

# $\epsilon$ -pT<sub>E</sub>X について

北川 弘典\*

version 180226, 2018 年 3 月 9 日

## 目次

1	はじめに	2
2	$\epsilon$ -T <sub>E</sub> X 拡張について	2
3	$\Omega$ 由来の機能 (旧名称 : FAM256 パッチ)	5
4	pdfT <sub>E</sub> X 由来の機能	8
5	バージョン番号	10
6	<code>\lastnodechar</code> プリミティブ	10
7	<code>\lastnodesubtype</code> プリミティブ	12
8	<code>\epTeXinputencoding</code> プリミティブ	13

---

\* <http://osdn.jp/projects/ptex/wiki/>, e-mail: [h\\_kitagawa2001@yahoo.co.jp](mailto:h_kitagawa2001@yahoo.co.jp)

## 1 はじめに

$\varepsilon$ -pTeX は、東京大学理学部数学科 3 年生対象の 2007 年度の授業「計算数学 II」<sup>\*1</sup>において北川が作成したプログラムである。もともとは pTeX 3.1.10 を基盤として、 $\varepsilon$ -TeX 2.2 相当の機能や 10 進 21 桁の浮動小数点演算を追加したものであったが、今では次の点が変わっている。

- TeX Live 2011 に取り込まれるにあたり、 $\varepsilon$ -TeX をベースにして、その上に pTeX 拡張やその他追加機能を載せる方針へと変更された。
- 浮動小数点演算の機能は 090927 版 (2009/9) から削除されている<sup>\*2</sup>。

製作の動機や作業過程などについては、詳しくは [1] を参照して欲しいけれども、大雑把に言うとも、動機は以下のように要約できる。

- pTeX は、TeX が持っている「レジスタ 1 種類につき 256 個まで」という制限をひきずっており、現状でも非常に多数のパッケージを読み込ませたりすると制限にぶち当たってしまう。
- 一方、 $\varepsilon$ -TeX 拡張ではこれが「レジスタ 1 種類につき 32768 個まで」と緩和されており、欧文で標準となっている pdfTeX やその後継の LuaTeX、及び XeTeX でも  $\varepsilon$ -TeX の機能が取り込まれている。
- そうすると、pTeX だけが制限をレジスタ制限を引きずっているのは世界から取り残されることになるのではないか。

## 2 $\varepsilon$ -TeX 拡張について

前に述べたように、 $\varepsilon$ -TeX は TeX の拡張の一つである。 $\varepsilon$ -TeX のマニュアル [15] には、開発目的が以下のように述べられている。

The  $\mathcal{N}\mathcal{T}\mathcal{S}$  project intends to develop an ‘New Typesetting System’ ( $\mathcal{N}\mathcal{T}\mathcal{S}$ ) that will eventually replace today’s TeX3. The  $\mathcal{N}\mathcal{T}\mathcal{S}$  program will include many features missing in TeX, but there will also exist a mode of operation that is 100% compatible with TeX3. It will, necessarily, require quite some time to develop  $\mathcal{N}\mathcal{T}\mathcal{S}$  to maturity and make it widely available.

Meanwhile  $\varepsilon$ -TeX intends to fill the gap between TeX3 and the future  $\mathcal{N}\mathcal{T}\mathcal{S}$ . It consists of a series of features extending the capabilities of TeX3.

$\mathcal{N}\mathcal{T}\mathcal{S}$  がどうなったのか僕は知らない。しかし、少なくとも  $\varepsilon$ -TeX 拡張自体は実用的な物であり、そのせいか  $\aleph$  (Aleph), pdfTeX, XeTeX などの他の拡張にもマージされており、ほとん

---

<sup>\*1</sup> <http://ks.ms.u-tokyo.ac.jp/>.

<sup>\*2</sup> TeX ソース中で浮動小数点演算を行う手段としては、例えば LaTeX3 の機能 (l3fp) や、xint パッケージバンドルがあるので、そちらを利用して欲しい。

どの人が  $\epsilon$ -T<sub>E</sub>X 拡張を使うことができるようになっている\*<sup>3</sup>.

$\epsilon$ -T<sub>E</sub>X 拡張で追加される機能について、詳しくは [15] を参照して欲しいが、[1] 中の 4.2 節「 $\epsilon$ -T<sub>E</sub>X の機能」から一部改変して引用する。

$\epsilon$ -T<sub>E</sub>X には Compatibility mode と Extended mode の 2 つが存在し、前者では  $\epsilon$ -T<sub>E</sub>X 特有の拡張は無効になるのでつまらない。後者がおもしろい。

拡張機能を使うにはファイル名を渡すときに \* をつけるかコマンドラインオプションとして `-etex` スイッチをつければいいが、 $\epsilon$ -T<sub>E</sub>X 拡張に関わる追加マクロは当然ながらそれだけでは駄目である。「plain マクロ for  $\epsilon$ -T<sub>E</sub>X」(`etex.fmt` というのが一番マシかな) では自動的に追加マクロである `etex.src` が呼ばれる。L<sup>A</sup>T<sub>E</sub>X 下ではちょうど `etex.src` に対応した `etex` パッケージを読み込む必要がある。

### ■レジスタの増加

最初に述べたように、T<sub>E</sub>X では 6 種類のレジスタが各 256 個ずつ利用できる。それぞれのレジスタには `\dimen75` などのように 0 ~ 255 の番号で指定できる他、予め別名の定義をしておけばそれによって指定することもできる。これらのいくつかは特殊な用途に用いられる（例えば `\count0` はページ番号などのように）ことになっているので、さらに `user` が使えるレジスタは減少する。

