The cjwplain Package*
(PLAIN TEX under LATEX 2ε)

Colin J. Wynne†

1998/08/31

Contents

Introduction

I first started using TEX some two-and-a-half years ago, having been introduced to it by several TEXnophiles in my college math department. I was aware from the

*This file has version 1.2.

†E-Mail at: cwynne@brutus.mts.jhu.edu, cwynne@jhu.edu .

start that there was a somehow 'bastardised' version of this very good program which went by the name 'LaTeX'—invariably referred to by my TeX mentors as 'LameTeX'. Most of you have probably heard this epithet before.

Well, I count it as a good thing that I was discouraged from using LaTeX at first, as I ended up writing quite a lot of TeX code for myself before I ever got around to actually reading *The TeXbook* in its entirety, which I did only a few months ago. My first real project for TeX was writing a large macro set for my undergraduate thesis—table of contents, marks for the running heads, chapter and section delineation and so forth. This growing library was expanded as I decided I wanted a good set of macros for writing outlines and by requirements for various papers, such as using endnotes in lieu of footnotes. This rather haphazard collection of mine underwent a major change when I found the macro package for NFSS (version 1) under Plain TeX, of which I promptly took advantage.

This year I finally got my own computer, mainly to run TeX. Given my newly purchased copy of *The TeXbook* and some free time, I began to try to organise that cluster of code. Having learned something in the meanwhile about generic markup, and why it is preferable, I started rewriting for more generalisation. Also in the meanwhile, LaTeX 2_ε had come along, greatly enhancing LaTeX's own use of generic markup. It also standardised the NFSS, which I had so come to appreciate. Basically, between the various chunks of LaTeX 2_ε which I had already hacked to work under my custom format and the movement towards increasingly generic code by both myself and the LaTeX folks, that a convergence was taking place, so I finally decided to give LaTeX 2_ε a serious looking-at.

I started by printing out the documented source code. I liked a lot of what I saw—but there were two problems. Some of my favourite bits of Plain TeX got left by the wayside. For doing a lot of mathematics, I still find `\eqalign` to be the easiest way of aligning a bunch of related equations. First, it involves less typing than a `\begin`...`\end` pair, and I don't often need equation numbers—something not easily done away with under vanilla LaTeX. The bigger concern was that I had a bunch of source files that were written under Plain TeX (or, rather, under *my* Plain TeX), and I didn't want to have to make the dozens of minor modifications necessary to get them to work under LaTeX.

So, I decided to learn how LaTeX does things and to do so by writing a package that would, at its simplest, let me add a `\documentclass` and a `\begin{document}` line to one of my existing Plain TeX sourc files and get it to compile under LaTeX.

Thus, I have written my first LaTeX package. I consider the main feature to be the ability to very easily add NFSS commands to a document written under Plain TeX. Secondly, maybe it will help convince some other Plain TeX die-hards to give LaTeX a try, inasmuch as all of their standard commands will be supported. Finally, it should let those who use LaTeX exclusively to easily deal with Plain TeX files if the need arises.

Feel free to let me know if you find this package useful or, of course, if you find any bugs or wish to suggest improvements.

This Package

This package is built over the file `ltplain.dtx`, or, more correctly, over those parts of `ltplain.dtx` which were changes to or omissions of the original PLAIN TEX source. Some parts, specifically font changes, have not been reproduced in their entirety, due basically to the fact that such would be a pointless exercise. See the comments in Section 2.5 for the explanation.

Finally, this document prints with all source code because I feel the source itself, and the modifications to it, are the best documentation.

1 Package Options

According to the documented LATEX 2_ε source file `ltplain.dtx`,

> LATEX includes almost all of the functionality of Knuth's original 'Basic Macros' That is, the plain TEX format described in Appendix B of the TEXBook.

It seems to me that removing the qualifying 'almost' would be no bad thing.

The idea behind the available options is that a given user may need only certain aspects of PLAIN TEX added back in for a document. Furthermore, the additional code can sometimes be specified in different ways—*i.e.*, either strictly according to the definitions of PLAIN TEX or in a manner syntactically identical to PLAIN TEX but functionally grounded in LATEX. The overall goal, though, is completeness; I have therefore included everything, in one form or another, even when I can't think of a reason why some things would be necessary.

