クラスファイルの中味を眺めてみる

私立文系初級ユーザー

2009 年 10 月 3 日

概　要

jarticle.cls の中味を，文系初級ユーザーの私が分かる範囲で眺めてみます。その前にまず，クラスファイルを眺めるに当たってある程度前提となることがらに関して（これまた私が理解している範囲についてのみですが）簡単に確認をします。

本文書は二部構成になっています。第 II 部がメインのつもりで，jarticle.cls の中味を眺めています。第 I 部は，第 II 部のための準備です。

第 II 部の直接のネタ元は，PracTEX Journal の記事 [10] で(1)，第 I 部のネタ元は memoir のマニュアルの “Appendix B: LaTeX and TeX” [9] です(2)。

第 I 部

1　基本的なファイルとその解説ドキュメント

はじめに，(LA)TEX を構成するファイルと，その解説ドキュメントについて，私の不十分な理解に基づいてですけれど，整理しておこうと思います。

すごく曖昧な知識なのですが，以下の①〜⑦のうち，①と⑤のみが，いわゆるプログラムというか実行ファイルで，残りのものはマクロ・パッケージです。現代のマシンパワーであれば，マクロを毎回読み込ませてもそれほど時間は掛からないのではないかとも素人考えでは思ってしまうのですが，これらのマクロは，一旦 “.fmt” という内部形式にしたものを，TEX なり pTEX なりに読み込ませているらしいです（フォーマットファイルにするのは効率の問題からだけではなくて，ハイフネーションパターンのことなんかも関係するらしいのですけど）。

以下では ⓐ が実体で，ⓑ がその解説ドキュメントです。

① TEX

- ⓐ `tex.exe` (?);
  ← `tex.p` 〔$\xleftarrow{\text{tangle}}$ `tex.web`〕;
  $\xrightarrow{\text{web2c}}$ `tex.c` (?)
- ⓑ `tex.tex` 〔$\xleftarrow{\text{weave}}$ `tex.web`〕;
  TEX Reference Manual [2];
  TEXbook [1];
  TEX by Topic [3], Impatient [4]

「裸の TEX」とか「組版エンジンとしての TEX」とも称される，TEX そのものです（現在のバージョンは TeX82 Version 3.1415926, February 2008(3)）。TEX というプログラムのソースは，クヌース先生提唱になる “Literate

(1) もちろん，PracTEX Journal の記事以前にも，クラスファイル（スタイルファイル）を読み解くという文献はありました。邦語文献に限ってみても：

- 奥村晴彦編著『LATEX 入門：美文書作成のポイント』（技術評論社・1994 年）の第 13 章「スタイルファイルの作り方」
- 藤田眞作『LATEX 本づくりの八衢』（アジソン・ウェスレイ・パブリッシャーズ・ジャパン・1996 年）の第 II 部「スタイルファイル版面設計」
- 乙部厳己・江口庄英『pLATEX 2_ε for Windows Another Manual Vol.2 Extend Kit』（ソフトバンクパブリッシング・1997 年）の Chapter10「クラスファイル」
- ページ・エンタープライゼズ『LATEX 2_ε【マクロ&クラス】プログラミング基礎解説』（技術評論社・2002 年）の第 11 章「11.1 一般的書籍のクラスファイル例」

等々，錚々たる名著の数々が，この話題を取り上げています。なお，いずれも文献でも，新たなクラスファイル（大抵は book.cls 相当）の構築を意図した解説となっていますが，本文書第 II 部は，ただ粛々と jarticle.cls の中味を追うだけのものです。

(2) 第 I 部で取り上げるトピックとその叙述の程度は，memoir のマニュアルの “Appendix B” に準じます。なにしろ私は私立文系初級ユーザーなものですから，“Appendix B” で引用されているハムレットの台詞を更にもじるならば：

There are more things in heaven and TEX, novice comrades,
Than are dreamt of in *my* philosophy!

というのが正直なところです…。

(3) TEX のバージョンというのはどうもよく分からないのですけど，TEX82 のバージョン番号を，TEX のバージョンと呼ぶみたいです。ですから，TEX82 のバージョン 0, 1, 2, 3 をそれぞれ TEX version 0, TEX version 1, TEX version 2, TEX version 3 と称しているようです。TEXbook の扉裏にも “This manual describes TEX Version 3.0.” と書いてありますし。*cf.* Philip Taylor, “*Computer Typesetting or Electronic Publishing?*”, in: TUGboat, Vol. 17 (1996), No.4.

Programming” という方法を実装した WEB という言語で書かれています。“tex.web” を tangle というプログラムに通すと “tex.p” という Pascal で書かれた TEX のソースが取り出せて，他方 “tex.web” を weave というプログラムで処理すると，Pascal のソースとその解説ドキュメントを TEX で整形した “tex.tex” が取り出せます。最近では，web2c というプログラムを使って，tex.p は tex.c とか何か，C 言語に変換してから実行ファイルにコンパイルするらしいですけど，そのあたりのことはよく分かりません。TEX の実行ファイルが tex.exe であるのかどうか（裸の TEX が tex.exe かどうか）も，ホントはかなり怪しいです。すいません。

ひとつの “.web” ファイルから，コードそのものと，コードとその解説ドキュメントとを取り出せるというのは，“.dtx” ファイルに似ていますね（.dtx は “Literate Programming” の LATEX 2ε版みたいなものなので，似ていて当然なのですが）。

私はプログラミングのプの字も知らないので，「Pascal で書かれた TEX というプログラム自体の解説」を見てもさっぱり分からないので，タイプセットすると 500 ページ以上にもなる tex.tex については，TEX の歴史について書かれた最初の 1 ページしか読んでいません[4]。

さて，ユーザーの立場からは，「TEX そのもの」は 325 個のプリミティブの集合体といえます。文献 [2] は，全プリミティブについてのレファレンスです。TEXbook [1] はクヌース先生の手になる「マニュアル」ですが，プリミティブについてだけなく，次の plain TEX までを含んでいます。また，索引が充実しているとはいえ飽くまでマニュアルであって，レファレンスではありません。

文献 [3] と文献 [4] も，プリミティブだけではなく，plain TEX までを含めての解説が載っています。

幸いなことに，ここに挙げた文献 [1]〜[4] はすべて，原著（英語版）であればオンラインで無料で読むことができます（TEXbook だけは，オンラインで見ようとするなら建前上 “texbook.tex” を見ることになりますけど）。

② plain TEX

- ⓐ plain.tex （→ plain.fmt）
- ⓑ ファイル plain.tex の中に書かれたコメント;
  TEXBOOK [1] の “Appendix B: Basic Control Sequences”

クヌース先生が書かれた，マクロ・パッケージです（現在のバージョンは Version 3.141592653, February 2008）。プリミティブだけで原稿を作成するのは大変ですし，plain.tex はマクロ・パッケージの見本として①と一緒に提供されたので，一般には①-ⓐと②-ⓐを合わせたものを「TEX」と呼んでいるようです。クヌース先生が書かれたマクロ・パッケージで公開されているものには，TEXbook を執筆された際に作られた “manmac.tex” ですとか，WEB のマニュアルのために書かれた “webmac.tex” などもあります。

マクロとしての plain.tex のドキュメントは，plain.tex の中に直接にコメントとして書かれていることの他には，TEXbook の “Appendix B” が丸々，plain.tex の解説となっています。

plain TEX に関するユーザー向けの文献が TEXbook[1] であって，また，文献 [3] や文献 [4] にも一部説明があります。

③ LATEX2.09

- ⓐ lplain.tex, lhyphen.tex, lfonts.tex, latex.tex （→ latex209.fmt）
- ⓑ ファイル lplain.tex, lhyphen.tex, lfonts.tex, latex.tex の中に書かれたコメント;
  各種 “.doc” ファイル;
  LATEX2.09 のマニュアル [5]

LATEX2.09 のマニュアル [5] の前書きによりますと，ランポートさんが LATEX を作り始めたのは 1982 年頃とのことです[5]。最終的には LATEX“2.09” というバージョンで呼ばれることになるこのマクロ・パッケージの実体は，③-ⓐに挙げた 4 つのファイルです。

クヌース先生による plain.tex の内容の多くは lplain.tex に引き継がれ，そして，新たに 3 つのファイルが付け加わりますが，そのメインは latex.tex です。plain.tex がコメント込みで 1,200 行ほどであるのに対して，latex.tex は 9,000 行近くもあります。

ランポートさんは，TEX の詳細を知らないユーザーでも LATEX を使えるように作ってくださいましたが，その反面，LATEX から始めて TEX へ遡るのは，なかなか大変な道のりになりました。ひとつには，LATEX では，行分割やページ分割の「ペナルティ」ですとか，脚註やフロートの一般形である「インサート」といった TEX の概念の一部をユーザーに見せないようにしているためです。もうひとつには，マクロ・パッケージ LATEX は，TEX のプリミティブと，「プリミティブを組み合わせて作った内部処理用のコマンド」とを組み合わせて成り立っているからです。つまり LATEX の舞台裏を覗こうとした場合には，LATEX の知識だけでは歯が立たなくって，TEX のプリミティブについての知識が不可欠なのだと分かります（但し，ランポートさんは，ご自分が作成した「内部処理用コマンド」については，その定義も使用法も latex.tex 内でコメントとして詳細に解説をしてくださっていますので，内部処理用コマンドについてならば，ユーザーの手許に解説ドキュメントがあることになります）。

また，現在でいう .cls ファイルや .sty ファイルに相当する，LATEX2.09 時代の「スタイルファイル」については，その解説ドキュメントが “.doc” ファイルという形で存在していました。つまり，現在であればひとつの

---

[4] そこには，1977 年の TEX のプロトタイプや TEX78 は SAIL という言語で書かれていたことなどが記されています。なお，TEX のバグ取りの記録は “errorlog.tex” として公開されています。これを見ると，1978 年 3 月 10 日に始まり，2002 年 9 月 30 日までに至る夥しい数の修正がまるで日記のように事細かに記録されています。バグ取りについてだけでなく，“Today I’m working on the user manual.” (21 Jun 1978) とか，1983 年 12 月 9 日には “Dinner party with 36 guests to celebrate TEX’s coming of age.” なんてことが書いてあって，パラパラ眺めるとちょっと面白いです。

[5] TUGBOAT, Vol. 22 (2001) No.1/2 のランポートさんへのインタビュー記事（http://www.tug.org/TUGboat/Articles/tb22-1-2/tb70lamp.pdf）には，次のようにあります：
“When Don was creating TEX80(?), the second version of TEX, the popular macro package at the time was one written by Max Diaz.—I’ve forgotten its name. I was in the process of starting to write a book, and I found Diaz’s macros inadequate. So, I needed to write a set of macros for the book. I figured that, with a little extra effort, I could make a macro package that could be used by other people as well. That was the origin of LATEX.”

.dtx ファイルからコードもドキュメントも得られますが，当時は，コメント込みの “.doc” ファイルと，.doc ファイルからコメント行を削除した “.sty” ファイルとの 2 本立てで配布されていました（でもこれだとメンテが大変そうですね）。

ユーザーコマンドの解説書はもちろん，LaTeX2.09 のマニュアル [5] です。

④ LaTeX 2ε

- ⓐ latex.ltx をはじめとする各種 “.ltx” ファイル
  〔← 沢山の “.dtx” ファイル〕（→ latex.fmt）
- ⓑ source2e.tex;
  各種 “.dtx” ファイル;
  LaTeX3 プロジェクトによるドキュメント;
  LaTeX 2ε のマニュアル [6];
  The LaTeX Companion [15]

ランポートさんの LaTeX2.09 と，LaTeX3 プロジェクトによる LaTeX 2ε との間には，Frank Mittelbach さんと Rainer Schöpf さんによる “Mainz Project” というのがあったらしいです（NFSS や，array.sty, ftnright.sty, multicol.sty, theorem.sty, verbatim.sty ）。マインツ・プロジェクトに関する文献というのはほとんど見当たらないのですけど，ftp://tug.org/historic/macros/latex209/ には “nfss-9203”，“nfss-9209”，“nfss-9307” というフォルダがあって，そこに当時のファイルが保存されています(6)。

さて，LaTeX2.09 では，解説ドキュメントは，latex.tex などのファイルの中に直接コメントとして書かれていましたので，それを読むには適当なエディタか何かで開いて，閲覧するなり印刷するなりする必要がありました。

LaTeX 2ε では，doc と docstrip とを使って，“Literate Programming” を実現しています(7)。“.dtx” という拡張子の documented (annotated) tex source を doc を使って処理すると，コードとその解説ドキュメントを併せてタイプセットすることが出来，docstrip を使うと，.dtx ファイルからドキュメント部分を取り去ってコードのみを取り出すことができる仕組みになっています（実際には，“.dtx” ファイルを記述するのに使われているクラスファイルが内部で doc を読み込んでいて，他方 “.ins” ファイルは内部で docstrip.tex を読み込んでいるので，表面的にはこの仕組みがユーザーの目に触れづらいきらいはあります）。

LaTeX2.09 は ③-ⓐ に挙げた 4 つのファイルから成っていましたが，LaTeX 2ε ではこれらを 40 個ほどの .dtx ファイルに切り分けています（もちろん，2.09 時代のコードとドキュメントだけをただ切り分けたのではなくて，2ε で新たに付け加えられたコードとそのドキュメントなども合わせたものを，全部で約 40 個のファイル群に整理し直したということです）。これら 40 個ほどの .dtx ファイルを，“2ekernel” というオプションを docstrip に与えて処理したものが，“latex.ltx” ということになります。これが，LaTeX 2ε そのもの，LaTeX 2ε のカーネル部分のコードです。

latex.ltx は，latex.fmt を作るために，ドキュメント部分を全部削ってコードのみを残したものです。上級者の方はよく「latex.ltx に書いてある」ということをおっしゃいますが，素人がコードを読む場合には，やはり解説ドキュメントが欲しいです。ここで，個々の .dtx ファイルを LaTeX で処理してももちろん構わないのですが，LaTeX 2ε のソースをまとめてタイプセットするための “source2e.tex” というファイルが予め用意されています(8)。source2e を使うと，コマンドインデックスも作成できますので，後々検索するのにとても重宝します。

LaTeX2.09 のときは，「スタイルファイル」についてのドキュメントは .doc というファイルで提供されていましたが，LaTeX 2ε では，カーネルだけでなく，クラスファイルやパッケージについても，.dtx ファイルの形式が採用されています。なので，標準のクラスファイルについてのドキュメントは “classes.dtx” というファイルで，その他標準に含まれているパッケージについても，コードと解説ドキュメントが .dtx ファイルひとつにまとめられています。

なお，LaTeX 2ε の場合には，LaTeX3 プロジェクトが作成したドキュメント（ltx3info.tex; cfgguide.tex, clsguide.tex, cyrguide.tex, encguide.tex, fntguide.tex, modguide.tex, usrguide.tex ）も同梱されているはずです。これらは元々 LaTeX2.09 を使っていたようなユーザーを念頭に書かれていると思われますが，一通り目を通しておくと，全体的な見通しがよくなるのではないかと思われます（この他にも，LaTeX 2ε の配布物にはものすご

(6) ちなみに，ftp://tug.org/historic/macros/latex-saildart/latex-0.92/manual/ と ftp://tug.org/historic/macros/latex-saildart/latex-2.0-1.0/manual/ には，LaTeX2.09 のマニュアル [5] のドラフト（1983 年版）が収録されています。

(7) 念のために補足しますが，ひとつのソースから複数の産出物を取り出せることを “Literate Programming” というわけではもちろんないです。勝手な解釈ですけれど，従来の「コードに部分的に解説コメントを付加する」というスタイルに対して，「コードとその解説ドキュメントとを融合させて，解説されるコードを解説ドキュメントの中に入れ込む」みたいなやり方が “Literate Programming” なのではないかと。はずしてたらすいません。

(8) memoir の “Appendix B” には次のようにあります：
“The LaTeX kernel is full of internal commands and a few are mentioned in Lamport. There is no place where you can go to get explanations of all the LaTeX commands, but if you run LaTeX on the source2e.tex file which is in the standard LaTeX distribution you will get the commented kernel code. The index of the commands runs to about 40 double column pages. Each class and package introduce new commands over and above those in the kernel.”
また，UK-TUG の “The UK TeX FAQ: Your 438 Questions Answered: version 3.19, date 2009-06-10” (http://www.tex.ac.uk/cgi-bin/texfaq2html?introduction=yes; CTAN にもあります）の “331 The definitions of LaTeX commands” の項目には，“The printed kernel is a nice thing to have, but it’s unwieldy and sits on my shelves, seldom used.” なんて書かれちゃってますけど，とりあえず索引込みでタイプセットして，あとは印刷はせずに pdf にでもしておけばいいのではないかと思います。
なお，「40 個ほどの .dtx ファイル」と言いましたが，具体的にどの .dtx ファイルが LaTeX のカーネルを構成しているのかについては，/base フォルダの中を見ればもちろん分かりますけど，“latex.ltx” 冒頭のソースファイル一覧ですとか，source2e.tex の “\DocInclude” の行の部分にも書いてあります。source2e.tex のほうには，

```
\DocInclude{ltplain} % LaTeX version of Knuth's plain.tex
```

という風に，ファイルについての簡単なコメントも書いてあります。

く沢山のドキュメントが含まれています)。

最後に，LaTeX 2εのユーザー向け文献は LaTeX のマニュアル [6] で，更に詳しい解説書が LaTeX Companion [15] です。LaTeX Companion については，「パッケージを沢山紹介している本」みたいに言われることがありますが，実際には LaTeX 2εの詳細マニュアルといえるものだと思います(9)。

⑤ pTeX

- ⓐ `ptex.exe` (?)
  ← `ptex-src-3.1.11.tar.gz` の中に入ってる何か
- ⓑ ASCII の HP(`http://ascii.asciimw.jp/pb/ptex/help/index.html`)；
  「日本語 TeX」[18]

pTeX のバージョンについては，アスキーの HP(`http://ascii.asciimw.jp/pb/ptex/base/intro.html`) に次のような表が掲載されています：

年	バージョン	トピック
1987	j1.0	最初のバージョン
1987	j1.1	NEC PC9801 シリーズ用の日本語 MicroTeX 発売
1990	j1.7 p1.0.9	縦組への対応
1995	p2.0	TeX 3 に対応
2002	p3.0	使用条件を BSD ライセンスに

同じページの説明に「とくに，縦組機能をサポートしているバージョンを，横組だけの日本語 TeX と区別し，pTeX (Publishing TeX) と呼んでいます。」とありますので，頭に「j」がついてるバージョンは，横組み専用の日本語 TeX だったのでしょう(10)。

pTeX のソースについては，私はまったく分かってません。アスキーの HP で配布されている `ptex-src-3.1.11.tar.gz` の中にある“`ptex-base.ch`”あたりが何か大事そうだな，と推測できる程度です。すいません。

pTeX のドキュメントについては，アスキーの「pTeX のプリミティブとレジスタ」というページ (`http://ascii.asciimw.jp/pb/ptex/help/index.html`) に，文字通り pTeX で追加されたプリミティブとレジスタについての説明が載っています。

pTeX の`/doc/ptex/`フォルダの中にはアスキーの方が書かれたドキュメントが入っていますが(11)，その中に「日本語 TeX」[18] という文書 (`jtexdoc.tex`) があって，こちらにも pTeX のプリミティブについての説明が書いてあります。

⑥ pLaTeX2.09

- ⓐ `jlplain.tex`, `hyphen.tex`, `jlfonts2.tex`, `latex.tex`, `kinsoku.tex`
  (→ `jlatex209.fmt`)；
  `plplain.tex`, `hyphen.tex`, `plfonts.tex`, `latex.tex`, `platex.tex`, `kinsoku.tex`
  (→ `platex209.fmt`)
- ⓑ ⓐの各ソースファイルの中に書かれたコメント;
  各種“`.doc`”ファイル;
  pLaTeX2.09 のマニュアル [7]

アスキーの「旧バージョンの pLaTeX ソースファイル」というページ (`http://ascii.asciimw.jp/pb/ptex/base/platex-obsolete.html`) に置いてある `platex209.tar.gz` には jLaTeX2.09 と pLaTeX2.09 が収録されていますが，その中心となるファイルは，⑥-ⓐに挙げたものです。縦組み対応の pLaTeX2.09 のほうには jLaTeX2.09 にはない“`platex.tex`”というファイルが含まれていますが，これは pLaTeX 2εにおける“`plext.sty`”(12)に相当するものだと思います（つまり，タテ組み関連のマクロ集です)。

pLaTeX2.09 の場合も，LaTeX2.09 同様，ソースファイルの中にコメントでマクロの説明が（英語で）書かれています。また，スタイルファイルの解説は，オリジナルの LaTeX2.09 付属の `.doc` ファイルがそのまま収録されているようですが，“`jreport.doc`”と“`jrep10.doc`”だけは，ちゃんと日本語で書かれています。

pLaTeX2.09 のユーザーマニュアルが文献 [7] なのかどうかは，実はよく分かりません。文献 [7] よりも前にもマニュアルはあったんだと思いますが，未見です。

⑦ pLaTeX 2ε

- ⓐ `platex.ltx` (`plcore.ltx` + `latex.ltx`)
  〔← `plvers.dtx`, `plfonts.dtx`, `plcore.dtx`〕
  (→ `platex.fmt`)
- ⓑ `pldoc.tex`
  (← `plvers.dtx`, `plfonts.dtx`, `plcore.dtx`, `plext.dtx`, `pl209.dtx`, `kinsoku.dtx`, `jclasses.dtx`, `jltxdoc.dtx`);
  pLaTeX 2εのマニュアル [8]

ようやく pLaTeX 2ε まで辿り着きました。pLaTeX 2ε に同梱されている“`platex.dtx`”に，pLaTeX 2εの概要が説明されていますが，それを読みますと，`platex.fmt` を作るための `platex.ltx` は，`plcore.ltx` と `latex.ltx` から作られ，そして，`plcore.ltx` は，`plvers.dtx`,

(9) 実のところ，LaTeX のマニュアル [6] と，LaTeX Companion [15]，それに UK-TUG の FAQ 集 [前掲註 (8)] があれば，文系初級レベルの疑問なら，8 割くらいは解決するような気もします。

(10) 文献 [7] は 3 冊組になっていますが，そのうちの 1 冊『縦組リファレンスガイド』の冒頭に次のような説明があります：
「現在，この日本語化には 2 つのバージョンがあります。1 つは，オリジナルの英語版 TeX システムで横組みの文章がきれいに組版できるようにしたバージョン，もう 1 つは縦組/横組の混在した市販の書籍や新聞，雑誌などを意識したバージョンです。
（1）日本語 TeX version 2.99 j1.7
（2）日本語 TeX version 2.99 j1.7 – p1.0.9G
後者には『出版 (publish)』を意味する『p』のバージョンが明記されており，日本語特有の処理や縦組を行う際に必要と思われる機能が『プリミティブ (primitive)』として，つまりマクロではなく組み込み型のコマンドとして新規にいくつか追加されています。」

(11) 環境によってはフォルダが異なるかも知れませんけれど，`ptex-texmf-2.5.tar.gz` だと，`/doc/ptex/` です。そこには，“`jtexdoc.tex`”の他には，〔表題のない〕“`jfm.tex`”，“`jtex.tex`”〔倉沢良一「TeX システムの日本語化」(昭和 62 年 3 月)〕，“`ptexdoc.tex`”〔濱野尚人・田村明史・倉沢良一「TeX の出版への応用」〕，“`ptexskip.tex`”〔中野賢「pTeX2.1.5 における数式の前後」(1997 年 8 月 8 日)〕というファイルが収められています。

(12) (La)TeX に関係するファイルやコマンドにはよく，“e” (e.g.: `\edef`, Eplain, epic) ですとか“x” (e.g.: `\xdef`, tabularx, xcolor, `dvipdfmx`) とかが付いていますが，これらはいずれも“*extended*”とか“*expanded*”という意味ですよね。

plfonts.dtx, plcore.dtx の 3 ファイルから作られるようです。

フォーマットファイルを作るための 3 つの.dtx ファイルの他に，pLaTeX 2ε を構成している.dtx ファイルが，5 つあります（plext.dtx, pl209.dtx, kinsoku.dtx, jclasses.dtx, jltxdoc.dtx）。これらの解説ドキュメントを参照するには，個々の.dtx ファイルをそれぞれ pLaTeX で処理してもいいのですが，LaTeX 2ε における source2e.tex に相当する “pldoc.tex” というファイルが用意されています。pldoc.tex を使うと，全体のコマンドインデックスも作れますので，検索する際にとても便利です。また，LaTeX 2ε の場合には，source2e.tex は classes.dtx は読み込みませんが，pldoc.tex のほうは jclasses.dtx も読み込みますので，日本語の標準クラスファイルの解説ドキュメントもまとめてタイプセットすることができます（変更履歴とコマンドインデックス込みで 150 ページほどのドキュメントです）。

pLaTeX 2ε のユーザーマニュアルは文献 [8] です。

2 いくつかの基本事項

ちょっと前置きが長くなり過ぎました。ここからが，第 II 部のための準備です。

① カテゴリーコードとか\def とかボックスとかの話

よく，「TeX では \, {, }, $, &, #, ^, _, %, ~ の 10 個の文字は特別な意味を持つのでそのままでは出力できません」というような説明が，入門書の冒頭に書かれていると思います[13]。これが「カテゴリーコード」と関係しています。

ここでは話を簡単にするために，英語の場合についてだけを考えます。TeX では，文字は 0〜15 の「カテゴリー」に分類されています。そして，通常出力できる文字は，カテゴリーコードが 11 の文字（英文字 “letter”）と，12 の文字（その他の文字 “other”）だけです。「英文字」は，a〜z，A〜Z の文字で，「その他の文字」は，上の 10 個の文字と英文字とを除いた，通常キーボードから入力できるその他の文字（数字を含む）と考えていいと思います（あっ，でもスペースは除きます）。

ということは，上に挙げた 10 個の文字のカテゴリーコードは，11 でも 12 でもないということです。実際，この 10 個の文字のカテゴリーコードは次のようになっています：

\	{	}	$	&	#	^	_	%	~
0	1	2	3	4	6	7	8	14	13

この，デフォルトでの文字とカテゴリーコードとの対応表から，カテゴリーコードの意味も推測できます。つまり，カテゴリーコード 0 が「エスケープ文字」，1 が「グループ開始文字」，2 が「グループ終了文字」，3 が「数式モードへの出入りの文字」，4 が「表組みやタビングの際の区切り文字」，6 が「パラーメータ文字」，7 が「上付き文字」，8 が「下付き文字」，14 が「コメント文字」で，そして 13 が「アクティブ文字」というわけです（「アクティブ文字」というのが聞き慣れないかも知れませんが，カテゴリーコード 13 の文字は，“\” なしでコマンドになれる文字で，デフォルトでは “~” のみです）。カテゴリーコード 5, 9, 10, 15 が出てきてませんが，これらはひとまず初級段階では気にしなくてもいいです（実は「スペース」のカテゴリーコードは 10 です）。

それで，多くの場合，文字のカテゴリーコードを変える必要というのは（少なくとも初級の間は）滅多にないですし安易に変更すると大抵困ったことになってしまうのですけど，カテゴリーコードを変える場合には，\catcode というプリミティブを使います：

```
\catcode ⟨charcode⟩ = ⟨number⟩
```

⟨charcode⟩ は，対象文字の「文字コード（数字）」で，⟨number⟩ は「カテゴリーコード」の数字です。例えば，文字 “a” の文字コードは 97 なので，文字 “a” のカテゴリーコードを（デフォルトの 11 から）13 に変えるとすると，

```
\catcode 97 = 13
```

とします。

ここで，TeX における整数の扱いについて簡単にまとめておきます。TeX では，10 進数だけでなく，8 進数や 16 進数も扱うことができます。8 進表記にする場合には数字の前にシングルクォート（’）を置いて，16 進表記ならダブルクォート（"）を前置します。したがって，今の例も，

```
\catcode ’141 = ’15
\catcode "61 = "D
```

のように書くこともできます。さらには，便利なことに，TeX では文字の前にバッククォート（‘）を置くと，その文字の文字コードを表わすことになります。ですから，例えば私がさっき，「文字 “a” の文字コードは 97 なので，…」と書いたときも，“a” の文字コードを知るためには実は文字コード表を見たのではなくって，原稿には次のように書いています：

```
文字‘‘a’’の文字コードは\number ‘a なので，…
```

\number というプリミティブを使っていますが，これは，数値を 10 進表記で出力するものです。したがって，次の例は，いずれも，“97” と出力されます：

```
\number ’141
\number "61
\number ‘a
```

カテゴリーコードが 11 以外の文字の場合には，文字の前にまずバックスラッシュ（エスケープ文字）を置いて，その前にバッククォートを置きます。バックスラッシュはカテゴリーコードが 11 の文字の前に置いた場合には特に何もしませんので，常にバッククォートとバックスラッシュをセットにして文字に前置しても構いません。

\char というプリミティブは，指定した文字コードの文字を出力するプリミティブですが，以上を踏まえますと，次の例はいずれも文字 “a” を出力するということが分かります：

```
\char 97
\char ’141
\char "61
\char ‘a
\char ‘\a
```

