

The
L_AT_EX
Wizard's Manual

Volume I

Copyright © 1991 by Michael Spivak
All Rights Reserved

A note on the L^AT_EX's Wizard's Manual

At present the Wizard's Manual is available only in this form, laboriously printed on a laser printer: The manual was divided into several sections, and for each section two .dvi files were produced, one for printing the odd-numbered pages, one for the even-numbered pages.

As a result of this procedure, which was carried out fairly hastily, several anomalies may occur, like running heads on otherwise blank pages, or two footnotes on a page both numbered 1: these anomalies are artifacts that do not occur when the file is handled normally.

In the unlikely event that there is sufficient interest in this Wizard's Manual to warrant publication as a book, a special rate will be given to those who have ordered the manual in its current form.

Meanwhile, updates will be issued as bugs are discovered and corrected, or as new features are added.

The idea of making an index is too horrible to contemplate—many, many entries would probably have dozens of citations. However, when the manual seems to have reached a stable state I will provide a “source” code. This will not be a .tex file, but a sort of ASCII representation of the book, so that one can use a text editor to search for anything.

Meanwhile, of course, please report any misprints, mistakes, omissions, bugs, etc., either by mail,

TEXplorators
3701 W. Alabama, Suite 450-273
Houston, TX, 77027 (U.S.A.)

or by e-mail,

spivak@rice.edu

CONTENTS

Part I Preliminaries

Chapter 1. Introduction	3
1.1 <i>AMS-TeX</i> Conventions	3
1.2 Constructions from <i>AMS-TeX</i>	5
1.3 Changes to <i>AMS-TeX</i> ; <code>\local</code> and <code>\global</code> assignments	7
Chapter 2. Getting started with <i>LAMS-TeX</i>	9
Chapter 3. Changes to <i>AMS-TeX</i>	12
3.1 <code>\Err@</code>	12
3.2 <code>\atdef@</code>	12
3.3 Tests	15
3.4 Spaces after control sequence names in error messages	16
3.5 Line breaking	18
3.6 <code>\alloc@</code> , <code>\newcount@</code> , and <code>\newbox@</code>	19
3.7 Lists	21
3.8 Skipping spaces in <code>\futurelet</code> 's	24
3.9 <code>\loop</code>	26
3.10 A la français	28
Chapter 4. Numbering styles	34
Chapter 5. Printing cardinal and ordinal numbers	38
Chapter 6. Inhibiting expansion	43
Chapter 7. Invisibility	45
7.1 Invisible constructions	45
7.2 Special considerations for invisible constructions	46

Chapter 8. Special considerations for <code>\everypar</code>	49
Chapter 9. <code>\page</code>	50
Chapter 10. Indexing	52
10.1 The <code>.ndx</code> file	52
10.2 <code>\indexproofing</code>	53
10.3 Converting tokens to type 12	54
10.4 The <code>\starparts@</code> and <code>\windex@</code> routines	56
10.5 Indexing	59
10.6 Changes to the L ^A T _E X Manual	60
10.7 Invisibility	61
10.8 Other delimiters for index entries	61
10.9 <code>\idefine</code> and <code>\iabbrev</code>	63

Part II Labels and Cross References

Chapter 11. The <code>\label</code> mechanism	69
11.1 Constructions that can be given <code>(label)</code> 's	69
11.2 Restrictions	71
11.3 Consequences of these restrictions	71
11.4 <code>\Initialize</code>	72
11.5 The question of fonts	73
11.6 Storing <code>(label)</code> 's	74
11.7 <code>\ref</code> and its relatives	76
11.8 <code>\label</code>	76
11.9 <code>\pagelabel</code>	77
Chapter 12. Beginning the document	80
12.1 Preliminaries	80
12.2 <code>\document</code>	81

Chapter 13. Labels	84
13.1 <code>\label</code>	84
13.2 <code>\pagelabel</code>	86
Chapter 14. Cross-Referencing	89
14.1 Preliminaries	89
14.2 <code>\ref</code> and its relatives	91
Chapter 15. Reading auxiliary files	94
15.1 <code>\readlax</code>	94
15.2 Style files	96

