55}。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。

5 その他の補助機能

5.1 文字コード変換, 漢数字

先述 (1.5 節) の通り, 和文文字の文字コードを数値で指定するには内部コードで表現する必要があるが, それは EUC-JP/Shift-JIS/Unicode と様々である。異なるエンコード間でも同じ文字を数値で直接表現できるように, 文字コード変換を行う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^{*56}。これらのプリミティブも `ptexenc` を使用しているので, Unicode とレガシーエンコードの間の変換では表 1 に従う。

▶ `\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。

^{*55} `\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)。

^{*56} 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。

たとえば、`\char\jis"346E`、`\char\eut"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 `upTeX` で実装されていたが、
`pTeX 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。`pTeX 3.10.0` で追加した。

▶ `\tojis <number>`

内部コードから JIS コードへの変換を行う。`pTeX 4.1.0` で追加した。

文字コードとして不正な値を与えても “! Bad character code (...)” エラーは発生
しない (下記参照)。なお、JIS コード、EUC コード、Shift-JIS コードについて扱えるのは
JIS X 0208 の範囲に限られる (1 節でも述べた通り。すなわち JIS X 0213 には非対応)。

 `pTeX 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：

- 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` 以降では、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^{*57} …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。(戻り値を統一)

▶ `\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=\eut"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)。

^{*57} 内部 `eut` における `\eut`、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。



`\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^{*58}。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 3.8.3 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`



以上のプリミティブ (`\kansujichar` を除くすべて) 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 (TeX82 の `\number`, `\romannumeral` と同様)。以下の例は内部コード EUC-JP の場合。

```
\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
```

- `\kuten, \jis, \euc, \sjis, \ucs, \toucs, \tojis` → 変換結果をカテゴリーコード 12 の欧文文字トークン列で返す。
- `\kansuji` → 変換結果を和文文字トークン列で返す^{*59}。

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、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。

^{*58} 内部 Unicode の 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ノ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。

^{*59} 例外的に、内部 Unicode の 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カテゴリーコード 12 の欧文文字トークンになる [8]。また、upTeX では生成する和文文字トークンにその時点での和文カテゴリーコードが与えられるが、もし `\kcatcode` が 15 ならば 18 扱いになる (`\forcecjktoken` と同様)。

5.2 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，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 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 函数に由来する。

5.3 バージョン番号

コミュニティ版の pTeX 系列独自で，TeX Live 2018 以降にバージョン番号を返す機能を (ε-)upTeX に追加した。

▶ `\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 3.8.0 で導入された。

この追加と同時に，(ε-)upTeX では以下が追加されている。

▶ `\uptexversion`, `\uptexrevision` (upTeX のみ)

upTeX のバージョン番号は `ux.y` 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。 `\uptexversion` は `x` の値を内部整数で返し， `\uptexrevision` はその後ろの「.y」

を文字列で返す.

さらに ε -(u)pTeX には `\epTeXversion` プリミティブがある (`./eptexdoc.pdf` を参照).

この文書は ε -upTeX で処理しているので, バージョン番号全体を表示すると

This is e-upTeX p4.1.2-u2.01-250202-2.6

となる. 最後の「-2.6」は ε -TeX のバージョン番号である.

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。

6 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 (...).
```

というエラーが発生する (... はその文字コード)。

 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
```

7 和文に対応したプリミティブ

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

先に 3 節でも述べた通り、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`

`\tfont` については 3.2 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) からも値を取得できる。

▶ `\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^{*60}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、その文字コードはオリジナルの TeX では 0–255 のみが許されるが、pTeX 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。



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。

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、upTeX 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 [14]。