更に，\chardef というプリミティブを使うと整数に別名を付けることができます。ltdirchk.dtx で，

(13) また，「<, >, |」は数式モードで使わないと「¡, ¿, — 」となってしまう，ということも書いてあるかと思いますが，これはエンコードが “OT1” のとき（の cmr フォント）のことなので，カテゴリーコードとは別の話です。

```
\chardef\active=13
```

と定義されているため，整数 13 には “\active” という別名が付けられています[14]。文字のカテゴリーコードを 13 にするとアクティブ文字にすることができて，上級者の方はよく使われるので，13 という数字に特に意味を持たせているわけです。以上をまとめますと，最初の「文字 “a” のカテゴリーコードを 13 に変更する」という例（\catcode 97 = 13）は，次のようにも書けることが分かると思います：

```
\catcode `\a = \active
```

これだといかにも「文字 “a” を active にする」という感じが出てますよね（これは飽くまで例ですので，“a” というコマンドを作っても意味はないのですけど，もしも本当に “a” というコマンドを作るのであれば，まず a を active にした後で，\def a{...}という感じに，コマンド a〔バックスラッシュはつきません〕の定義をすることになります）。

さて，\chardef の話が出たついでに，カーネルやクラスファイルでよく使われる，@が付いた定数についてもまとめておきます：

コマンド	値
\m@ne	−1
\z@	0
\@ne	1
\tw@	2
\thr@@	3
\sixt@@n	16
\@xxxii	32
\@cclv	255

コマンド	値
\@cclvi	256
\@m	1000
\@M	10000
\@Mi	10001
\@Mii	10002
\@Miii	10003
\@Miv	10004
\@MM	20000

左側の表の \m@ne と \z@を除いたものが \chardef で別名が付けられていて，右側の表のものは皆 \mathchardef で名前が付けられています（\m@ne は \count22 の別名でその値が “−1” にしてあります．\z@ はディメンションでその値を “0pt” にしてあって，数値が必要な文脈でディメンションが使われた場合には sp 単位の整数値に換算されるという仕組みで “数値 0” としても使用されます．カウンタやディメンションについては次の節で扱います）。

\chardef や \mathchardef は整数しか扱えないこともあって，以下のものは ltfssbas.dtx で，マクロとして定義されています：

コマンド	値
\@vpt	5
\@vipt	6
\@viipt	7
\@viiipt	8
\@ixpt	9
\@xpt	10

コマンド	値
\@xipt	10.95
\@xiipt	12
\@xivpt	14.4
\@xvii	17.28
\@xxpt	20.74
\@xxvpt	24.88

@が出てきたので，次は \makeatletter と \makeatother の話に移りましょう。これらは ltdefns.dtx で以下のように定義されています：

```
\def\makeatletter{\catcode`\@11\relax}
\def\makeatother{\catcode`\@12\relax}
```

これまでの説明で意味は大体分かると思いますが，実はイコール（=）はいつもオプションなので，この例のようにイコールは入れなくても構わなくて，あと，\makeatletter や \makeatother の次に数字が続いた場合に，11 や 12 じゃない数字になってしまうのを防ぐために，11 と 12 の後ろに “\relax” が入れてあります（多分）。

それで，つまりは “@” のカテゴリーコードを 11 にしたり，12 に戻したりしているわけですが，それは，コントロールシーケンスに含まれる文字列は，カテゴリーコードが 11 の文字でないといけないからです。

そうなると，TEX におけるマクロの定義の仕方についても見てみないといけません。マクロを定義する場合の文字の並び順を，変テコな色付きの箱で描いてみました（色自体には意味はありません．下付きの数字はカテゴリーコードのつもりです）：

ここでオレンジの部分は，紫の部分のプレフィクスで，

- \long
- \global
- \outer

というプリミティブのどれかか，又はその組み合わせ（順番は問いません），あるいはオレンジの部分がない場合もあります（個々のコマンドの説明は後回しにします）。次の紫の部分は \def に相当するコマンドです。デフォルトでは，

- \def
- \gdef
- \edef
- \xdef

というプリミティブのいずれかです（これらも説明は後ほど）。そして，赤い箱を 2 つ並べましたが，ここが定義されるコントロールシーケンスのつもりです。カテゴリーコード 0 のエスケープ文字の次に，カテゴリーコード 11 の文字が続いた場合を図示したつもりです。実は，コマンドになれるものというのは，次のいずれかです：

- カテゴリーコード 0 の文字の次に，カテゴリーコード 11 の文字が 1 個以上続く文字列（コントロール・ワード）
- カテゴリーコード 0 の文字の次に，カテゴリーコード 11 以外の文字が 1 個だけ続くもの（コントロール・シンボル）
- カテゴリーコード 13 の文字（アクティブ文字．この場合はカテゴリーコード 0 の文字を前置しません）

このために，カテゴリーコードが 12 である@や数字は，コントロール・ワードに含ませることができないわけです。カテゴリーコードが 12 の文字であっても，コントロール・シンボルにはなれるので，\@ や \5 というマクロは定義可能ですし（\@は実際ありますね），他にも “\,” とか，あとは “\%”，“\$” のような「特殊文字を出力するコントロール・シンボル」はおなじみだと思います。

(14) いえ，“\chardef” という名前のプリミティブなのですから，実は \active は \char13 の別名です。ですから，原稿の中で \active と書くと，（そのときのエンコーディングでの）文字コード 13 に相当する文字が出力されます。更には，コントロールシーケンスの意味を出力するプリミティブ \meaning を使って “\meaning\active” としてみると，“\char"D” と出力されます（\meaning の出力は 16 進表記なので，"D は 10 進表記の 13 です）。でも，0〜255 の整数に別名を付けるのにも，この \chardef は使われます（0 以上の 256 個の整数ということからも，\chardef は元々文字コードに別名をつけるためのものだと分かりますね）。“\number\active” も，“\the\active” もいずれもちゃんと “13” と出力されます。なお，255 より大きい整数に別名をつけるには，同様に \mathchardef が流用 (?) されます。

コントロール・ワードに@を含ませることができるようにするためには，@の文字コードをデフォルトの 12（“other”）から 11（“letter”）に変更すればいいわけで，それが \makeatletter（*make-@-letter*）がやっていることで，その逆が \makeatother（*make-@-other*）です。

続く茶色の部分は，引数の指定（パラメータ文字と数字）で，それにカテゴリーコード 1 のグループ開始文字（緑色），マクロが展開された際の置換文字列（赤色）と来て，そして最後がカテゴリーコード 2 のグループ終了文字（緑色）です。

例えば，カテゴリーコードの割り振りがデフォルトのママの状態で，頭に \long を置いて，\def を使って，引数をひとつ取る “\command” というマクロを定義するとすると，次のようになります：

\long \def \ 0 command 11 # 6 1 12 { 1 ⟨*text*⟩ } 2

ここで “⟨*text*⟩” と書いた部分が，\command が展開された際の置換文字列です。引数を取れるように定義するのであれば，普通はこの置換文字列の中で “#1” が使われることになります（置換文字列の部分で “#1” を使わないと，引数はただ捨てられてしまうことになります）。

定義されるマクロが引数を取らないのならば，茶色の部分は不要で，逆に，複数の引数を取るマクロの場合には，引数は 9 個まで指定できます（#1～#9）。引数を指定する茶色の部分では，引数は #1 から順に並べないといけませんが，置換文字列の中では，引数を使う順番や実際に使うか否かは問いません。

また，\def（やその仲間）による引数指定の部分には，「パターンマッチ」という機能があります。これが LaTeX の \newcommand にはない \def の強みです。例えば \def\commandA#1#2#3{...} という定義の場合には，\commandA を使う際には，\commandA{a}{b}{c} みたいにただ引数を並べることになりますが，この引数指定の部分に，区切り文字（デリミタ）を含ませることができるのです。つまり，例えば，\def\commandB#1[#2]#3 のように定義した場合には，\commandB の次の文字から “[” の前までがひとつ目の引数で，次に “[” から “]” までの間が二つ目の引数，そして “]” の後ろが三つ目の引数，ということになります。そして，\commandB を使う際には，このパターン通りに引数を指定しないと，エラーになります。

さて，\def に似たコマンドとして，\let というプリミティブがあります。これは，

```
\let⟨cmda⟩=⟨cmdb⟩
```

のような形で使って，\let された時点での ⟨*cmdb*⟩ の意味を，⟨*cmda*⟩ にコピーするというものです。コピーされるのは「\let された時点での」⟨*cmdb*⟩ の意味なので，仮にその後 ⟨*cmdb*⟩ が再定義されたとしても，⟨*cmda*⟩ は，⟨*cmdb*⟩ の再定義前の意味のママです。

ここで，⟨*cmda*⟩ は，元々存在していても，または存在していなくてもどちらでも構いません。⟨*cmda*⟩ が元々存在していたのであれば，\let の時点で ⟨*cmda*⟩ は ⟨*cmdb*⟩ の意味に上書きされ，もしも ⟨*cmda*⟩ が存在していなかった場合には，新たに ⟨*cmda*⟩ が作られて ⟨*cmdb*⟩ のコピーとなります。

\let を使えば，コマンドのコピーが作れてしまうわけですから，それで，さきほど「…紫の部分は \def に相当するコマンドです」というような書き方をしました。例えば，

```
\let\COMMAND=\def
```

とすると，以後は \COMMAND は \def とまったく同じ働きのコマンドとして使えます。また，この \COMMAND を使って \def を再定義してしまえば，\def はもう，\def ではなくなってしまいます（おかしな言い方ですけど．TeX ではプリミティブの意味まで再定義可能となっています）。

既に存在しているコマンドの機能を別のコマンドにコピーするとか，未定義エラーを避けるために一旦 \relax とか \@empty とかにコマンドを \let しておくというような場合の他に，\let がよく使われるのは，コマンドを再定義するとき（名前を変えずに機能を変えたい場合）です。元のコマンドの機能を一旦別の名前で保存しておいて，それから再定義することが多いですし，更には，再定義の中で，保存しておいた元の機能を使う，ということもよくあります。

どういうことかといいますと，例えば，“®” という記号を出力する \textregistered というコマンドがありますけど，これを，\textregistered という名前のままで，上付きにするように変更したいとします。この場合，次のようにするのが常套パターンです：

```
\let\originaltextregistered\textregistered
\def\textregistered{%
   \textsuperscript{\originaltextregistered}}
```

ちょっと長ったらしいコマンドを例に取ってしまったせいで分かりづらいかもしれませんが，まず，最初の行では，\textregistered を \originaltextregistered という名前でコピーしています（@を使える状態でならば，\orig@textregistered とかとしたほうが，見やすくなるでしょう）。次の 2 行では，\textregistered を，\originaltextregistered を使って再定義しています。このように再定義をした後では，\textregistered は，再定義前と同じ名前のママで，“®” と上付きで出力されるようになるわけです。

それでは，ここで，さきほど先送りしておいたオレンジの部分と紫の部分に入るプリミティブについて簡単に見ておきます。

\long というプレフィクスは，置換文字列の部分に空行や \par が含まれることを許可します。\long がない状態では，マクロの置換文字列の部分に複数の段落が含まれるとエラーになるようになっています。LaTeX の “\newcommand” の定義には元々 \long が含まれているので，\newcommand の場合には，置換文字列の部分が複数の段落になっても大丈夫です。逆に，LaTeX で置換文字列の部分に複数段落を許さない形で定義をするには “\newcommand*” を使います。次の \global は，定義がグループを抜けても有効になるようにします。通常，\def による定義に限らず，TeX での代入などはそのときどきのグループにローカルなので，グループを出てしまうとなくなってしまう（か元に戻る）のですが，\global を前置すると，当該グループの外に出ても有効なママになるようにしてくれます。そして，\outer というのは，これを前置して定義したコマンドが，他のコマンドの引数になることを禁止するものです。

\gdef は “\global\def” の省略形で，\xdef は “\global\edef” の省略形です。それでは，\edef って何かというと，これはちょっと私のような文系初級には難しいです。頑張って説明してみますと，\edef は，置換文字列の部分にマクロなど展開可能なものが含まれている場合には，\edef とした時点ですべて展開し切ってから定義をします。逆にいうと，\def で定義されたマクロの場合には，マクロが展開されるのは，定義時ではな

くて，その定義されたマクロが使われた時点であって，しかも，置換文字列の部分のマクロは一気に展開し尽くされるのではなくて，一皮ずつ剥いていく感じで順番に置き換えられていくだけです。

あまり的確な例が浮かばないのですけど，次のようなマクロを考えてみます：

```
\newcommand{\alphabet}{}
\newcommand{\append}[1]{%
  \edef\alphabet{#1 \alphabet}}
```

なんとも面白くない例で申し訳ありませんが，このように定義してから，

```
\append{a}\alphabet\par
\append{b}\alphabet\par
\append{c}\alphabet\par
```

を実行すると，次のような結果になります：

```
a
b a
c b a
```

何をやってるのかといいますと，3 行目の

```
\edef\alphabet{#1 \alphabet}
```

という部分で，\append が実行される度に，\append の引数と，その時点での\alphabet を展開し切ったものとを，\alphabet に格納するように定義をしています。ここで \edef の代わりに \def を使ってしまうと，\alphabet は \alphabet に置換されるので，延々終わりがなくて，TEX に “! TeX capacity exceeded, sorry” と言われてしまいます。1 行目で \alphabet を定義しているのは，\append を最初に使ったときに \edef が展開する \alphabet が存在してないといけないからです（3 行目で \edef でなく \def にした場合には，\alphabet は結局 3 行目で再定義されることになるので，1 行目の空引数に展開されたりはしません．なお，\par は改段落をさせるために入れただけなので，ここでは特に意味はありません）。

さて，なんかちょっと難しくなって来てますけど，次は \csname と \endcsname というプリミティブのペアの話に移ります。この二つは，

```
\csname ⟨string⟩\endcsname
```

のような形で使って，⟨string⟩ をコントロールシーケンスにしてくれるというものです。⟨string⟩ の部分にはコントロールシーケンスが含まれていても構わないのですが，展開できるものでないとダメです（ですから，ここの文字列には，プリミティブが含まれていたりとか，プリミティブを含む文字列に展開されるマクロとかは，不可です）。

例えば ⟨string⟩ が “command” という 7 文字だとした場合（⟨string⟩ には展開可能なものが含まれていないというケース）には，“\command” というコントロールシーケンスが元々存在していたり，予め定義してあれば，それが実行されます（未定義の場合には \relax になります）。

これらを使うと何が嬉しいかというと，ひとつには，ここで ⟨string⟩ という文字列を構成する文字のカテゴリーコードは 11 でなくてもいいということと，もうひとつには，動的にコントロールシーケンスを作り出せるということです。

まずは，カテゴリーコードについてですが，先ほど，コントロール・ワードにはカテゴリーコード 11 の文字しか含めることができず，“@” のようなカテゴリーコード 12 の文字をコンロール・ワードに混ぜるためには，そのカテゴリーコードを 11 に変更しなくてはいけない，と書きました。でも，\csname と \endcsname の間に挟む文字列については，カテゴリーコード 12 の文字が含まれていても構わないのです。つまり，例えば，ただ “\s@mple” と書いた場合には，通常のルールにしたがって “\s” と文字列 “@mple” に分離してしまいますが，\csname と \endcsname を使えば，

```
\csname s@mple\endcsname
```

は “\s@mple” と同じ意味になります（この場合には，\makeatletter を使って “@” のカテゴリーコードを変更しなくてもいい，ということです）。でも，\s@mple というコントロールシーケンスは未定義なので，このままだと，“\csname s@mple\endcsname” は “\relax” になります。ところが予め，例えば，

```
\makeatletter
\def\s@mple{sample}
\makeatother
```

とでも\s@mple が定義してあれば：

\csname s@mple\endcsname
↓
\s@mple
↓
sample

となります。以上をまとめてみますと，例えば，

```
1  [\csname t@st\endcsname]
2
3  \makeatletter
4  \def\t@st{test}
5  \makeatother
6
7  [\csname t@st\endcsname]
8
9  \makeatletter
10 *\t@st*
11 \makeatother
```

とすると，その結果は次のようになります：

```
[]
[test]
*test*
```

1 行目と 7 行目はまったく同じですが，1 行目の時点では \t@st は未定義なので，出力は “[]” となり，3〜5 行目で \t@st を定義しているので，7 行目の出力は “[test]” となります。なお，\csname と \endcsname を使わない場合には，\makeatletter で “@” のカテゴリーコードを変えないと，\t@st は使えません（9〜11 行目）。

以上では “@” を例にとったので，\makeatletter を使えば “@” を含んだコントロールシーケンスを定義できましたが，“@” 以外の，カテゴリーコードが 11 でない文字を含んだコントロールシーケンスを定義するには，どうしたらよいのでしょうか（ひとつひとつ \catcode でカテゴリーコードを変更する?）。

このような要請に対応できるように，LATEX では “\@namedef” というコマンドが用意されています。

\@namedef は ltdefns.dtx で，次のように定義されています（1 行に収まらなかったので改行しています）：

```
\def\@namedef#1{%
   \expandafter\def\csname #1\endcsname}
```

“\expandafter” というまったく初級向きでないプリミティブが出てきてしまいましたが，ここでも頑張って説明してみます。\csname と \endcsname で作ったコントロールシーケンスを定義しようとした場合，素朴に考えると，

\def\csname ⟨*string*⟩\endcsname{⟨*text*⟩}

とでもなりそうですが，これだと「\csname を定義する」ことになってしまいますので，うまくいきません。望みどおりの結果を得るには，\def よりも先に，まず色をつけた部分（\csname ⟨*string*⟩\endcsname）のほうを実行して，それから \def が働いてくれないといけません。こういうときのために用意されているプリミティブが，\expandafter です[15]。ここでは \def の前に \expandafter を置くと，\def を一旦留保して，次の “\csname ⟨*string*⟩\endcsname” を実行し，それから，その実行結果の前に \def を戻します。これでめでたく

\def\⟨*string*⟩{⟨*text*⟩}

と定義できることになります。\@namedef が行っているのはこの前半部分ですので，実際に \@namedef を使う場合には，

\@namedef{⟨*string*⟩}{⟨*text*⟩}

のようにすることになります。なお，逆に，“\⟨*string*⟩” を使う場合の “\csname ⟨*string*⟩\endcsname” についてですが，このままでは随分と長いので，LaTeX では，“\@nameuse” というコマンドにしてあります（ltdefns.dtx より）：

```
\def\@nameuse#1{\csname #1\endcsname}
```

実際に使う場合には，

\@nameuse{⟨*string*⟩}

という風になり，\@nemeuse の引数にはバックスラッシュは付けません。

\@namedef と \@nameuse には，コマンド名に既に “@” が入っちゃってるので，通常の原稿の中で使う場合には，結局 \makeatletter が必要になります（ですから，元々カーネルやクラスファイルの内部で使うために用意されているのでしょうね）。

続いて，\csname と \endcsname を使えると嬉しいのは，動的にコマンドを作れるという点です。次のような例を考えてみます：

```
\newcommand{\grade}[2]{%
   \csname #1point\endcsname{#2}}

\newcommand{\excellentpoint}[1]{!#1!}
\newcommand{\goodpoint}[1]{*#1*}
\let\fairpoint\goodpoint
\let\passingpoint\goodpoint
\newcommand{\failurepoint}[1]{#1...}
```

1〜2 行目では，引数を 2 つとる “\grade” というマクロを定義しています。ここで 1 つ目の引数は “excellent”, “good”, “fair”, “passing”, “failure” のいずれか 1 つに決まっているということにして，2 つ目の引数は，100 点満点での点数ということにします。

\grade の 1 つ目の引数は，置換文字列の部分で “\csname #1point\endcsname” に渡されますので，1 つ目の引数の文字列に応じて，この部分は “\excellentpoint”, “\goodpoint”, “\fairpoint”, “\passingpoint”, “\failurepoint” に展開されます。そして，4〜8 行目では，これら 5 つのコマンドを，それぞれ引数を 1 つとるマクロとして定義しています。つまり，\grade の 2 つ目の引数は，下請けの “\excellentpoint” 等々の必須引数ということになります。こうしておいてから，

```
\grade{excellent}{95}\par
\grade{good}{80}\par
\grade{fair}{70}\par
\grade{passing}{60}\par
\grade{failure}{40}\par
```

を実行してみますと，次のようになります：

```
!95!
*80*
*70*
*60*
40...
```

こんなに単純な例では全然ありがたみを実感できないと思いますけど，第 II 部をご覧いただきますと，ページの「マーク」を設定する部分ですとか，「enumerate 環境」の設定部分とかに，\csname と \endcsname が使われています。

そろそろこの節ちょっと長くなり過ぎな気もしてきましたが，最後に memoir の “Appendix B” に沿って，「モード」と「ボックス」について，ごくごく簡単に見ておきます。

TeX はいつでも，次の 6 つのモードのどれかひとつにいます：

- 水平モード，限定水平モード
- 垂直モード，内部垂直モード
- 数式モード，ディスプレイ数式モード

うーんと平たくいうと，水平モードでは（普通は左から右へ）行を作っていて，垂直モードでは（普通は上から下へ）ページを，そして数式モードでは数式を，作っています。TeX は最初は垂直モードにいて，段落を開始するモノに出会うと水平モードに移行し，段落が終わるとまた垂直モードに戻ります。「水平ボックス」の中では限定水平モードになり，「垂直ボックス」の中では内部垂直モードになります。

水平ボックスと垂直ボックスを作るプリミティブは，それぞれ：

\hbox to ⟨*dimen*⟩ {⟨*text*⟩}
\vbox to ⟨*dimen*⟩ {⟨*text*⟩}

[15] 文系初級でも理解できる \expandafter の解説としては，Stephan v. Bechtolsheim, *A Tutorial on \expandafter*, TUGBOAT, Vol. 9 (1988), No. 1, pp. 57–61. (http://www.tug.org/TUGboat/Articles/tb09-1/tb20bechtolsheim.pdf) がお勧めです。

です（ホントは\vtop とか\vcenter とかというのもありますが，割愛）。ここで，“to ⟨*dimen*⟩” はオプションで，サイズ指定をする部分です[16]。\hbox の場合はヨコ幅のサイズで，\vbox の場合はタテの長さになります。サイズを指定しない場合には，ボックスの中身である ⟨*text*⟩ の自然なサイズになります（ホントは垂直ボックスの場合は幅の指定も\hsize とかでしなくちゃいけないけど，これまた割愛）。

LaTeX の “\mbox” や “\makebox” は \hbox を利用して作られていて，“\parbox” や “minipage 環境” は \vbox で作られています。このことから推測できるかも知れませんけど，\hbox の中では改行できません。ずーっとヨコに文字が続いていくだけです。サイズ指定をしても，そのサイズ以下だと “underfullhbox” といわれ，サイズを越えると “overfullhbox” といわれます。他方 \vbox の場合には，指定された横幅（指定がなければ行幅）で行が折り返されることになります（\vbox の場合も，指定のタテの長さに中味の量が合わなければやはり，“overfullvbox” だとか “underfullvbox” だと TeX にいわれます）。

なお，“\hbox to” は ltdefns.dtx で次のようにひとつにまとめられています[17]：

```
\def\hb@xt@{\hbox to}
```

ところで，先ほど「TeX は最初は垂直モードにいて，段落を開始するモノに出会うと水平モードに移行する」と言いましたが，\hbox は，段落を開始しません。ですから，TeX が垂直モードにいるときに

```
\hbox{あ}\hbox{い}
```

とすると，

```
あ
い
```

と出力されます。そこで，垂直モードを抜けて水平モードに明示的に移行する場合には，plain TeX 以来のマクロである

```
\leavevmode
```

を使います（LaTeX 2ε での定義は ltplain.dtx にあります）。\leavevmode は TeX が水平モードにいるときには何もせず，垂直モードにいるときには水平モードへと移行します。したがって，

```
\leavevmode\hbox{あ}\hbox{い}
```

とすれば，

```
  あい
```

となります（行頭には\parindent 分のアキが入っています）。

② レジスタとかの話

TeX には，変数を一時的に記憶しておいてくれる「レジスタ」というものがあります。入れておける変数の種類に応じて，

- \count レジスタ（\count0 〜\count255）
- \dimen レジスタ（\dimen0 〜\dimen255）
- \skip レジスタ（\dimen0 〜\dimen255）
- \muskip レジスタ（\muskip0 〜\muskip255）
- \box レジスタ（\box0 〜\box255）
- \toks レジスタ（\toks0 〜\toks255）

の 6 種類が，どれも 0 番〜255 番までの 256 個ずつ用意されています[18]。

「\count レジスタ」が記憶するのは整数で，「\dimen レジスタ」の場合はディメンション，「\skip レジスタ」ではグルー，「\muskip レジスタ」は数式グルーで，「\box レジスタ」が記憶するのはボックス，そして最後に「\toks レジスタ」の場合はトークン・リストというものを入れておくことができます。

TeX が扱える整数の範囲は，$\pm 2^{31}$ の間だそうで，つまり，-2147483647 から 2147483647 までとのことです。

ディメンションというのは，数値と単位を合わせたもので，簡単にいうと「長さ」です。オリジナルの TeX では，in, pc, pt, bp, cm, mm, dd, cc, sp, em, ex という 11 種類の単位が使えます（pTeX では他に zh, zw などもあります）。これらは「キーワード」なので，TeX がディメンションやグルーを予定している文脈で使われると，単なる文字列ではなく，「単位」とみなされます。なお，TeX が長さを期待しているところで，ただ “0” とすると，“! Illegal unit of measure (pt inserted).” と言われてしまうので，「長さ 0」を指定する際には，何でもいいので何か単位を添えないといけません。

TeX の内部では，ディメンションは sp という単位に換算されて扱われているそうです（$1\,\mathrm{sp} = 1/65536\,\mathrm{pt}$）。そして，TeX が扱えるディメンションの範囲は $\pm 2^{30} - 1\,\mathrm{sp}$ らしいので，これはポイント換算で $\pm 16384\,\mathrm{pt}$，メートル換算だと約 $\pm 5.7583\,\mathrm{m}$ の間ということになります。

グルーというのは，伸縮長を伴った，長さです。「⟨*dimension*⟩ plus⟨*dimension*⟩ minus⟨*dimension*⟩」という形になりますが，冒頭の “⟨*dimension*⟩” が，このグルーの自然長で，次の “plus⟨*dimension*⟩” が伸張部，最後の “minus⟨*dimension*⟩” が収縮部です（いずれもディメンション，つまり単位つき数値です）。なお，伸縮部には fil, fill, filll という単位 (?) も使えます。

ディメンションが期待されている文脈でグルーが使われると，伸縮部が捨てられます。逆に，グルーが期待されている文脈でディメンションが使われると，伸縮部が “plus 0pt minus 0pt” であるグルーとして扱われます。ちなみに，LaTeX の “\newlenght” で作られる長さは，ディメンションではなくて，グルーです。

---

(16) ここに出てきた “to” のようなものを，TeX ではなぜか「キーワード」と呼んでいます。特定の文脈でのみ，意味を持つ単語で，例えばこの to も，普通の原稿の中に出てくればもちろん “to” と出力されますが，\hbox など特定のプリミティブの後ろに来ると，サイズ指定子の働きをすることになっています。次のものが TeX の全キーワードです：at, bp, by, cc, cm, dd, depth, em, ex, fil, height, in, l, minus, mm, mu, pc, plus, pt, scaled, sp, spread, to, true, width。

(17) TeX では「トークン」という言葉が出てきますが，これは，個々の文字または，コントロールシーケンスのことだそうです。コントロールシーケンスの場合はいくら長い名前でも，1 トークンで，文字（スペースも含む）は，1 個 1 個が 1 トークンとのことで，それで，“\hbox to” を “\hb@xt@” にすると，スペースを含めて 3 トークン減らせるらしいのです。同様に，ltdefns.dtx では，\def\@height{height}, \def\@depth{depth}, \def\@width{width}, \def\@minus{minus}, \def\@plus{plus}という置き換えがされています。先にまとめておいた “@” 付きの定数についても，同じくトークン数を節約するために置き換えをしているわけです（例えば，“10000” は 5 トークンですが，“\@M” なら 1 トークンで済みます）。

(18) ε–TeX では 32768 個ずつに拡張されているらしいです。

整数が期待されている文脈でディメンションやグルーが現われると，sp 単位に換算された整数値として扱われます。グルーの場合には伸縮部が捨てられるというのは，ここでも同様です。

レジスタは 256 個ずつ用意されていますが，一部は (LA)TEX 自体が予約していますし[19]，更にはクラスファイルやパッケージなどもレジスタを使っていますので，ユーザーが勝手に「○○番の××レジスタ」という風にレジスタの番号を直接指定して使うと，既に使われているレジスタとバッティングしてしまう可能性があります。それに，番号を指定して使わなければならないとしたら，かなり使いづらいです（「えーっと，紙のヨコ幅は 83 番の \dimen レジスタに保存して，タテの長さは 137 番の \dimen レジスタ，それから，章番号には 200 番の \count レジスタを使おう」なんていうのは混乱の元です）。