$\epsilon$ -T<sub>E</sub>X では、追加のレジスタとして番号で言うと 256 ~ 32767 が使用できるようになった。上の pdf によると最初の 0 ~ 255 と違って若干の制限はあるようだが、それは些細な話である。追加された（各種類あたり） $32768 - 256 = 32512$  個のレジスタは、メモリの効率を重視するため `sparse register` として、つまり、必要な時に始めてツリー構造の中で確保されるようになっている。

### ■式が使用可能に

T<sub>E</sub>X における数量の計算は充実しているとはいえない。例えば、

$$\backslash\dimen123 \leftarrow (\backslash\dimen42 + \backslash@tempdima)/2$$

という計算を元々の T<sub>E</sub>X で書こうとすると、

```
\dimen123=\dimen42
\advance\dimen123by\@tempdima
\dimen123=0.5\@tempdima
```

のように書かないといけない。代入、加算代入、乗算代入、除算代入ぐらいしか演算が用意されていない状態になっている（上のコードのように、 $d_2 += 0.8d_1$  というような定数倍を冠することは平気）。

$\epsilon$ -T<sub>E</sub>X では、そのレジスタの演算に、他のプログラミング言語で使われているような数式の表現が使えるようになった。上の PDF では実例として

$$\backslash\ifdim \backslash\dimexpr (2\text{pt}-5\text{pt})*\backslash\dimexpr 3-3*13/5\relax + 34\text{pt}/2<\backslash\wd20$$

---

\*<sup>3</sup> L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> 2017-01-01 からは、L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> のフォーマット作成段階から  $\epsilon$ -T<sub>E</sub>X 拡張が必須となった ([18])。

が書かれている。これは,

$$32\text{pt} = (2\text{pt} - 5\text{pt})(3 - \text{div}(3 \cdot 13, 5)) + \frac{34\text{pt}}{2} < \text{\box20 の幅}$$

が真か偽かを判定していることになる。

### ■ `\middle primitive`

$\text{T}_{\text{E}}\text{X}$  に `\left`, `\right` という primitive があり, それを使えば括弧の大きさが自動調整されるのはよく知られている。 $\varepsilon\text{-T}_{\text{E}}\text{X}$  では, さらに `\middle primitive` が追加された。

具体例を述べる。

$$\left\{ n + \frac{1}{2} \mid n \in \omega \right\} \left\{ n + \frac{1}{2} \mid n \in \omega \right\}$$

これは以下の source で出力したものである：

```
\def\set#1#2{\setbox0=\hbox{\displaystyle #1,#2$}%
\left\{\, \vphantom{\copy0}\#1 \,\right|!\left.\, \,
\vphantom{\copy0}\#2 \,\right\}
\def\eset#1#2{\left\{\, \, #1 \,\middle|\, \, #2 \,\right\}
\[\set{n+\frac{1}{2}}{n\in \omega} \eset{n+\frac{1}{2}}{n\in \omega} \]
```

両方とも集合の表記を行うコマンドである。 $\text{T}_{\text{E}}\text{X}$  流の `\set` では 2 つの `\left`, `\right` の組で実現させなければならず, そのために | の左側と右側に入る式の最大寸法を測定するという面倒な方法を使っている。その上, この定義では `\textstyle` 以下の数式 (文中数式とか) ではそのまま使えず, それにも対応させようとすると面倒になる。一方,  $\varepsilon\text{-T}_{\text{E}}\text{X}$  流の `\eset` では, 何も考えずに `\left`, `\middle`, `\right` だけで実現できる。

### ■ $\text{T}_{\text{E}}\text{X} \rightarrow \text{X}_{\text{E}}\text{T}$ ( $\text{TeX} \rightarrow \text{XeT}$ )

left-to-right と right-to-left を混植できるという機能であるらしい。ヘブライ語あたりの組版に使えるらしいが, よく知らない。ここでの RtoL は LtoR に組んだものを逆順にしているだけの気がする。

とりあえず一目につきそうな拡張機能といったらこれぐらいだろうか。他にも tracing 機能や条件判断文の強化などあるが, そちら辺はパツとしないのでここで紹介するのは省略することにしよう。

$\varepsilon\text{-pT}_{\text{E}}\text{X}$  ではここに述べた代表的な機能を含め, ほとんどすべての機能を実装しているつもりである\*4。ただ,  $\text{T}_{\text{E}}\text{X} \rightarrow \text{X}_{\text{E}}\text{T}$  を和文で使うと約物の位置がずれたり空白がおかしかったりするけれども, その修正は大変に思えるし, 苦労して実装する意味があるのか疑問なので放置している。

$\text{pT}_{\text{E}}\text{X}$  拡張では,  $\text{T}_{\text{E}}\text{X}$  と比較して `dir_node` と `disp_node` という 2 種類のノードが追加された。前者は, 現在のリストの中に違う組方向の box を挿入する際に寸法を補正するために作

---

\*4 さらに, レジスタの個数については, 各種類 65536 個まで使えるようになっている (次節参照)。

られ、`\hbox` や `\vbox` のコンテナとなっている。また後者は、欧文文字のベースライン補正のために使われる。

$\epsilon$ -pTeX 110102 まではこれらのノードも `\lastnodetype` の値として出力させるようにした。しかし、両者ともに  $\epsilon$ -pTeX が自動的に挿入する（ユーザーが意識する必要はない）ノードであることから、 $\epsilon$ -pTeX 110227 以降では `dir_node` と `disp_node` は `\lastnodetype` の対象とする「最後のノード」とはならないようにしている\*5。