***Part III Particular Constructions Allowing Labels
and their associates***

Chapter 16. Displayed formulas	99
16.1 Invisibility	99
16.2 Localizing labels	104
16.3 <code>\tag</code>	104
16.4 <code>\align</code>	109
16.5 <code>\alignat</code> and <code>\xalignat</code>	114
16.6 <code>\gather</code>	118
Chapter 17. New counters	119
17.1 <code>\newcounter</code>	119
17.2 <code>\usecounter</code>	122
Chapter 18. Lists	128
18.1 Style choices	128
18.2 Counters, etc.	131
18.3 Other preliminaries	132
18.4 <code>\list</code>	134
18.5 <code>\item</code>	137

18.6	<code>\runitem@</code>	141
18.7	<code>\inlevel</code>	143
18.8	<code>\outlevel</code>	144
18.9	<code>\endlist</code>	144
Chapter 19. <code>\describe</code> and <code>\margins</code>		148
19.1	<code>\describe</code>	148
19.2	<code>\margins</code>	149
Chapter 20. <code>\nopunct</code>, <code>\nospace</code>, and <code>\overlong</code>		154
20.1	<code>\nopunct</code> , <code>\nospace</code> , and <code>\overlong</code>	154
20.2	Using the flags	159
Chapter 21. <code>\demo</code>		161
Chapter 22. <code>\claim's</code>		165
22.1	Preliminaries	165
22.2	<code>\claimformat@</code>	166
22.3	Further preliminaries	168
22.4	Starting a <code>\claim</code>	169
22.5	Starting a <code>\claim@c</code>	171
22.6	Starting a <code>\claim@q</code>	173
22.7	Finishing off	173
22.8	<code>\endclaim</code>	175
22.9	<code>\newclaim</code>	175
22.10	<code>\shortenclaim</code>	181
22.11	Customizing <code>\claim's</code>	185
Chapter 23. Heading levels		187
23.1	The <code>.toc</code> file	187
23.2	Preliminaries	187
23.3	Different levels of <code>\HL</code>	188
23.4	The <code>\HL</code> construction	188
23.5	The <code>\hl</code> construction	195
23.6	Other elements of heading levels	198
23.7	Writing long token lists	199

23.8	<code>\HLtoc@</code> and <code>\hltoc@</code>	201
23.9	<code>\mainfile</code>	207
23.10	Creating heading levels	208
23.11	Initializations	211
23.12	<code>\aftertoc@</code>	212
23.13	Order of heading levels	213
23.14	Naming header levels	214
23.15	<code>\Initialize</code>	221

Chapter 24. Accessing and controlling counters, styles, etc. 224

Chapter 25. Footnotes 240

25.1	Preliminaries	240
25.2	<code>\footnote@</code>	241
25.3	Fancy footnote numbering	247
25.4	<code>\footmark</code>	249
25.5	<code>\foottext</code>	256
25.6	<code>\footnote</code>	258

Part IV Miscellaneous Constructions

Chapter 26. Literal mode 261

26.1	In-line literal mode	261
26.2	Displayed literal mode	264
26.3	Notes for the wary	266
26.4	Prohibiting page breaks	267
26.5	Indentation	268
26.6	TAB's	268
26.7	Widow control	270
26.8	Page breaks	272
26.9	<code>\Litbox</code>	274
26.10	The general definition	275
26.11	Nicer syntax	281

Chapter 27. Literal mode in heading levels	288
27.1 Literal mode in <code>\HL</code> and <code>\hl</code>	289
27.2 The general definitions	294
Chapter 28. Title, author, etc., in the default style	298
Chapter 29. The bibliography	301
29.1 <code>\cite</code>	302
29.2 Features of L ^A T _E X's bibliography macros	303
29.3 Storing the fields	313
29.4 Starting the bibliography macros	315
29.5 <code>\bibinfo@</code>	318
29.6 Additional flags	320
29.7 <code>\bib</code>	321
29.8 The basic construction	323
29.9 <code>\no</code> , <code>\key</code> ,	325
29.10 Manipulating the <code>\vbox</code> 'es	328
29.11 Line breaking commands	329
29.12 Adding punctuation before a field	332
29.13 <code>\endbib@</code>	334
29.14 <code>\endbib</code> , <code>\morebib</code> , <code>\anotherbib</code> , and <code>\transl</code>	341
Chapter 30. Interfacing with BIB_TE_X	345
30.1 <code>\UseBibTeX</code>	346
30.2 The <code>bibtex.tex</code> file	351
Chapter 31. <code>\purge'ing</code> and <code>\unpurge'ing</code>	358