それで，plain TEX 以来，未使用のレジスタに名前を付けて順番に割り当てていくというマクロがちゃんと，用意されています。「\count レジスタ」には “\newcount”，「\dimen レジスタ」なら “\newdimen”，「\skip レジスタ」の場合は “\newskip”，「\muskip レジスタ」には “\newmuskip”，「\box レジスタ」だったら “\newbox”，そして「\toks レジスタ」では “\newtoks” を使うと，それぞれ未使用のレジスタに，順番に別名を付けることができます（これらのマクロは LATEX 2ε では ltplain.dtx で定義してあります）。私の勝手な印象ですけど，レジスタに別名をつけると「パラメータ」と呼ばれることが多いような気がします。

ここで，例えば，

```
\newcount\testcount
\meaning\testcount
```

とかいう風に実験してみると，

```
\count89
```

のように出力されます。

何をしたかといいますと，\newcount を使って，試しに “\testcount” というパラメータを作ってみて，それでそれが何番の \count レジスタに割り当てられたのかを \meaning で確認しています。この場合だと，既に 88 番までの \count レジスタが使われていて，89 番の \count レジスタに “\testcount” という名前が付けられたということことが分かります（いろいろな環境により，実際の番号は変わってきます．これは飽くまで例です）[20]。

なお，\newbox については，同じように，

```
\newbox\testbox
\meaning\testbox
```

と実験してみると，

```
\char"29
```

という風に出力されます。つまり，上で作った “\testcount” は “\count89” の別名ですが，この “\testbox” は “\box41” の別名ではなくて，“\char41” の別名となっています。これは，\box レジスタの操作が，\box レジスタの番号を指定する方式になっているからです（例えば，41 番の \box レジスタの中身をコピーする場合には，“\copy\box41” とするのではなくて，“\copy41” と指定するので，“41” に別名を付ける必要があるわけです）。

\new... を使うとレジスタに別名を付けられるわけですが，LATEX では，一時使用のためのテンポラリなレジスタについても予め，いくつか名前を付けて用意しています[21]：

種類	テンポラリ・レジスタ
\count	\@tempcnta, \@tempcntb
\dimen	\@tempdima, \@tempdimb, \@tempdimc
\skip	\@tempskipa, \@tempskipb
\box	\@tempboxa
\toks	\@temptokena

これらには名前に “@” が入ってますし，カーネルやパッケージの内部などでかなり頻繁に使われているので，初級の間は安易に利用しないほうが無難かと思われます（どれも ltalloc.dtx で宣言されています）。

---

[19] 0 番から 22 番までの \count レジスタと，255 番の \box レジスタは (LA)TEX が使っています（例えば，22 番の \count レジスタの別名が “\m@ne” で，その値が “−1” だということは前に言いましたよね）。また，\count レジスタ以外のレジスタの 0 番から 9 番については，ユーザーが自由に使ってよいことになっていますが，初級レベルでは手を出さないほうが無難です。LATEX が用意してくれている “\newcounter” や “\newlength”，“\newsavebox” を使って新たにパラメータを宣言したほうが安全です（レジスタを余計に消費することにはなってしまいますが）。

[20] それでは，今この状態だと 90 番から 255 番の \count レジスタが空いているのかというと，必ずしもそうではありません。「インサート」という，脚註やフロートの一般形は，同じ番号の\count, \dimen, \skip, \box レジスタをセットにして使うので，新たなインサートごとに 254 番から順に，下にくだって番号が割り当てられることになっているからです。ltplain.dtx には次のような説明があります：

```
\count10=22 % allocates \count registers 23, 24, ...
\count11=9 % allocates \dimen registers 10, 11, ...
\count12=9 % allocates \skip registers 10, 11, ...
\count13=9 % allocates \muskip registers 10, 11, ...
\count14=9 % allocates \box registers 10, 11, ...
\count15=9 % allocates \toks registers 10, 11, ...
...
\count20=255 % allocates insertions 254, 253, ...
```

したがって，10，11，12，13，14，15 番の \count レジスタの値を調べれば，現在何番までの \count, \dimen, \skip, \muskip, \box, \toks レジスタが使用済みかが分かります。そして，20 番の \count レジスタの値を調べれば，上から何番までが使用済みかも分かるわけです。尤も分かったからといって，私のような初級ユーザーには格別使い途はないのですけれど…。

[21] 前掲註 (19) で，\count レジスタ以外のレジスタの 0 番から 9 番は自由に使っていいことになっていると言いましたが，これらのうちのいくつかにも，ltplain.dtx で別名が付けられています：\countdef\count@=255, \dimendef\dimen@=0, \dimendef\dimen@i=1, \dimendef\dimen@ii=2, \skipdef\skip@=0, \toksdef\toks@=0. つまり，\count255 の別名が “\count@” で，\dimen0 には “\dimen@”，\dimen1 は “\dimen@i”，\dimen2 には “\dimen@ii” という名前が付けられていて，そして \skip0 と \toks0 にもそれぞれ “\skip@”，“\toks@” というような別名が付けられています。

以下では，まず，\box レジスタと \toks レジスタの操作について簡単に見た後で，\count，\dimen，\skip レジスタの操作について見てみることにします（\muskip レジスタについては私は分からないので割愛します）。

\box レジスタが記憶するのは「ボックス」ですので，つまりは \hbox や\vbox などです。レジスタにボックスを代入するには，

\setbox⟨*register no*⟩ = ⟨*box*⟩

というプリミティブを使います（ここで ⟨*register no*⟩ は \box レジスタの番号または番号の別名）。

代入してある \box レジスタの中味を取り出すには，

\box⟨*register no*⟩

ですとか，

\copy⟨*register no*⟩

というプリミティブを使います。“\box” だと，中味を取り出すとレジスタが空になりますが，他方 “\copy” はコピーですので，コピー後もレジスタの中味は残ります（この他に，ボックスごと取り出したりコピーしたりするのではなくて，ボックスの中身だけを取り出したりコピーしたりする \unhbox, \unhcopy, \unvbox, \unvcopy なんていうのもありますけど，割愛）。いずれの場合も，\box レジスタを指定するには，その番号（⟨*register no*⟩）や別名を指定します。

ボックスの高さ，深さ，幅を取得するには，“\ht”，“\dp”，“\wd” という，プリミティブの ⟨*dimen*⟩ パラメータが用意されています。ボックスの指定にはやはり，その番号または番号の別名を指定します。

例として第 II 部の「3 フォント」という部分にある，次のような設定箇所を見てみましょう：

```
\setbox0\hbox{\char\euc"A1A1}%
\setlength\Cht{\ht0}
\setlength\Cdp{\dp0}
\setlength\Cwd{\wd0}
```

1 行目ではまず，「\hbox に “\char\euc"A1A1” という文字（全角の空白文字）を入れたもの」を，0 番の \box レジスタに代入しています（イコールはオプションなので，省略されています）。そして，予め別の場所で宣言してある \Cht，\Cdp，\Cwd という ⟨*dimen*⟩ パラメータに，\box0 の高さ・深さ・幅の値を代入しています。

これを LaTeX-ey に書き直してみますと，次のようになるかと思います：

```
\newsavebox{\charbox}
\sbox{\charbox}{\char\euc"A1A1}
\settoheight{\Cht}{\usebox{\charbox}}
\settodepth{\Cdp}{\usebox{\charbox}}
\settowidth{\Cwd}{\usebox{\charbox}}
```

0 番の \box レジスタを使う代わりに，1 行目で “\charbox” というボックスを宣言しました。そして 2 行目では，その \charbox に “\char\euc"A1A1” を入れて，3〜5 行目では，LaTeX の \settoheight，\settodepth，\settowidth を使って，\Cht，\Cdp，\Cwd に，\charbox の高さ・深さ・幅の値を代入しています。

続いて \toks レジスタについてですが，トークン・リストを代入するには，

\toks⟨*register no*⟩ = {⟨*token list*⟩}

とします。左辺は直接番号を指定せずに，別名の \toks パラメータでももちろんいいです。ここでイコールは相変わらずオプションですけど，⟨*token list*⟩ を囲んでいるブレースは省略できません。\toks レジスタに保存してある ⟨*token list*⟩ を取り出すには，

\the

というプリミティブを \toks レジスタに前置します（\the は，\toks レジスタだけでなく，〔\box レジスタを除いた〕ほとんどのレジスタやパラメータの値を出力するのに使えます）。

つまらない例しか思いつかないのですが，例えば，

```
\toks0={\TeX{} and \LaTeX}
This paper describes some fundamentals on \the\toks0.\par

\toks1={japanese grammar}
\toks0=\toks1
This paper describes some fundamentals on \the\toks0.
```

とでもしてみますと，

This paper describes some fundamentals on TeX and LaTeX.

This paper describes some fundamentals on japanese grammar.

という風になります。

さて次は，\count，\dimen，\skip レジスタの操作に移ります。これらのレジスタに値を代入するには，いずれについてもイコールの左辺にレジスタ，右辺に値を指定します（でも，イコールはやはりオプションです）。まず，\count レジスタから：

\count⟨*register no*⟩ = ⟨*number*⟩

ここで “\count⟨*register no*⟩” の部分は，直接 \count レジスタの番号を指定するのではなくて，別名を付けた \count パラメータでも，もちろん構いません。

ところで，「LaTeX のカウンタ」というのは，バックスラッシュがつかない形式でした。そのあたりがどうなっているのかというと，ltcounts.dtx で定義されている，LaTeX の \newcounter の下請けの \@definecounter の中味を見ると分かります（一箇所だけ改行を変えています）：

```
\def\@definecounter#1{%
   \expandafter\newcount\csname c@#1\endcsname
   \setcounter{#1}\z@
   \global\expandafter\let\csname cl@#1\endcsname\@empty
   \@addtoreset{#1}{@ckpt}%
   \global\expandafter\let\csname p@#1\endcsname\@empty
   \expandafter
   \gdef\csname the#1\expandafter\endcsname\expandafter
     {\expandafter\@arabic\csname c@#1\endcsname}}
```

ここで “#1” は，\newcounter の引数ですが，“⟨*counter*⟩” だということにしましょう。そうしますと，

- まず 2 行目では\newcount を使って “\c@⟨*counter*⟩” という \count パラメータが作られています。
- 4 行目では，“\cl@⟨*counter*⟩” が作られて \@empty に \let されています。
- 同様に 6 行目では “\p@⟨*counter*⟩” が作られ，\@empty に \let されています。
- そして，8〜9 行目では “\the⟨*counter*⟩” が作られて，それが “\@arabic\c@⟨*counter*⟩” に \gdef されています。

つまり，バックスラッシュなしの LaTeX の ⟨*counter*⟩ という名前のカウンタは，内部的には “\c@⟨*counter*⟩” という \count パラメータとして存在しているということです。

更に，\@definecounter の中では，この引数 ⟨*counter*⟩ を使って “\cl@⟨*counter*⟩”, “\p@⟨*counter*⟩”, “\the⟨*counter*⟩” という 3 つのコントロールシーケンスが同時に作られています。

\cl@⟨*counter*⟩ はカウンタの親子関係の設定に使われ，\p@⟨*counter*⟩ は相互参照の際の番号表示のプレフィクスに使われるものですが，\@definecounter での初期設定ではいずれもひとまず \@empty に \let されています。

\the⟨*counter*⟩ は，\c@⟨*counter*⟩ の出力形式ということになりますが，初期設定ではアラビア数字表記にされているということも分かります。

なお，「LaTeX のカウンタ」を \count パラメータに変換するためのコマンドも用意されていて，ltcounts.dtx で，

```
\def\value#1{\csname c@#1\endcsname}
```

と定義されています。“⟨*counter*⟩” は「LaTeX におけるカウンタ」の「名前」であって，その「値」にアクセスするためには ⟨*counter*⟩ の実体である “\c@⟨*counter*⟩” とする必要があるわけですが，それを “\value” というコマンドにしておいてくれています。

続いて，\dimen レジスタ，\skip レジスタに値を代入するには，\count レジスタ同様，

```
\dimen⟨register no⟩ = ⟨dimension⟩
\skip⟨register no⟩ = ⟨glue⟩
```

とします。ここでも “\dimen⟨*register no*⟩” のところは，\dimen パラメータの名前でもいいですし，同様に “\skip⟨*register no*⟩” は，\skip パラメータの名前でももちろん構いません。

なお，LaTeX であればこういうときは “\setlength” を使いますね。ちょっとまた，つまらない実験をしてみます：

```
1  \newdimen\testdimen
2  \testdimen=10pt \the\testdimen\par
3  \testdimen10mm \the\testdimen\par
4  \setlength{\testdimen}{20pt} \the\testdimen
5
6  \newskip\testskip
7  \testskip=5pt plus 2pt minus 3pt
8  \the\testskip\par
9  \testskip10mm plus 4pt minus 6pt
10 \the\testskip\par
11 \setlength{\testskip}{20pt plus 8pt minus 12pt}
12 \the\testskip
```

予想通りだとは思いますが，出力は次のようになります：

```
10.0pt
28.4527pt
20.0pt
5.0pt plus 2.0pt minus 3.0pt
28.4527pt plus 4.0pt minus 6.0pt
20.0pt plus 8.0pt minus 12.0pt
```

1 行目と 6 行目では，“\testdimen” や “\testskip” を宣言して，これらのパラメータに 2 行目と 7 行目ではイコールを使って値を代入し，3 行目と 9 行目ではオプションのイコールを省略して代入してみました。そして，4 行目と 11 行目では LaTeX の \setlength を使っています（なお，\setlength を使わない場合には，代入する値の最後に “\relax” を補わないとうまく代入できないときがあります）。

3 行目では，単位を pt ではなく mm で指定していますが，\the による出力は pt 単位に換算されたものとなります。9 行目のように，複数の単位を混ぜたりしても大丈夫です（\the による出力はやはり pt 単位となります）。

それではここで，

```
\testdimen=5pt plus 2pt minus 3pt \the\testdimen
```

としてみると，どうなるでしょうか？ こうなります：

```
plus 2pt minus 3pt 5.0pt
```

今度も予想通りでしたか？ \dimen レジスタにはディメンションしか入れられないので，その値としてグルーを指定すると，自然長の部分のみが \testdimen に保存されて，残りの伸縮長の部分はそのまま文字列として出力されてしまいます。そして，その後ろに，\testdimen の中味が \the によって出力されているわけです（なお，LaTeX の “\newlength” で作られる長さはディメンションではなくてグルーなので，このようなことにはなりません）。

この節の最後に，\count，\dimen，\skip レジスタの四則演算について見ておきましょう：

```
\advance ⟨count⟩ by ⟨number⟩
\advance ⟨dimen⟩ by ⟨dimension⟩
\advance ⟨skip⟩ by ⟨glue⟩
```

```
\multiply ⟨count⟩|⟨dimen⟩|⟨skip⟩ by ⟨number⟩
\divide ⟨count⟩|⟨dimen⟩|⟨skip⟩ by ⟨number⟩
```

足し算や引き算には \advance，掛け算には \multiply，割り算には \divide というプリミティブを使います。by はオプションです。

ここで ⟨*count*⟩ は \count レジスタないし \count パラメータのつもりで，同じく ⟨*dimen*⟩ や ⟨*skip*⟩ も，レジスタないしパラメータだと思ってください。

⟨*number*⟩ は整数です（正負どちらもアリです）。もしも小数を指定した場合には，小数点より前の部分が ⟨*number*⟩ になり，小数点以降の部分はそのまま文字列として出力されてしまいます。また，割り算の商では，小数点以下が切り捨てられます。

\count パラメータで試してみましょう：

```
1 \newcount\testcount
2 \testcount=3 \the\testcount\par
3 \advance\testcount by 7 \the\testcount\par
4 \multiply\testcount by 3 \the\testcount\par
5 \divide\testcount by 6 \the\testcount
6
7 \advance\testcount 2.4\par
8 \the\testcount\par
9 \divide\testcount 8 \the\testcount
```

出力は次のようになります：

```
3
10
30
5
.4
7
0
```

1 行目で \testcount を宣言して，2 行目でそのパラメータに 3 を代入，3 行目ではそれに 7 を足してその和が 10，4 行目ではそれに 3 を掛けてその積が 30，5 行目ではそれを 6 で割ってその商が 5 となっています。

7 行目では 2.4 を足そうとしていますが，しかし，“.4” がそのまま出力されていて，和としては “7” が出力されています。そして，9 行目では 7 を 8 で割っているので，その商の小数点より前の部分は 0 です。

整数が期待されている文脈で \dimen や \skip が現われると，sp 単位に換算された整数として扱われます：

```
\testcount=196608
\testdimen=1pt
\divide\testcount by \testdimen \the\testcount
```

3 行目では，整数をディメンションで割っているわけではなくって，整数 “196608” を，1pt を sp に換算した “65536” で割っていることになります。よって，この出力は “3” となります。

ディメンションやグルーの場合には，\multiply を使う以外に掛け算をする方法があります。レジスタやパラメータの前に，係数を付けることができるのです。この場合は，係数は整数でなくてもいいので，実数倍することが可能になります。但し，⟨*skip*⟩ の場合には，\multiply を使ったときには伸縮部も掛け算されますが，係数を付けた場合には（係数が整数であっても小数であっても）伸縮部は捨てられてしまいます。やってみましょう：

```
\testdimen=10pt
\testdimen=1.5\testdimen \the\testdimen

\testskip=4pt plus 2pt minus 3pt
\the\testskip\par
\multiply \testskip by 2 \the\testskip\par
\testskip=3\testskip\relax \the\testskip\par
\testskip=4pt plus 2pt minus 3pt
\testskip 1.5\testskip\relax \the\testskip\par
\setlength{\testskip}{1.5\testskip} \the\testskip
```

出力は次のようになります：

```
15.0pt
4.0pt plus 2.0pt minus 3.0pt
8.0pt plus 4.0pt minus 6.0pt
24.0pt
6.0pt
9.0pt
```

1 行目では \testdimen に 10pt を代入して，2 行目では，それを 1.5 倍しています[22]。なお，“\testdimen=1.5\testdimen” だなんて，頭の中がグルグル回ってしまいそうですが，\edef のときみたいに考えると，理解しやすいかも知れません。つまり，「その時点での \testdimen の値を 1.5 倍したもの（右辺）」を，左辺の「\testdimen というパラメータ」に代入しているということです。

4 行目では \testskip にグルーを代入して，5 行目で出力し，6 行目で \multiply を使って 2 を掛けると，ちゃんと伸縮部も 2 倍されています。ところが，7 行目で 3 を係数にすると，伸縮部が捨てられて，出力は “24.0pt” となっています。8 行目でもう一度グルーを代入し直してから，9 行目で 1.5 を掛けてみると，やはり伸縮部は捨てられて，出力は “6.0pt” となっています。10 行目では LaTeX の \setlength を使ってみました。

あー，あと，ディメンションにグルーを足したり引いたりした場合にも，グルーの伸縮部は無視されて，足したり引いたりするのに使われるのは，グルーの自然長の部分です。逆に，グルーにディメンションを足したりするときには，ディメンションは伸縮長が plus 0pt minus 0pt のグルー扱いになります。つまり，

```
\testdimen3pt
\testskip 4pt plus 2pt minus 3pt

\advance\testdimen\testskip \the\testdimen\par
\advance\testskip\testdimen\relax \the\testskip
```

とすると，出力は，

```
7.0pt
11.0pt plus 2.0pt minus 3.0pt
```

となります。

③ \if とかの話

「いくつかの基本事項」の最後の節です。条件判断なんて，まったくもって文系初級向きのトピックではないので，ものすごく端折ります。jarticle.cls の中味を眺めるだけあれば，\ifnum と \iftdir，あとは \ifcase とか \newif くらいで間に合うのですけど，一応一通り列挙だけはしておきます。

TeX における条件判断は，

```
\if...⟨condition⟩ ⟨true text⟩ \fi
```

か，または，

```
\if...⟨condition⟩
   ⟨true text⟩
\else
   ⟨false text⟩
\fi
```

という形をとります。

“\if...⟨*condition*⟩” の部分が真のときには ⟨*true text*⟩ が実行されます。また，二つ目の例のように \else 以下がある場合には，“\if...⟨*condition*⟩” の部分が真でないとき（つまり偽のとき）には，⟨*false text*⟩ のほうが実行されます。

一つ目の例のように \else 以下がない場合には，“\if...⟨*condition*⟩” の部分が偽のときには何もされません。逆に，二つ目の例で，⟨*true text*⟩ の部分がないこともあります。その場合には，“\if...⟨*condition*⟩” の部分が真のときには何もされず，偽のときには ⟨*false text*⟩ が実行されるということになります。

“\else” と “\fi” はいずれもプリミティブで，そして，“\if...” の部分は，以下のプリミティブのうちのどれかです：

- \if
- \ifcat
- \ifx
- \ifnum

[22] 実はちょっと，ズルをしています。どういうことかというと，TeX では長さは sp 単位で計算がされているので，pt 単位で計算するとキレイな値にならないことがあります。例えば，\testdimen=5pt として，これを 1.2 倍してみると，“6pt” ではなくて，“5.99998pt” と出力されます。ここでは，キレイな数値になる例を意図的に選びました。

- \ifodd
- \ifdim
- \ifhbox, \ifvbox, \ifvoid
- \ifeof
- \ifcase
- \ifhmode, \ifvmode, \ifmmode, \ifinner
- \ifture, \iffalse

(\ifhmode, \ifvmode, \ifmmode, \ifinner, \ifture, \iffalse の場合には，⟨condition⟩ の部分はありません．)

また，pTEX の場合には，更に以下のプリミティブが追加されています：

- \iftbox, \ifybox
- \iftdir, \ifydir, \ifmdir

(こちらについても，\iftdir, \ifydir, \ifmdir の場合は，⟨condition⟩ の部分はないです．)

…やっぱり，これを全部扱うのは荷が重過ぎるので，以下では第 II 部で必要なものだけにしておきます。すいません。

まず，\ifnum は次のような形式で使われます：

```
\ifnum ⟨number1⟩ ⟨rel⟩ ⟨number2⟩
   ⟨true text⟩
\else
   ⟨false text⟩
\fi
```

⟨condition⟩ に相当する “⟨number1⟩ ⟨rel⟩ ⟨number2⟩” の部分で，整数 ⟨number1⟩ と整数 ⟨number2⟩ の関係をテストしています。⟨rel⟩ は，“<”，“=”，“>” のいずれかです。またまたつまらない例を考えてみました：

```
\newcounter{Number}
\newcommand{\checknumber}{%
  \ifnum \value{Number} > 0
    The ‘‘Number’’ is positive.
  \else
    The ‘‘Number’’ is negative.
  \fi}

\setcounter{Number}{-3}
Number = \theNumber\par
\checknumber

\addtocounter{Number}{10}
Number = \theNumber\par
\checknumber
```

もう結果を見るまでもないでしょうけど，出力はこうなります：

```
Number = −3
The ‘‘Number’’ is negative.
Number = 7
The ‘‘Number’’ is positive.
```

“\value{Number}” の部分は “\c@Number” としても同じですけど，その場合には，\makeatletter で “@” のカテゴリーコードを変えとかないといけないですね。

次は，\iftdir について見てみましょう。これは，「現在の組み方向がタテ組みか否か」という判断なので，⟨condition⟩ の部分はありません：

```
\iftdir
   ⟨true text⟩
\else
   ⟨false text⟩
\fi
```

現在の組み方向がタテ組みの場合には ⟨true text⟩ が実行され，ヨコ組みの場合には ⟨false text⟩ が実行されます。\iftdir は，第 II 部では「6.3.1 enumerate 環境」や，「6.3.2 itemize 環境」，「6.3.3 description 環境」，それに「9 今日の日付」の部分に出てきます。

続いて，\ifcase の形式は，次のようになります：

```
\ifcase ⟨number⟩ ⟨text for 0⟩
   \or ⟨text for 1⟩ \or ⟨text for 2⟩ ...
   ...
   \or ⟨text for N⟩
   [\else ⟨text for anything else⟩]
\fi
```

ここで ⟨number⟩ は整数で，“\else ⟨text for anything else⟩” の部分はオプションです。そして，“\or” は，\else や \fi 同様，プリミティブです（なお，この例では形式を揃えるために敢えて改行をしていますが，この場合には各行の行末に “%” を補わないと，余計なスペースが入ってしまうと思います）。

\ifcase を除く条件判断では，判断結果は，“true” か “false” のどちらかなのですが，この \ifcase の場合は，整数 ⟨number⟩ の値に応じて 3 つ以上の判断結果を利用することができます。

⟨number⟩ に相当する部分が 0 の場合は ⟨text for 0⟩ が実行され，⟨number⟩ が 1 なら ⟨text for 1⟩ が…という風に，⟨number⟩ が N のときには⟨text for N⟩ が実行され，⟨number⟩ が N を越えた場合や，負の整数のときには ⟨text for anything else⟩ の部分が実行されます。

\ifcase の使用例としては，例えば ltcounts.dtx に，次のような定義があります：

```
\def\@alph#1{%
  \ifcase#1\or a\or b\or c\or d\or e\or f\or
   g\or h\or i\or j\or k\or l\or m\or n\or
   o\or p\or q\or r\or s\or t\or u\or v\or
   w\or x\or y\or z\else\@ctrerr\fi}
```

この “\@alph” は，カウンタの値に応じて英小文字を出力するコマンド “\alph” の内部で使われているものですが，カウンタの値が 1〜26 の場合には，それぞれ対応する英小文字が出力されます。最初の \or の前は空なのでカウンタの値が 0 だと何も出力されず，他方，カウンタの値が負の整数だったり 27 以上の整数の場合には，エラーメッセージ \@ctrerr（=“Counter too large.”）が表示されます。

さて，最後に \newif を取り上げて，第 II 部のための準備は終わりということにしましょう。

少しだけ一般的な話から始めます。さっきの \iftdir みたいに，「現在の状態が，A か B のどちらか」で，処理を分けたいということはよくあります。それを実現するには，いくつか方法があると思いますが，ひとつには，カウンタを使うという手があります。例えば，A という状態ではカウンタ “\testcount” の値を 1 にして，B という状態では\testcount の値を 2 にでもして，あとは \ifnum を使って場合を分ければいいわけです。この場合には，何も状態は 2 つに限る必要はなくって，状態 C なら \testcount の値を 3 にして，“\ifnum \testcount=1”，“\ifnum \testcount=2”，“\ifnum \testcount=3” を使えば，三つの場合を使い分けることもできます。これは，第 II 部だと，「4.3 ページレイアウト」の部分で採用されているやり方です。

また，マクロを使うという手もあるでしょう。そのときどきの状態にあわせて，適当なマクロを再定義するようにすれば，場合分けに使うことができます。例えば，第 II

部の「2.15 オプションの実行」の部分では，“\@ptsize” というマクロを，クラスオプションに合わせて “0” か “1” か “2” に定義し直すことで，読み込むべきファイル名 “jsize1\@ptsize.clo” の判断に利用しています。

以上は 3 つ以上の状態の場合分けでしたが，2 つの状態のどちらかの判断ができさえすればよいのであれば，\count レジスタを無駄遣いしたりしないで済む方法があります。それが “\newif” を使って，新たな条件判断テスト “\if⟨*switch*⟩” を導入するというものです。

```
\newif\if⟨switch⟩
```

とすると，

- \let\if⟨*switch*⟩\iffalse
- \def{\⟨*switch*⟩true}{\let\if⟨*switch*⟩\iftrue}
- \def{\⟨*switch*⟩false}{\let\if⟨*switch*⟩\iffalse}

に展開されます（\newif は plain TEX から LATEX2.09 の lplain.tex へと引き継がれた後，LATEX 2ε では ltdefns.dtx で再インプリメントされています）。

ちょっと見づらいかもしれませんけど，\newif によって “\if⟨*switch*⟩”，“\⟨*switch*⟩true”，“\⟨*switch*⟩false” という 3 つのコントロールシーケンスが作られています。

この節の最初に “\if...” に相当するプリミティブの一覧を挙げた際，“\iftrue” と “\iffalse” の説明をしませんでしたが，これらはそれぞれ，「常に真」「常に偽」を意味します。したがって，

```
\if...
   ⟨true text⟩
\else
   ⟨false text⟩
\fi
```

で “\if...” が “\iftrue” であれば，⟨*true text*⟩ が実行され，“\if...” が “\iffalse” であれば ⟨*false text*⟩ が実行されます。

ここで，“\if...” が “\if⟨*switch*⟩” だと考えると，ようやく話がつながります：

```
\if⟨switch⟩
      ⟨true text⟩
\else
      ⟨false text⟩
\fi
```

という条件判断は，この条件判断よりも前の箇所のどこかで “\⟨*switch*⟩true” と書いておけば，⟨*true text*⟩ が実行され，逆に，“\⟨*switch*⟩false” と書いておけば，⟨*false text*⟩ のほうが実行される，というわけです。

最後はちょっと駆け足で，結局 memoir の “Appendix B” を全部カバーすることもできませんでしたけど，これくらいでなんとか，第 II 部へと進めるかと思います。

参考文献

[1] Donald E. Knuth, The TEXbook, Addison-Wesley, 1986. 〔斎藤信男監修・鷺谷好輝訳『改訂新版 TEX ブック』（アスキー・1992 年）〕. (CTAN:systems/knuth/dist/tex/texbook.tex)

[2] David Bausum, TEX Reference Manual, Kluwer, 2002. (http://www.tug.org/utilities/plain/cseq.html 他の URL もあります)

[3] Victor Eijkhout, TEX by Topic, Addison-Wesley, 1992. 〔富樫秀昭訳『TEX by Topic』（アスキー・1999 年）〕. (CTAN:info/texbytopic/TeXbyTopic.pdf 他の URL もあります)

[4] Paul W. Abrahams with Karl Berry, Kathryn A. Hargreaves , TEX for the Impatient, Addison-Wesley, 1990. 〔渡辺了介訳『明快 TEX』（アジソン・ウェスレイ・パブリッシャーズ・ジャパン・1997 年）〕. (CTAN:info/impatient/book.pdf)

[5] Leslie Lamport, LATEX, Addison-Wesley, 1986. 〔Edgar Cook・倉沢良一監訳・大野俊治・小暮博道・藤浦はる美訳『文書処理システム LATEX』（アスキー出版局・1990 年）〕.