-1: none (empty list)	5: mark node	11: glue node
0: char node	6: adjust node	12: kern node
1: hlist node	7: ligature node	13: penalty node
2: vlist node	8: disc node	14: unset node
3: rule node	9: whatsit node	15: math mode nodes
4: ins node	10: math node	

`\currentifttype` における条件判断文とそれを表す数字との対応は、以下のようになっている。21–28 が、pTeX 拡張で追加された条件判断文に対応する。29 の `\ifpdfprimitive` は pdfTeX 由来のプリミティブ（後述）である。

1: <code>\if</code>	11: <code>\ifhbox</code>	21: <code>\iftdir</code>
2: <code>\ifcat</code>	12: <code>\ifvbox</code>	22: <code>\ifydir</code>
3: <code>\ifnum</code>	13: <code>\ifx</code>	23: <code>\ifddir</code>
4: <code>\ifdim</code>	14: <code>\ifeof</code>	24: <code>\ifmdir</code>
5: <code>\ifodd</code>	15: <code>\iftrue</code>	25: <code>\iftbox</code>
6: <code>\ifvmode</code>	16: <code>\iffalse</code>	26: <code>\ifybox</code>
7: <code>\ifhmode</code>	17: <code>\ifcase</code>	27: <code>\ifdbbox</code>
8: <code>\ifmmode</code>	18: <code>\ifdefined</code>	28: <code>\ifmbox</code>
9: <code>\ifinner</code>	19: <code>\ifcsname</code>	29: <code>\ifpdfprimitive</code>
10: <code>\ifvoid</code>	20: <code>\iffontchar</code>	

### 3 Ω 由来の機能（旧名称：FAM256 パッチ）

$\epsilon$ -pTeX には、掲示板 TeX Q & A の山本氏の書き込み [2] に刺激されて作った、本節でに説明する Ω の一部機能を使えるようにするパッチが存在する。これは FAM256 パッチと呼ばれ、今までは「 $\epsilon$ -pTeX 本体とは一応別扱いで、 $\epsilon$ -pTeX の配布、及び W32TeX, TeX Live のバイナリでは標準で有効になっていただけ」という扱いであったが、それでは利用者が混乱するので「FAM256 パッチは  $\epsilon$ -pTeX 160201 以降からは切り離さない」とここで宣言する。本ドキュメントの最後のページ\*6にちょっとしたサンプルを載せてある。

本節で述べる追加機能は `extendend mode` でなくても有効になっている。ただし、後に説明する「レジスタが各種類 65536 個まで」は、`extended mode` の時に限り有効になる。

\*5 最後のノードが `dir_node` であった場合、`\lastnodetype` はそのノードが格納している `hlist_node` か `vlist_node` の種類を返す。

\*6 ただし、ソースファイルで言えば `fam256d.tex`（本文）と `fam256p.tex`（preamble 部）に対応する。

## ■数式フォント制限の緩和

$\Omega$  の大きな特徴としては、 $\mathrm{T}_{\mathrm{E}}\mathrm{X}$  内部のデータ構造を倍の領域を用いるように改変し<sup>\*7</sup>、 $\mathrm{T}_{\mathrm{E}}\mathrm{X}$  に従来から存在していた「256 個制限」を  $2^{16}$  個にまで緩和したことが挙げられる。同様に、 $\Omega$  では ([2] にもあるように) 数式フォントを同時に 256 個まで用いることができ、各フォントも 65536 文字まで許されるようになっている。

$\varepsilon\text{-pT}_{\mathrm{E}}\mathrm{X}$  では、中途半端だが、数式フォント 1 つあたりの使用可能文字数は 256 個のままで、同時に数式フォントを 256 個まで使えるようにしている。基本的には  $\Omega$  と同様の方法を用いているが、内部でのデータ構造に違いがある (数字はすべて bit 幅) :

	category	family	char	math code	delimiter code
$\mathrm{T}_{\mathrm{E}}\mathrm{X}82$	3	4	8	$3 + 4 + 8 = 15$	$3 + \underbrace{4 + 8}_{\text{small}} + \underbrace{4 + 8}_{\text{large}} = 27$
$\Omega$	3	8	16	$3 + 8 + 16 = 27$	$(3 + \underbrace{8 + 16}_{\text{small}}, \underbrace{8 + 16}_{\text{large}}) = (27, 24)$
$\varepsilon\text{-pT}_{\mathrm{E}}\mathrm{X}$	3	8	8	$3 + 8 + 8 = 21$	$(3 + \underbrace{8 + 8}_{\text{small}}, \underbrace{8 + 8}_{\text{large}}) = (19, 16)$

$\mathrm{T}_{\mathrm{E}}\mathrm{X}$  に本来あったプリミティブは互換性維持のために同じ動作とする必要があるので、16 番から 255 番のフォントを利用する際には別のプリミティブが必要となる。(実装自体に  $\Omega$  の流儀を使っているから) ここでは、 $\Omega$  のプリミティブ名を流用することにした。すなわち、以下のプリミティブが追加されている<sup>\*8</sup>。