Part V Islands and the Output Routine

Chapter 32. Packaging figures, tables, . . . , with captions	363
32.1 Preliminaries	363
32.2 Starting an <code>\island</code>	366

32.3	Starting a <code>\caption</code>	372
32.4	Formatting a <code>\caption</code>	375
32.5	<code>\ticwrite@</code>	381
32.6	<code>\Htrim@</code>	383
32.7	Other accoutrements for <code>\endisland</code>	386
32.8	<code>\endisland</code>	386
32.9	<code>\newisland</code>	391
32.10	Manipulating <code>\island</code> 's	391
Chapter 33. An overview; placing the packaged figures, tables, . . .		395
33.1	<code>\place</code>	395
33.2	Automatic placement	396
33.3	Setting things up	399
33.4	How <code>\Aplace</code> works	399
33.5	How the <code>\output</code> routine works	405
33.6	When insertions float	416
33.7	What happens to an <code>\Hbyw</code> ?	418
Chapter 34. <code>\Aplace</code>, <code>\AAplace</code> and <code>\Bplace</code>		419
34.1	Figures, etc., within <code>\Par . . . \endPar</code>	419
34.2	<code>\place@</code>	421
34.3	<code>\Aplace</code> and <code>\AAplace</code>	424
34.4	<code>\Bplace</code>	426
34.5	Changing <code>\pagecontents</code>	429
34.6	<code>\breakisland@</code> and <code>\printisland@</code>	430
34.7	<code>\bottomfigs@</code>	430
34.8	<code>\resetdimtopins@</code>	431
Chapter 35. <code>\Cplace</code>, <code>\Mplace</code>, and <code>\MXplace</code>		435
35.1	<code>\Place@</code>	435
35.2	<code>\Cplace@</code>	436
35.3	<code>\Mplace@</code> and <code>\MXplace@</code>	437
35.4	<code>\endPar</code>	441

Chapter 36. The <code>\output</code> routine: Ta-ran-ta-ra! Ta-ran-ta-ra!	
Ta-ran-ta-ra!	449
36.1 <code>\plainoutput</code>	449
36.2 <code>\pagebody</code>	452
36.3 <code>\pagecontents</code>	453
36.4 And we are done!	459
36.5 When <code>\box255</code> is too small	460
36.6 The endgame	461
36.7 The endgame once again	461
36.8 The endgame once again	464
36.9 The endgame once again	464
36.10 The endgame once again	464
36.11 The endgame once again	464
36.12 The endgame once again	465
36.13 A final warning	465

Part VI Front and Back Matter

Chapter 37. Front Matter (Table of Contents, List of Figures, Tables, etc.)	471
37.1 <code>lamstex.stf</code> preliminaries	471
37.2 Setting an entry	472
37.3 Further preliminaries for the table of contents	477
37.4 Starting the <code>\maketoc</code> command	479
37.5 Redefining <code>\HL</code> and <code>\hl</code>	480
37.6 <code>\NameHL</code> and <code>\Namehl</code>	483
37.7 <code>\maketoc</code>	484
37.8 Lists of Figures, Tables, etc.	485
37.9 <code>Fini</code>	490
Chapter 38. Back matter; the index	491
38.1 Preliminaries	491
38.2 <code>\LETTER</code> and <code>\Entry</code>	495
38.3 <code>\Page</code> , etc.	502

38.4	<code>\Xref</code> , etc.	503
38.5	Preliminaries for double columns	506
38.6	<code>\makeindex</code>	507
38.7	<code>\combinecols@</code>	509
38.8	<code>\doublecolumns@</code>	511
38.9	<code>\balancecolumns@</code>	519
Chapter 39. The index program		524
39.1	The <code>.ndx</code> file	524
39.2	The index program	526

Part VII The Style Files

Introduction	537	
Chapter 40. The paper style	538	
40.1	Basic settings	538
40.2	Fonts and point sizes	539
40.3	The <code>.toc</code> levels	544
40.4	Setting up heading levels	545
40.5	Footnotes	550
40.6	Additional “top matter” and “end matter” constructions	551
40.7	Bibliography	555
40.8	<code>\maketoc</code>	556
Chapter 41. The book style	559	
41.1	Basic settings	559
41.2	Fonts and point sizes	560
41.3	The <code>.toc</code> levels	560
41.4	Flushing out figures	560
41.5	<code>\part</code> and <code>\chapter</code>	561
41.6	<code>\plainoutput</code>	568
41.7	Other heading levels	574
41.8	Footnotes	578