There are nineteen regular and two special options available for the `cjwplain` package. (All are entered in standard LATEX form, as optional arguments to the `\usepackage` command. I only call two of them 'special' so as to draw attention to them.)

1.1 Regular Options

`Options` The regular options are:

outerallocs	diagnostics	plainskips
outerallocsoff	diagnosticsoff	plainskipsoff
strictline	tabbing	strictitem
strictlineoff	tabbingoff	strictitemoff
eqalign	magnification	plainoutput
eqalignoff	magnificationoff	plainoutputoff
strictfootnotes	altfootnotes	footnotesoff

Note that most of these options come in ⟨*option*⟩/⟨*option*⟩`off` pairs. These are particularly useful in conjunction with the special options (1.2) or to toggle the default options. By default, the options `diagnostics`, `tabbing` and `eqalign` are active (just what I tend to use...).

The actual options will be explained in section 2. Keep in mind, though, that some options affect others—for example selecting one of `strictfootnotes` and `altfootnotes` will automatically turn the other off; you can, however, disable both forms either with `footnotesoff` or by giving the two separate `...off` commands separately. Also, `plainoutput` requires `strictfootnotes` and `magnification`, but `plainoutputoff` itself does not disable the PLAIN TEX footnote macros or magnification.

1.2 Special Options

`none` `all` Two options, called `none` and `all`, are available to allow maximum flexibility. These function because `cjwplain` calls the starred command `\ProcessOptions*` and therefore processes options in the order specified to `\usepackage`, and not the package's internal declaration order. Thus, to make only PLAIN TEX's tabbing commands available, one would use the call

```
\usepackage[none,tabbing]{cjwplain}
```

and to use everything while leaving LATEX's `\item` command alone one would enter the command

```
\usepackage[all,strictitemoff]{cjwplain}
```

in the preamble.

2 The Code

2.1 Declarations

The options are implemented as `\if` statements, as that seemed to me to be the easiest way of including or excluding relatively large sections of code. First we allocate the `\if`s.

```
⟨*package⟩
\newif\if@outerallocs      \@outerallocsfalse
\newif\if@diagnostics      \@diagnosticstrue
\newif\if@plainskips       \@plainskipsfalse
\newif\if@strictline       \@strictlinetrue
\newif\if@tabbing          \@tabbingtrue
\newif\if@strictitem       \@strictitemfalse
\newif\if@eqalign          \@eqaligntrue
\newif\if@strictfootnotes  \@strictfootnotesfalse
\newif\if@altfootnotes     \@altfootnotesfalse
\newif\if@plainoutput      \@plainoutputfalse
\newif\if@magnification    \@magnificationfalse
```

Now we declare how the options affect these `\if` tests.

```
\DeclareOption{outerallocs}{\@outerallocstrue}
\DeclareOption{outerallocsoff}{\@outerallocsfalse}

\DeclareOption{diagnostics}{\@diagnosticstrue}
\DeclareOption{diagnosticsoff}{\@diagnosticsfalse}
```

```
18
19 \DeclareOption{plainskips}{\@plainskipstrue}
20 \DeclareOption{planiskipsoff}{\@plainskipsfalse}
21
22 \DeclareOption{strictline}{\@strictlinetrue}
23 \DeclareOption{strictlineoff}{\@strictlinefalse}
24
25 \DeclareOption{tabbing}{\@tabbingtrue}
26 \DeclareOption{tabbingoff}{\@tabbingfalse}
27
28 \DeclareOption{strictitem}{\@strictitemtrue}
29 \DeclareOption{strictitemoff}{\@strictitemfalse}
30
31 \DeclareOption{eqalign}{\@eqaligntrue}
32 \DeclareOption{eqalignoff}{\@eqalignfalse}
```

We will have two possible ways of providing a `\footnote` command. As these are mutually exclusive, we make sure that they cannot both be true.

```
33 \DeclareOption{strictfootnotes}{%
34   \@strictfootnotestrue \@altfootnotesfalse}
35 \DeclareOption{altfootnotes}{%
36   \@altfootnotestrue \@strictfootnotesfalse}
37 \DeclareOption{footnotesoff}{%
38   \@altfootnotesfalse \@strictfootnotesfalse}
39
40 \DeclareOption{magnification}{\@magnificationtrue}
41 \DeclareOption{magnificationoff}{\@magnificationfalse}
```