[6] Leslie Lamport, LATEX: A Document Preparation System: User's Guide and Reference Manual, 2nd ed., Addison-Wesley, 1994. 〔阿瀬はる美訳『文書処理システム LATEX 2ε』（ピアソン・エデュケーション・1999 年）〕.

[7] アスキー書籍編集部『パーソナル日本語 TEX 縦組対応版』（アスキー・1994 年）.

[8] 中野賢『日本語 LATEX 2ε ブック』（アスキー・1996 年）.

[9] Peter Wilson, The Memoir Class for Configurable Typesetting: User Guide, 7th ed., May 2008. (CTAN:macros/latex/contrib/memoir/memman.pdf)

[10] Peter Flynn, *Rolling your own Document Class: Using LATEX to keep away from the Dark Side*, The PracTEX Journal, 2006 (No. 4). (http://www.tug.org/pracjourn/2006-4/flynn/flynn.pdf)

[11] Johannes Braams, David Carlisle, Alan Jeffrey, Leslie Lamport, Frank Mittelbach, Chris Rowley, Rainer Schöpf, LATEX 2ε Sources, 2005/12/01. (source2e.tex)

[12] Leslie Lamport, Frank Mittelbach, Johannes Braams, Standard Document Classes for LATEX version 2e, 2003/12/03. (classes.dtx)

[13] Ken Nakano, The pLATEX 2ε Sources, 2006/11/10. (pldoc.tex)

[14] The LATEX3 Project, LATEX 2ε for class and package writers, 2006/02/15. (clsguide.pdf)

[15] Frank Mittelbach, Michel Goossens et al., The LATEX Companion, 2nd ed., Addison-Wesley, 2004.

[16] 池田望「pLATEX 2εHandbook」Version 0.3 (1997/10/15 → 2005/5/25). (http://tutimura.ath.cx/~nob/tex/handbk.pdf)

[17] 中野賢「pLATEX ニュース」第 1 号（1997 年 2 月）. (plnews01.tex)

[18] 倉沢良一「日本語 TEX」（1988–1989 年）. (jtexdoc.tex)

第 II 部

はじめに

以下では，*jclasses.dtx* に記述されている順番に沿って，jarticle.cls (2006/06/27 v1.6) と jsize10.clo (2006/06/27 v1.6) の中味を眺めてみようと思います（「1 オプションスイッチ」～「10 初期設定」の見出しも *jclasses.dtx* のものです）。jarticle.cls のコードは背景色を黄色っぽい色にして，jsize10.clo のコードは青っぽい色の背景色にしてあります。

まず，ファイルの冒頭には，DOCSTRIP によって付加されたコメントがあります〔1～39 行〕。*jclasses.dtx* を元にして，DOCSTIP に `article,yoko` というオプションを与えて生成したのが本ファイル jarticle.cls である，ということが書いてあります。あとは，著作権に関する注意ですね[1]。

```
%%
%% This is file ‘jarticle.cls’,
%% generated with the docstrip utility.
%%
%% The original source files were:
%%
%% jclasses.dtx (with options: ‘article,yoko’)
%%
%% IMPORTANT NOTICE:
%%
%% For the copyright see the source file.
%%
%% Any modified versions of this file must be renamed
%% with new filenames distinct from jarticle.cls.
%%
%% For distribution of the original source see the terms
%% for copying and modification in the file jclasses.dtx.
%%
%% This generated file may be distributed as long as the
%% original source files, as listed above, are part of the
%% same distribution. (The sources need not necessarily be
%% in the same archive or directory.)
%% File: jclasses.dtx
%% \CharacterTable
%%  {Upper-case   \A\B\C\D\E\F\G\H\I\J\K\L\M\N\O\P\Q\R\S\T\U\V\W\X\Y\Z
%%   Lower-case   \a\b\c\d\e\f\g\h\i\j\k\l\m\n\o\p\q\r\s\t\u\v\w\x\y\z
%%   Digits       \0\1\2\3\4\5\6\7\8\9
%%   Exclamation  \!    Double quote \"    Hash (number) \#
%%   Dollar       \$    Percent      \%    Ampersand     \&
%%   Acute accent \’    Left paren   \(    Right paren   \)
%%   Asterisk     \*    Plus         \+    Comma         \,
%%   Minus        \-    Point        \.    Solidus       \/
%%   Colon        \:    Semicolon    \;    Less than     \<
%%   Equals       \=    Greater than \>    Question mark \?
%%   Commercial at \@   Left bracket \[    Backslash     \\
%%   Right bracket \]   Circumflex   \^    Underscore    \_
%%   Grave accent \‘    Left brace   \{    Vertical bar  \|
%%   Right brace  \}    Tilde        \~}
%%
```

(1) p(LA)TEX の著作権表示は，p(LA)TEX 付属のファイル “COPYRIGHT” や “COPYRIGHT.jis” にあります。

次の 4 行〔40〜43 行〕では，必要なフォーマットファイルの種類が指定され，また，このファイルの名前が名乗ってあります。\NeedsTeXFormat{pLaTeX2e}とあるので，\documentclass{jarticle}としたソースを pLaTeX でなく LaTeX で処理しようとしたりすると，エラーになります（でも，逆は大丈夫です．つまり，\documentclass{article}と指定したソースでも pLaTeX で処理できるように，pLaTeX では\NeedsTeXFormat が再定義されています）。なお，\ProvidesClass の引数部分は，\listfiles を指定した際のファイル一覧の情報源にもなっています。

\NeedsTeXFormat

→ cf. *plcore.dtx*

\ProvidesClass

```
40 \NeedsTeXFormat{pLaTeX2e}
41 \ProvidesClass{jarticle}
42   [2006/06/27 v1.6
43  Standard pLaTeX class]
```

さて，それでは，以下順番に中味を見ていこうと思いますが：

- \NeedsTeXFormat や\ProvidesClass をはじめ，クラスファイルの中で使われるコマンドについては，clsguide.pdf [14] や，LaTeX Companion [15] の “Appendix A.4: Package and class file structure” に説明があります。
- LaTeX のマニュアル [6] に載っているパラメータについては，その意味にいちいち言及をしませんが，パラメータの意味についてオンラインで確認するとしたら，“pLaTeX 2ε Handbook” [16] が便利だと思います。
- 以下の記述の主たる情報源は，*jclasses.dtx* の他には，source2e [11], *classes.dtx* [12], pldoc [13] の 3 つです。

なお，文系初級ユーザーの私が理解していない部分については，*jclasses.dtx* の記述をそのままコピーしておきます（逆に，*jclasses.dtx* の説明が分かりやすい場合にも，該当部分をただコピーしちゃいます）。数式絡みの部分とかファイルの入出力関係とかは全然分かりません…。すいません。

1 オプションスイッチ

ここでは，後々条件分岐に使うためのスイッチが，予め宣言されたり定義されたりしています：

- \newif を使って，スイッチを 6 つ新設[2]：
  - \if@landscape \@landscapefalse 〔45 行〕 → 2.3 横置きオプション
  - \if@restonecol[3] 〔47 行〕 → 6.0.1 表題； 8.3 索引
  - \if@titlepage \@titlepagefalse 〔48〜49 行〕 → 2.9 表題ページオプション
  - \if@stysize[4] \@stysizefalse 〔53 行〕 → 2.1 用紙オプション
  - \if@enablejfam \@enablejfamtrue 〔54 行〕 → 2.13 日本語ファミリ…
  - \if@mathrmmc \@mathrmmcfalse 〔55 行〕 → 2.13 日本語ファミリ…

[2] 以下では簡略に「スイッチ\if...」という書き方をしますが，恐らく正しくは，スイッチ自体は if に続く文字列です。つまり，例えば「\if@landscape」は「@landscape というスイッチのテスト（判断）」であって，スイッチ @landscape をオンにするのが\@landscapetrue で，オフにするのが\@landscapefalse ということなのだと思います。

[3] “*restore to one column*” とか “*restore to twoclumn from onecolumn* ” でしょうか。

[4] “*[LaTeX2.09 document] style [paper] size*” でしょうね。

- 紙のサイズに応じてページレイアウトを設定するためのスイッチに使うカウンタを新設（\c@@paper）〔44 行〕。 → 2.1 用紙オプション
  \c@@paper
- 指定されたポイントサイズに対応するオプションファイル（jsize10|11|12.clo）を読み込むスイッチに使うために，オプションで指定されたポイントサイズの一の位の値の保持用マクロを定義（\@ptsize）〔46 行〕。 → 2.2 サイズオプション
  \@ptsize

後ほどトンボオプションで使うために，カウンタを使った「時（\hour）」と「分（\minute）」の計算も，ここで行われています〔50〜52 行〕。 → 2.4 トンボオプション
\hour
\minute

```
44 \newcounter{@paper}
45 \newif\if@landscape \@landscapefalse
46 \newcommand{\@ptsize}{}
47 \newif\if@restonecol
48 \newif\if@titlepage
49 \@titlepagefalse
50 \hour\time \divide\hour by 60\relax
51 \@tempcnta\hour \multiply\@tempcnta 60\relax
52 \minute\time \advance\minute-\@tempcnta
53 \newif\if@stysize \@stysizefalse
54 \newif\if@enablejfam \@enablejfamtrue
55 \newif\if@mathrmmc \@mathrmmcfalse
```

カウンタ \hour と \minute については，*plcore.dtx* に説明があります：
\year
\month
\day
\time

from: *plcore.dtx*

TEX には，年月日を示す数値を保持しているカウンタとして，それぞれ \year, \month, \day がプリミティブとして存在します。しかし，時分については，深夜の零時からの経過時間を示す \time カウンタしか存在していません。そこで，pLATEX 2ε では，時分を示すためのカウンタ \hour と \minute を作成しています。
何時か（\hour）を得るには，\time を 60 で割った商をそのまま用います。何分か（\minute）は，\hour に 60 を掛けた値を \time から引いて算出します。ここではカウンタを宣言するだけです。実際の計算は，クラスやパッケージの中で行なっています。

```
\newcount\hour
\newcount\minute
```

\if@enablejfam と \if@mathrmmc はよく分からないので，早速 *jclasses.dtx* の説明をコピーしておきます：
\if@enablejfam
\if@mathrmmc

from: *jclasses.dtx*

\if@enablejfam \@enablejfamtrue
日本語ファミリを宣言するために用いるフラグです。

\if@mathrmmc \@mathrmmcfalse
和欧文両対応の数式文字コマンドを有効にするときに用いるフラグです。マクロの展開順序が複雑になるのを避けるため，デフォルトでは false としてあります。

2 オプションの宣言

2.1〜2.14 は，jarticle.cls で指定できるクラスオプションの設定の説明です（「2.10 右左起こしオプション」は jarticle.cls にはありませんが）。

2.1 用紙オプション

まずは，通常の A4, A5, B4, B5 用の用紙サイズオプションです。それぞれ，カウンタ\c@@paper の値が 1，2，3，4 にセットされます。このカウンタは，後出の「4.3.2 本文領域」や「4.3.3 マージン」の設定の際の，条件分岐に使われます(5)。

→ 4.3 ページレイアウト

a4paper
a5paper
b4paper
b5paper

```
\DeclareOption{a4paper}{\setcounter{@paper}{1}%
  \setlength\paperheight {297mm}%
  \setlength\paperwidth {210mm}}
\DeclareOption{a5paper}{\setcounter{@paper}{2}%
  \setlength\paperheight {210mm}
  \setlength\paperwidth {148mm}}
\DeclareOption{b4paper}{\setcounter{@paper}{3}%
  \setlength\paperheight {364mm}
  \setlength\paperwidth {257mm}}
\DeclareOption{b5paper}{\setcounter{@paper}{4}%
  \setlength\paperheight {257mm}
  \setlength\paperwidth {182mm}}
```

次は，通常の指定よりも本文領域を広くするオプション（a4j，a5j，b4j，b5j）です。用紙のサイズや\c@@paper の設定は上と同じですが，スイッチ\if@stysize が\@stysizetrue にセットされ，これが後出の「4.3.2 本文領域」や「4.3.3 マージン」の設定の際に効いてきます。

a4j
a5j
b4j
b5j

```
\DeclareOption{a4j}{\setcounter{@paper}{1}\@stysizetrue
  \setlength\paperheight {297mm}%
  \setlength\paperwidth {210mm}}
\DeclareOption{a5j}{\setcounter{@paper}{2}\@stysizetrue
  \setlength\paperheight {210mm}
  \setlength\paperwidth {148mm}}
\DeclareOption{b4j}{\setcounter{@paper}{3}\@stysizetrue
  \setlength\paperheight {364mm}
  \setlength\paperwidth {257mm}}
\DeclareOption{b5j}{\setcounter{@paper}{4}\@stysizetrue
  \setlength\paperheight {257mm}
  \setlength\paperwidth {182mm}}
```

続いて a4p，a5p，b4p，b5p です：

a4p
a5p
b4p
b5p

```
\DeclareOption{a4p}{\setcounter{@paper}{1}\@stysizetrue
  \setlength\paperheight {297mm}%
  \setlength\paperwidth {210mm}}
\DeclareOption{a5p}{\setcounter{@paper}{2}\@stysizetrue
  \setlength\paperheight {210mm}
  \setlength\paperwidth {148mm}}
\DeclareOption{b4p}{\setcounter{@paper}{3}\@stysizetrue
  \setlength\paperheight {364mm}
  \setlength\paperwidth {257mm}}
\DeclareOption{b5p}{\setcounter{@paper}{4}\@stysizetrue
  \setlength\paperheight {257mm}
  \setlength\paperwidth {182mm}}
```

オプション a4j，a5j，b4j，b5j；a4p，a5p，b4p，b5p については，pLaTeX 2ε のマニュアル [8] にも格別記述はないのですが，「pLaTeX ニュース」の第 1 号 [17] に説明がありました：

(5) でも，実際には\@stysizetrue のときにしか，このカウンタ\c@@paper による分岐はされてないように見えます。

from: plnews01.tex

本文領域の広いレイアウト

jLaTeX 2.09 や pLaTeX 2.09 とともに使われていた，a4j, a5j, b4j, b5j, a4p, a5p, b4p, b5p のスタイルファイルと同等のレイアウトをするためのクラスオプションを追加しました。これらのオプションを指定すると，デフォルトで設定されている本文領域よりも広いレイアウトで文章を作成することができます。

オプション名は，以前のスタイルファイル名と同じです。最後が “j” で終わるものは横組専用，“p” で終わるものは縦横両用のスタイルでしたが，pLaTeX 2ε ではとくに区別をしていません。“j” で終わるオプションも “p” で終わるオプションも縦横両用です。

2.2 サイズオプション

後々対応するオプションファイル（jsize10|11|12.clo）を読み込むための準備として，オプションで指定された本文のポイントサイズに応じて，マクロ \@ptsize をここで再定義します。 → 2.15 オプションの実行

\@ptsize

互換モードの場合，または，10pt オプションの場合には，\@ptsize が “0” と定義されます（\@ptsize は，展開されると “0” になるということ）。LaTeX2.09 互換モードとの条件分岐に使うスイッチ \if@compatibility は，*ltclass.dtx* で宣言されています。

10pt

```
\if@compatibility
  \renewcommand{\@ptsize}{0}
\else
  \DeclareOption{10pt}{\renewcommand{\@ptsize}{0}}
\fi
```

同様に，\@ptsize は，11pt オプションの場合は “1” に，12pt オプションの場合は “2” へと展開されることになります：

11pt
12pt

```
\DeclareOption{11pt}{\renewcommand{\@ptsize}{1}}
\DeclareOption{12pt}{\renewcommand{\@ptsize}{2}}
```

2.3 横置きオプション

用紙を横置きにするオプションです。landscape を指定すると，スイッチ \if@landscape が \@landscapetrue にセットされ，また，\@tempdima を使って，\paperheight と \paperwidth の値が入れ替えられています。 → 4.3.2 本文領域

landscape

```
\DeclareOption{landscape}{\@landscapetrue
  \setlength\@tempdima{\paperheight}%
  \setlength\paperheight{\paperwidth}%
  \setlength\paperwidth{\@tempdima}}
```

2.4 トンボオプション

トンボをつけるオプションです。ファイル名と日時も併せて印刷する tombow オプションと，トンボのみで文字列ナシの tombo オプションとが用意されています。

tombow
tombo

ここでスイッチに使われている \iftombow や \iftombowdate，トンボ用の罫線の太さである \@tombowwidth，トンボと一緒に印刷する文字列を保持してい

る \@bannertoken，そして実際にトンボを作成する \maketombowbox 等々は，*plcore.dtx* で定義されています(トンボを出力する \@outputtombow の定義も，*plcore.dtx* にあります)。

```
\DeclareOption{tombow}{%
  \tombowtrue \tombowdatetrue
  \setlength{\@tombowwidth}{.1\p@}%
  \@bannertoken{%
     \jobname\space:\space\number\year/\number\month/\number\day
      (\number\hour:\number\minute)}
  \maketombowbox}
```

```
\DeclareOption{tombo}{%
  \tombowtrue \tombowdatefalse
  \setlength{\@tombowwidth}{.1\p@}%
  \maketombowbox}
```

2.5 面付けオプション

mentuke

このオプションを指定すると，tombo オプションを指定した場合と同じ位置に文章が印刷されますが，トンボ自体は印刷しないためのオプションのようです：

```
\DeclareOption{mentuke}{%
  \tombowtrue \tombowdatefalse
  \setlength{\@tombowwidth}{\z@}%
  \maketombowbox}
```

2.6 組方向オプション

tate

tate オプションを指定すると，\begin{document}のところで pTEX のプリミティブである\tate が挿入されて，本文が縦組みになります。\adjustbaseline は *plfonts.dtx* で定義されていて，和文と欧文のセンターが揃うようにベースラインをシフトするものです。

```
\DeclareOption{tate}{%
  \AtBeginDocument{\tate\message{《縦組モード》}%
                   \adjustbaseline}%
}
```

2.7 両面，片面オプション

→ 4.3 ページレイアウト

oneside
twoside

oneside を指定すると，スイッチ \if@twoside が \@twosidefalse にセットされ，twoside を指定すると，\@twosidetrue にセットされます。片面か両面かの判断は，「4.3.2 本文領域」や「4.3.3 マージン」の設定の際に効いてきます。スイッチ \if@twoside は，*ltoutput.dtx* で宣言されています。

```
\DeclareOption{oneside}{\@twosidefalse}
\DeclareOption{twoside}{\@twosidetrue}
```

2.8 二段組オプション

onecolumn とすると \if@twocolumn が \@twocolumnfalse に，twocolumn だと，\@twocolumntrue に。\if@twocolumn は，*ltoutput.dtx* で宣言されています。

→ 4.3 ページレイアウト

onecolumn
twocolumn

```
\DeclareOption{onecolumn}{\@twocolumnfalse}
\DeclareOption{twocolumn}{\@twocolumntrue}
```

2.9 表題ページオプション

titlepage とすると \if@titlepage が \@titlepagetrue となり，notitlepage だと，\@titlepagefalse になります。このスイッチは「6.0.1 表題」や「6.0.2 概要」で使われます。

→ 6 文書コマンド

titlepage
notitlepage

```
\DeclareOption{titlepage}{\@titlepagetrue}
\DeclareOption{notitlepage}{\@titlepagefalse}
```

2.10 右左起こしオプション

jarticle.cls にはないオプションですが，互換モードかまたは openright と指定すると，\if@openright が \@openrighttrue となり，openany とすると \@openrightfalse となります（jreport.cls のデフォルトは openany で，jbook.cls のデフォルトは openright）。\chapter の定義の中で，このスイッチに応じた条件分岐がされます（\chapter を奇数頁起こしにするか否か）。

openright
openany

```
\if@compatibility
\@openrighttrue
\else
\DeclareOption{openright}{\@openrighttrue}
\DeclareOption{openany}{\@openrightfalse}
\fi
```

2.11 数式のオプション

leqno[6] を指定するとクラスオプションファイル leqno.clo がここで読み込まれ，同様に，fleqn[7] を指定すると fleqn.clo が読み込まれます。

leqno
fleqn

```
\DeclareOption{leqno}{\input{leqno.clo}}
\DeclareOption{fleqn}{\input{fleqn.clo}}
```

2.12 参考文献のオプション

openbib を指定すると，クラスファイルの最後の部分で，マクロ \@openbib@code と \newblock が再定義されます。

→ 8.2 参考文献

openbib

「8.2 参考文献」のところで，\@openbib@code は元々 \let\@openbib@code\@empty と代入されていて，また，\newblock は \newcommand{\newblock}{\hskip.11em \@plus.33em\@minus.07em}と定義されています（734 行，715 行）。

(6) “*left [side] equation no*” とかですかね。

(7) “*flush left equaion*” でしょうか。

```
\DeclareOption{openbib}{%
  \AtEndOfPackage{%
   \renewcommand\@openbib@code{%
      \advance\leftmargin\bibindent
      \itemindent -\bibindent
      \listparindent \itemindent
      \parsep \z@
      }%
   \renewcommand\newblock{\par}}}
```

133〜136 行をより LaTeX-ey に書き直すと：

```
   \addtolength{\leftmargin}{\bibindent}
   \setlength{\itemindent}{-\bibindent}
   \setlength{\listparindent}{\itemindent}
   \setlength{\parsep}{0pt}
```

という感じでしょうか。パラメータの意味や「オープンスタイル」については，LaTeX のマニュアル [6] に説明があります。

→ cf. 6.3 リスト環境

2.13 日本語ファミリ宣言の抑制，和欧文両対応の数式文字

ここはよく分からないので，*jclasses.dtx* から丸写ししておきます：

→ 7 フォントコマンド
disablejfam
mathrmmc

from: *jclasses.dtx*

pLaTeX 2ε は，このあと，数式モードで直接，日本語を記述できるように数式ファミリを宣言します。しかし，TeX で扱える数式ファミリの数が 16 個なので，その他のパッケージと組み合わせた場合，数式ファミリを宣言する領域を超えてしまう場合があるかもしれません。そのときには，残念ですが，そのパッケージか，数式内に直接，日本語を記述するのか，どちらかを断念しなければなりません。このクラスオプションは，数式内に日本語を記述するのをあきらめる場合に用います。

`disablejfam` オプションを指定しても `\textmc` や `\textgt` などを用いて，数式内に日本語を記述することは可能です。

`mathrmmc` オプションは，`\mathrm` と `\mathbf` を和欧文両対応にするためのクラスオプションです。

```
\if@compatibility
  \@mathrmmctrue
\else
  \DeclareOption{disablejfam}{\@enablejfamfalse}
  \DeclareOption{mathrmmc}{\@mathrmmctrue}
\fi
```

ついでに「pLaTeX ニュース」[17] からも転記しておきます：

from: plnews.tex

フォントファミリ

pLaTeX 2ε の特徴の一つに，数式内にも直接，日本語を記述することができることが挙げられます。しかし AMS のパッケージや PostScript 用のパッケージを用いた場合，

```
No room for a new \mathgroup .
```

や

```
Too many math alphabets used in version normal.
```

などのエラーが表示される場合があります。

これらのエラーは，数式内に直接，記述できるフォントファミリとして TEX が扱えるのが最大 16 個ということから起こっています。このエラーを回避するには，用いるフォントファミリの数を 16 個以内にするしかありません。
そこで，pLATEX 2ε では，日本語を数式内に直接記述はできなくなるけれども，必要なパッケージをロードできる（かもしれない）ようにするためのオプション disablejfam をクラスファイルに用意しました。disablejfam オプションを指定すれば，フォントファミリを節約することができます。ただし，宣言している数は一つだけですので，用いるパッケージによっては効果がないかもしれないことに注意をしてください。

2.14 ドラフトオプション

draft を指定すると \overfullrule の太さが 5pt に設定され，final とすると 0pt に設定されます。 *draft* *final*

```
\DeclareOption{draft}{\setlength\overfullrule{5pt}}
\DeclareOption{final}{\setlength\overfullrule{0pt}}
```

2.15 オプションの実行

以上で宣言したオプションのうち，まずデフォルトで設定するオプションを \ExecuteOptions で実行し，その後で，ユーザーがクラスオプションで指定したオプションを \ProcessOptions で実行します： *\ExecuteOptions* *\ProcessOptions*

```
\ExecuteOptions{a4paper,10pt,oneside,onecolumn,final}
\ProcessOptions\relax
```

そして，「2.2 サイズオプション」で再定義しておいた \@ptsize を使って，対応する jsize10|11|12.clo ファイルをここで読み込みます： *jsize10.clo* *jsize11.clo* *jsize12.clo*

```
\input{jsize1\@ptsize.clo}
```

jclasses.dtx はここからは一旦，jsize10|11|12.clo の説明になっていますので，本文書も，次の「3 フォント」の部分では，jsize10.clo の中味を眺めることにしましょう。

3 フォント

jsize10.clo の最初の 43 行は，jarticle.cls の冒頭と同様，DOCSTRIP によるコメントと，\NeedsTeXFormat の行と \ProvidesFile の行です。
そして，続く 44〜84 行に，基本の文字サイズが 10pt の場合の，各種サイズ変更コマンドが定義されています。
最初の \normalsize，\small，\footnotesize の三つについては，文字サイズと行送りの他に，

- \abovedisplayskip *\abovedisplayskip*
- \belowdisplayskip *\belowdisplayskip*
- \abovedisplayshortskip *\abovedisplayshortskip*
- \belowdisplayshortskip *\belowdisplayshortskip*

の値と，

- \@listi

\@listi

の各種パラメータの設定がなされます（\belowdisplayskip についてはいずれの場合も \abovedisplayskip が代入されています．また，\normalsize の \@listi には \@listI が代入されるだけです）。

→ 6.3 リスト環境

フォントサイズを定義するコマンド \@setfontsize は *ltfssini.dtx* で定義されていて，使い方は *jclasses.dtx* に次のようにあります：

\@setfontsize

from: *jclasses.dtx*

\@setfontsize\size⟨*font-size*⟩⟨*baselineskip*⟩

⟨*font-size*⟩	これから使用する，フォントの実際の大きさです。
⟨*baselineskip*⟩	選択されるフォントサイズ用の通常の\baselineskip の値です（実際は，\baselinestretch × ⟨*baselineskip*⟩ の値です）。

まずは，\normalsize を（再）定義して，そして，\normalsize を実行します：

\normalsize

```
\renewcommand{\normalsize}{%
   \@setfontsize\normalsize\@xpt{15}%
  \abovedisplayskip 10\p@ \@plus2\p@ \@minus5\p@
  \abovedisplayshortskip \z@ \@plus3\p@
  \belowdisplayshortskip 6\p@ \@plus3\p@ \@minus3\p@
   \belowdisplayskip \abovedisplayskip
   \let\@listi\@listI}
\normalsize
```

45〜49 行をまた，LaTeX-ey に書き直してみますと：

```
  \@setfontsize{\normalsize}{10}{15}%
  \setlength{\abovedisplayskip}{10pt plus 2pt minus 5pt}
  \setlength{\abovedisplayshortskip}{0pt plus 3pt}
  \setlength{\belowdisplayshortskip}{6pt plus 3pt minus 3pt}
  \setlength{\belowdisplayskip}{\abovedisplayskip}
```

なお，\normalsize は *ltfntcmd.dtx* で次のように定義されているので，クラスファイルで改めて適切に「再定義」する必要があるようです：

from: *ltfntcmd.dtx*

```
\def\normalsize{%
  \@latex@error {The font size command \protect\normalsize\space
             is not defined:\MessageBreak
             there is probably something wrong with
             the class file}\@eha
}
```

ここで，0 番のボックスレジスタに和文フォントの「全角空白」文字（euc"A1A1）を入れて，その高さ・深さ・幅をそれぞれ \Cht，\Cdp，\Cwd に設定しています。また，\Cvs と \Chs にはそれぞれ，\baselineskip と \wd0 の値を設定しています。

\Cht
\Cdp
\Cwd
\Cvs
\Chs

```
\setbox0\hbox{\char\euc"A1A1}%
\setlength\Cht{\ht0}
\setlength\Cdp{\dp0}
\setlength\Cwd{\wd0}
\setlength\Cvs{\baselineskip}
\setlength\Chs{\wd0}
```

これらのパラメータの意味については，*plfonts.dtx* を写しちゃいます：

from: *plfonts.dtx*

ここでは，和文フォントの幅や高さなどを格納する変数について説明をしています。

頭文字が大文字の変数は，ノーマルサイズの書体の大きさで，基準値となります。これらは，jart10.clo〔ママ〕などの補助クラスファイルで設定されます。

小文字だけからなる変数は，フォントが変更されたときに（\selectfont 内で）更新されます。

\Cht, \cht
: \Cht は基準となる和文フォントの文字の高さを示します。\cht は現在の和文フォントの文字の高さを示します。なお，この“高さ”はベースラインより上の長さです。