^{*60} 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ればオリジナルの 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 [20] 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 [20] 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)。upTeX u1.35 以降では、合成文字用拡張文字コードも用いられる (1.4.1 節)。
- 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] t-tk, 「[upTeX+dvipdfmx] 異体字セレクト、Unicode 合成文字」, 2018/01/28,
<https://github.com/texjporg/tex-jp-build/issues/46>
- [12] 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>
- [13] aminophen, 「TeX Live 2019 での \inhibitglue の挙動変更【予定】」, 2019/02/06,
<https://okumuralab.org/tex/mod/forum/discuss.php?d=2566>
- [14] aminophen, 「upTeX の \if と \ifcat」, 2019/01/17,
<https://github.com/texjporg/tex-jp-build/issues/68>
- [15] 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>
- [16] 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>
- [17] 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>
- [18] t-tk, 「[ptexenc] 入力ファイルの文字コードの自動判定」, 2022/06/05,
<https://github.com/texjporg/tex-jp-build/issues/142>
- [19] aminophen, 「[upTeX] JIS-encoded TFM」, 2022/10/15,
<https://github.com/texjporg/tex-jp-build/issues/149>
- [20] TUG DVI Standards Working Group, *The DVI Driver Standard, Level 0*.
<https://ctan.org/pkg/dvistd>
- [21] h-kitagawa, 「[XeTeX] BMP 超のコントロールシンボル」, 2024/02/25,
<https://github.com/texjporg/tex-jp-build/issues/167>
- [22] t-tk, 「[upTeX] ofm 読み込みと符号位置 256 以上の欧文文字トークン・ノード」,
2024/08/18,
<https://github.com/texjporg/tex-jp-build/issues/170>
- [23] Ideographic Variation Database, 2022/09/13,
<https://www.unicode.org/ivd/data/2022-09-13/>
- [24] Emoji Sequence Data, 2024/05/01,
<https://www.unicode.org/Public/emoji/16.0/emoji-sequences.txt>

索引

Symbols	
<code>\accent</code>	41
<code>\autospacing</code>	30
<code>\autoxspacing</code>	30
<code>\char</code>	40
<code>\chardef</code>	40
<code>\disablecjktoken</code>	17
<code>\disinhibitglue</code>	31
<code>\dtou</code>	32
<code>\enablecjktoken</code>	17
<code>\euc</code>	35
<code>\font</code>	40
<code>\fontdimen</code>	40
<code>\fontname</code>	40
<code>\forcecjktoken</code>	17
<code>\if</code>	41
<code>\ifcat</code>	41
<code>\ifdbx</code>	33
<code>\ifddir</code>	33
<code>\ifjfont</code>	24
<code>\ifmbox</code>	33
<code>\ifmdir</code>	33
<code>\iftbox</code>	33
<code>\iftdir</code>	33
<code>\iftfont</code>	24
<code>\ifybox</code>	33
<code>\ifydir</code>	33
<code>\inhibitglue</code>	31
<code>\inhibitxspcode</code>	29
<code>\jcharwidowpenalty</code>	28
<code>\jfam</code>	25
<code>\jfont</code>	22
<code>\jis</code>	35
<code>\kanjiskip</code>	28
<code>\kansuji</code>	36
<code>\kansujichar</code>	36
<code>\kcatcode</code>	15, 16
<code>\kchar</code>	22
<code>\kchardef</code>	22
<code>\kuten</code>	35
<code>\noautospacing</code>	30
<code>\noautoxspacing</code>	30
<code>\postbreakpenalty</code>	27
<code>\prebreakpenalty</code>	27
<code>\ptexfontname</code>	26
<code>\ptexlineendmode</code>	19
<code>\ptexminorversion</code>	38
<code>\ptexrevision</code>	38
<code>\ptextracingfonts</code>	25
<code>\ptexversion</code>	38
<code>\scriptbaselineshiftfactor</code>	34
<code>\scriptscriptbaselineshiftfactor</code>	34
<code>\showmode</code>	30
<code>\sjis</code>	35
<code>\tate</code>	32
<code>\tbaselineshift</code>	34
<code>\textbaselineshiftfactor</code>	34
<code>\tfont</code>	22
<code>\tojis</code>	36
<code>\toucs</code>	36
<code>\ucs</code>	36
<code>\uptexrevision</code>	38
<code>\uptexversion</code>	38
<code>\xkanjiskip</code>	29
<code>\xspcode</code>	29
<code>\ybaselineshift</code>	34
<code>\yoko</code>	32
H	
H	38
Q	
Q	38