- ▶  $\backslash\mathrm{omathcode}\langle 8\text{-bit number}\rangle=\langle 27\text{-bit number}\rangle$
- ▶  $\backslash\mathrm{omathcode}\langle 8\text{-bit number}\rangle$
- ▶  $\backslash\mathrm{omathchar}\langle 27\text{-bit number}\rangle$
- ▶  $\backslash\mathrm{omathaccent}\langle 27\text{-bit number}\rangle$
- ▶  $\backslash\mathrm{omathchardef}\langle\text{control sequence}\rangle=\langle 27\text{-bit number}\rangle$
- ▶  $\backslash\mathrm{odelcode}\langle 8\text{-bit number}\rangle=\langle 27\text{-bit number}\rangle\langle 24\text{-bit number}\rangle$
- ▶  $\backslash\mathrm{odelimiter}\langle 27\text{-bit number}\rangle\langle 24\text{-bit number}\rangle$
- ▶  $\backslash\mathrm{oradical}\langle 27\text{-bit number}\rangle\langle 24\text{-bit number}\rangle$

ここで、27 bit とか 24 bit の自然数の意味については、上の表の  $\Omega$  の行を参照して欲しい。上に書いた内部のデータ構造から推測できる通り、 $\backslash\mathrm{omathchar}$  等の character code の指定に使われる 16 bit の数値のうち実際に使われるのは下位 8 bit であり、上位 8 bit は無視される。例えば、 $\backslash\mathrm{omathchar}"4012345$  と  $\backslash\mathrm{omathchar}"4010045$  は内部表現としては全く同じである。

<sup>\*7</sup> 詳しい話は `texk/web2c/texmfmem.h` 中の共用体 `memoryword` の定義を参照。大雑把に言うと、1 つの「メモリ要素」に 2 つの 32 bit 整数を同時に格納できるようになっている。

<sup>\*8</sup>  $\Omega$  では  $\langle 8\text{-bit number}\rangle$  のところが  $\langle 16\text{-bit number}\rangle$  になっている。

なお、`\odelcode`  $\langle 8\text{-bit number} \rangle$  として `delimiter code` を取得しようとしても、現時点のパッチでは、うまく動作しない\*9。

L<sup>A</sup>T<sub>E</sub>X において数式フォントを同時に 16 個以上使うには、`\omathchar` などのプリミティブに対応したマクロを使う必要がある。最近の pL<sup>A</sup>T<sub>E</sub>X (2016/11/29 以降) はこれを部分的にサポートしていて、`\DeclareMathAlphabet` で使うことのできる数式用アルファベットの上限だけは 256 個に拡張されている。だが、これだけでは記号類の定義に用いられる `\DeclareMathSymbol` や `\DeclareMathDelimiter` が `\omathchar` や `\odelcode` を使用しないので不十分である。実験的と書かれてはいるが、山本氏による「最低限のパッケージ」[4]、またはこれを最新の L<sup>A</sup>T<sub>E</sub>X に追従してまとめ直された `mathfam256` パッケージ\*10を使うのが手っ取り早いような気がする。

### ■無限のレベル

T<sub>E</sub>X では、`glue` の伸縮量に `fil`, `fill`, `filll` という 3 つの無限大のレベルが存在し、1 が多いほど無限大のオーダーが高くなっていた。Ω では、「inter-letter spacing のために」`fi` という、有限と `fil` の中間にあたる無限大のレベルが付け加えられ、`\hfi`, `\vfi` という 2 つのプリミティブも追加された。そこで、この無限大レベル `fi` も採用することにした。

実装方法は、大まかには Ω で `fi` の実装を行っている change file `omfi.ch` の通りであるのだが、これに pT<sub>E</sub>X や ε-T<sub>E</sub>X に伴う少々の変更を行っている。

- プリミティブ `\pagefistretch` を新たに定義している。
- `\gluestretchorder`, `\glueshrinkorder` の動作を ε-T<sub>E</sub>X のそれと合わせた。具体的には、ある適当な `glue` `\someglue` の伸び量を  $\langle stretch \rangle$  とおくと、

$$\backslash\gluestretchorder\backslashsomeglue = \begin{cases} 0 & \langle stretch \rangle \text{ が高々 } \text{fi} \text{ レベルの量} \\ 1 & \langle stretch \rangle \text{ がちょうど } \text{fil} \text{ レベルの無限量} \\ 2 & \langle stretch \rangle \text{ がちょうど } \text{fill} \text{ レベルの無限量} \\ 3 & \langle stretch \rangle \text{ がちょうど } \text{filll} \text{ レベルの無限量} \end{cases}$$

となっている。内部では `fi` レベルが 1, `fil` レベルが 2, ……として処理している。

### ■レジスタについて

Ω では (前にも書いたが) データ構造の変更が行われ、それによってレジスタが各種類あたり 65536 個使えるようになっている。

一方、ε-T<sub>E</sub>X では、256 番以降のレジスタを専用の `sparse tree` に格納することにより、32767 番までのレジスタの使用を可能にしていた。このツリー構造を分析してみると、65536 個までレジスタを拡張するのはさほど難しくないことのように思われた。具体的には、ツリーの階層を 1 つ増やしてみた (だから、おそらく各種類あたり  $16 \cdot 32768 = 524288$  個まで使えるとは思いますが、これはきりが悪い)。そこで、ε-pT<sub>E</sub>X では ε-T<sub>E</sub>X 流の方法を用いながらも、レジスタをさらに 65536 個まで増やしている。

\*9 51 bit 自然数を返さないといけませんからねえ。やる気があれば検討してみます。

\*10 <https://ctan.org/pkg/mathfam256>.

## 4 pdfTeX 由来の機能

開発中の L<sup>A</sup>T<sub>E</sub>X3 では、 $\epsilon$ -T<sub>E</sub>X 拡張の他に、pdfT<sub>E</sub>X で導入された `\pdfstrcmp`（又はその同等品）が必要となっており、もはや純粋な  $\epsilon$ -T<sub>E</sub>X ですら L<sup>A</sup>T<sub>E</sub>X3 を利用することはできない状況である ([5, 6, 7]). その他にも、pdfT<sub>E</sub>X 由来のいくつかのプリミティブ ([17]) の実装が日本の T<sub>E</sub>X ユーザからあり、ほとんど pdfT<sub>E</sub>X における実装をそのまま真似する形で実装している.