\Cdp, \cdp
: \Cdp は基準となる和文フォントの文字の深さを示します。\cdp は現在の和文フォントの文字の深さを示します。なお，この“深さ”はベースラインより下の長さです。

\Cwd, \cwd
: \Cwd は基準となる和文フォントの文字の幅を示します。\cwd は現在の和文フォントの文字の幅を示します。

\Cvs, \cvs
: \Cvs は基準となる行送りを示します。ノーマルサイズの \baselineskip と同値です。\cvs は現在の行送りを示します。

\Chs, \chs
: \Chs は基準となる字送りを示します。\Cwd と同値です。\chs は現在の字送りを示します。

次の \small と \footnotesize の設定項目は，\normalsize と同じです：

\small
\footnotesize

```
\newcommand{\small}{%
  \@setfontsize\small\@ixpt{11}%
  \abovedisplayskip 8.5\p@ \@plus3\p@ \@minus4\p@
  \abovedisplayshortskip \z@ \@plus2\p@
  \belowdisplayshortskip 4\p@ \@plus2\p@ \@minus2\p@
  \def\@listi{\leftmargin\leftmargini
              \topsep 4\p@ \@plus2\p@ \@minus2\p@
              \parsep 2\p@ \@plus\p@ \@minus\p@
              \itemsep \parsep}%
  \belowdisplayskip \abovedisplayskip}
```

```
\newcommand{\footnotesize}{%
  \@setfontsize\footnotesize\@viiipt{9.5}%
  \abovedisplayskip 6\p@ \@plus2\p@ \@minus4\p@
  \abovedisplayshortskip \z@ \@plus\p@
  \belowdisplayshortskip 3\p@ \@plus\p@ \@minus2\p@
  \def\@listi{\leftmargin\leftmargini
              \topsep 3\p@ \@plus\p@ \@minus\p@
              \parsep 2\p@ \@plus\p@ \@minus\p@
              \itemsep \parsep}%
  \belowdisplayskip \abovedisplayskip}
```

残りのサイズ変更コマンドでは，文字サイズと行送りが設定されるだけです：

\scriptsize
\tiny
\large
\Large
\LARGE
\huge
\Huge

```
\newcommand{\scriptsize}{\@setfontsize\scriptsize\@viipt\@viiipt}
\newcommand{\tiny}{\@setfontsize\tiny\@vpt\@vipt}
\newcommand{\large}{\@setfontsize\large\@xiipt{17}}
\newcommand{\Large}{\@setfontsize\Large\@xivpt{21}}
\newcommand{\LARGE}{\@setfontsize\LARGE\@xviipt{25}}
\newcommand{\huge}{\@setfontsize\huge\@xxpt{28}}
\newcommand{\Huge}{\@setfontsize\Huge\@xxvpt{33}}
```

4 レイアウト

4.1 用紙サイズの決定

jclasses.dtx の記述はここで，jarticle.cls に戻ってきます。見出しは「用紙サイズの決定」となっていますけど，設定されているのは二段組の際の段間の幅（\columnsep）と，段間に引かれる罫線の太さ（\columnseprule）です。スイッチ \if@stysize が \@stysizetrue のときの段間隔は和文の 2 文字分で，\@stysizefalse のときは 10pt となっています。また，段間の罫線の太さは 0pt なので，デフォルトでは罫線は引かれません。

\columnsep
\columnseprule

なお，\columnsep と \columnseprule は *ltoutput.dtx* で宣言されています。

```
\if@stysize
  \setlength\columnsep{2\Cwd}
\else
  \setlength\columnsep{10\p@}
\fi
```

```
\setlength\columnseprule{0\p@}
```

4.2 段落の形

ここで設定されているのは，

- \lineskip と \normallineskip
- \baselinestretch, \parskip, \parindent
- \@lowpenalty, \@medpenalty, \@highpenalty

の値です。

\baselinestretch，\parskip，\parindent は LaTeX のマニュアル [6] にも出てきますが，他のパラメータは TeX のプリミティブや内部コマンドですね…。

\lineskip はプリミティブで，他方 \normallineskip は *ltplain.dtx* で宣言されています：

\lineskip
\normallineskip

from: *jclasses.dtx*

\lineskip, \normallineskip
これらの値は，行が近付き過ぎたときの TeX の動作を制御します。

```
\setlength\lineskip{1\p@}
\setlength\normallineskip{1\p@}
```

\baselinestretch[8]，\parskip，\parindent はおなじみですね：

\baselinestretch
\parskip
\parindent

```
\renewcommand{\baselinestretch}{}
\setlength\parskip{0\p@ \@plus \p@}
\setlength\parindent{1\Cwd}
```

[8] *jclasses.dtx* には，\baselinestretch の説明として，「これは，\baselineskip の倍率を示すために使います。デフォルトでは，何もしません。このコマンドが “empty” でない場合，\baselineskip の指定の plus や minus 部分は無視されることに注意してください。」とあるのですが，確かにスキップに係数を掛けると伸縮部がなくなっちゃうとは思うのですけど，\baselineskip には plus や minus 部分は元々ないように見えます。→ 3. フォント

from: *jclasses.dtx*

\nopagebreak と \nolinebreak コマンドは，これらのコマンドが置かれた場所に，ペナルティを起いて〔ママ〕，分割を制御します。置かれるペナルティは，コマンドの引数によって，\@lowpenalty, \@medpenalty, \@highpenalty のいずれかが使われます。

```
\@lowpenalty  51
\@medpenalty 151
\@highpenalty 301
```

カウンタ \@lowpenalty, \@medpenalty, \@highpenalty は *ltfinal.dtx* で宣言されています。

\@lowpenalty
\@medpenalty
\@highpenalty

ここではひとまず，\nopagebreak の定義のうち，ペナルティと関係する部分だけを *ltspace.dtx* から抜き出しておきます：

from: *ltspace.dtx*

```
\def\nopagebreak{\@testopt\@no@pgbk4}

\def\@no@pgbk #1[#2]{%
  \ifvmode
    \penalty #1\@getpen{#2}%
  \else
    \@bsphack
    \vadjust{\penalty #1\@getpen{#2}}%
    \@esphack
  \fi}

\def\@getpen#1{\ifcase #1 \z@ \or \@lowpenalty\or
        \@medpenalty \or \@highpenalty
        \else \@M \fi}
```

\nopagebreak[0]|[1]|[2]|[3]|[4] が，\penalty=0|\@lowpenalty|\@medpenalty|\@highpenalty|10000 のそれぞれに相当するのですね（デフォルトは [4]．\@no@pgbk の “#1” って何でしょう?)。

4.3 ページレイアウト

4.3.1 縦方向のスペース

ここからはまた，jsize10.clo の中味です（ページレイアウトは，本文の文字サイズに依存するので)。まず，\headheight，\headsep，\topskip，\footskip の設定からです：

\headheight
\headsep
\topskip
\footskip

```
\setlength\headheight{12\p@}
\setlength\headsep{25\p@}
\setlength\topskip{1\Cht}
\setlength\footskip{30\p@}
```

\maxdepth については *jclasses.dtx* を写しておきます…：

\maxdepth

from: *jclasses.dtx*

TEX のプリミティブレジスタ \maxdepth は，\topskip と同じような働きをします。\@maxdepth レジスタは，つねに \maxdepth のコピーでなくてはいけません。これは \begin{document}の内部で設定されます。TEX と LATEX2.09 では，\maxdepth は 4pt に固定です。LATEX 2ε では，\maxdepth + \topskip を基本サイズの 1.5 倍にしたいので，\maxdepth を \topskip の半分の値で設定します。

```
\if@compatibility
  \setlength\maxdepth{4\p@}
\else
  \setlength\maxdepth{.5\topskip}
\fi
```

4.3.2 本文領域

次に，\textwidth の設定です（\Cwd は和文フォントの文字幅）： *\textwidth*

まずは LaTeX2.09 モードから：

```
\if@compatibility
  \if@stysize
    \ifnum\c@@paper=2 % A5
      \if@landscape
        \setlength\textwidth{47\Cwd}
      \else
        \setlength\textwidth{28\Cwd}
      \fi
    \else\ifnum\c@@paper=3 % B4
      \if@landscape
        \setlength\textwidth{75\Cwd}
      \else
        \setlength\textwidth{60\Cwd}
      \fi
    \else\ifnum\c@@paper=4 % B5
      \if@landscape
        \setlength\textwidth{60\Cwd}
      \else
        \setlength\textwidth{37\Cwd}
      \fi
    \else % A4 ant other
      \if@landscape
        \setlength\textwidth{73\Cwd}
      \else
        \setlength\textwidth{47\Cwd}
      \fi
    \fi\fi\fi
  \else
    \if@twocolumn
      \setlength\textwidth{52\Cwd}
    \else
      \setlength\textwidth{327\p@}
    \fi
  \fi
```

そして，LaTeX 2εモード：

```
\else
  \if@stysize
    \if@twocolumn
      \setlength\textwidth{.8\paperwidth}
    \else
      \setlength\textwidth{.7\paperwidth}
    \fi
  \else
    \setlength\@tempdima{\paperwidth}
    \addtolength\@tempdima{-2in}
```

```
    \setlength\@tempdimb{327\p@}
    \if@twocolumn
      \ifdim\@tempdima>2\@tempdimb\relax
        \setlength\textwidth{2\@tempdimb}
      \else
        \setlength\textwidth{\@tempdima}
      \fi
    \else
      \ifdim\@tempdima>\@tempdimb\relax
        \setlength\textwidth{\@tempdimb}
      \else
        \setlength\textwidth{\@tempdima}
      \fi
    \fi
  \fi
\fi
```

幾重にもネストされてて一瞬クラっとしてしまいますが，要は LaTeX2.09 の互換モードか否かで分岐して，あとは紙のサイズごとに \textwidth の長さを決めているだけのことです(9)：

- LaTeX2.09 互換モードの場合（\@compatibilitytrue）
  - a4j|a5j|b4j|b5j|a4p|a5p|b4p|b5p の場合（\@stysizetrue）
    - A5 用紙の場合（\c@@paper=2）
      - 用紙横置きの場合（\@landscapetrue）：\textwidth=47\Cwd
      - 用紙縦置きの場合（\@landscapefalse）：\textwidth=28\Cwd
    - B4 用紙の場合（\c@@paper=3）
      - 用紙横置きの場合（\@landscapetrue）：\textwidth=75\Cwd
      - 用紙縦置きの場合（\@landscapefalse）：\textwidth=60\Cwd
    - B5 用紙の場合（\c@@paper=4）
      - 用紙横置きの場合（\@landscapetrue）：\textwidth=60\Cwd
      - 用紙縦置きの場合（\@landscapefalse）：\textwidth=37\Cwd
    - A4 用紙その他の場合（\c@@paper=2|3|4 以外の場合）
      - 用紙横置きの場合（\@landscapetrue）：\textwidth=73\Cwd
      - 用紙縦置きの場合（\@landscapefalse）：\textwidth=47\Cwd
  - a4paper|a5paper|b4paper|b5paper の場合（\@stysizefalse）
    - 二段組の場合（\@twocolumntrue）：\textwidth=52\Cwd
    - 一段組の場合（\@twocolumnfalse）：\textwidth=327pt
- LaTeX 2ε モードの場合
  - a4j|a5j|b4j|b5j|a4p|a5p|b4p|b5p の場合（\@stysizetrue）
    - 二段組の場合（\@twocolumntrue）：\textwidth=.8\paperwidth
    - 一段組の場合（\@twocolumnfalse）：\textwidth=.7\paperwidth
  - a4paper|a5paper|b4paper|b5paper の場合（\@stysizefalse）

    ```
    \setlength{\@tempdima}{\paperwidth-2in}
    \setlength{\@tempdimb}{327pt}
    ```

    - 二段組の場合（\@twocolumntrue）
      - \@tempdima > 654pt (=2\@tempdimb) の場合：\textwidth=654pt
      - \@tempdima > 654pt でない場合：\textwidth=\@tempdima
    - 一段組の場合（\@twocolumnfalse）
      - \@tempdima > 327pt (=\@tempdimb) の場合：\textwidth=327pt
      - \@tempdima > 327pt でない場合：\textwidth=\@tempdima

\textwidth の長さが決まったら，LaTeX 2ε の内部コマンド \@settopoint を使って，\textwidth の長さをポイントの整数倍に丸めています： *\@settopoint*

(9) …なんてエラそうに場合分けをしてはみたものの，ホントにこれで合っているのでしょうか???

```
\@settopoint\textwidth
```

\@settopoint は *ltlength.dtx* で次のように定義されています：

from: *ltlength.dtx*

```
\def\@settopoint#1{\divide#1\p@\multiply#1\p@}
```

TEX は，ユーザーに対しては長さを pt に換算して教えてくれますが，内部では 1/65536 pt（= 1sp）を単位にしているのだそうです。ですので，65536sp の整数倍であれば，ポイント換算時に小数点以下が発生しません。ここでは，TEX の割り算では商の小数点以下が切り捨てられることと，整数が必要な文脈でディメンションが指定された場合にはその値は sp に換算された整数値になるという仕組みによって，上の \@settopoint の計算で，長さをポイントの整数倍に丸めることができるわけです（\p@ = 1pt = 65536sp → 65536）。

続いて，\textheight の長さです（\Cvs=\baselineskip）。ここでも細かく場合分けがされていて，\textheight が確定した後で，最後に \@settopoint でポイントの端数を削っています。 *\textheight*

LATEX2.09 互換モード：

```
\if@compatibility
  \if@stysize
    \ifnum\c@@paper=2 % A5
      \if@landscape
        \setlength\textheight{17\Cvs}
      \else
        \setlength\textheight{28\Cvs}
      \fi
    \else\ifnum\c@@paper=3 % B4
      \if@landscape
        \setlength\textheight{38\Cvs}
      \else
        \setlength\textheight{57\Cvs}
      \fi
    \else\ifnum\c@@paper=4 % B5
      \if@landscape
        \setlength\textheight{22\Cvs}
      \else
        \setlength\textheight{35\Cvs}
      \fi
    \else % A4 and other
      \if@landscape
        \setlength\textheight{27\Cvs}
      \else
        \setlength\textheight{43\Cvs}
      \fi
    \fi\fi\fi
    \addtolength\textheight{\topskip}
  \else
  \setlength\textheight{578\p@}
  \fi
```

LATEX 2ε モード：

```
\else
  \if@stysize
    \setlength\textheight{.75\paperheight}
  \else
    \setlength\@tempdima{\paperheight}
```

```
    \addtolength\@tempdima{-2in}
    \addtolength\@tempdima{-1.5in}
    \divide\@tempdima\baselineskip
    \@tempcnta\@tempdima
    \setlength\textheight{\@tempcnta\baselineskip}
  \fi
\fi
\addtolength\textheight{\topskip}
```

```
\@settopoint\textheight
```

4.3.3 マージン

続いて，マージンの設定です。まずは \topmargin から： *\topmargin*

LaTeX2.09 互換モード：

```
\if@compatibility
  \if@stysize
    \setlength\topmargin{-.3in}
  \else
    \setlength\topmargin{27\p@}
  \fi
```

LaTeX 2ε モード：

```
\else
  \setlength\topmargin{\paperheight}
  \addtolength\topmargin{-\headheight}
  \addtolength\topmargin{-\headsep}
  \addtolength\topmargin{-\textheight}
  \addtolength\topmargin{-\footskip}
  \if@stysize
    \ifnum\c@@paper=2 % A5
      \addtolength\topmargin{-1.3in}
    \else
      \addtolength\topmargin{-2.0in}
    \fi
  \else
    \addtolength\topmargin{-2.0in}
  \fi
  \addtolength\topmargin{-.5\topmargin}
\fi
```

```
\@settopoint\topmargin
```

\marginparsep と \marginparpush の設定です： *\marginparsep* *\marginparpush*

```
\if@twocolumn
  \setlength\marginparsep{10\p@}
\else
  \setlength\marginparsep{10\p@}
\fi
```

```
\setlength\marginparpush{5\p@}
```

\oddsidemargin, \evensidemargin, \marginparwidth です。ここでも, pt 単位で決め打ちするのではなく紙のサイズを元に計算で設定する場合には, \@settopoint でポイントの整数値に丸めています。

\oddsidemargin
\evensidemargin
\marginparwidth

またまた，まずは LaTeX2.09 互換モード：

```
\if@compatibility
    \if@twoside
      \setlength\oddsidemargin {44\p@}
      \setlength\evensidemargin {82\p@}
      \setlength\marginparwidth {107\p@}
    \else
     \setlength\oddsidemargin {60\p@}
     \setlength\evensidemargin {60\p@}
     \setlength\marginparwidth {90\p@}
  \fi
  \if@twocolumn
     \setlength\oddsidemargin {30\p@}
     \setlength\evensidemargin {30\p@}
     \setlength\marginparwidth {48\p@}
  \fi
  \if@stysize
    \if@twocolumn\else
      \setlength\oddsidemargin{0\p@}
      \setlength\evensidemargin{0\p@}
    \fi
  \fi
```

そしてやはり，LaTeX 2_ε モード：

```
\else
  \setlength\@tempdima{\paperwidth}
  \addtolength\@tempdima{-\textwidth}
  \if@twoside
    \setlength\oddsidemargin{.4\@tempdima}
  \else
    \setlength\oddsidemargin{.5\@tempdima}
  \fi
  \addtolength\oddsidemargin{-1in}
  \setlength\evensidemargin{\paperwidth}
  \addtolength\evensidemargin{-2in}
  \addtolength\evensidemargin{-\textwidth}
  \addtolength\evensidemargin{-\oddsidemargin}
  \@settopoint\oddsidemargin % 1999.1.6
  \@settopoint\evensidemargin
  \if@twoside
    \setlength\marginparwidth{.6\@tempdima}
    \addtolength\marginparwidth{-.4in}
  \else
    \setlength\marginparwidth{.5\@tempdima}
    \addtolength\marginparwidth{-.4in}
  \fi
  \ifdim \marginparwidth >2in
    \setlength\marginparwidth{2in}
  \fi
  \@settopoint\marginparwidth
\fi
```

ふ〜, 長かったですねぇ。こんなに長々と見てきましたが, PracTeX Journal の記事 [10] では LaTeX2.09 互換モードの部分は全部省略されていて，また，ペー

ジレイアウトには geometry を使いましょうと書いてあります…。

4.4 脚 注

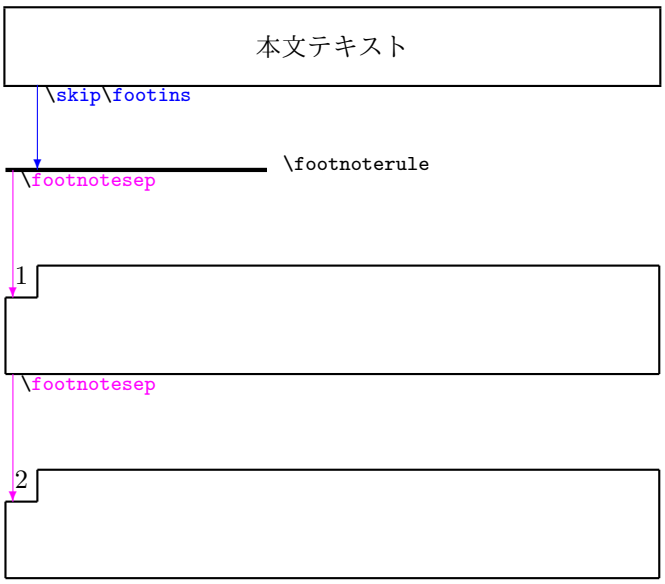
\footnotesep は LaTeX のマニュアル [6] にも書いてありますが, \skip\footins は載っていませんね。なお，この 6.65pt という値は，本文の文字サイズが 10pt のときの footnotesize の \baselineskip (= 9.5pt) の，0.7 倍（\strut の高さ）ということらしいです。

\footnotesep
\skip\footins

from: *jclasses.dtx*

\skip\footins は，本文の最終行と最初の脚注との間の距離です。

```
278 \setlength\footnotesep{6.65\p@}
279 \setlength{\skip\footins}{9\p@ \@plus 4\p@ \@minus 2\p@}
```



4.5 フロート

from: *jclasses.dtx*

すべてのフロートパラメータは，LaTeX のカーネルでデフォルトが定義されています。そのため，カウンタ以外のパラメータは \renewcommand で設定する必要があります。

4.5.1 フロートパラメータ

ページ上でテキストとフロートが混在する際の間隔（グルー）の設定です：

\floatsep
\textfloatsep
\intextsep
\dblfloatsep
\dbltextfloatsep

```
280 \setlength\floatsep  {12\p@ \@plus 2\p@ \@minus 2\p@}
281 \setlength\textfloatsep{20\p@ \@plus 2\p@ \@minus 4\p@}
282 \setlength\intextsep {12\p@ \@plus 2\p@ \@minus 2\p@}
283 \setlength\dblfloatsep  {12\p@ \@plus 2\p@ \@minus 2\p@}
284 \setlength\dbltextfloatsep{20\p@ \@plus 2\p@ \@minus 4\p@}
```

次の 6 つのパラメータは LaTeX のマニュアル [6] には載っていませんが，フロートのみが独立のページに出力される際の，フロートの上下やフロート同士の間に置かれるグルーの設定です（頭に “dbl” がついているものは，2 段組の際の段抜きフロートの場合）：

\@fptop
\@fpsep
\@fpbot
\@dblfptop
\@dblfpsep
\@dblfpbot

```
285 \setlength\@fptop{0\p@ \@plus 1fil}
286 \setlength\@fpsep{8\p@ \@plus 2fil}
287 \setlength\@fpbot{0\p@ \@plus 1fil}
288 \setlength\@dblfptop{0\p@ \@plus 1fil}
289 \setlength\@dblfpsep{8\p@ \@plus 2fil}
290 \setlength\@dblfpbot{0\p@ \@plus 1fil}
```

4.5.2 フロートオブジェクトの上限値

本文の文字サイズに依存する設定をほぼ終えて，ここでまた *jclasses.dtx* は jarticle.cls に戻ってきます。ページに置くことができるフロートの「数」の上限値をカウンタで設定しています：

\c@topnumber
\c@bottomnumber
\c@totalnumber
\c@dbltopnumber

```
164 \setcounter{topnumber}{2}
165 \setcounter{bottomnumber}{1}
166 \setcounter{totalnumber}{3}
167 \setcounter{dbltopnumber}{2}
```

今度は，数ではなく，ページ上で占めることが許される「割り合い」の設定です（カウンタは整数しか扱えないので，ここではマクロで定義されています）：

\topfraction
\bottomfraction
\textfraction
\floatpagefraction
\dbltopfraction
\dblfloatpagefraction

```
168 \renewcommand{\topfraction}{.7}
169 \renewcommand{\bottomfraction}{.3}
170 \renewcommand{\textfraction}{.2}
171 \renewcommand{\floatpagefraction}{.5}
172 \renewcommand{\dbltopfraction}{.7}
173 \renewcommand{\dblfloatpagefraction}{.5}
```

5 ページスタイル

ちょっと横着をして，*jclasses.dtx* から説明を転記します：

\ps@foo
\@evenhead
\@oddhead
\@evenfoot
\@oddfoot

from: *jclasses.dtx*

pLaTeX 2ε では，つぎの 6 種類のページスタイルを使用できます。empty は latex.dtx〔ママ〕で定義されています。

empty	ヘッダにもフッタにも出力しない
plain	フッタにページ番号のみを出力する
headnombre	ヘッダにページ番号のみを出力する
footnombre	フッタにページ番号のみを出力する
headings	ヘッダに見出しとページ番号を出力する
bothstyle	ヘッダに見出し，フッタにページ番号を出力する

ページスタイル foo は，\ps@foo コマンドとして定義されます。

\@evenhead, \@oddhead, \@evenfoot, \@oddfoot:

これらは \ps@... から呼び出され，ヘッダとフッタを出力するマクロです。

〔…中略…〕

これらの内容は，横組の場合は \textwidth の幅を持つ \hbox に入れられ，縦組の場合は \textheight の幅を持つ \hbox に入れられます。

実際には “myheadings” というページスタイルも定義しているので，pLaTeX でデフォルトで使えるページスタイルは全部で 7 種類だと思います。\ps@empty と \ps@plain が定義をされているのは *ltpage.dtx* においてですね。この “empty” と “plain” という 2 つのスタイルはカーネル（← *ltpage.dtx*）で定義済みのため，クラスファイルで改めて定義しなくても使えます。empty，plain，headings，myheadings の 4 種類は元々の LaTeX のクラスファイルでも定義されているスタイルですが，pLaTeX では更に headnombre，footnombre，bothstyle の 3 種類を使えるように定義してあるというわけです。

それで，\pagestyle{foo}という風にユーザーコマンド \pagestyle の引数としてページスタイル “foo” が指定できるためには，“\ps@foo” というマクロが定義されてなくてはいけなくて，その \ps@foo の中では，\@evenhead, \@oddhead, \@evenfoot, \@oddfoot が定義をされてなくっちゃいけない，ということです。

5.1 マークについて

ここでもまずは，*jclasses.dtx* を引き写しておきます：

\markboth
\markright
\leftmark
\rightmark
\@mkboth

from: *jclasses.dtx*

ヘッダに入る章番号や章見出しは，見出しコマンドで実行されるマークコマンドで決定されます。ここでは，実行されるマークコマンドの定義を行なっています。これらのマークコマンドは，TeX の \mark 機能を用いて，‘left’ と ‘right’ の 2 種類のマークを生成するように定義しています。

\markboth{⟨LEFT⟩}{⟨RIGHT⟩}: 両方のマークに追加します。
\markright{⟨RIGHT⟩}: ‘右’ マークに追加します。

\leftmark:
\@oddhead，\@oddfoot，\@evenhead，\@evenfoot マクロで使われ，現在の “左” マークを出力します。\leftmark は TeX の \botmark コマンドのような働きをします。初期値は空でなくてはいけません。

\rightmark:
\@oddhead，\@oddfoot，\@evenhead，\@evenfoot マクロで使われ，現在の “右” マークを出力します。\rightmark は TeX の \firstmark コマンドのような働きをします。初期値は空でなくてはいけません。

マークコマンドの動作は，左マークの ‘範囲内の’ 右マークのために合理的になっています。たとえば，左マークは \chapter コマンドによって変更されます。そして右マークは \section コマンドによって変更されます。しかし，同一ページに複数の \markboth コマンドが現れたとき，おかしな結果となることがあります。

\tableofcontents のようなコマンドは，\@mkboth コマンドを用いて，あるページスタイルの中でマークを設定しなくてはなりません。\@mkboth は，\ps@... コマンドによって，\markboth（ヘッダを設定する）か，\@gobbletwo（何もしない）に \let されます。

ここでは \mark，\botmark，\firstmark といった TeX のプリミティブに深入りすることはせずに，文系初級的理解にとどめます（なので，不正確極まりない説明です）。まず，\markboth というマクロは引数を 2 つとって，それらを「左マーク」「右マーク」としてその場所にマークを付けるものであって，他方 \markright は引数がひとつで，その引数をその場所に「右マーク」としてマークするものです。そして，予めこのようにしてマークしておいた文字列を，後からページを出力する際に取得するには，\leftmark で（そのページ内最後の）「左マーク」を，\rightmark で（そのページ内最初の）「右マーク」を取り出すことができる仕組みになっていま

す（当該ページにマークがない場合には，前のページのマークがそれぞれ引き継がれていきます）。

「ヘッダに入る章番号や章見出しは，見出しコマンドで実行されるマークコマンドで決定されます。」ということなので，\section とか \subsection といった見出しコマンドの中で \markboth や \markright が使われていて，その見出しの場所でマークを付けていることになります。そのマークの中味が，\section や \subsection の引数（＋見出しの番号とか）ということになるはずです。

plain，headnombre，footnombre といったページスタイルではマークは関係ないので，ちょっと話を先取りすることになってしまっていますが，headings スタイルの定義の 197〜202 行の部分を見てみましょう。定義の中の定義であることから “#” が重ねて使ってあることや，カウンタ \c@secnumdepth の値によって見出しの番号を付するかどうかの条件分岐のところなどを省略して，且つ LaTeX-ey に書き直してみますと，次のようになります：

```
\renewcommand{\sectionmark}[1]{%
   \markboth{\thesection.\hspace{1zw}#1}{}%
}

\renewcommand{\subsectionmark}[1]{%
   \markright{\thesubsection.\hspace{1zw}#1}%
}
```

したがって，\markboth による左マークは「セクション番号（\thesection）＋ピリオド（.）＋全角 1 文字分のアキ（\hspace{1zw}）＋ \sectionmark の引数（#1）」で右マークは空っぽ（{}），\rightmark による右マークは「サブセクション番号＋ピリオド＋全角 1 文字分のアキ＋ \subsectionmark の引数」ということになっています。そして，こうしてマークしたものを，\leftmark や \rightmark として利用することができます（194〜195 行）：

```
\def\@evenhead{\thepage\hfil\leftmark}%
\def\@oddhead{{\rightmark}\hfil\thepage}%
```