To use PLAIN TEX's entire output routine will require that magnification code as well as PLAIN TEX style footnotes be defined.

```
42 \DeclareOption{plainoutput}{%
43   \@plainoutputtrue \@strictfootnotestrue
44   \@altfootnotesfalse \@magnificationtrue}
45 \DeclareOption{plainoutputoff}{\@plainoutputfalse}
```

The two special options are given.

```
46 \DeclareOption{none}{%
47   \@outerallocsfalse    \@eqalignfalse
48   \@diagnosticsfalse    \@plainskipsfalse
49   \@strictfootnotesfalse\@strictlinefalse
50   \@altfootnotesfalse   \@tabbingfalse
51   \@magnificationfalse  \@strictitemfalse
52   \@plainoutputfalse}
53
54 \DeclareOption{all}{%
55   \@outerallocstrue    \@eqaligntrue
56   \@diagnosticstrue    \@plainskipstrue
57   \@strictfootnotestrue\@strictlinetrue
58   \@altfootnotestrue   \@tabbingtrue
59   \@magnificationtrue  \@strictitemtrue
60   \@plainoutputtrue}
```

Finally we define a default option handling routine. I prefer only a warning as opposed to an error.

```
61 \DeclareOption*{%
62   \PackageWarning{cjwplain}{Unknown option `\CurrentOption'}}
```

Now that all the options are declared, we process them in the order specified in the package call.

```
\ProcessOptions*
```

2.2 Allocation Calls: outerallocs

\newcount \newdimen \newskip \newbox \newwrite \newfam

Originally Plain TeX had all allocation macros (`\newcount`, etc.) defined as `\outer`. LaTeX redefines several of them to be non-outer. Careful consideration has failed to yield to me why these would need to be rewritten as `\outer` in this package—any Plain TeX file which expects `\outer` definitions would not call them in a non-outer position, and any other files would themselves have redefined versions of the macros.

Since, however, it is such a small change, we will provide it. Note: Using the `outerallocs` option will break a good deal of standard LaTeX code, namely the standard macros for counters and lengths. This means you probably do *not* want to use it. It is here only for completeness's sake.

```
\if@outerallocs

\outer\def\newcount{\alloc@0\count\countdef\insc@unt}
\outer\def\newdimen{\alloc@1\dimen\dimendef\insc@unt}
\outer\def\newskip{\alloc@2\skip\skipdef\insc@unt}

\outer\def\newbox{\alloc@4\box\chardef\insc@unt}
\outer\def\newwrite{\alloc@7\write\chardef\sixt@@n}
\outer\def\newfam{\alloc@8\fam\chardef\sixt@@n}

\fi
```

2.3 Error Processing: diagnostics

Any Plain TeX afficianados using this package will feel more comfortable to have the standard values for error processing information. One change, though. LaTeX 2_ε uses a *counter* named `errorcontextlines`, and not a count.

```
\if@diagnostics

\showboxbreadth=5
\showboxdepth=3
\setcounter{errorcontextlines}{5}

\fi
```

2.4 Skips: plainskips

When the `plainskips` option is selected, the three `\...skip` macros should unconditionally leave horizontal mode and insert a skip, like in Plain TeX.

```
\if@plainskips
  \def\smallskip{\vskip\smallskipamount}
  \def\medskip{\vskip\medskipamount}
  \def\bigskip{\vskip\bigskipamount}
\fi
```

2.5 Fonts

A package already exists whereby oldstyle font commands can be given, namely `oldlfont`. Furthermore, one can use `rawfonts`, if necessary, to load in such specific fonts as `\ninebf`, etc.

```
%\font\tenrm=cmr10 % roman text
    ⋮
%\textfont\ttfam=\tentt
```

2.6 The `\line` Macro: strictline

`\line` `\latex@line`

Now we get to the first tricky part. The `\line` macro needs to be available to the `picture` environment in LaTeX, as well as restoring the original Plain TeX definition for our usage here. The good news is that LaTeX *only* uses `\line` inside of the `picture` environment. So we employ the following solution: we keep the definition of `\@@line` as per LaTeX convention, and give in any case a user accessible `\plainline`.

```
\let\plainline\@@line

\if@strictline

%\def\@@line{\hbox to\hsize} % Defined in |ltplain.dtx|
```

Now we define an internal name for the standard LaTeX macro and restore the Plain TeX definition.

```
\let\latex@line\line
\let\line\@@line
```