現在の  $\epsilon$ -pT<sub>E</sub>X で利用できる pdfT<sub>E</sub>X 由来のプリミティブの一覧を以下に示す. これらは extended mode でないと利用できない.

### ► `\pdfstrcmp` *<general text>* *<general text>*

2 つの引数を文字列化したものを先頭バイトから比較し、結果を  $-1$ （第 1 引数の方が先）、 $0$ （等しい）、 $1$ （第 2 引数の方が先）として文字列で返す.

比較する文字列中に和文文字がある場合には、( $\epsilon$ -pT<sub>E</sub>X の内部漢字コードにかかわらず) UTF-8 で符号化して比較する. そのため、例えば

```
\pdfstrcmp{あ}{\e3\81\83} % 「あ」は UTF-8 で E38182
```

の実行結果は  $-1$  である.

### ► `\pdfpagewidth`, `\pdfpageheight`

ページの「幅」「高さ」を表す内部長さであるが、ここで言う「幅」は「字送り方向」のことではなく、物理的な意味である.

この 2 つの内部長さを設定するだけでは dvi に何の影響も与えない. すぐ後で述べる `\pdflastxpos`, `\pdflastypos` による出力位置の取得の際の原点位置を設定するためだけに使われ、初期値は  $0$  である.

### ► `\pdflastxpos`, `\pdflastypos`

`\pdfsavepos` が置かれた場所の、dvi における出力位置を返す内部整数（読み取り専用）. 原点はページの（物理的な意味の）左下隅であり、 $y$  軸は（物理的な）上方向に向かって増加する.

- ページの物理的な幅と高さはすぐ上の `\pdfpagewidth`, `\pdfpageheight` で設定する. これらの内部長さが  $0$  であった場合は、`\shipout` されたボックスの寸法と `\hoffset`（または `\voffset`）の値から自動的に計算される.
- pT<sub>E</sub>X では横組・縦組と組方向が複数あるので、`\pdflastxpos`, `\pdflastypos` の値の座標系を「物理的な」向きとすべきか、それとも「組方向に応じた」向きとすべきかは悩みどころである. 110227 版以降、現在までの版では上記のように物理的な向きとしている.

### ► `\pdfcreationdate`

エンジン起動時の時刻を、`D:20180309184338+09'00'` の形式で表した文字列に展開する. これは standalone パッケージを  $\epsilon$ -pT<sub>E</sub>X で扱うために 2013/06/05 に実装された



プリミティブであるが、現在時刻の「秒」まで得るためにも使用できる (T<sub>E</sub>X82 では分単位でしか取得できない)。

$\varepsilon$ -pT<sub>E</sub>X においてプリミティブを実装した当初は「最初にこのプリミティブが実行された時刻を…」としていたが、161030 版から pdfT<sub>E</sub>X と同じ挙動に修正した。

► `\pdffilemoddate <filename>`, `\pdffilesize <filename>`

それぞれ `<filename>` の更新時刻 (`\pdfcreationdate` と同じ形式) とファイルサイズを表す文字列に展開する。これらも standalone パッケージのために  $\varepsilon$ -pT<sub>E</sub>X に実装されたプリミティブである。

► `\pdffiledump [offset <offset>] length <length> <filename>`

`<filename>` で与えられたファイル名の `<offset>` バイト目 (先頭は 0) から `<length>` バイトを読み込み、16 進表記 (大文字) したものに展開される。

本プリミティブは Heiko Oberdiek 氏による `bmptsize` パッケージを  $\varepsilon$ -pT<sub>E</sub>X でも使うために角藤さんが実装したものである (2014/05/06)。

► `\pdfshellescape`

`\write18` による shell-escape が利用可能になっているかを示す内部整数 (読み取り専用)。0 ならば不許可, 1 ならば許可, 2 ならば restricted shell-escape<sup>\*11</sup>である。

本プリミティブは T<sub>E</sub>X ユーザの集い 2014 でリクエストを受けて実装された ([9])。

► `\pdfmdfivesum [file] <general text>`

引数 `<general text>` の MD5 ハッシュ値か、あるいは `file` が指定された場合はファイル名が `<general text>` のファイルの MD5 ハッシュ値を計算する。

前者の「引数の MD5 ハッシュ値を計算する」場合には、`\pdfstrcmp` と同じように和文文字を UTF-8 で符号化してから計算する。

このプリミティブは [11] 以降の議論を元に、角藤さんがリクエストしたもので、2015/07/04 に  $\varepsilon$ -pT<sub>E</sub>X に実装されている。

► `\pdfprimitive`, `\ifpdfprimitive`

`\pdfprimitive` は次に続く制御綴がプリミティブと同じ名称であった場合に、プリミティブ本来の意味で実行させるものである。例えば

`\pdfprimitive\par`

は、`\par` が再定義されていようが、本来の `\par` の意味 (段落終了) となる。また、`\ifpdfprimitive` は、次に続く制御綴が同名のプリミティブの意味を持っていれば真、そうでなければ偽となる条件判断文である。

これらのプリミティブは 2015/07/15 版の `expl3` パッケージで使われた ([12]) ことを受けて実装されたものだが、現在ではこれらのプリミティブは使われていない。