さて，ということは，\section の定義の中で \sectionmark が使われていて，そこで \section の引数（＝見出しの文字列）が \sectionmark にも渡されているはずで，同様に，\subsection の定義の中では \subsectionmark が使われているはずです。

でも，そう考えて source2e の索引を探したり，latex.ltx を検索したりしても，次のようなものしか見つかりません：

```
\let\sectionmark\@gobble
\let\subsectionmark\@gobble
\let\subsubsectionmark\@gobble
\let\paragraphmark\@gobble
\let\subparagraphmark\@gobble
```

どれも，引数をひとつ取って呑み込むだけの \@gobble に \let されてるだけです。実は，（\section 以下の）見出しコマンドの中で \...mark が使われているのは，「6.1 章見出し」に出てくる内部コマンド \@startsection の更に下請けの内部コマンド \@sect の定義の次のような部分にです：

```
\def\@sect#1#2#3#4#5#6[#7]#8{%
...
   \csname #1mark\endcsname{#7}%
...
}
```

ここで，#1 は “section” とか “subsection” とかで，#7 は見出しコマンドの必須引数ないしオプション引数〔＝見出しの文字列〕です（オプション引数があれば，それはそのまま #7 になり，オプション引数がない場合は必須引数 #8 のコピーが #7 になります．*cf.* 後掲註 (14)，後掲 Excursus）。

このようにして，\section の場合には \sectionmark がつくられ，\subsection の場合には \subsectionmark がつくられる，という風に動的にマークコマンドが設定されています。

そして，\@mkboth というのは，目次（図目次・表目次を含む）や thebibliography 環境，theindex 環境などの定義に入れてあって，引数をふたつとる形になっています：

→ 8.1 目次
→ 8.2 参考文献
→ 8.3 索引
→ 10 初期設定

```
664:  \@mkboth{\contentsname}{\contentsname}
703:  \@mkboth{\listfigurename}{\listfigurename}
709:  \@mkboth{\listtablename}{\listtablename}
717:  \@mkboth{\refname}{\refname}
739:  \@mkboth{\indexname}{\indexname}
```

\@mkboth 自体はどこでも定義されていないので(10)，\@mkboth の挙動を，各ページスタイルの中で決めておく必要があるわけです。一般に，柱を入れるのであれば \markboth に \let し，柱を入れないページスタイルであるなら \@gobbletwo（ふたつの引数をただ呑み込んじゃうマクロ）に \let することになっています。

5.2 plain ページスタイル

\ps@plain

plain スタイルでは \@oddhead と \@evenhead を空にし，そしてヘッダを空にするのですから\@mkboth は \@gobbletwo に \let しています。\@oddfoot ではノンブルをセンタリングして，\@evenfoot にはその \@oddfoot を \let しています。\ps@jpl@in については，次節に *jclasses.dtx* を引用します。

```
174  \def\ps@plain{\let\@mkboth\@gobbletwo
175    \let\ps@jpl@in\ps@plain
176    \let\@oddhead\@empty
177    \def\@oddfoot{\reset@font\hfil\thepage\hfil}%
178    \let\@evenhead\@empty
179    \let\@evenfoot\@oddfoot}
```

5.3 jpl@in ページスタイル

\ps@jpl@in

from: *jclasses.dtx*

> jpl@in スタイルは，クラスファイル内部で使用するものです。LaTeX では，book クラスを headings としています。しかし，\tableofcontnts コマンドの内部では plain として設定されるため，一つの文書でのページ番号の位置が上下に出力されることになります。
>
> そこで，pLaTeX 2ε では，\tableofcontents や \theindex のページスタイルを jpl@in にし，実際に出力される形式は，ほかのページスタイルで \let をしています。したがって，headings のとき，目次ページのページ番号はヘッダ位置に出力され，plain のときには，フッタ位置に出力されます。

```
180  \let\ps@jpl@in\ps@plain
```

(10) 「10 初期設定」の 791 行で \pagestyle{plain}と設定されていて，そして plain スタイルの中で \@mkboth は \@gobbletwo に \let されているので，未定義エラーにはなりません。

欧文の出版物などを見ますと「一つの文書でのページ番号の位置が上下に出力される」というのは格別おかしくはないとも思われるのですけど，日本的組版感覚では，文書全体でスタイルが揃うのが望ましいのでしょうか。

それで，元々の LaTeX のクラスファイルでは，

- （表題ページを独立ページにしない場合の）`\maketitle`
- （book.cls や report.cls の）`\part`
- `\chapter`
- `theindex` 環境

で `\thispagestyle{plain}` が使われており，また，

- book.cls や report.cls の `\tableofcontents` の定義の中では `\chapter*` が使われている

ため，ページスタイルを仮に `empty` や `headings` に指定していても，

- 独立ページにしない場合の表題ページ
- （book.cls や report.cls の）部扉
- 章の開始ページ
- 索引の開始ページ
- 目次の開始ページ

では，デフォルトだとフッタの中央にノンブルが出力されます。

これに対して pLaTeX では，上記の各所で `\thispagestyle{plain}` の代わりに `\thispagestyle{jpl@in}` としてあって，そして `jpl@in` の実際のスタイルは，各ページスタイルの中で特定のスタイルに `\let` することにして，全体のページスタイルを揃えるようにしているわけです。

5.4 headnombre ページスタイル

`headnombre` スタイルでは，ヘッダの小口側にノンブルを入れ，フッタは空にしています。また，`\@mkboth` は `\@gobbletwo` に，`\ps@jpl@in` は `\ps@headnombre` に `\let` されています。

\ps@headnombre

```
\def\ps@headnombre{\let\@mkboth\@gobbletwo
   \let\ps@jpl@in\ps@headnombre
  \def\@evenhead{\thepage\hfil}%
  \def\@oddhead{\hfil\thepage}%
  \let\@oddfoot\@empty\let\@evenfoot\@empty}
```

5.5 footnombre ページスタイル

`footnombre` スタイルでは，フッタの小口側にノンブルを入れ，ヘッダを空にしています。`\@mkboth` は `\@gobbletwo` に，`\ps@jpl@in` は `\ps@footnombre` に `\let` されています。

\ps@footnombre

```
\def\ps@footnombre{\let\@mkboth\@gobbletwo
   \let\ps@jpl@in\ps@footnombre
  \def\@evenfoot{\thepage\hfil}%
  \def\@oddfoot{\hfil\thepage}%
  \let\@oddhead\@empty\let\@evenhead\@empty}
```

5.6 headings スタイル

headings スタイルでは，ヘッダの小口側にノンブルを，ノド側に \section の見出しないし \subsection の見出しを入れ，フッタは空にしています（片面印刷の場合は \section の見出しのみ）。\@mkboth は \markboth に，\ps@jpl@in は \ps@headnombre に \let されています。 *\ps@headings*

```
\if@twoside
  \def\ps@headings{\let\ps@jpl@in\ps@headnombre
    \let\@oddfoot\@empty\let\@evenfoot\@empty
    \def\@evenhead{\thepage\hfil\leftmark}%
    \def\@oddhead{{\rightmark}\hfil\thepage}%
    \let\@mkboth\markboth
    \def\sectionmark##1{\markboth{%
       \ifnum \c@secnumdepth >\z@ \thesection.\hskip1zw\fi
       ##1}{}}%
    \def\subsectionmark##1{\markright{%
       \ifnum \c@secnumdepth >\@ne \thesubsection.\hskip1zw\fi
       ##1}}%
  }
\else % if not twoside
  \def\ps@headings{\let\ps@jpl@in\ps@headnombre
    \let\@oddfoot\@empty
    \def\@oddhead{{\rightmark}\hfil\thepage}%
    \let\@mkboth\markboth
  \def\sectionmark##1{\markright{%
     \ifnum \c@secnumdepth >\m@ne \thesection.\hskip1zw\fi
     ##1}}%
  }
\fi
```

5.7 bothstyle スタイル

bothstyle スタイルでは，ヘッダの小口側には \section の見出しないし \subsection の見出し，フッタの小口側にはノンブルを入れています（片面印刷の場合は \section の見出しのみ）。\@mkboth は \markboth に，\ps@jpl@in は \ps@footnombre に \let されています[11]。 *\ps@bothstyle*

```
\if@twoside
  \def\ps@bothstyle{\let\ps@jpl@in\ps@footnombre
    \def\@evenhead{\leftmark\hfil}% right page
    \def\@evenfoot{\thepage\hfil}% right page
    \def\@oddhead{\hfil\rightmark}% left page
    \def\@oddfoot{\hfil\thepage}% left page
  \let\@mkboth\markboth
  \def\sectionmark##1{\markboth{%
     \ifnum \c@secnumdepth >\z@ \thesection.\hskip1zw\fi
     ##1}{}}%
  \def\subsectionmark##1{\markright{%
     \ifnum \c@secnumdepth >\@ne \thesubsection.\hskip1zw\fi
     ##1}}%
  }
\else % if one column
  \def\ps@bothstyle{\let\ps@jpl@in\ps@footnombre
    \def\@oddhead{\hfil\rightmark}%
```

(11) ちなみに，216〜219 行のコメントの right と left は，逆だと思います。

```
    \def\@oddfoot{\hfil\thepage}%
    \let\@mkboth\markboth
  \def\sectionmark##1{\markright{%
     \ifnum \c@secnumdepth >\m@ne \thesection.\hskip1zw\fi
     ##1}}%
  }
\fi
```

5.8 myheading スタイル

myheadings スタイル[12]では，ヘッダの小口側にノンブルを入れ，ノド側には \leftmark と \rightmark を入れていますが，\sectionmark と \subsectionmark はリセットされているので，ユーザーが自分で \markboth や \markright でマークをしなくてはなりません。\@mkboth は \@gobbletwo に，\ps@jpl@in は \ps@plain に \let されています。

\ps@myheadings

```
\def\ps@myheadings{\let\ps@jpl@in\ps@plain%
  \let\@oddfoot\@empty\let\@evenfoot\@empty
  \def\@evenhead{\thepage\hfil\leftmark}%
  \def\@oddhead{{\rightmark}\hfil\thepage}%
  \let\@mkboth\@gobbletwo
  \let\sectionmark\@gobble
  \let\subsectionmark\@gobble
}
```

ページスタイルについても，PracTEX Journal の記事 [10] では，fancyhdr を使うことを薦めています…。

6 文書コマンド

6.0.1 表 題

ここでは，titlepage 環境と，\maketitle が定義されています。

\tilte，\author，\date は *ltsect.dtx* で定義済みなので，*jclasses.dtx* ではコメントアウトされています：

\title
\author
\date

from: *jclasses.dtx*

文書のタイトル，著者，日付の情報のための，これらの 3 つのコマンドは latex.dtx 〔ママ〕で提供されています。これらのコマンドは次のように定義されています。

```
%\newcommand*{\title}[1]{\gdef\@title{#1}}
%\newcommand*{\author}[1]{\gdef\@author{#1}}
%\newcommand*{\date}[1]{\gdef\@date{#1}}
```

\date マクロのデフォルトは，今日の日付です。

```
%\date{\today}
```

まずは，LaTeX2.09 互換モードの場合の titlepage 環境の定義です。\twocolumntrue の場合は，\begin{titlepage}の定義内で \@restonecoltrue となるので，\end{titlepage}の

titlepage

(12) 見出しは「myheading スタイル」となってますけど。

定義内で\twocolumn に戻されています。また，表題ページの \c@page カウンタを “0” にしているので，表題ページの次のページのノンブルが “1” になります。

```
\if@compatibility
\newenvironment{titlepage}
    {%
     \if@twocolumn\@restonecoltrue\onecolumn
     \else\@restonecolfalse\newpage\fi
     \thispagestyle{empty}%
     \setcounter{page}\z@
    }%
    {\if@restonecol\twocolumn\else\newpage\fi
    }
```

続いて，LaTeX 2ε モードの場合の titlepage 環境です。やはり \if@twocolumn と \if@restonecol とを使って，二段組の場合は，\begin{titlepage}のところで一時的に一段組にした後，\end{titlepage}のところで二段組に戻しています。また，表題ページの次のページのノンブルは，片面印刷の場合には “1” となり，両面印刷の場合には “2” となります。

→ cf. 8.3 索引

```
\else
\newenvironment{titlepage}
    {%
      \if@twocolumn
        \@restonecoltrue\onecolumn
      \else
        \@restonecolfalse\newpage
      \fi
      \thispagestyle{empty}%
      \setcounter{page}\@ne
    }%
    {\if@restonecol\twocolumn \else \newpage \fi
     \if@twoside\else
        \setcounter{page}\@ne
     \fi
    }
\fi
```

\p@thanks については，*jclasses.dtx* をただ転記しておきます… :

\p@thanks

from: *jclasses.dtx*

> 縦組のときは，\thanks コマンドを \p@thanks に \let します。このコマンドは \footnotetext を使わず，直接，文字を \@thanks に格納していきます。

```
\def\p@thanks#1{\footnotemark
  \protected@xdef\@thanks{\@thanks
    \protect{\noindent$\m@th^\thefootnote$~#1\protect\par}}}
```

最初に，表題ページを独立させる場合の \maketitle の定義です :

\maketitle

```
\if@titlepage
  \newcommand{\maketitle}{\begin{titlepage}%
  \let\footnotesize\small
  \let\footnoterule\relax
  \let\footnote\thanks
  \null\vfil
```

```
  \vskip 60\p@
  \begin{center}%
    {\LARGE \@title \par}%
    \vskip 3em%
    {\Large
     \lineskip .75em%
      \begin{tabular}[t]{c}%
        \@author
      \end{tabular}\par}%
      \vskip 1.5em%
    {\large \@date \par}%     % Set date in \large size.
  \end{center}\par
  \@thanks\vfil\null
  \end{titlepage}%
```

この後は後処理です。一度 \maketitle を使うと，後は使えなくなります：

```
  \setcounter{footnote}{0}%
  \global\let\thanks\relax
  \global\let\maketitle\relax
  \global\let\p@thanks\relax
  \global\let\@thanks\@empty
  \global\let\@author\@empty
  \global\let\@date\@empty
  \global\let\@title\@empty
  \global\let\title\relax
  \global\let\author\relax
  \global\let\date\relax
  \global\let\and\relax
  }%
```

次に，表題ページを独立させない場合の，\maketitle の定義です。実際の出力は \@maketitle に下請けに出しています：

```
\else
  \newcommand{\maketitle}{\par
  \begingroup
    \renewcommand{\thefootnote}{\fnsymbol{footnote}}%
    \def\@makefnmark{\hbox{\ifydir $\m@th^{\@thefnmark}$
      \else\hbox{\yoko$\m@th^{\@thefnmark}$}\fi}}%
     \long\def\@makefntext##1{\parindent 1em\noindent
       \hbox to1.8em{\hss$\m@th^{\@thefnmark}$}##1}%
    \if@twocolumn
      \ifnum \col@number=\@ne \@maketitle
      \else \twocolumn[\@maketitle]%
      \fi
    \else
      \newpage
      \global\@topnum\z@ % Prevents figures from going at top of page.
      \@maketitle
    \fi
     \thispagestyle{jpl@in}\@thanks
  \endgroup
```

ここからはやはり後処理部分です：

```
  \setcounter{footnote}{0}%
  \global\let\thanks\relax
  \global\let\maketitle\relax
  \global\let\p@thanks\relax
```

```
  \global\let\@thanks\@empty
  \global\let\@author\@empty
  \global\let\@date\@empty
  \global\let\@title\@empty
  \global\let\title\relax
  \global\let\author\relax
  \global\let\date\relax
  \global\let\and\relax
  }
```

そして下請けの \@maketitle の定義です： *\@maketitle*

```
  \def\@maketitle{%
  \newpage\null
  \vskip 2em%
  \begin{center}%
  \let\footnote\thanks
    {\LARGE \@title \par}%
    \vskip 1.5em%
    {\large
      \lineskip .5em%
      \begin{tabular}[t]{c}%
        \@author
      \end{tabular}\par}%
    \vskip 1em%
    {\large \@date}%
  \end{center}%
  \par\vskip 1.5em}
\fi
```

307 行と 339 行で \and が \relax に \let されていますが, \and は *ltsect.dtx* で次のように定義されています：

```
\def\and{%                % \begin{tabular}
  \end{tabular}%
  \hskip 1em \@plus.17fil%
  \begin{tabular}[t]{c}}% % \end{tabular}
```

6.0.2 概 要

abstract 環境の定義です。\@titlepagetrue だと, 独立したページになります。 *abstract*

```
\if@titlepage
  \newenvironment{abstract}{%
      \titlepage
      \null\vfil
      \@beginparpenalty\@lowpenalty
      \begin{center}%
        {\bfseries\abstractname}%
        \@endparpenalty\@M
      \end{center}}%
      {\par\vfil\null\endtitlepage}
\else
  \newenvironment{abstract}{%
    \if@twocolumn
      \section*{\abstractname}%
    \else
      \small
```

```
      \begin{center}%
        {\bfseries\abstractname\vspace{-.5em}\vspace{\z@}}%
      \end{center}%
      \quotation
    \fi}{\if@twocolumn\else\endquotation\fi}
\fi
```

6.1 章見出し

jclasses.dtx の「6.1 章見出し」の部分には，見出しだけで中味が書いてありません。以前のバージョンでは何か解説してあったのを，今では削除されたのでしょうか…。

6.2 マークコマンド

from: *jclasses.dtx*

\...mark コマンドを初期化します。これらのコマンドはページスタイルの定義で使われます（第 5 節参照）。これらのたいていのコマンドは latex.dtx ですでに定義されています。

```
%\newcommand*{\sectionmark}[1]{}
%\newcommand*{\subsectionmark}[1]{}
%\newcommand*{\subsubsectionmark}[1]{}
%\newcommand*{\paragraph}[1]{}
%\newcommand*{\subparagraph}[1]{}
```

「5.1 マークについて」の部分で既に引用していますが，マークコマンドは実際には *ltsect.dtx* で，次のように初期化されています：

```
\let\sectionmark\@gobble
\let\subsectionmark\@gobble
\let\subsubsectionmark\@gobble
\let\paragraphmark\@gobble
\let\subparagraphmark\@gobble
```

6.2.1 カウンタの定義

\c@secnumdepth のデフォルトの深さを設定して，それから章立て用のカウンタを宣言し，それぞれのカウンタの出力形式を決めています。\c@secnumdepth は *ltsect.dtx* で宣言されています。

```
\setcounter{secnumdepth}{3}                                  \c@secnumdepth

\newcounter{part}                                            \c@part
\newcounter{section}                                         \c@section
\newcounter{subsection}[section]                             \c@subsection
\newcounter{subsubsection}[subsection]                       \c@subsubsection
\newcounter{paragraph}[subsubsection]                        \c@paragraph
\newcounter{subparagraph}[paragraph]                         \c@subparagraph
```

```
\renewcommand{\thepart}{\@Roman\c@part}
\renewcommand{\thesection}{\@arabic\c@section}
\renewcommand{\thesubsection}{\thesection.\@arabic\c@subsection}
\renewcommand{\thesubsubsection}{%
  \thesubsection.\@arabic\c@subsubsection}
\renewcommand{\theparagraph}{%
  \thesubsubsection.\@arabic\c@paragraph}
\renewcommand{\thesubparagraph}{%
  \theparagraph.\@arabic\c@subparagraph}
```

\thepart
\thesection
\thesubsection
\thesubsubsection
\theparagraph
\thesubparagraph

6.2.2 前付け，本文，後付け

\frontmatter，\mainmatter，\backmatter は jbook.cls でしか用意されていません：

\frontmatter
\mainmatter
\backmatter

```
\newcommand\frontmatter{%
  \if@openright \cleardoublepage \else \clearpage \fi
  \@mainmatterfalse\pagenumbering{roman}}
\newcommand{\mainmatter}{%
  \if@openright \cleardoublepage \else \clearpage \fi
  \@mainmattertrue\pagenumbering{arabic}}
\newcommand{\backmatter}{%
  \if@openright \cleardoublepage \else \clearpage \fi
  \@mainmatterfalse}
```

6.2.3 ボックスの組み立て[13]

jclasses.dtx のこの部分では，見出しのマクロ定義に使われる内部コマンドである\@startsection と \secdef について説明がされていますが，本文書では\@startsection の説明は「6.2.6 下位レベルの見出し」のところに移動しています。

というわけで，先に \secdef の説明から：

\secdef

from: *jclasses.dtx*

\secdef マクロは，見出しコマンドを \@startsection を用いないで定義するときに使います。このマクロは，2 つの引数を持ちます。

\secdef⟨unstarcmds⟩⟨starcmds⟩

⟨unstarcmds⟩　見出しコマンドの普通の形式で使われます。
⟨starcmds⟩　　* 形式の見出しコマンドで使われます。

\secdef は次のようにして使うことができます。

```
\def\chapter {... \secdef \CMDA \CMDB }
\def\CMDA [#1]#2{....} % \chapter[...]{...} の定義
\def\CMDB #1{....} % \chapter*{...} の定義
```

\secdef は，\part や \chapter の内部で使われているマクロですが，星の有無で分岐して，例えば\def\command{\secdef{\CMDA}{\CMDB}}とすると，星ナシの “\command” のときには “\CMDA” となり（且つ，オプション引数がない場合には，必須引数がオプション引数にコピーされます），星アリで “\command*” だと “\CMDB”

[13] *classes.dtx* には “Building blocks” とありますが…。

に展開される仕組みになっています。通常，見出しコマンドは，[] で囲まれたオプション引数を取れるように定義しますが，オプション引数がとれるのは星ナシの場合です。星の有無，オプション引数の有無で，\secdef による “\command” の挙動を整理すると，次のようになります(14)：

- \command{foo} → \CMDA[foo]{foo}
- \command[bar]{foo} → \CMDA[bar]{foo}
- \command*{foo} → \CMDB{foo}

6.2.4 part レベル

\part の定義です。jarticle.cls での \part の定義は，jreport.cls や jbook.cls に比べてとても簡単になっています。\@afterindenttrue とすると，見出しの後の最初の段落で字下げされます（元々の LaTeX のクラスファイルの見出しコマンドでは，ここが \@afterindentfalse になっています）。 *\part*

まず，\secdef を使って，星の有無で分岐します：

- \part{title} → \@part[title]{title}
- \part[toc-title]{title} → \@part[toc-title]{title}
- \part*{title} → \@spart{title}

```
396 \newcommand{\part}{\par\addvspace{4ex}%
397   \@afterindenttrue
398   \secdef\@part\@spart}
```

星ナシの場合の \part の本体です（399〜415 行）： *\@part*

```
399 \def\@part[#1]#2{%
```

最初は目次に関する設定です。\c@secnumdepth が 0 以上の場合には，\c@part をインクリメントして，“.toc” ファイルには “\prepartname\thepart\postpartname \hspace{1zw}#1” が送られ（“#1” は見出しの文字列．\part のオプション引数に相当する部分），\c@secnumdepth が −1 以下の場合には “.toc” には見出し文字列のみが送られます。また，マークはどちらも空にしています。 → 10 初期設定

```
400   \ifnum \c@secnumdepth >\m@ne
401     \refstepcounter{part}%
402     \addcontentsline{toc}{part}{%
403       \prepartname\thepart\postpartname\hspace{1zw}#1}%
404   \else
405     \addcontentsline{toc}{part}{#1}%
406   \fi
407   \markboth{}{}%
```

続いて，part を実際に組版する部分です。部見出し（\part の必須引数 “#2”）は \huge\bfseries で組まれ，また，\c@secnumdepth が 0 以上の場合には，番号部分が \Large\bfseries で組まれます。

(14) \command{foo}が\CMDA[foo]{foo}になるというのが不思議に思われるでしょうけれど，これは \secdef の内部で使われている \@dblarg というマクロの働きによるものです。

```
{\parindent\z@\raggedright
 \interlinepenalty\@M\reset@font
 \ifnum \c@secnumdepth >\m@ne
   \Large\bfseries\prepartname\thepart\postpartname
   \par\nobreak
 \fi
 \huge\bfseries#2\par}%
\nobreak\vskip3ex\@afterheading}
```

星アリの場合の \part の本体です。この場合は，“.toc” ファイルには何も送られず，番号もつかず，部見出しの文字列が \huge\bfseries で組まれるだけです。 \@spart

```
\def\@spart#1{{%
 \parindent\z@\raggedright
 \interlinepenalty\@M\reset@font
 \huge\bfseries#1\par}%
 \nobreak\vskip3ex\@afterheading}
```

415 行と 420 行に \@afterheading という後処理のマクロが入れてありますが，397 行の\@afterindenttrue は，この \@afterheading とペアで，見出しの次の最初の段落の字下げをコントロールしているらしいです。でも，\@afterheading の定義を見ても，初級の私には意味が分かりませんでした…。

6.2.5 chapter レベル

jarticle.cls には \chapter はありません。定義自体は \part にとてもよく似ています。違いは，\if@openright や \if@mainmatter で分岐していたり，\chaptermark の設定が入っていたり，\@chapter は chapter の実際の組版を \@makechapterhead に下請けに出し，\@schapter は \@makeschapterhead に下請けに出しているところくらいです。

6.2.6 下位レベルの見出し

「6.2.3 ボックスの組み立て」からこちらに引越しさせた \@startsection の説明です（ここではしかも，*classes.dtx* のほうから引用をします）： \@startsection

from: *classes.dtx*

7.2.1 Building blocks

The macro \@startsection has 6 required arguments, optionally followed by a *, an optional argument and a required argument:

\@startsection⟨*name*⟩⟨*level*⟩⟨*indent*⟩⟨*beforeskip*⟩⟨*afterskip*⟩⟨*style*⟩optional * [⟨*altheading*⟩]⟨*heading*⟩

It is a generic command to start a section, the arguments have the following meaning:

⟨*name*⟩	The name of the user level command, e.g., ‘section’.
⟨*level*⟩	A number, denoting the depth of the section – e.g., chapter = 1, section = 2, etc. A section number will be printed if ⟨*level*⟩ <= the value of the secnumdepth counter.
⟨*indent*⟩	The indentation of the heading from the left margin.
⟨*beforeskip*⟩	The absolute value of this argument gives the skip to leave above the heading. If it is negative, then the paragraph indent of the text following the heading is suppressed.
⟨*afterskip*⟩	If positive, this gives the skip to leave below the heading, else it gives the skip to leave to the right of a run-in heading.

⟨*style*⟩	Commands to set the style of the heading.
*	When this is missing the heading is numbered and the corresponding counter is incremented.
⟨*altheading*⟩	Gives an alternative heading to use in the table of contents and in the running heads. This should be present when the * form is used. [*sic*] (15)
⟨*heading*⟩	The heading of the new section.

A sectioning command is normally defined to `\@startsection` and its first six arguments.

`\part` や `\chapter` では，`\if@afterheading` で見出しの後の最初の段落の字下げを制御していますが，`\section` 以下の見出しでは，`\@startsection` の第 4 引数 ⟨*beforeskip*⟩ の正負をスイッチに使っています（LaTeX の元々のクラスファイルでは，`\section` 以下の設定で第 4 引数の ⟨*beforeskip*⟩ を負にして，見出し後のインデントを抑制しています）：

article.cls:

```
\newcommand\section{\@startsection
   {section}% name
   {1}% level
   {\z@}% indent
   {-3.5ex \@plus -1ex \@minus -.2ex}% beforeskip
   {2.3ex \@plus.2ex}% afterskip
   {\normalfont\Large\bfseries}}% style
```

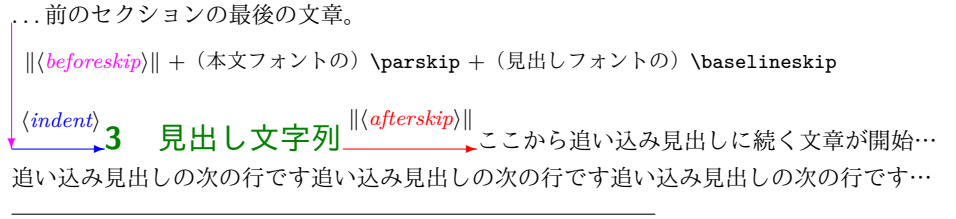
jarticle.cls:

```
\newcommand{\section}{\@startsection
   {section}% name
   {1}% level
   {\z@}% indent
   {1.5\Cvs \@plus.5\Cvs \@minus.2\Cvs}% beforeskip
   {.5\Cvs \@plus.3\Cvs}% afterskip
   {\reset@font\Large\bfseries}}% style
```

また，第 5 引数 ⟨*afterskip*⟩ の正負で，見出しをディスプレイ見出しにするか，追い込み（run-in）見出しにするかの制御をしています。第 5 引数 ⟨*afterskip*⟩ の正負による，ディスプレイ見出しと追い込み見出しの違いを図示してみるとこんな感じです：

ディスプレイ見出しの例（*afterskip* >0）：

追い込み見出しの例（*afterskip* <0）：



(15) *ltsect.dtx* からも引用しておきます：“If ‘*’ is missing, then increment the counter. If it is present, then there should be no [⟨*altheading*⟩] argument.”