The definitions of `\leftline`, `\rightline` and `\centerline` can be left as is (though users depending upon personal redefinitions of `\line` for special effects in these macros should simply put their redefinition into the macro `\@@line`).

```
%\def\leftline#1{\@@line{#1\hss}}
%\def\rightline#1{\@@line{\hss#1}}
%\def\centerline#1{\@@line{\hss#1\hss}}
```

Now we make a patch to the definition of `\@picture` (the workhorse macro for the `picture` environment) which will restore the LaTeX definition only within that environment.

```
\def\@picture(#1,#2)(#3,#4){%
  \let\line\latex@line%
  \@picht#2\unitlength
  \setbox\@picbox\hbox to#1\unitlength\bgroup
    \hskip -#3\unitlength
    \lower #4\unitlength\hbox\bgroup
      \ignorespaces}

\fi
```

2.7 Tab Alignments: tabbing

The tabbing macros from Plain TeX use the `\newif` construction, so must occur at an `\outer` level. Thus, they are included in a separate package.

```
\if@tabbing
  \InputIfFileExists{cjwpltab.clo}{}{%
    \PackageWarning{cjwplain}{Option `cjwpltab.clo' not found.}
    \@tabbingfalse}
\fi
⟨/package⟩
⟨*tabbing⟩
```

\cleartabs
\settabs
\tabalign
\+

LaTeX may have it's own tabbing environment, but I like Plain TeX's. The only potential conflict I saw was with the `\+` macro. However, LaTeX only defines `\+` inside of the `tabbing` environment itself, so there should be absolutely no problem.

```
\newif\ifus@ \newif\if@cr
\newbox\tabs \newbox\tabsyet \newbox\tabsdone

\def\cleartabs{\global\setbox\tabsyet\null \setbox\tabs\null}
\def\settabs{\setbox\tabs\null \futurelet\next\sett@b}
\let\+=\relax % in case this file is being read in twice
\def\sett@b{\ifx\next\+\let\next\relax
\def\next{\afterassignment\s@tt@b\let\next}%
  \else\let\next\s@tcols\fi\next}
\def\s@tt@b{\let\next\relax\us@false\m@ketabbox}
\def\tabalign{\us@true\m@ketabbox} % non-\outer version of \+
\outer\def\+{\tabalign}
\def\s@tcols#1\columns{\count@#1 \dimen@\hsize
  \loop\ifnum\count@>\z@ \@nother \repeat}
\def\@nother{\dimen@ii\dimen@ \divide\dimen@ii\count@
  \setbox\tabs\hbox{\hbox to\dimen@ii{}\unhbox\tabs}%
  \advance\dimen@-\dimen@ii \advance\count@\m@ne}

\def\m@ketabbox{\begingroup
  \global\setbox\tabsyet\copy\tabs
  \global\setbox\tabsdone\null
  \def\cr{\@crtrue\crcr\egroup\egroup
\ifus@\unvbox\z@\lastbox\fi\endgroup
\setbox\tabs\hbox{\unhbox\tabsyet\unhbox\tabsdone}}%
  \setbox\z@\vbox\bgroup\@crfalse
\ialign\bgroup&\t@bbox##\t@bb@x\crcr}

\def\t@bbox{\setbox\z@\hbox\bgroup}
\def\t@bb@x{\if@cr\egroup % now \box\z@ holds the column
  \else\hss\egroup \global\setbox\tabsyet\hbox{\unhbox\tabsyet
      \global\setbox\@ne\lastbox}% now \box\@ne holds its size
\ifvoid\@ne\global\setbox\@ne\hbox to\wd\z@{}%
\else\setbox\z@\hbox to\wd\@ne{\unhbox\z@}\fi
\global\setbox\tabsdone\hbox{\box\@ne\unhbox\tabsdone}\fi
  \box\z@}
⟨/tabbing⟩
⟨*package⟩
```

2.8 Itemising: strictitem

\@@item
\plainitem

Now we have another problem, namely the `\item` macro. I unfortunately see no way to get around the fact that `\item` is a general macro in LaTeX, and that the

formats are completely different: i.e., PLAIN TEX expects the ⟨*label*⟩ to be the one mandatory argument, whereas LATEX's `\item` macro takes the ⟨*label*⟩ as an optional argument. Thus, the best I can think of is the following. We redefine PLAIN TEX's `\item` after standard LATEX practice,

```
\def\@@item{\par\hang\textindent}
```

and we `\let` it to something accessible in normal documents, the command `\plainitem`.

```
\let\plainitem\@@item
```