41.9	Bibliography	578
41.10	book.stf	578
41.11	book.stb	583
Chapter 42.	The letter style	590



Chapter 1. Introduction

This manual is intended for T_EX wizards pondering the intricacies of various L_AM_S-T_EX constructions, as well as for T_EXnicians designing style files for L_AM_S-T_EX who need more detailed information than that provided by the L_AM_S-T_EX Style File Designers Manual. Although certain points about T_EX receive detailed explanation, many sections presuppose considerable T_EXpertise, which it would be impractical to try to provide within the scope of this already lengthy manual.

Despite the manual's lengthiness, the division into chapters and sections allows specialized constructions used for one part of L_AM_S-T_EX to be separated from those used in other parts. Of course, the various chapters are not completely independent, and Part I should probably be perused by everyone.

1.1. L_AM_S-T_EX Conventions. We will not be analyzing the file `amstex.tex` itself, since a detailed description of L_AM_S-T_EX is given in the file `amstex.doc`. Nevertheless, certain L_AM_S-T_EX conventions must be mentioned here, because they are used throughout `lamstex.tex`.

First of all, L_AM_S-T_EX uses the "scratch" tokens `\next`, `\next@`, `\nextii@`, `\nextiii@`, ... In order to keep the number down, many definitions will, for example, define `\nextiv@` back in terms of `\next@`, `\nextii@`, etc.

The `amstex.doc` file mentions the peculiar contortions that are used to avoid difficulties that might arise when a definition has a clause like

```
\def\next@{ ... \next ... }
```

since a previous `\futurelet\next` may have let `\next` be something that is `\outer`.

In L_AM_S-T_EX, on the other hand, it is simply quite out of the question to allow anything to be `\outer`. Something like `\claim` can't be `\outer`, for example, because then things like `\newpre\claim` wouldn't work. But even something like `\bye` can't be `\outer` because it might easily occur right after a point where L_AM_S-T_EX has to subject the `next` token to some sort of test (see the small print notes on pages 100 and 146). Consequently, although we will continue to reserve `\next` as the token of choice in all `\futurelet` constructions, we will finesse this whole problem by making sure that *nothing* in L_AM_S-T_EX is `\outer`.

The only `\outer` things in plain TeX are the ASCII form-feed `^^L`, the `\new...` constructions (`\newcount`, `\newdimen`, ...), the `\+` from the `\settabs` construction, the `\beginsection` and `\proclaim` constructions, and `\bye`. \LaTeX redefines `^^L`, the `\new...` constructions, `\+`, and `\bye` so that they are not `\outer`, and it makes `\beginsection` be undefined (\LaTeX has its own system of “heading levels”), while $\mathcal{A}\mathcal{M}\mathcal{S}$ -TeX has already made `\proclaim` be undefined (until a style file is read in).

It should be mentioned that although we no longer have to worry about `\next` being `\outer`, some precautions are still in order—see page 23.

Another $\mathcal{A}\mathcal{M}\mathcal{S}$ -TeX convention involves constructions like

```
\if... \def\next@{\csa}\else\def\next@{\csb}\fi\next@
```

(It is assumed that the user of this manual understands why this is required instead of simply `\if... \csa\else\csb\fi` whenever `\csa` or `\csb` has an argument.)

In TUGBOAT, Volume 8, No. 2, Kabelschacht points out that this can be replaced by

```
\if... \expandafter\csa\else\expandafter\csb\fi
```

We will call this the “K-method”; it is often used without explicit mention. (As pointed out in the `amstex.doc` file, however, this method is not always valid or practicable).