---

<sup>\*11</sup> あらかじめ「安全」と認められたプログラム (`texmf.cnf` 中で指定する) のみ実行を許可する仕組み。

► `\pdfuniformdeviate`  $\langle number \rangle$ , `\pdfnormaldeviate`

`\pdfuniformdeviate` は、0 以上  $\langle number \rangle$  未満の一様分布に従う乱数（整数値）を生成する。`\pdfnormaldeviate` は、平均値 0、標準偏差 65536 の正規分布に従う乱数（整数値）を生成する。

現在の乱数生成の種の値は、`\pdfrandomseed` で取得できる（読み取り専用）。種の初期化にはシステムのマイクロ秒単位での現在時刻情報が使われる。また、種の値は `\pdfsetrandomseed`  $\langle number \rangle$  によって特定の値に設定可能である。

L<sup>A</sup>T<sub>E</sub>X3 の l3fp において、2016/11/12 あたりから実装された乱数生成機能 ([13]) をサポートするために 161114 版から実装された。

► `\pdfelapsedtime`, `\pdfresettimer`

`\pdfelapsedtime` は、エンジン起動からの経過時間を “scaled seconds” すなわち 1/65536 秒単位で返す。この値は `\pdfresettimer` によって再び 0 にリセットできる。すぐ上の乱数生成プリミティブと同時に実装された。

## 5 バージョン番号

pT<sub>E</sub>X p3.8.0 に `\ptexversion` が実装されたのと同時に、 $\varepsilon$ -pT<sub>E</sub>X でもバージョン番号を取得する `\epTeXversion` プリミティブが 180121 版から追加された。

► `\epTeXversion`

$\varepsilon$ -pT<sub>E</sub>X のバージョン番号（例えば 180121）を内部整数で返す。 $\varepsilon$ -pT<sub>E</sub>X 起動時のバナーでは  $\varepsilon$ -T<sub>E</sub>X, pT<sub>E</sub>X のバージョン番号も表示されるので、それを再現しようとする以下のようなになる。

```
This is e-pTeX, Version 3.14159265-%  
p\number\ptexversion.\number\ptexminorversion\ptexrevision-%  
\number\epTeXversion-\number\epTeXversion\epTeXrevision ...
```

## 6 `\lastnodechar` プリミティブ

本プリミティブは T<sub>E</sub>X ユーザの集い 2014 でリクエストを受けて実装された ([9]) プリミティブで、extended mode でしか利用できない。詳細な背景説明・仕様は [10] に譲る。

pT<sub>E</sub>X では

これは、`\textmc{『ほげ党宣言』}` の……

という入力からは

これは、『ほげ党宣言』の……

という出力が得られ、コンマと二重鍵括弧の間が全角空きになってしまうことが以前から知られている。

この現象は、(展開し続けた結果)「,」の直後のトークンが「『」ではないことによって、「,」の後の半角空きと、「『」の前の半角空きが両方入ってしまうという pTeX の和文処理の仕様<sup>\*12</sup>による。min10 フォントメトリックで「ちょっと」を組むと「よっ」の間が詰まるという不具合は有名であるが、「ちょ{ }っと」と空グループを挟むことで回避されるのも、同じ理由である。

`\lastnodechar` プリミティブは、上で述べた「書体変更命令を間に挟むと和文間グルーが『まともに』ならない」という状況を改善する助けになることを目指して実装された。

#### ► `\lastnodechar`

現在構築中のリストの「最後のノード」が文字由来であれば、そのコード番号（内部コード）を内部整数として返す。

上記「最後のノード」では、pTeX によって自動挿入される

- JFM によって入るグルー
- 行末禁則処理のために挿入されるペナルティ
- 欧文文字のベースライン補正用のノード

は無視される。また、「最後のノード」が欧文文字のリガチャであった場合は、リガチャそれ自身のコード番号ではなく、最後の構成要素の文字のコード番号を返す。「最後のノード」が文字を表すものでなかった場合は、`-1` が返る。

例えば、`\lastnodechar` を使って

これは、`\the\lastnodechar\textmc{『ほげ党宣言』}`……

と入力すると、

これは、`41380『ほげ党宣言』`……

のようになり、`\lastnodechar` 実行時の「最後のノード」（文字「,」を表す）の内部コード<sup>\*13</sup>が得られる。これによって、`\textmc` 等の命令の直前の文字を知ることができるので、あとは TeX マクロ側でなんとかできるだろう、という目論見である。

また、上記の説明にあるとおり、

`abcfi\the\lastnodechar, abc\char"1C \the\lastnodechar`

は

`abcfi105, abcfi28`


となり、見た目では同じ「fi」が `\lastnodechar` 実行時の「最後のノード」であるかのように

---

<sup>\*12</sup> TeX82 の欧文のカーニングや合字処理も同じような仕様になっている。例えば `W\relax oWo` からは `WoWo` という出力になり、`W` と `o` の間のカーニングが `\relax` によって挿入されなくなったことがわかる。

<sup>\*13</sup> 内部コードが EUC の場合は `"A1A4 = 41380`、SJIS の場合は `"8143 = 33091` となる。

見えても、それが合字の場合（左側）では `\lastnodechar` の実行結果は最後の構成要素「i」のコード番号 105 となる。

 「これは、」とソース中に入力したときのノードの状態を `\showlists` で調べてみると次のようになり、本当に一番最後のノードは JFM によって挿入される二分空きの空白（「,」と通常の和文文字の間に入るはずのもの）であることがわかる。

```
### yoko direction, horizontal mode entered at line 465
\hbox(0.0+0.0)x9.24683
\JY1/hmc/m/it/10 こ
\JY1/hmc/m/it/10 れ
\JY1/hmc/m/it/10 は
\penalty 10000(for kinsoku)
\JY1/hmc/m/it/10 ,
\glue(refer from jfm) 4.62341 minus 4.62341
```