\section，\subsection，\subsubsection の設定です。\@startsection の第 5 引数が正なので，ディスプレイ見出しになります：

```
\newcommand{\section}{\@startsection{section}{1}{\z@}%
  {1.5\Cvs \@plus.5\Cvs \@minus.2\Cvs}%
  {.5\Cvs \@plus.3\Cvs}%
  {\reset@font\Large\bfseries}}
```
\section

```
\newcommand{\subsection}{\@startsection{subsection}{2}{\z@}%
  {1.5\Cvs \@plus.5\Cvs \@minus.2\Cvs}%
  {.5\Cvs \@plus.3\Cvs}%
  {\reset@font\large\bfseries}}
```
\subsection

```
\newcommand{\subsubsection}{\@startsection{subsubsection}{3}{\z@}%
  {1.5\Cvs \@plus.5\Cvs \@minus.2\Cvs}%
  {.5\Cvs \@plus.3\Cvs}%
  {\reset@font\normalsize\bfseries}}
```
\subsubsection

\paragraph，\subparagraph の設定です。\@startsection の第 5 引数が負なので，追い込み見出しになります。

```
\newcommand{\paragraph}{\@startsection{paragraph}{4}{\z@}%
  {3.25ex \@plus 1ex \@minus .2ex}%
  {-1em}%
  {\reset@font\normalsize\bfseries}}
```
\paragraph

```
\newcommand{\subparagraph}{\@startsection{subparagraph}{5}{\z@}%
  {3.25ex \@plus 1ex \@minus .2ex}%
  {-1em}%
  {\reset@font\normalsize\bfseries}}
```
\subparagraph

日本語の出版物では，見出しの次の最初の段落も普通は字下げをするので，\section 以下の見出しコマンドの \@startsection の第 4 引数は，いずれも正になっています。

\@startsection の第 2 引数がきれいに 1, 2, 3, 4, 5 と並んでいますが，これが，よく見かける

command	level
\part	level 0
\section	level 1
\subsection	level 2
\subsubsection	level 3
\paragraph	level 4
\subparagraph	level 5

という見出しコマンドの「レベル」を決定している部分ですね。（\part については 400 行の “\ifnum \c@secnumdepth >\m@ne” という行がレベルを決めています．なお，jbook.cls や jreport.cls には \chapter があるので，\chapter のレベルが 0 で，\part のレベルが −1 に設定されています）。

見出しのレベルは，見出しを組版する際と，見出しを “.toc” に送る際，そして \mark... による柱を設定する際に，見出しに番号を付するか否かの条件分岐に使われています（\c@secnumdepth の値とレベルとが比較されています）。前掲「6.2.4 part レベル」の \@part と似たような定義が，\@startsection の下請けの \@sect の中にあります[16]。

→ 5 ページスタイル

[16] PracTEX Journal の記事 [10] では，\@startsection には深入りせずに，sectsty を使うことが薦められています。

Excursus

「jarticle.cls の中味をただ粛々と追うだけです」と言ったことにちょっと反する気もするので，以下は “Excursus” として入れておきます：

(1) \@startsection の末尾では，\secdef と同じ仕組みで分岐をして，\@sect と \@ssect に下請けに出しています。\@startsection の 6 つの引数が a, b, c, d, e, f と与えられている場合に，星の有無，オプション引数の有無で \section の分岐を整理すると，次のようになります：

- \section{title} → \@sect{a}{b}{c}{d}{e}{f}[title]{title}
- \section[toc-title]{title} → \@sect{a}{b}{c}{d}{e}{f}[toc-title]{title}
- \section*{title} → \@ssect{c}{d}{e}{f}{title}

(2) \@sect の中で，“.toc” ファイルに送る文字列に番号を付けるか否かを決めているのは，次の部分です：

```
\addcontentsline{toc}{#1}{%
  \ifnum #2>\c@secnumdepth \else
    \protect\numberline{\csname the#1\endcsname}%
  \fi
  #7}}%
```

ここで，“#1” は \@startsection の ⟨*name*⟩，“#2” は ⟨*level*⟩，そして “#7” は見出しコマンドのオプション引数部分に当たります（オプション引数がない場合には必須引数のコピー）。なお，“.toc” に送られた情報を使って実際に目次にその見出し項目を記載するか否かは，\c@tocdepth で制御されます（→ 8.1 目次）。

(3) 見出しを組版する際に番号を付すか否かを決定しているのは，\@sect の次の部分です：

```
\ifnum #2>\c@secnumdepth
  \let\@svsec\@empty
\else
  \refstepcounter{#1}%
  \protected@edef\@svsec{\@seccntformat{#1}\relax}%
\fi
```

\c@secnumdepth の値を見出しのレベルより小さく設定すると，\@svsec が \@empty に \let され，見出しレベルより大きく設定すると，\@svsec には \@seccntformat を展開したものが格納されます。この \@svsec は，次の箇所で使われています：

```
#6{%
  \@hangfrom{\hskip #3\relax\@svsec}%
    \interlinepenalty \@M #8\@@par}%
```

ここで “#6” は\@startsection の ⟨*style*⟩，“#3” は ⟨*indent*⟩ に当たり，“#8” は見出しコマンドの必須引数です。つまり，⟨*indent*⟩ だけ \hskip して，\@svsec を置いて，見出し文字列を配置し，これらの書式が ⟨*style*⟩ というわけです。

\@seccntformat のデフォルトの設定は，次のようになっています：

```
\def\@seccntformat#1{\csname the#1\endcsname\quad}
```

したがって，この\@seccntformat の定義を変更すると，\thesection などを再定義することなしに，見出し番号まわりの体裁のみを変更することができます。

6.2.7 付 録

from: *jclasses.dtx*

\appendix

article クラスの場合，\appendix コマンドは次のことを行ないます。

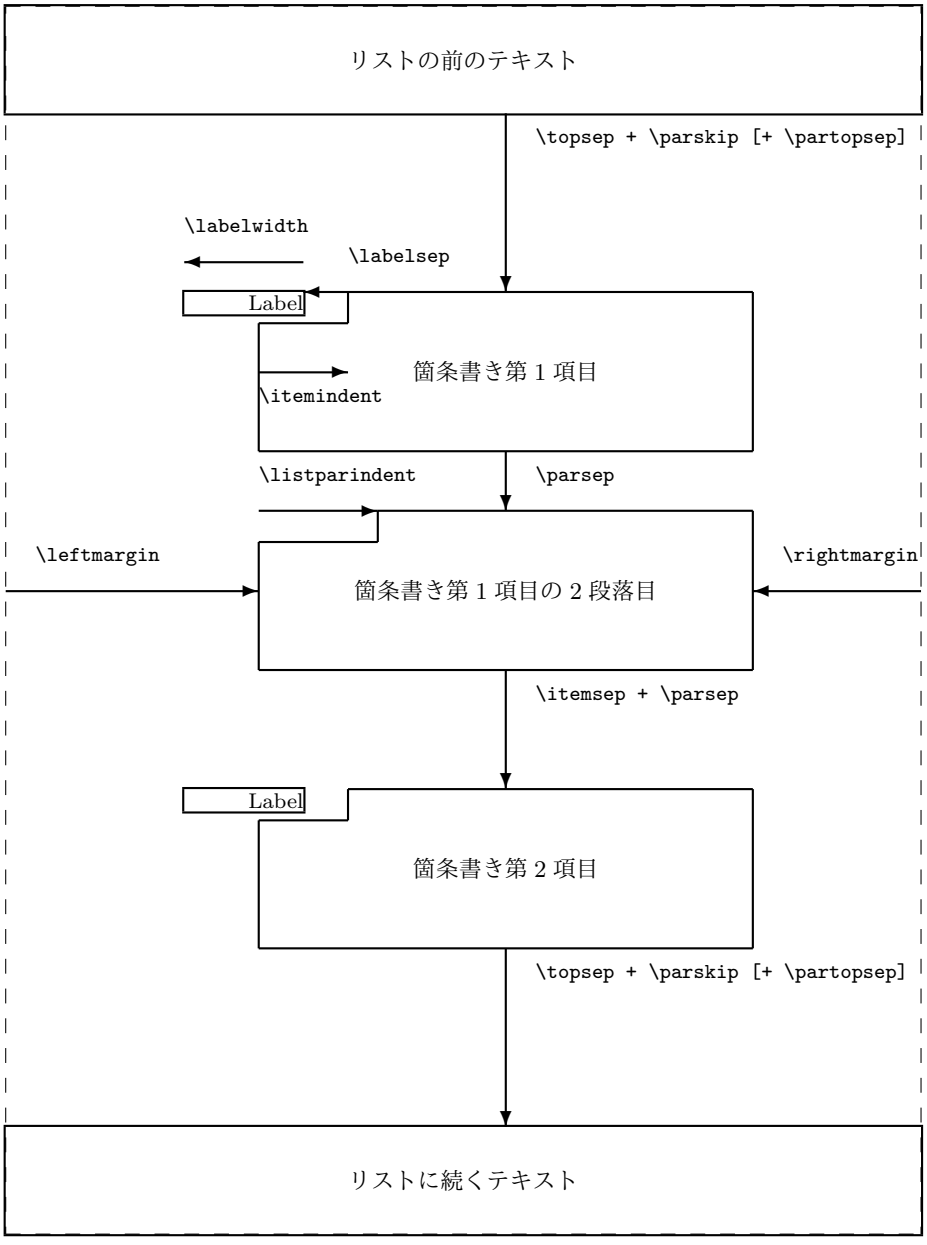
- section と subsection カウンタをリセットする。
- \thesection を英小文字で〔ママ〕出力するように再定義する[17]。

```
441 \newcommand{\appendix}{\par
442   \setcounter{section}{0}%
443   \setcounter{subsection}{0}%
444   \renewcommand{\thesection}{\@Alph\c@section}}
```

[17] 英「大」文字で出力ですよね…。

6.3 リスト環境

ここではリスト環境のパラメータの値が設定されています。まず最初に，おなじみのリスト環境のパラメータの図を入れておきます：



from: *jclasses.dtx*

リスト環境のデフォルトは次のように設定されます。

まず，`\rigtmargin` [*sic*]，`\listparindent`，`\itemindent` をゼロにします。そして，K 番目のレベルのリストは `\@listK` で示されるマクロが呼び出されます。ここで 'K' は小文字のローマ数字で示されます。たとえば，3 番目のレベルのリストとして `\@listiii` が呼び出されます。`\@listK` は `\leftmargin` を `\leftmarginK` に設定します。

\@listi ～ \@listvi の設定で使うための \leftmargin の設定です。まずは一番外側のリスト環境のための \leftmargini のデフォルトの値から：

\leftmargini

```
\if@twocolumn
  \setlength\leftmargini {2em}
\else
  \setlength\leftmargini {2.5em}
\fi
```

from: *jclasses.dtx*

次の 3 つの値は，\labelsep とデフォルトラベル（‘(m)’，‘vii.’，‘M.’）の幅の合計よりも大きくしてあります。

\leftmarginii
\leftmarginiii
\leftmarginiv

```
\setlength\leftmarginii {2.2em}
\setlength\leftmarginiii {1.87em}
\setlength\leftmarginiv {1.7em}
```

5 番目と 6 番目のレベルのリスト環境用の \leftmargin のデフォルトの値の設定です：

\leftmarginv
\leftmarginvi

```
\if@twocolumn
  \setlength\leftmarginv {.5em}
  \setlength\leftmarginvi{.5em}
\else
  \setlength\leftmarginv {1em}
  \setlength\leftmarginvi{1em}
\fi
```

続いて，\labelsep と \labelwidth の設定です：

\lebelsep
\lebelwidth

```
\setlength \labelsep {.5em}
\setlength \labelwidth{\leftmargini}
\addtolength\labelwidth{-\labelsep}
```

from: *jclasses.dtx*

\@beginparpenalty, \@endparpenalty
　これらのペナルティは，リストや段落環境の前後に挿入されます。
\@itempenalty
　このペナルティは，リスト項目の間に挿入されます。

\@beginparpenalty
\@endparpenalty
\@itempenalty

```
\@beginparpenalty -\@lowpenalty
\@endparpenalty -\@lowpenalty
\@itempenalty   -\@lowpenalty
```

ここからは，またちょっとだけ jsize10.clo です。\partopsep の初期値です（リスト環境の前後に空行があると，\parskip + \topsep に更に \partopsep が加えられます）：

\partosep

```
\setlength\partopsep{2\p@ \@plus 1\p@ \@minus 1\p@}
```

そして，\@listi ～ \@listvi の各種パラメータの設定です：

from: *jclasses.dtx*

\@listi は，\leftmargin，\parsep，\topsep，\itemsep などのトップレベルの定義をします。この定義は，フォントサイズコマンドによって変更されます（たとえ

ば，\small の中では “小さい” リストパラメータになります)。
このため，\normalsize がすべてのパラメータを戻せるように，\@listI は \@listi のコピーを保存するように定義されています。

\@listi

```
\def\@listi{\leftmargin\leftmargini
  \parsep 4\p@ \@plus2\p@ \@minus\p@
  \topsep 8\p@ \@plus2\p@ \@minus4\p@
  \itemsep4\p@ \@plus2\p@ \@minus\p@}
\let\@listI\@listi
```

```
\@listi
```

from: *jclasses.dtx*

下位レベルのリスト環境のパラメータの設定です。これらは保存用のバージョンを持たないことと，フォントサイズコマンドによって変更されないことに注意をしてください。言い換えれば，このクラスは，本文サイズが \normalsize で現れるリストの入れ子についてだけ考えています。

\@listii
\@listiii

```
\def\@listii{\leftmargin\leftmarginii
   \labelwidth\leftmarginii \advance\labelwidth-\labelsep
   \topsep 4\p@ \@plus2\p@ \@minus\p@
   \parsep 2\p@ \@plus\p@ \@minus\p@
   \itemsep\parsep}
```

```
\def\@listiii{\leftmargin\leftmarginiii
   \labelwidth\leftmarginiii \advance\labelwidth-\labelsep
   \topsep 2\p@ \@plus\p@\@minus\p@
   \parsep\z@
   \partopsep \p@ \@plus\z@ \@minus\p@
   \itemsep\topsep}
```

\@listiv ～ \@listvi では \leftmargin と \labelwidth が設定されるだけですね：

\@listiv
\@listv
\@listvi

```
\def\@listiv {\leftmargin\leftmarginiv
              \labelwidth\leftmarginiv
              \advance\labelwidth-\labelsep}
```

```
\def\@listv {\leftmargin\leftmarginv
             \labelwidth\leftmarginv
             \advance\labelwidth-\labelsep}
```

```
\def\@listvi {\leftmargin\leftmarginvi
              \labelwidth\leftmarginvi
              \advance\labelwidth-\labelsep}
```

6.3.1 enumerate 環境

jarticle.cls に戻ります。enumerate 環境の初期設定です。まずは，カウンタ \c@enumi ～ \c@enumiv の出力形式の設定から：

\theenumi
\theenumii
\theenumiii
\theenumiv

```
\renewcommand{\theenumi}{\@arabic\c@enumi}
\renewcommand{\theenumii}{\@alph\c@enumii}
\renewcommand{\theenumiii}{\@roman\c@enumiii}
\renewcommand{\theenumiv}{\@Alph\c@enumiv}
```

続いて \theenumi 〜 \theenumiv を使って，\list 環境の第 1 引数となるラベルの体裁を決めています：

\labelenumi
\labelenumii
\labelenumiii
\labelenumiv

```
\newcommand{\labelenumi}{\theenumi.}
\newcommand{\labelenumii}{(\theenumii)}
\newcommand{\labelenumiii}{\theenumiii.}
\newcommand{\labelenumiv}{\theenumiv.}
```

そして，\ref で参照される際に \theenumii 〜 \theenumiv に前置されるプレフィクスの設定です：

\p@enumii
\p@enumiii
\p@enumiv

```
\renewcommand{\p@enumii}{\theenumi}
\renewcommand{\p@enumiii}{\theenumi(\theenumii)}
\renewcommand{\p@enumiv}{\p@enumiii\theenumiii}
```

from: *jclasses.dtx*

トップレベルで使われたときに，最初と最後に半行分のスペースを開けるように，変更します。この環境は，ltlists.dtx で定義されています。

…というのは 483〜484 行のことなのでしょうけど，\iftdir で組み方向のテストをしているので，この「変更」はタテ組みのときの設定のような気がします。一応，*ltlists.dtx* からオリジナルの enumerate 環境の定義も見てみますと：

```
\def\enumerate{%
  \ifnum \@enumdepth >\thr@@\@toodeep\else
    \advance\@enumdepth\@ne
    \edef\@enumctr{enum\romannumeral\the\@enumdepth}%
      \expandafter
      \list
        \csname label\@enumctr\endcsname
        {\usecounter\@enumctr\def\makelabel##1{\hss\llap{##1}}}%
  \fi}
\let\endenumerate =\endlist
```

と，なっていて，ヨコ組みに関しては特には変更してないみたいですし。

enumerate

```
\renewenvironment{enumerate}
  {\ifnum \@enumdepth >\thr@@\@toodeep\else
   \advance\@enumdepth\@ne
   \edef\@enumctr{enum\romannumeral\the\@enumdepth}%
   \list{\csname label\@enumctr\endcsname}{%
      \iftdir
         \ifnum \@listdepth=\@ne \topsep.5\normalbaselineskip
           \else\topsep\z@\fi
         \parskip\z@ \itemsep\z@ \parsep\z@
         \labelwidth1zw \labelsep.3zw
         \ifnum \@enumdepth=\@ne \leftmargin1zw\relax
           \else\leftmargin\leftskip\fi
         \advance\leftmargin 1zw
      \fi
         \usecounter{\@enumctr}%
         \def\makelabel##1{\hss\llap{##1}}}%
  \fi}{\endlist}
```

まず，*ltlists.dtx* で宣言されているカウンタ \@enumdepth の深さで分岐して，\@toodeep は *lterror.dtx* で定義されている “Too deeply nested” というエ

ラーメッセージ，\@enumctr はカウンタ \@enumdepth の値に応じて “enumi”〜“enumvi” と展開されて，それが \list の第 1 引数で “\labelenumi”〜“\labelenumiv” として使われています。

\usecounter と \makelabel は LaTeX のマニュアル [6] にも載ってますけど，\list 環境のデフォルトの挙動を変更するときに使うものらしいです。

\usecounter{\@enumctr}は，やはりカウンタ \@enumdepth の値に応じて \usecounter{enumi}〜\usecounter{enumiv}と展開されて，そうすると，カウンタ enumi 〜 enumiv が \item ごとにインクリメントするようになるみたいです。

\def\makelabel#1{\hss\llap{#1}}は，TeX のプリミティブの \hss と plain TeX 以来のマクロ \llap（LaTeX 2ε では *ltboxes.dtx* に定義があります）を使って，ラベルを \labelwidth 幅の箱の右端に押し付けて左側にはみ出させるようにようにしています（LaTeX-ey に書くと \makebox[\labelwidth][r]{#1} みたいな感じ）。

6.3.2 itemize 環境

itemize 環境の設定です。まずはラベルから。\labelitemii だけタテ組みとヨコ組みでは記号を変えているのですね：

\labelitemi
\labelitemii
\labelitemiii
\labelitemiv

```
\newcommand{\labelitemi}{\textbullet}
\newcommand{\labelitemii}{%
  \iftdir
     {\textcircled{˜}}
  \else
     {\normalfont\bfseries\textendash}
  \fi
}
\newcommand{\labelitemiii}{\textasteriskcentered}
\newcommand{\labelitemiv}{\textperiodcentered}
```

from: *jclasses.dtx*

> トップレベルで使われたときに，最初と最後に半行分のスペースを開けるように，変更します。この環境は，ltlists.dtx で定義されています。

…というのはやはり，タテ組みのことなのではないかと。ここでも *ltlists.dtx* からオリジナルを見てみますと：

```
\def\itemize{%
  \ifnum \@itemdepth >\thr@@\@toodeep\else
    \advance\@itemdepth\@ne
    \edef\@itemitem{labelitem\romannumeral\the\@itemdepth}%
    \expandafter
    \list
      \csname\@itemitem\endcsname
      {\def\makelabel##1{\hss\llap{##1}}}%
  \fi}
\let\enditemize =\endlist
```

となっていて，ヨコ組みについては変更はなさそうです。

itemize

```
\renewenvironment{itemize}
  {\ifnum \@itemdepth >\thr@@\@toodeep\else
   \advance\@itemdepth\@ne
   \edef\@itemitem{labelitem\romannumeral\the\@itemdepth}%
   \expandafter
   \list{\csname \@itemitem\endcsname}{%
      \iftdir
         \ifnum \@listdepth=\@ne \topsep.5\normalbaselineskip
           \else\topsep\z@\fi
         \parskip\z@ \itemsep\z@ \parsep\z@
         \labelwidth1zw \labelsep.3zw
         \ifnum \@itemdepth =\@ne \leftmargin1zw\relax
           \else\leftmargin\leftskip\fi
         \advance\leftmargin 1zw
      \fi
         \def\makelabel##1{\hss\llap{##1}}}%
   \fi}{\endlist}
```

\usecounter を使わないところ以外は，enumerate 環境と似たような仕組みですね。

6.3.3 description 環境

from: *jclasses.dtx*

description 環境を定義します。縦組時には，インデントが 3 字分だけ深くなります。

description

```
\newenvironment{description}
  {\list{}{\labelwidth\z@ \itemindent-\leftmargin
   \iftdir
     \leftmargin\leftskip \advance\leftmargin3\Cwd
     \rightmargin\rightskip
     \labelsep=1zw \itemsep\z@
     \listparindent\z@ \topskip\z@ \parskip\z@ \partopsep\z@
   \fi
          \let\makelabel\descriptionlabel}}{\endlist}
\newcommand{\descriptionlabel}[1]{%
   \hspace\labelsep\normalfont\bfseries #1}
```

やはり念のため見ておきましょうか。*classes.dtx* からです：

```
\newenvironment{description}
             {\list{}{\labelwidth\z@ \itemindent-\leftmargin
                      \let\makelabel\descriptionlabel}}
             {\endlist}
\newcommand*\descriptionlabel[1]{\hspace\labelsep
                                 \normalfont\bfseries #1}
```

6.3.4 verse 環境

verse 環境の設定です。欧文用の article.cls の設定とまったく同じママのようですね：

verse

```
\newenvironment{verse}
  {\let\\\@centercr
   \list{}{\itemsep\z@ \itemindent -1.5em%
           \listparindent\itemindent
           \rightmargin\leftmargin \advance\leftmargin 1.5em}%
           \item\relax}{\endlist}
```

6.3.5 quotation 環境

quotation 環境の設定も，article.cls と一緒です： *quotation*

```
\newenvironment{quotation}
  {\list{}{\listparindent 1.5em%
           \itemindent\listparindent
           \rightmargin\leftmargin
           \parsep\z@ \@plus\p@}%
           \item\relax}{\endlist}
```

6.3.6 quote 環境

そしてやはり，quote 環境の場合も： *quote*

```
\newenvironment{quote}
  {\list{}{\rightmargin\leftmargin}%
           \item\relax}{\endlist}
```

6.4 フロート

ここでは，フロートに関してクラスファイルで設定しておくべきことについて説明がされていますが，*classes.dtx* から丸写ししておきます…：

from: *classes.dtx*

7.6 Floating objects

The file `latex.dtx` only defines a number of tools with which floating objects can be defined. This is done in the document class. It needs to define the following macros for each floating object of type `TYPE` (e.g., `TYPE` = figure).

`\fps@TYPE` The default placement specifier for floats of type `TYPE`.

`\ftype@TYPE` The type number for floats of type `TYPE`. Each `TYPE` has associated a unique positive `TYPE` number, which is a power of two. E.g., figures might have type number 1, tables type number 2, programs type number 4, etc.

`\ext@TYPE` The file extension indicating the file on which the contents list for float type `TYPE` is stored. For example, `\ext@figure` = 'lof'.

`\fnum@TYPE` A macro to generate the figure number for a caption. For example, `\fnum@TYPE` == 'Figure `\thefigure`'.

`\@makecaption` ⟨*num*⟩⟨*text*⟩ A macro to make a caption, with ⟨*num*⟩ the value produced by `\fnum@...` and ⟨*text*⟩ the text of the caption. It can assume it's in a `\parbox` of the appropriate width. This will be used for *all* floating objects.

The actual environment that implements a floating object such as a figure is defined using the macros `\@float` and `\end@float`, which are defined in `latex.dtx`.

An environment that implements a single column floating object is started with `\@float{TYPE}[`⟨*placement*⟩`]` of type `TYPE` with ⟨*placement*⟩ as the placement

specifier. The default value of ⟨*PLACEMENT*⟩ is defined by \fps@TYPE.

The environment is ended by \end@float. E.g., \figure == \@float{figure}, \endfigure == \end@float.

とりあえず，“TYPE” というフロートを使うには，\fps@TYPE（フロートの配置のデフォルト），\ftype@TYPE（フロート “TYPE” の識別番号．2 の累乗にする），\ext@TYPE（フロートの目次データが送られるファイルの拡張子），\fnum@TYPE（キャプションの番号部分）をそれぞれ定義して，それで，TYPE 環境は：

```
\newenvironment{TYPE}{\@float{TYPE}}{\end@float}
```

という風に定義することになっている，ということが書いてあるみたいですね。

6.4.1 figure 環境

\c@figure
\fps@figure
\ftype@figure
\ext@figure
\fnum@figure
figure
figure*

というわけで，まずは figure 環境の場合の諸定義です（星アリは 2 段組の段抜きの場合）：

```
\newcounter{figure}
\renewcommand{\thefigure}{\@arabic\c@figure}
\def\fps@figure{tbp}
\def\ftype@figure{1}
\def\ext@figure{lof}
\def\fnum@figure{\figurename~\thefigure}
\newenvironment{figure}
               {\@float{figure}}
               {\end@float}
\newenvironment{figure*}
               {\@dblfloat{figure}}
               {\end@dblfloat}
```

6.4.2 table 環境

\c@table
\fps@table
\ftype@table
\ext@table
\fnum@table
table
table*

続いて table 環境の場合：

```
\newcounter{table}
\renewcommand{\thetable}{\@arabic\c@table}
\def\fps@table{tbp}
\def\ftype@table{2}
\def\ext@table{lot}
\def\fnum@table{\tablename~\thetable}
\newenvironment{table}
               {\@float{table}}
               {\end@float}
\newenvironment{table*}
               {\@dblfloat{table}}
               {\end@dblfloat}
```

6.5 キャプション

\abovecaptionskip
\belowcaptionskip

そして，キャプションの定義です。まずはキャプションの上下のアキの宣言とその値の設定から：

```
\newlength\abovecaptionskip
\newlength\belowcaptionskip
\setlength\abovecaptionskip{10\p@}
\setlength\belowcaptionskip{0\p@}
```

それから \@makecaption そのものの定義です： *\@makecaption*

```
\long\def\@makecaption#1#2{%
  \vskip\abovecaptionskip
  \iftdir\sbox\@tempboxa{#1\hskip1zw#2}%
    \else\sbox\@tempboxa{#1: #2}%
  \fi
  \ifdim \wd\@tempboxa >\hsize
    \iftdir #1\hskip1zw#2\relax\par
      \else #1: #2\relax\par\fi
  \else
    \global \@minipagefalse
    \hbox to\hsize{\hfil\box\@tempboxa\hfil}%
  \fi
  \vskip\belowcaptionskip}
```

組み方向で分岐した後，キャプション番号（“#1”）とキャプション文字列（“#2”）とを \@tempboxa に入れて，その幅（\wd\@tempboxa）と行幅（\hsize〔=\textwidth〕）を比較して，\@tempboxa の幅が行幅よりも短い場合にはキャプションをセンタリングにしていますね[18]。

PracTEX Journal の記事 [10] では，キャプションをいじる場合には ccaption を使いましょうと書いてあります。

6.6 コマンドパラメータの設定

細々としたパラメータの諸設定です。

6.6.1 array と tabular 環境

\arraycolsep，\tabcolsep，\arrayrulewidth，\doublerulesep のデフォルト値です： *\arraycolsep* *\tabcolsep* *\arrayrulewidth* *\doublerulesep*

```
\setlength\arraycolsep{5\p@}
\setlength\tabcolsep{6\p@}
\setlength\arrayrulewidth{.4\p@}
\setlength\doublerulesep{2\p@}
```

6.6.2 tabbing 環境

\tabbingsep の設定です： *\tabbingsep*

```
\setlength\tabbingsep{\labelsep}
```

6.6.3 minipage 環境

minipage 環境内の本文と脚註の距離です： *\skip\@mpfootins*

```
\skip\@mpfootins = \skip\footins
```

(18) \@minipagefalse のあたりは初級には難しすぎます…。

6.6.4 framebox 環境

\fboxsep と \fboxrule のデフォルト値の設定です： *\fboxsep* *\fboxrule*

```
\setlength\fboxsep{3\p@}
\setlength\fboxrule{.4\p@}
```

6.6.5 equation と eqnarray 環境

\c@equation の出力形式の設定です。\c@equation は *ltmath.dtx* で宣言されています。 *\c@equation*

```
\renewcommand{\theequation}{\@arabic\c@equation}
```