The command `\itemitem` can be taken care of directly.

```
\def\itemitem{\par\indent \hangindent2\parindent \textindent}
```

Now a user will have to replace all occurrences of `\item{foo}` with either `\item[foo]` or `\plainitem{foo}` (I imagine the choice will depend upon one's editor's facilities with regexps...). It's not perfect, but it's the only way I can think of to provide maximum compatibility. Of course, we will still give the option, `strictitem`, of using just the original definition, but that will probably not be terribly convenient for anyone trying to add LATEX features on top of an existing PLAIN TEX source. Thus, we will also provide the (slightly longwinded) replacement `\latexitem`.

```
\if@strictitem
  \let\latexitem\item
  \let\item\@@item
\fi
```

2.9 Miscellaneous

2.9.1 Sectioning

I have personally never used the PLAIN TEX `\beginsection` macro, but somebody might have...

```
\outer\def\beginsection#1\par{\vskip\z@ plus.3\vsize\penalty-250
  \vskip\z@ plus-.3\vsize\bigskip\vskip\parskip
  \message{#1}\leftline{\bf#1}\nobreak\smallskip\noindent}
```

2.9.2 Proclamations

Once again we will leave LATEX's NFSS based redefinition, this time for the `\proclaim` command, in place.

```
%\outer\def\proclaim #1. #2\par{\medbreak
%  \noindent{\bfseries#1.\enspace}{\slshape#2\par}%
%  \ifdim\lastskip<\medskipamount \removelastskip\penalty55\medskip\fi}
```

2.9.3 Paragraph Formatting

I have done some simple tests of LATEX's `\raggedright` macro, and it seems to me that it mimics the functionality of the PLAIN TEX macro of the same name. Therefore I see no reason to redefine it as part of this package.

```
%\def\raggedright{%
%  \rightskip\z@ plus2em \spaceskip.3333em \xspaceskip.5em\relax}
```

Another LATEX font change will also be left as is.

```
%\def\ttraggedright{\reset@font\ttfamily\rightskip\z@ plus2em\relax}
```

2.9.4 Accents and Miscellaneous

These should work as is for PLAIN TEX documents.

```
%\def\_{\leavevmode \kern.06em \vbox{\hrule \@width.3em}}
%\def\AA{\leavevmode\setbox0\hbox{h}\dimen@\ht0\advance\dimen@-1ex%
%  \rlap{\raise.67\dimen@\hbox{\char'27}}A}
```

Nor do I see a reason to change these back to PLAIN TEX definitions.

```
%\def\pd#1{\oalign{#1\crcr\hidewidth\sh@ft{08}.\hidewidth}}
%\def\d{\protect\pd}
%
%\def\pb#1{\oalign{#1\crcr\hidewidth\sh@ft{29}%
%\vbox to.2ex{\hbox{\char22}\vss}\hidewidth}}
%\def\b{\protect\pb}
%
%\def\pc#1{\setbox\z@\hbox{#1}\ifdim\ht\z@=1ex\accent24 #1%
%  \else{\ooalign{\unhbox\z@\crcr\hidewidth\char24\hidewidth}}\fi}
%\def\c{\protect\pc}
%
%\def\pt#1{{\edef\next{\the\font}\the\textfont1\accent127\next#1}}
%\def\t{\protect\pt}
```

The LATEX definition of `\ldots` is more or less identical to the PLAIN TEX macro `\dots`. So we will leave this alone, too.

```
%\def\dots{\ldots}
```

These changes, as others before, only add functionality without seeming to limit PLAIN TEX usage, so no change will be made.

```
%\def\hrulefill{\leavevmode\leaders\hrule\hfill\kern\z@}
%\def\dotfill{\leavevmode\cleaders
%     \hbox{$\m@th \mkern1.5mu.\mkern1.5mu$}\hfill\kern\z@}
%
%\def\longrightarrow{\protect\@lra}
%  \def\@lra{\relbar\joinrel\rightarrow}
%\def\longleftarrow{\protect\@lla}
%   \def\@lla{\leftarrow\joinrel\relbar}
```