しかし、数段落上の説明の通り、`\lastnodechar` は pTeX の和文処理によって自動的に挿入されたこれら JFM 由来の空白を無視する。

「最後のノード」を見ているので、

これは、`\relax\sffamily{}\the\lastnodechar\textmc{『ほげ党宣言』}`……

などとノードに関係しないものが途中にあっても、それは単純に無視されて

これは、41380『ほげ党宣言』……

となる。

## 7 `\lastnodesubtype` プリミティブ

ε-pTeX 180226 では、「グルーが明示的な `\hskip` 由来か、それとも JFM 由来のものか」などをユーザが取得できるようにするため、`\lastnodesubtype` 命令を新設した。

### ► `\lastnodesubtype`

現在のリストの最後のノードの subtype を取得する。実際に有用なのは以下の場合：

最後のノードがグルー (`\lastnodetype = 11`) のとき

0: 明示的な <code>\hskip</code> , <code>\vskip</code>	14: <code>\xspaceskip</code>
1: <code>\lineskip</code>	15: <code>\parfillskip</code>
2: <code>\baselineskip</code>	16: <code>\kanjiskip</code>
3: <code>\parskip</code>	17: <code>\xkanjiskip</code>
4: <code>\abovedisplayskip</code>	18: <code>\thinmuskip</code>
5: <code>\belowdisplayskip</code>	19: <code>\medmuskip</code>
6: <code>\abovedisplayshortskip</code>	20: <code>\thickmuskip</code>
7: <code>\belowdisplayshortskip</code>	21: JFM 由来グルー
8: <code>\leftskip</code>	98: <code>\nonscript</code>
9: <code>\rightskip</code>	99: <code>\mskip</code>
10: <code>\topskip</code>	100: <code>\leaders</code>
11: <code>\splittopskip</code>	101: <code>\cleaders</code>

12: <code>\tabskip</code>	102: <code>\xleaders</code>
13: <code>\spaceskip</code>	

最後のノードがカーン (`\lastnodetype = 12`) のとき

0: カーニング	2: アクセント由来	99: <code>\mkern</code>
1: 明示的な <code>\kern</code>	3: イタリック補正 <code>\/</code>	

最後のノードがペナルティ (`\lastnodetype = 13`) のとき

0: 明示的な <code>\penalty</code>	2: 禁則処理由来
1: <code>\jcharwidowpenalty</code>	

## 8 `\epTeXinputencoding` プリミティブ

現在読み込んでいるファイルの文字コードを切り替えるプリミティブであり、2016/02/01 に阿部紀行さんによって実装された。詳細は実装者の解説記事 [14] を参照してほしいが、おおまかに述べると以下のようなになるだろう。

### ► `\epTeXinputencoding` $\langle encoding \rangle$

現在読み込んでいるファイルの文字コードを  $\langle encoding \rangle$  に変更する。実際に変更されるのは「次の行」であり、また現在のファイルからさらに `\input` 等で読まれたファイルには効力を及ぼさない。

$\langle encoding \rangle$  の値は、基本的には pTeX の `-kanji` オプションで指定できる値 (`euc`, `sjis`, `jis`, `utf8`) である。

## 参考文献

- [1] 北川 弘典, 「計算数学 II 作業記録」, 2008.  
<https://osdn.jp/projects/eptex/document/resume/ja/1/resume.pdf> ほか, 本 pdf と同じディレクトリにある `resume.pdf` がそれにあたる.
- [2] 山本 和義, 「数式 fam の制限と luatex」, 掲示板「TeX Q & A」, 2009/02/12.  
<http://oku.edu.mie-u.ac.jp/~okumura/texfaq/qa/52744.html>
- [3] 山本 和義, 「Re: 数式 fam の制限と luatex」, 掲示板「TeX Q & A」, 2009/02/16.  
<http://oku.edu.mie-u.ac.jp/~okumura/texfaq/qa/52767.html>
- [4] 山本 和義, 「数式 fam 拡張マクロ for e-pTeX 等」, 掲示板「TeX Q & A」, 2009/02/21.  
<http://oku.edu.mie-u.ac.jp/~okumura/texfaq/qa/52799.html>
- [5] 河原, 「パッケージとディストリビューションについて」, 掲示板「TeX Q & A」, 2010/12/16. <http://oku.edu.mie-u.ac.jp/~okumura/texfaq/qa/55464.html>
- [6] 角藤 亮, 「Re: パッケージとディストリビューションについて」, 掲示板「TeX Q & A」, 2010/12/19. <http://oku.edu.mie-u.ac.jp/~okumura/texfaq/qa/55478.html>
- [7] zrbabbler, 「LaTeX3 と expl3 パッケージ」, ブログ「マクロツイーター」内, 2010/12/22.  
<http://d.hatena.ne.jp/zrbabbler/20101222/1293050561>