Another frequently used convention of $\mathcal{A}\mathcal{M}\mathcal{S}$ -TeX is “compressed format”. We often have to make definitions of the form

```
\def\cs{\futurelet\next\cs@}
\def\cs@{\ifx\next(something or other)%
  \def\next@{...\cs@@...}\else
  \def\next@{...\cs@@@...}\fi
\next@}
\def\cs@@{...}
\def\cs@@@{...}
```

But this uses up three new control sequence names, `\cs@`, `\cs@@`, and `\cs@@@`, just for this one construction. The “compressed format” uses the same names

$\backslash\text{next@}$, $\backslash\text{nextii@}$, etc., over and over again, simply redefining them within each definition:

```

\def\cs{%
  \def\next@{\ifx\next(something or other)%
    \def\next@{... \nextii@...}\else
    \def\next@{... \nextiii@...}\fi
  \next@}%
\def\nextii@{...}%
\def\nextiii@{...}%
\futurelet\next\next@}

```

Notice that the “first” clause $\backslash\text{futurelet}\backslash\text{next}\backslash\text{next@}$ has to be made last (and, although it looks strange at first, it’s perfectly legitimate to have $\backslash\text{next@}$ defined in terms of $\backslash\text{next@}$ in this situation).

Compressed format makes things go a little slower, since $\backslash\text{next@}$, etc., have to be redefined all the time, but seems worth it, especially since it is usually used for major formatting constructions that introduce a lot of space anyway.

$\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\mathcal{T}\mathcal{E}\mathcal{X}$ also uses the construction

```
\Invalid@\controlseq
```

to make $\backslash\text{controlseq}$ give an error message. As explained in `amstex.doc`, this is recommended for any control sequences (often discovered via a $\backslash\text{futurelet}$) that function as “syntax” for other control sequences, and consequently shouldn’t be encountered on their own.

1.2. Constructions from $\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\mathcal{T}\mathcal{E}\mathcal{X}$. Some more specific $\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\mathcal{T}\mathcal{E}\mathcal{X}$ code should also be mentioned. First of all, the code

```
(A) \ifx\amstexloaded@\relax\catcode'\@=\active
     \endinput\else\let\amstexloaded@=\relax\fi
```

appears near the beginning of the `amstex.tex` file. This prevents the file `amstex.tex` from being loaded twice by making $\backslash\text{amstexloaded@}$ be undefined if `amstex.tex` hasn’t been loaded, but $\backslash\text{relax}$ if it has. This is a necessity because of the two lines

```

\let\ic@=/
\def/{\unskip\ic@}

```

that occur later (compare *The T_EXbook*, pp. 382–383).

Testing `\amstexloaded@` also allows other macro packages to tell whether `amstex.tex` has already been loaded, which is important for L_AM_S-T_EX, as we will see in the next chapter.

L_AM_S-T_EX also introduces two new counters, and a new token list,

```
\newcount\count@@
\newcount\count@@@
\toksdef\toks@@=2
```

in addition to the counter `\count@` and token list `\toks@` provided by `plain.tex`; these are also used in L_AM_S-T_EX (see section 3 for the choice of 2 in the `\toksdef`).

Furthermore, L_AM_S-T_EX introduces the abbreviations

```
\def\FN@{\futurelet\next}
\def\DN@{\def\next@}
\def\DNii@{\def\nextii@}
\def\RIfM@{\relax\ifmmode}
\def\RIfMIIfI@{\relax\ifmmode\ifinner}
\def\setboxz@h{\setbox\z@\hbox}
\def\wdz@{\wd\z@}
\def\boxz@{\box\z@}
```

These are used throughout L_AM_S-T_EX also. When we show `lamstex.tex` code, however, we will usually expand out these definitions, to make things easier to read. Similarly, certain control sequences from `plain`, like `\z@`, `\p@`, etc., will usually be expanded out for the sake of readability. Moreover, in constructions like

```
\count@=...
\let\next@=\relax
```

and so forth, we will often add the optional `=` signs that are normally omitted in the code.

We will frequently use the L_AM_S-T_EX control sequence `\eat@` defined by

```
\def\eat@#1{}
```


$\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$ introduces the token $\backslash\text{space@}$ that has been $\backslash\text{let}$ equal to a space. It is often used after

```
 $\backslash\text{futurelet}\backslash\text{next}\backslash\text{foo}$ 
```

constructions where $\backslash\text{foo}$ has to do something special if the next token is a space. In many cases, $\backslash\text{foo}$ must skip over that space, and then execute $\backslash\text{goo}$. The standard $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$ way of doing this is with the code

```
 $\backslash\text{ifx}\backslash\text{next}\backslash\text{space@}\backslash\text{def}\backslash\text{next@}.\{\backslash\text{goo}\}\backslash\text{else}$   
 $\backslash\text{def}\backslash\text{next@}.\{\backslash\text{goo}\}\backslash\text{fi}\backslash\text{next@}.$ 
```

The $.$ after the $\backslash\text{next@}$ makes the space ‘visible’ to $\mathcal{T}\mathcal{E}\mathcal{X}$.