2.9.5 Ending the Document

We simply add the `\bye` macro back in, though the `\end` should be changed to the LATEX `\end{document}`.

```
\outer\def\bye{\end{document}}
```

2.9.6 Math Commands

Operators and other math-mode font-related changes will be ignored with as other NFSS alterations already mentioned.

```
%\def\log{\mathop{\rm log}\nolimits}
%\def\lg{\mathop{\rm lg}\nolimits}
%\def\deg{\mathop{\rm deg}\nolimits}
```

⋮

```
%\def\bmod{\mskip-\medmuskip\mkern5mu
%  \mathbin{\rm mod}\penalty900\mkern5mu\mskip-\medmuskip}
%\def\pmod#1{\allowbreak\mkern18mu({\rm mod}\,\,#1)}
```

Various \matrix type command, including \bordermatrix and \cases have similarly been rewritten for NFSS commands under LaTeX.

2.10 Math Alignment: eqalign

\eqalign
\eqalignno
\leqalignno

If the eqalign option has been selected, we add \eqalign back in, as well as related macros. Notice that we use the \@centering macro provided by LaTeX, because LaTeX itself reserves \centering. We also supply a user macro \plaincentering in case such is needed.

```
\let\plaincentering\@centering

\if@eqalign

\def\eqalign#1{\null\,\vcenter{\openup\jot\m@th
  \ialign{\strut\hfil$\displaystyle{##}$&$\displaystyle{{}##}$\hfil
      \crcr#1\crcr}}\,}

\def\eqalignno#1{\displ@y \tabskip\@centering
  \halign to\displaywidth{\hfil$\@lign\displaystyle{##}$\tabskip\z@skip
    &$\@lign\displaystyle{{}##}$\hfil\tabskip\@centering
    &\llap{$\@lign##$}\tabskip\z@skip\crcr
    #1\crcr}}
\def\leqalignno#1{\displ@y \tabskip\@centering
  \halign to\displaywidth{\hfil$\@lign\displaystyle{##}$\tabskip\z@skip
    &$\@lign\displaystyle{{}##}$\hfil\tabskip\@centering
    &\kern-\displaywidth\rlap{$\@lign##$}\tabskip\displaywidth\crcr
    #1\crcr}}

\else
```

eqnarray*

If the user does not choose this option, we will instead define an eqnarray* environment which does not number equations.

```
\@namedef{eqnarray*}{%
  \let \\ \cr $$\null\,\vcenter{\openup\jot\m@th
  \ialign{\strut\hfil$\displaystyle{##}$&$\displaystyle{{}##}$\hfil\crcr}}}
\@namedef{endeqnarray*}{\crcr}

\fi
```

2.11 Output Routine: plainoutput

The output routines also involve \newif commands, and are therefore also relegated to a separate package.

```
\if@plainoutput
  \InputIfFileExists{cjwplout.clo}{}{%
    \PackageWarning{cjwplain}{Option 'cjwplout.clo' not found.}
    \@tabbingfalse}
\fi
```

⟨/package⟩

⟨*output⟩

\headline
\footline
\pageno
\folio

If the user wishes to use the entire Plain TeX output routine, we first redefine

the normal versions of `headline`, `footline` and `pageno`, as well as related macros. We will use NFSS definitions in place of `\tenrm`.

```
\countdef\pageno=0 \pageno=1 % first page is number 1
\newtoks\headline \headline={\hfil} % headline is normally blank
\newtoks\footline \footline={\hss\reset@font\folio\hss}
  % footline is normally a centered page number in font \tenrm
\def\nopagenumbers{\footline{\hfil}} % blank out the footline
\def\folio{%
  \ifnum\pageno<\z@ \romannumeral-\pageno \else\number\pageno \fi}
\def\advancepageno{\ifnum\pageno<\z@ \global\advance\pageno\m@ne
  \else\global\advance\pageno\@ne \fi} % increase |pageno|
```

`\raggedbottom` `\normalbottom` We also supply the `\raggedbottom` macro and its counterpart, `\normalbottom`.

```
\newif\ifr@ggedbottom
\def\raggedbottom{\topskip 10\p@ plus60\p@ \r@ggedbottomtrue}
\def\normalbottom{\topskip 10\p@ \r@ggedbottomfalse}
                                          % undoes \raggedbottom
```