- [8] 角藤 亮, 「Re: e-pTeX 101231」, 掲示板「 $\mathrm{T}_{\mathrm{E}}\mathrm{X}$  Q & A」, 2011/01/01.  
<http://oku.edu.mie-u.ac.jp/~okumura/texfaq/qa/55528.html>
- [9] Dora TeX, 「Re: \pdfshellescape, \lastnodechar の実装」,  $\mathrm{T}_{\mathrm{E}}\mathrm{X}$  Forum, 2014/11/19.  
<http://oku.edu.mie-u.ac.jp/tex/mod/forum/discuss.php?d=1435#p8053>
- [10] 北川 弘典, 「\lastnodechar プリミティブについて」, 2014/12/15.  
<https://ja.osdn.net/projects/eptex/wiki/lastnodechar>
- [11] Joseph Wright, “[XeTeX] \(\pdf)mdfivesum”, 2015/07/01,  
<http://tug.org/pipermail/xetex/2015-July/026044.html>
- [12] tat tsan, 「[expl3 / e(u)ptex] 2015/07/15 版 expl3 パッケージが、(u)platex で通らない」,  $\mathrm{T}_{\mathrm{E}}\mathrm{X}$  Forum, 2015/07/26.  
<http://oku.edu.mie-u.ac.jp/tex/mod/forum/discuss.php?d=1632>
- [13] Joseph Wright, “[tex-live] Random number primitives”, 2016/11/12,  
<http://tug.org/pipermail/tex-live/2016-November/039436.html>
- [14] 阿部 紀行, 「2016 年 2 月 2 日」, 2016/02/02.  
<http://abenori.blogspot.jp/2016/02/e-ptexetexinputencoding.html>.
- [15] The  $\mathcal{N}\mathcal{T}\mathcal{S}$  Team. *The  $\varepsilon$ - $\mathrm{T}_{\mathrm{E}}\mathrm{X}$  manual* (v2.0).  
 $\$TEXMFDIST/doc/etex/base/etex\_man.pdf$
- [16] J. Plaice, Y. Haralambous. *Draft documentation for the  $\Omega$  system*, 1999.  
 $\$TEXMFDIST/doc/omega/base/doc-1.8.tex$
- [17] Hàn Thế Thành et al. *The pdf $\mathrm{T}_{\mathrm{E}}\mathrm{X}$  user manual*, 2015.  
 $\$TEXMFDIST/doc/pdftex/manual/pdftex-a.pdf$
- [18] The  $\mathrm{L}^{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}3$  Project Team,  *$\mathrm{L}^{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$  News Issue 26*, 2017.  
 $\$TEXMFDIST/source/latex/base/ltnews26.tex$ ,  
<https://www.latex-project.org/news/latex2e-news/ltnews26.pdf>.

## 索引

<code>\currentifttype</code> , 5	<code>\pdffiledump</code> , 9
<code>\epTeXinputencoding</code> , 13	<code>\pdffilemoddate</code> , 9
<code>\epTeXversion</code> , 10	<code>\pdffilesizes</code> , 9
<code>\ifpdfprimitive</code> , 9	<code>\pdflastxpos</code> , 8
<code>\lastnodechar</code> , 11	<code>\pdflasttypos</code> , 8
<code>\lastnodesubtype</code> , 12	<code>\pdfmdfivesum</code> , 9
<code>\lastnodetype</code> , 5	<code>\pdfnormaldeviate</code> , 10
<code>\middle</code> , 4	<code>\pdfpageheight</code> , 8
<code>\odelcode</code> , 6	<code>\pdfpagewidth</code> , 8
<code>\odelimiter</code> , 6	<code>\pdfpritimive</code> , 9
<code>\omathaccent</code> , 6	<code>\pdfrandomseed</code> , 10
<code>\omathchar</code> , 6	<code>\pdfresettimer</code> , 10
<code>\omathchardef</code> , 6	<code>\pdfsavepos</code> , 8
<code>\omathcode</code> , 6	<code>\pdfsetrandomseed</code> , 10
<code>\oradical</code> , 6	<code>\pdfshellescape</code> , 9
<code>\pdfcreationdate</code> , 8	<code>\pdfstrcmp</code> , 8
<code>\pdfelapsedtime</code> , 10	<code>\pdfuniformdeviate</code> , 10

## Test source for FAM256 patch

本ソースは山本和義氏による「数式 fam の制限と luatex」(qa:52744) 中のコードをベースにしたものである。

■ More than 16 math font families.

ABCDEFGHIJKL fam = 19  
 AaAbAcAdAeAfAgAhAiAjAkAlAmAnAoAp fam35  
 AaAbAcAdAeAfAgAhAiAjAkAlAmAnAoAp fam51  
 AaAbAcAdAeAfAgAhAiAjAkAlAmAnAo**A**p fam**67**  
**A**a fam68 Ab fam69 Ac fam70 Ad fam71 roman

■ \omathchar etc.

`\mathchar"7F25 : %, \omathchar"7420125 : %`

meaning of `\langle`: macro:->`\delimiter "426830A ,`

meaning of `\lx`: macro:->`\odelimiter "4450068"030001`

`\lx:h, \bigl\lx:), \Bigl\lx:)`

`\the\mathcode'\f: 7166, \the\omathcode'\f: 7010066` (どちらも 16 進に変換した)

t>>>)))<

$$\sqrt{\cdot}, p, \sqrt{a}, p_a^{\blacksquare} \sqrt{\int_V f d\mu}, \sqrt{\int_V f d\mu}$$

`\odelcode` primitive による **delimiter code** の取得はうまく動かない：

`\the\delcode'\|: 2532108, \the\odelcode'\|: -1` (どちらも 10 進)

## ■ Infinite level “f”

■(fi)■(fil)■(fill)■	(filll)	■
■(fi)■(fil)■	(fill)	■
■(fi)■	(fil)	■
■	(fi)	■
	(fi)	(fi)
		(fi)
		■

■ 65536 registers

fuga! a 漢字仮名 sdf T<sub>E</sub>X ほげほげ<sub>E</sub>X  
589