7 フォントコマンド

ここは私の守備範囲外なので，*jclasses.dtx* を転記しておきます：

from: *jclasses.dtx*

> disablejfam オプションが指定されていない場合には，以下の設定がなされます。まず，数式内に日本語を直接，記述するために数式記号用文字に “JY1/mc/m/n” を登録します。数式バージョンが bold の場合は，“JY1/gt/m/n” を用います。これらは，\mathmc, \mathgt として登録されます。また，日本語数式ファミリとして \symmincho がこの段階で設定されます。mathrmmc オプションが指定されていた場合には，これに引き続き \mathrm と \mathbf を和欧文両対応にするための作業がなされます。この際，他のマクロとの衝突を避けるため \AtBeginDocument を用いて展開順序を遅らせる必要があります。
>
> disablejfam オプションが指定されていた場合には，\mathmc と \mathgt に対してエラーを出すだけのダミーの定義を与える設定のみが行われます。

```
\if@enablejfam
  \if@compatibility\else
    \DeclareSymbolFont{mincho}{JY1}{mc}{m}{n}
    \DeclareSymbolFontAlphabet{\mathmc}{mincho}
    \SetSymbolFont{mincho}{bold}{JY1}{gt}{m}{n}
    \jfam\symmincho
    \DeclareMathAlphabet{\mathgt}{JY1}{gt}{m}{n}
  \fi
  \if@mathrmmc
    \AtBeginDocument{%
    \reDeclareMathAlphabet{\mathrm}{\mathrm}{\mathmc}
    \reDeclareMathAlphabet{\mathbf}{\mathbf}{\mathgt}
  }%
  \fi
\else
  \DeclareRobustCommand{\mathmc}{%
    \@latex@error{Command \noexpand\mathmc invalid with\space
       ‘disablejfam’ class option.}\@eha
  }
  \DeclareRobustCommand{\mathgt}{%
    \@latex@error{Command \noexpand\mathgt invalid with\space
       ‘disablejfam’ class option.}\@eha
  }
\fi
```

from: *jclasses.dtx*

ここでは LaTeX 2.09 で一般的に使われていたコマンドを定義しています。これらのコマンドはテキストモードと数式モードのどちらでも動作します。これらは互換性のために提供をしますが，できるだけ `\text...` と `\math...` を使うようにしてください。

```
\DeclareOldFontCommand{\mc}{\normalfont\mcfamily}{\mathmc}
\DeclareOldFontCommand{\gt}{\normalfont\gtfamily}{\mathgt}
\DeclareOldFontCommand{\rm}{\normalfont\rmfamily}{\mathrm}
\DeclareOldFontCommand{\sf}{\normalfont\sffamily}{\mathsf}
\DeclareOldFontCommand{\tt}{\normalfont\ttfamily}{\mathtt}
\DeclareOldFontCommand{\bf}{\normalfont\bfseries}{\mathbf}
\DeclareOldFontCommand{\it}{\normalfont\itshape}{\mathit}
\DeclareOldFontCommand{\sl}{\normalfont\slshape}{\@nomath\sl}
\DeclareOldFontCommand{\sc}{\normalfont\scshape}{\@nomath\sc}
\DeclareRobustCommand*{\cal}{\@fontswitch\relax\mathcal}
\DeclareRobustCommand*{\mit}{\@fontswitch\relax\mathnormal}
```

8 相互参照

8.1 目 次

まずは目次についての説明を *jclasses.dtx* から：

from: *jclasses.dtx*

`\section` コマンドは，`.toc` ファイルに，次のような行を出力します。

`\contentsline{section}{`⟨*title*⟩`}{`⟨*page*⟩`}`

⟨*title*⟩ には項目が，⟨*page*⟩ にはページ番号が入ります。

`\section` に見出し番号が付く場合は，⟨*title*⟩ は，

`\numberline{`⟨*num*⟩`}{`⟨*heading*⟩`}`

となります。⟨*num*⟩ は `\thesection` コマンドで生成された見出し番号です。⟨*heading*⟩ は見出し文字列です。この他の見出しコマンドも同様です。

figure 環境での `\caption` コマンドは，`.lof` ファイルに，次のような行を出力します。

`\contentsline{figure}{\numberline{`⟨*num*⟩`}{`⟨*caption*⟩`}}{`⟨*page*⟩`}`

⟨*num*⟩ は，`\thefigure` コマンドで生成された図番号です。⟨*caption*⟩ は，キャプション文字列です。table 環境も同様です。

`\contentsline{`⟨*name*⟩`}`コマンドは，`\l@`⟨*name*⟩ に展開されます。したがって，目次の体裁を記述するには，`\l@chapter`，`\l@section` などを定義します。図目次のためには `\l@figure` です。これらの多くのコマンドは `\@dottedtocline` コマンドで定義されています。

目次項目の体裁は `\l@`⟨*name*⟩ というマクロが決めているとのことなので，本文目次での part の各行の体裁は `\l@part` が，section については `\l@section`，図目次の各行の体裁は `\l@figure` が，それぞれ担っていることになります。

仮に “`\SECTION`” という見出しマクロを考えてみます。例えば，第 2 ページ目に当たる部分でソースに “`\SECTION{はじめに}`” と書いたとすると，見出しコマンドの中で使われている `\addtcontentsline` によって “`.toc`” ファイルには，

```
\contentsline{SECTION}{\numberline{1}はじめに}{2}
```

という情報が送られます（\addcontentsline の定義は 655 行に，\numberline の定義は 639 行にあります）。もしも \c@secnumdepth の値を調整をして見出し番号を抑制している場合には，“.toc” の中味は

```
\contentsline{SECTION}{はじめに}{2}
```

となります。そして，これらはそれぞれ

```
\l@SECTION{\numberline{1}はじめに}{2}
\l@SECTION{はじめに}{2}
```

に展開されるというわけです。\contentsline は *ltsect.dtx* で

```
\def\contentsline#1{\csname l@#1\endcsname}
```

と定義されているからです。

さてここで，\l@SECTION の定義の仕方にはひとまず二つのやり方が考えられます。ひとつには，素直に引数を 2 つ取るマクロとして \l@SECTION を定義するというやり方と，もうひとつには，3 個以上の引数をとる別のマクロと \l@SECTION を置換するように定義するというものです[19]。\l@part や \l@section は前者の方法をとり，\l@subsection 以下は後者の方法をとっています。つまり，\l@subsection などの場合は，\l@SECTION の部分を，引数を 5 つとる汎用的な目次項目作成マクロの \@dottedtocline（を最初の 3 つの引数付き）で，置換しています（\l@part 等々は「8.1.1 本文目次」のところで定義されており，また，\@dottedtocline の説明についても「8.1.1 本文目次」のほうに引越しさせました）。

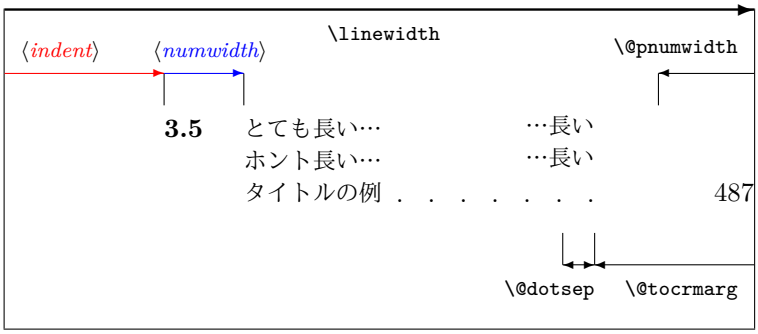
それでは，まず最初の設定は，\c@tocdepth の深さのデフォルトからです： *\c@topdepth*

```
632 \setcounter{tocdepth}{3}
```

続いて \@pnumwidth，\@tocrmarg，\@dotsep の初期値です： *\@pnumwidth* *\@tocrmarg* *\@dotsep*

```
633 \newcommand{\@pnumwidth}{1.55em}
634 \newcommand{\@tocrmarg}{2.55em}
635 \newcommand{\@dotsep}{4.5}
```

パラメータの図を入れておきましょう（⟨*indent*⟩ と ⟨*numwidth*⟩ については，後述の \@dottedtocline のところに説明があります）：



すぐ後で \@dottedtocline を再定義する際に使う \toclineskip をここで導入 *\toclineskip*

[19] \@startsection の場合によく似ていますね。\@startsection の下請けの \@sect は引数が 8 個必要ですが，\@startsection がそのうちの 6 個を引き連れて，“\section” 等々と置き換わるように定義されていました。→ Excursus

しています。これはオリジナルの LaTeX にはないパラメータで，\@dottedtocline による目次項目同士の間に挿入される長さです（尤も，637 行を見てお分かりのようにヨコ組みでは 0pt の設定なので格別違いはないのですが）。

```
\newdimen\toclineskip
\setlength\toclineskip{\z@}
```

from: *jclasses.dtx*

> \numberline マクロの定義を示します。オリジナルの定義では，ボックスの幅を \@tempdima にしていますが，この変数はいろいろな箇所で使われますので，期待した値が入らない場合があります。
> たとえば，pLaTeX 2ε での \selectfont は，和欧文のベースラインを調整するために \@tempdima 変数を用いています。そのため，\l@... マクロの中でフォントを切替えると，\numberline マクロのボックスの幅が，ベースラインを調整するときに計算した値になってしまいます。
> フォント選択コマンドの後，あるいは \numberline マクロの中でフォントを切替えてもよいのですが，一時変数を意識したくないので，見出し番号の入るボックスを \@lnumwidth 変数を用いて組み立てるように \numberline マクロを再定義します。

というわけでここで \@lnumwidth を導入して，それを使って \numberline を再定義しています：

\@lnumwidth
\numberline

```
\newdimen\@lnumwidth
\def\numberline#1{\hbox to\@lnumwidth{#1\hfil}}
```

ちなみに *ltsect.dtx* のオリジナルはこうです：

```
\def\numberline#1{\hb@xt@\@tempdima{#1\hfil}}
```

次に，\@dottedtocline と \addcontentsline を再定義しています。まずは \@dottedtocline から（“#1” は ⟨*level*⟩，“#2” は ⟨*indent*⟩，“#3” は ⟨*numwidth*⟩，“#4” は ⟨*title*⟩，“#5” は ⟨*page*⟩ です．説明はすぐ次の「8.1.1 本文目次」のところに）：

\@dottedtocline

```
\def\@dottedtocline#1#2#3#4#5{%
  \ifnum #1>\c@tocdepth \else
    \vskip\toclineskip \@plus.2\p@
    {\leftskip #2\relax \rightskip \@tocrmarg \parfillskip -\rightskip
     \parindent #2\relax\@afterindenttrue
     \interlinepenalty\@M
     \leavevmode
     \@lnumwidth #3\relax
     \advance\leftskip \@lnumwidth \hbox{}\hskip -\leftskip
     {#4}\nobreak
     \leaders\hbox{$\m@th \mkern \@dotsep mu.\mkern \@dotsep mu$}%
     \hfill\nobreak
     \hb@xt@\@pnumwidth{\hss\normalfont \normalcolor #5}%
     \par}%
  \fi}
```

オリジナルを *ltsect.dtx* から転記しますと，“\vskip\toclineskip \@plus.2\p@” の部分が元は “\vskip \z@ \@plus.2\p@” となっており，また *jclasses.dtx* の説明にありますように，\@lnumwidth のところでは元々 \@tempdima が使われていたことが分かります（他にも少し変更されてますね）：

```
\def\@dottedtocline#1#2#3#4#5{%
  \ifnum #1>\c@tocdepth \else
    \vskip \z@ \@plus.2\p@
    {\leftskip #2\relax \rightskip \@tocrmarg \parfillskip -\rightskip
     \parindent #2\relax\@afterindenttrue
     \interlinepenalty\@M
     \leavevmode
     \@tempdima #3\relax
     \advance\leftskip \@tempdima \null\nobreak\hskip -\leftskip
     {#4}\nobreak
     \leaders\hbox{$\m@th
        \mkern \@dotsep mu\hbox{.}\mkern \@dotsep
        mu$}\hfill
     \nobreak
     \hb@xt@\@pnumwidth{\hfil\normalfont \normalcolor #5}%
     \par}%
  \fi}
```

そして，\addcontentsline の再定義です： *\addcontentsline*

```
\def\addcontentsline#1#2#3{%
  \protected@write\@auxout
    {\let\label\@gobble \let\index\@gobble \let\glossary\@gobble
\@temptokena{\thepage}}%
    {\string\@writefile{#1}%
       {\protect\contentsline{#2}{#3}{\the\@temptokena}}}%
}
```

ltsect.dtx のオリジナルでは，\addcontentsline はその定義の中で \addtocontents に引数を渡していますが，pLaTeX では，\addtocontents に相当する部分を \addcontentsline に入れて，定義し直してますね（ファイルの入出力関係は全然分かりません…）：

```
\def\addcontentsline#1#2#3{%
  \addtocontents{#1}{\protect\contentsline{#2}{#3}{\thepage}}}
\long\def\addtocontents#1#2{%
  \protected@write\@auxout
     {\let\label\@gobble \let\index\@gobble \let\glossary\@gobble}%
     {\string\@writefile{#1}{#2}}}
```

8.1.1 本文目次

まず，\tableofcontents の定義です。ファイルの入出力の仕組みについては分からないのですが，図目次や表目次の定義とも比較してみますと，どうやら “\@starttoc{⟨*ext*⟩}” というところが大事そうです（あと，本当は，こんな風に \section の引数の中に \@mkboth を入れてしまうとマズいことになる場合がある，というのをどこかで聞いた気がします）： *\tableofcontents*

```
\newcommand{\tableofcontents}{%
  \section*{\contentsname
    \@mkboth{\contentsname}{\contentsname}%
  }\@starttoc{toc}%
}
```

そして，part の目次項目の体裁を決めている \l@part の定義です。part の目次のレベルは jarticle.cls でも，jbook.cls や jreport.cls と同じく，−1 になっていますね（ここで “#1” は ⟨*title*⟩で，“#2” は ⟨*page*⟩，あと \@lnumwidth は ⟨*numwidth*⟩ です）： *\l@part*

```
\newcommand*{\l@part}[2]{%
  \ifnum \c@tocdepth >-2\relax
    \addpenalty{\@secpenalty}%
    \addvspace{2.25em \@plus\p@}%
    \begingroup
    \parindent\z@\rightskip\@pnumwidth
    \parfillskip-\@pnumwidth
    {\leavevmode\large\bfseries
     \setlength\@lnumwidth{4zw}%
     #1\hfil\nobreak
     \hbox to\@pnumwidth{\hss#2}}\par
    \nobreak
    \if@compatibility
    \global\@nobreaktrue
    \everypar{\global\@nobreakfalse\everypar{}}%
    \fi
     \endgroup
  \fi}
```

続いて，section の目次項目の体裁である \l@section の定義です。\c@tocdepth が 1 以上に設定されると，section 項目が目次に掲載されます。

\l@section

```
\newcommand*{\l@section}[2]{%
  \ifnum \c@tocdepth >\z@
    \addpenalty{\@secpenalty}%
    \addvspace{1.0em \@plus\p@}%
    \begingroup
      \parindent\z@ \rightskip\@pnumwidth \parfillskip-\rightskip
      \leavevmode\bfseries
      \setlength\@lnumwidth{1.5em}%
      \advance\leftskip\@lnumwidth \hskip-\leftskip
      #1\nobreak\hfil\nobreak\hbox to\@pnumwidth{\hss#2}\par
    \endgroup
  \fi}
```

\l@subsection 以下の目次項目では，\@dottedtocline に下請けに出しています。こちらに場所を移動させた *jclasses.dtx* による \@dottedtocline の説明です：

from: *jclasses.dtx*

\contentsline⟨*name*⟩コマンドは，\l@⟨*name*⟩に展開されます。したがって，目次の体裁を記述するには，\l@chapter，\l@section などを定義します。図目次のためには \l@figure です。これらの多くのコマンドは \@dottedtocline コマンドで定義されています。このコマンドは次のような書式となっています。

\@dottedtocline{⟨*level*⟩}{⟨*indent*⟩}{⟨*numwidth*⟩}{⟨*title*⟩}{⟨*page*⟩}

⟨*level*⟩	“⟨*level*⟩ <= tocdepth” のときにだけ，生成されます。\chapter はレベル 0，\section はレベル 1，…です。
⟨*indent*⟩	一番外側からの左マージンです。
⟨*numwidth*⟩	見出し番号（\numberline コマンドの⟨*num*⟩）が入るボックスの幅です。

\l@subsection 以下の定義をまずは見てみます：

\l@subsection
\l@subsubsection
\l@paragraph
\l@subparagraph

```
\newcommand*{\l@subsection} {\@dottedtocline{2}{1.5em}{2.3em}}
\newcommand*{\l@subsubsection}{\@dottedtocline{3}{3.8em}{3.2em}}
\newcommand*{\l@paragraph} {\@dottedtocline{4}{7.0em}{4.1em}}
\newcommand*{\l@subparagraph} {\@dottedtocline{5}{10em}{5em}}
```

\@dottedtocline は引数を 5 つ必要としますが，\l@subsection 以下の \l@... は，「最初の引数 3 つを予め引き連れた \@dottedtocline」と，置換されるように定義されていますね。つまり，例えば，\subsection の場合を考えてみますと：

\contentsline{subsection}{⟨title⟩}{⟨page⟩}
↓
\l@subsection{⟨title⟩}{⟨page⟩}
↓
\@dottedtocline{2}{1.5em}{2.3em}{⟨title⟩}{⟨page⟩}

という風に，順に展開されていくことになります。

なお，\@dottedtocline の第 1 引数が，⟨lebel⟩となっていますが，\tocdepth の値をこのレベルと同じかまたはより高く設定すると，当該目次項目が目次に掲載されることになります（641 行で，⟨lebel⟩ と \tocdepth とを比較しています．part と section の目次のレベルはそれぞれ，668 行と 686 行で決まっています）。

目次の「レベル」は（part を除いて），見出しの「レベル」と同じに設定されていますが，これは恐らく，混乱を避けるためでしょう。いずれにしても，\secnumdepth は見出しに番号を付するか否かを制御し，他方 \tocdepth は当該レベルの項目を目次に掲載するか否かを決めています。

8.1.2 図目次と表目次

\listoffigures と \listoftables の定義は，\tableofcontents の定義と同じですね：

\listoffigures
\listoftables

```
\newcommand{\listoffigures}{%
    \section*{\listfigurename
  \@mkboth{\listfigurename}{\listfigurename}}%
  \@starttoc{lof}%
}
\newcommand*{\l@figure}{\@dottedtocline{1}{1.5em}{2.3em}}
```

```
\newcommand{\listoftables}{%
    \section*{\listtablename
  \@mkboth{\listtablename}{\listtablename}}%
  \@starttoc{lot}%
}
\let\l@table\l@figure
```

目次についても以上長々と眺めてきましたけど，PracTEXJournal の記事 [10] では，目次の設定には tocloft を使うことが薦められています。

8.2 参考文献

ここでは，thebibliography 環境を定義しています。

まず，ずーっと最初のほうの，「2.12 **参考文献のオプション**」のところに出てきた openbib オプション指定時に，\@openbib@code の再定義（＝リスト環境のパラメータ設定の追加）の際に使われる\bibindent をここで導入して，その初期値を決めています：

← 2.12 参考文献のオプション

\bibindent

```
\newdimen\bibindent
\setlength\bibindent{1.5em}
```

“.bib” ファイル中で “@book” とかの中のフィールド同士はただコンマで区切って入力しますが，BibTeX で処理した “.bbl” ファイルの中ではそこが \newblock で区切られていて，ここはその \newblock の初期設定です（間に小さなグルーを入れてフィールド項目を並べて行くようにしています．openbib 指定時には \newblock は \par に再定義されているので〔138 行〕，フィールド項目ごとに改行します〔正確には改段落ですが〕.）：

\newblock

```
715 \newcommand{\newblock}{\hskip .11em\@plus.33em\@minus.07em}
```

そして，thebibliography 環境です：

thebibliography

```
716 \newenvironment{thebibliography}[1]
717 {\section*{\refname\@mkboth{\refname}{\refname}}%
718   \list{\@biblabel{\@arabic\c@enumiv}}%
719        {\settowidth\labelwidth{\@biblabel{#1}}%
720         \leftmargin\labelwidth
721         \advance\leftmargin\labelsep
722         \@openbib@code
723         \usecounter{enumiv}%
724         \let\p@enumiv\@empty
725         \renewcommand\theenumiv{\@arabic\c@enumiv}}%
726   \sloppy
727   \clubpenalty4000
728   \@clubpenalty\clubpenalty
729   \widowpenalty4000%
730   \sfcode‘\.\@m}
731  {\def\@noitemerr
732    {\@latex@warning{Empty ‘thebibliography’ environment}}%
733   \endlist}
```

文献番号を振るカウンタとしては enumerate 環境の 4 番目のレベルのカウンタを流用していて，リスト環境の第 1 引数（ラベル）は「マクロ \@biblabel にカウンタ \c@enumiv の出力形式をアラビア数字にしたものを引数として与えた展開結果」で，\labelwidth は「\@biblabel の引数として，ユーザーが thebibliography 環境の引数として与えた文字列を入れたものを展開した，文字列の幅」，720〜722 行はリスト環境のパラメータの設定で（ここに \@openbib@code とか何かマクロを予め入れておけば，後からそれを使って設定を追加できるわけです），723 行は \item ごとにカウンタがインクリメントするにようにする設定でした。

あとは，726〜730 行では，overfull になりにくくしたり，文献リストでは略語がいろいろ使われるのでピリオドのあとのスペースの調整をしたり，改ページが不恰好にならないように設定がされたりしているらしいです。

\@openbib@code が未定義エラーとならないように，ここではとりあえず \@empty に \let されています：

\@openbib@code

```
734 \let\@openbib@code\@empty
```

このあと *jclasses.dtx* では，\@biblabel と \@cite のデフォルトについて，なぜか英語のままの記述が残っています（それぞれ，文献リスト内での文献番号の体裁と，本文中での文献番号の体裁の設定です）：

\@biblabel
\@cite

from: *jclasses.dtx*

\@biblabel

The label for a \bibitem[...] command is produced by this macro. The

default from latex.dtx is used.

```
% \renewcommand*{\@biblabel}[1]{[#1]\hfill}
```

\@cite

The output of the \cite command is produced by this macro. The default from latex.dtx is used.

```
% \renewcommand*{\@cite}[1]{[#1]}
```

\@biblabel と \@cite の定義は，*ltbibl.dtx* にあります。目次とか文献リストとかは，ファイルの入出力が絡むので，私の手には負えません…。

8.3 索 引

theindex 環境の定義です。LaTeX のマニュアル [6] にも書いてありますが，*MakeIndex* を使わないで手作業で簡単な索引を作る際に，使うものなのでしょう。 *theindex*

```
\newenvironment{theindex}
  {\if@twocolumn\@restonecolfalse\else\@restonecoltrue\fi
   \columnseprule\z@ \columnsep 35\p@
   \twocolumn[\section*{\indexname}]%
   \@mkboth{\indexname}{\indexname}%
   \thispagestyle{jpl@in}\parindent\z@
   \parskip\z@ \@plus .3\p@\relax
   \let\item\@idxitem}
  {\if@restonecol\onecolumn\else\clearpage\fi}
```

\begin{theindex} のところで，2 段組かどうかで \if@restonecol をセットして，\end{theindex} のところで，1 段組に戻しています。jarticle.cls の場合，\thispagestyle{jpl@in} が出てくるのは結局ここだけですね。 *\item* *\subitem* *\subsubitem* *\indexspace*

```
\newcommand{\@idxitem}{\par\hangindent 40\p@}
\newcommand{\subitem}{\@idxitem \hspace*{20\p@}}
\newcommand{\subsubitem}{\@idxitem \hspace*{30\p@}}
\newcommand{\indexspace}{\par \vskip 10\p@ \@plus5\p@ \@minus3\p@\relax}
```

theindex 環境での \item では，TeX のプリミティブである \hangindent を使ってぶら下げインデントにしています。あとは，\indexspace の初期値の設定ですね。

8.4 脚 注

脚註関係の初期設定 2 点です。まず，\footnoterule の再定義から： *\footnoterule*

```
\renewcommand{\footnoterule}{%
  \kern-3\p@
  \hrule width .4\columnwidth
  \kern 2.6\p@}
```

オリジナルは *ltfloat.dtx* で次のようになっていました：

from: *ltfloat.dtx*

\footnoterule

LaTeX keeps PLAIN TeX's \footnoterule as the default.

```
\def\footnoterule{\kern-3\p@
\hrule \@width 2in \kern 2.6\p@} % the \hrule is .4pt high
```

ほんの数行の定義ですけど，ここではTEXの垂直モードと水平モードが関係するので，実は初級向きではない気もしますが，TEXのプリミティブである \kern⟨*dimen*⟩ は，垂直モードのときには垂直方向に ⟨*dimen*⟩ だけ移動して，水平モードのときには水平方向に ⟨*dimen*⟩ だけ移動します。

また，TEXのプリミティブの \hrule は，垂直モードで，指定の高さ・幅・深さの水平線を引きます（値の指定は“\hrule height⟨*dimen*⟩ width⟨*dimen*⟩ depth⟨*dimen*⟩”で行い，デフォルトは，高さは 0.4pt，幅はそのときのボックスの幅，深さは 0pt です）。

というわけで，ここでは，まず上に 3pt 戻って，「太さ〔高さ〕が 0.4pt で長さ〔幅〕が 2inch なり 0.4\columnwidth なりの水平線」を引いて，それから下に 2.6pt 降りています。水平線の太さの分を打ち消すようにシフトしているのですね。

次の \@makefntext は，\footnote の下請けの \@footnotetext の中で以下のように使われていて，脚註文字列を組版する部分に当たります（定義は *ltfloat.dtx* より．ここで“#1”は \footnote の引数である脚註文字列）：

\@makefntext

```
\long\def\@footnotetext#1{\insert\footins{%
...
    \color@begingroup
      \@makefntext{%
        \rule\z@\footnotesep\ignorespaces#1\@finalstrut\strutbox}%
    \color@endgroup}}%
```

この \@makefntext の，jarticle.cls での定義です：

```
\newcommand\@makefntext[1]{\parindent 1em
  \noindent\hbox to 1.8em{\hss\@makefnmark}#1}
```

つまり，\@makefntext の定義中の“#1”の部分に，上の“\rule\z@\footnotesep\ignorespaces #1\@finalstrut\strutbox”が入るわけです。

脚註部分での最初の段落の字下げはナシで 2 段落目からは \parindent を 1em として，脚註の参照符（\@makefnmark）は幅 1.8em の箱に入れて右端に押し付けて，それから「幅が 0pt で高さが \footnotesep の支柱」を立ててた後，脚註文字列を置いていますね（末尾で \strutbox の深さ分の支柱を立ててる〔\@finalstrut\strutbox〕意味は，分かりません…）。

9 今日の日付

\today の表記を日本的にするために，再定義しています：

\today

```
\newif\if 西暦 \西暦false
\def\西暦{\西暦true}
\def\和暦{\西暦false}

\newcount\heisei \heisei\year \advance\heisei-1988\relax
```

```
\def\today{{%
  \iftdir
    \if 西暦
      \kansuji\number\year 年
      \kansuji\number\month 月
      \kansuji\number\day 日
    \else
      平成\ifnum\heisei=1 元年\else\kansuji\number\heisei 年\fi
      \kansuji\number\month 月
      \kansuji\number\day 日
    \fi
  \else
    \if 西暦
      \number\year~年
      \number\month~月
      \number\day~日
    \else
      平成\ifnum\heisei=1 元年\else\number\heisei~年\fi
      \number\month~月
      \number\day~日
    \fi
  \fi}}
```

ちなみにオリジナルの article.cls での定義はこんなに簡単です：

```
\def\today{\ifcase\month\or
  January\or February\or March\or April\or May\or June\or
  July\or August\or September\or October\or November\or December\fi
  \space\number\day, \number\year}
```

10 初期設定

文字通り，最後に各種の初期設定をしています。まずは，いろいろな文字列のデフォルトです：

```
\newcommand{\prepartname}{第}
\newcommand{\postpartname}{部}
\newcommand{\contentsname}{目 次}
\newcommand{\listfigurename}{図 目 次}
\newcommand{\listtablename}{表 目 次}
\newcommand{\refname}{参考文献}
\newcommand{\indexname}{索 引}
\newcommand{\figurename}{図}
\newcommand{\tablename}{表}
\newcommand{\appendixname}{付 録}
\newcommand{\abstractname}{概 要}
```

続いて，ページスタイルを “plain” にして，ノンブルはアラビア数字，ページ下端の不揃いを許します：

```
\pagestyle{plain}
\pagenumbering{arabic}
\raggedbottom
```

2 段組オプションがオンのときは 2 段組にして，オーバーフルが起こりにくいように \sloppy にし，2 段組オプションがオフのときは 1 段組とします：

```
\if@twocolumn
  \twocolumn
  \sloppy
\else
  \onecolumn
\fi
```

両面オプションがオンのときは\if@mparswitch を \@mparswitchtrue にし，両面オプションがオフのときは\@mparswitchfalse にします。\if@mparswitch は *ltoutput.dtx* で宣言されていて，オンのときは傍註が奇数ページと偶数ページとで左右に振り分けられます。\if@mparswitch がオフなら，ページの偶奇によらず，傍註は同じ側に出力されます（いずれの場合も普通は小口側です）。

```
\if@twoside
  \@mparswitchtrue
\else
  \@mparswitchfalse
\fi
```

ふ〜っ。当初予想したよりもずいぶんと長い道のりでした…。 

目 次