`\topinsert` `\midinsert` `\pageinsert` If the entire output routine is being used, we define the Plain TEX insertion macros as normal.

```
\newinsert\topins
\newif\ifp@ge \newif\if@mid
\def\topinsert{\@midfalse\p@gefalse\@ins}
\def\midinsert{\@midtrue\@ins}
\def\pageinsert{\@midfalse\p@getrue\@ins}
\skip\topins=\z@skip % no space added when a topinsert is present
\count\topins=1000 % magnification factor (1 to 1)
\dimen\topins=\maxdimen % no limit per page
\def\@ins{\par\begingroup\setbox\z@\vbox\bgroup} % start a \vbox
\def\endinsert{\egroup % finish the \vbox
   \if@mid \dimen@\ht\z@ \advance\dimen@\dp\z@ \advance\dimen@12\p@
  \advance\dimen@\pagetotal \advance\dimen@-\pageshrink
  \ifdim\dimen@>\pagegoal\@midfalse\p@gefalse\fi\fi
   \if@mid \bigskip\box\z@\bigbreak
   \else\insert\topins{\penalty100 % floating insertion
  \splittopskip\z@skip
  \splitmaxdepth\maxdimen \floatingpenalty\z@
  \ifp@ge \dimen@\dp\z@
  \vbox to\vsize{\unvbox\z@\kern-\dimen@}% depth is zero
  \else \box\z@\nobreak\bigskip\fi}\fi\endgroup}
```

`\plainoutput` Now we define the main part of the output routine. We use `\@@line` instead of `\line`, since `\@@line` is guaranteed to have the definition we want.

```
\def\plainoutput{\shipout\vbox{\makeheadline\pagebody\makefootline}%
  \advancepageno
  \ifnum\outputpenalty>-\@MM \else\dosupereject\fi}
\def\pagebody{\vbox to\vsize{\boxmaxdepth\maxdepth \pagecontents}}
\def\makeheadline{\vbox to\z@{\vskip-22.5\p@
  \@@line{\vbox to8.5\p@{}\the\headline}\vss}\nointerlineskip}
\def\makefootline{\baselineskip24\p@\@@line{\the\footline}}
\def\dosupereject{\ifnum\insertpenalties>\z@
                 % something is being held over
  \@@line{}\kern-\topskip\nobreak\vfill\supereject\fi}

\def\pagecontents{\ifvoid\topins\else\unvbox\topins\fi
```

```
  \dimen@=\dp\@cclv \unvbox\@cclv % open up \box255
  \ifvoid\footins\else % footnote info is present
    \vskip\skip\footins
    \footnoterule
    \unvbox\footins\fi
  \ifr@ggedbottom \kern-\dimen@ \vfil \fi}
```

Finally, we make the Plain TeX output routines active again.

```
  \output{\plainoutput}
⟨/output⟩
⟨*package⟩
```

2.11.1 Page Numbering, Running Heads and Miscellaneous

We can make the Plain TeX head and foot commands accessible (after a fashion), even if the entire output routine is not being used. LaTeX 2_ε provides the commands `\@oddhead` and `\@evenhead`, as well as their footline equivalents. Therefore we can give the following versions.

```
\if@plainoutput\else

\def\footline{\@ifnextchar ={\@@footline}{\@@footline=}}
\def\@@footline=#1{\gdef\@oddfoot{#1} \gdef\@evenfoot{#1}}

\def\headline{\@ifnextchar ={\@@headline}{\@@headline=}}
\def\@@headline=#1{\gdef\@oddhead{#1} \gdef\@evenhead{#1}}

\def\nopagenumbers{\let\@oddfoot\relax \let\@evenfoot\relax}
```

Since the Plain TeX `\headline` and `\footline` macros are actually tokens, we have to allow that the assignments to them can be made with an optional =. The use of `\@ifnextchar =` nicely takes care of that. This will alas be a slightly inefficient use of the macros, as many Plain TeX heads and feet already test for odd and even pages—but not all of them. So, we compromise.

Now we can simply make `\pageno` an equivalent for the LaTeX counter `\c@page`.

```
\let\pageno\c@page
```

This means that assignments can either be made in primitive fashion directly to `\pageno` or in LaTeXfashion to the `page` counter. We would also like to have the Plain TeX `\folio` macro. We are going to expand upon this slightly, though. I find `\folio` generally to be a useful command, so we will write it in such a way as to make it useful with LaTeX counters (as with, say, the `\arabic` macro) as well as with counts.

```
\newcommand{\folio}[1][\pageno]{\@folio{#1}}
\def\@folio#1{%
  \edef\@tempa{\string#1}%
  \expandafter\@ifundefined{c@\@tempa}% % Does the counter exist?
    {% %   No such counter.
      \ifnum #1 <\z@ \romannumeral-#1
        \else \number #1
      \fi
    }%
    {% %   It is a counter.
      \ifnum\value{#1}<\z@ \roman{#1} \else \arabic{#1} \fi}
  }
```

This is not, unfortunately, perfect, in that it must be used with an optional argument (`\folio[section]`) as opposed to the normal style (`\arabic{section}`). On the other hand, I can't think of many applications for `\folio` *other* than page numbering. At any rate, we finish page numbering with the incrementation command.

```
\def\advancepageno{\ifnum\pageno<\z@ \global\advance\pageno\m@ne
  \else\global\advance\pageno\@ne \fi}
```

One more bit from the PLAIN TEX output routine needs to be dealt with. Although LATEX's `\raggedbottom` macro will suffice to simulate PLAIN TEX's command of the same name, we need to add a `\let` command to enable PLAIN TEX's counterpart, `\normalbottom`.

```
\let\normalbottom\flushbottom
```

2.11.2 Insertions

If the PLAIN TEX output routine is not being used, we simulate the insertions using LATEX's `figure` environment.

```
\def\topinsert{\begin{figure}[t]}
\def\pageinsert{\begin{figure}[p]}
\def\midinsert{\begin{figure}[htpb]}
\def\endinsert{\end{figure}}

\fi
```

2.11.3 Footnotes: strictfootnotes, altfootnotes

We define `\@@footnote` as PLAIN TEX's footnoting macro.

```
%\newinsert\footins

\let\latex@footnote\footnote

\def\@@footnote#1{\let\@sf\empty % parameter #2 (the text) is read later
  \ifhmode\edef\@sf{\spacefactor\the\spacefactor}\/\fi
  #1\@sf\vfootnote{#1}}
\def\vfootnote#1{\insert\footins\bgroup
  \interlinepenalty\interfootnotelinepenalty
  \splittopskip\ht\strutbox % top baseline for broken footnotes
  \splitmaxdepth\dp\strutbox \floatingpenalty\@MM
  \leftskip\z@skip \rightskip\z@skip \spaceskip\z@skip
  \xspaceskip\z@skip
  \textindent{#1}\footstrut\futurelet\next\fo@t}
\def\fo@t{\ifcat\bgroup\noexpand\next \let\next\f@@t
  \else\let\next\f@t\fi \next}
\def\f@@t{\bgroup\aftergroup\@foot\let\next}
\def\f@t#1{#1\@foot}
\def\@foot{\strut\egroup}
\def\footstrut{\vbox to\splittopskip{}}
```

LATEX leaves these initializations for the `\footins` insert.

```
%\skip\footins=\bigskipamount % space added when footnote is present
%\count\footins=1000 % footnote magnification factor (1 to 1)
%\dimen\footins=8in % maximum footnotes per page
```

Now we have several options for how to really deal with footnotes. The easy answer is to do them entirely according to PLAIN TEX.

```
\if@strictfootnotes
  \let\footnote\@@footnote
\fi
```

The second option is to just use the LATEX `\footnote` command. This needs no rewriting, of course. The last option is to rewrite LATEX's `\footnote` macro to use the PLAIN TEX format instead of the LATEX format, which uses an optional argument.

```
\if@altfootnotes
  \def\footnote#1{\latex@footnote[#1]}
\fi
```

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
```

2.11.4 Magnification: magnification

The last part of PLAIN TEX for which we need to account is magnification. The magnification macros are easily reinstated, either as part of the overall PLAIN TEX output routine or standalone. Since the `\mag` primitive is not disabled, it *could* still be used in LATEX. However, LATEX does not itself work with `true` units any usage of `\magnification` could do some strange things to your page layouts.

```
\if@magnification
  \def\magnification{\afterassignment\m@g\count@}
  \def\m@g{\mag\count@
    \hsize6.5truein\vsize8.9truein\dimen\footins8truein}
\fi
```

This brings us to the end of the main package.

```
⟨/package⟩
```