As we shall see in section 3.8, $\mathcal{L}\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$ introduces a somewhat more economical approach to this problem.

By the way, a case like this, where something is part of the syntax for $\backslash\text{next@}$, is one of the situations where the K-method would not work.

There are a few more $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$ devices that are important in $\mathcal{L}\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$, but their discussion has been deferred until Chapter 3, since these devices are actually additions intended for later versions of $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$.

1.3. Changes to $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$; $\backslash\text{local}$ and $\backslash\text{global}$ assignments. Numerous lines of amstex.tex have been deleted in amstex1.tex because they are not used, or are modified, by lamstex.tex . Major changes of this sort are discussed at the appropriate points.

There are also numerous small changes. For example, all $\backslash\text{relaxnext@}$'s have been omitted, since $\mathcal{L}\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$ no longer needs that device for dealing with $\backslash\text{outer}$ constructions.

One change was necessary to avoid a conflict: $\backslash\text{roman}$ is now used in $\mathcal{L}\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$ for a numbering control sequence, whereas it previously had a different (extremely unlikely) use in $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$, as a control sequence to be used in math to produce a roman letter. The latter has now been changed from $\backslash\text{roman}$ to $\backslash\text{rom}$.

One other change should be mentioned explicitly: near the beginning of amstex1.tex the definitions

```
 $\backslash\text{def}\backslash\text{height}\{\text{height}\}$   
 $\backslash\text{def}\backslash\text{width}\{\text{width}\}$   
 $\backslash\text{def}\backslash\text{depth}\{\text{depth}\}$ 
```

have been inserted, so that ‘height’, ‘width’ and ‘depth’ can be replaced by the corresponding single tokens in the specifications for various `\hrules` and `\vrules`; these replacements save even more memory space in `lamstex.tex`, where rules occur much more frequently.

Finally, I have now conscientiously adhered to *The T_EXbook*’s recommendations (see pages 301 and 346) that assignments of variables either always be global or always be local. In most cases, the necessary changes have been minor (like changing some `\xdef`’s to `\edef`’s, or vice versa), but somewhat more extensive changes were required to ensure that `\setboxn` is always local for n even and global for n odd; these changes occur in the definitions of `\insplit@`, `\rendsplit@`, `\lendsplit@`, `\multline@@@`, `\rmultline@@@`, `\binrel@`, `\sideset@`, `\r@@t`, `\pmb@`, and perhaps one or two other places.

We will exercise comparable care regarding assignments of variables throughout L_AT_EX.

Chapter 2. Getting started with L_AM_S-T_EX

The first thing in `lamstex.tex`, after the copyright notice, is

```
\catcode'\@=11
```

to make `@` a letter, in order to create “private” control sequences that the casual user cannot type, as well as to access such private control sequences from `plain.tex` and `amstexl.tex`.

We will always adhere to the convention introduced here, using horizontal lines when we print actual `lamstex.tex` code, as opposed to examples, pieces of code, etc. We will often use different line breaks from the actual `lamstex.tex` code, so that it will fit better on these printed pages. (It should also be noted that when we give preliminary pieces of code we will often omit `%` signs at the ends of lines, although they are meticulously added when needed in the code itself.)

Since L_AM_S-T_EX is not supposed to be loaded unless A_MS-T_EX has already been loaded, the next code,

```
\ifx\amstexloaded@\relax\else  
\errmessage{AmS-TeX must be loaded before LamS-TeX}\fi
```

produces an error message if it hasn't—see the discussion of the code (A) on page 5.

We will adopt a different scheme for preventing `lamstex.tex` from being read in twice, one that doesn't create a new control sequence name, by using the fact that `lamstex.tex` will eventually define the control sequence `\laxread@` (see page 80), while A_MS-T_EX makes certain that `\undefined` is always undefined (see `amstex.doc`):

```
\ifx\laxread@\undefined\else\catcode'\@=\active\endinput\fi
```

[Other macro packages that need to know whether or not L_AM_S-T_EX has been loaded can use a similar test, or, if the status of `\undefined` isn't clear, they can use the test

```
\expandafter\ifx\csname laxread@\endcsname\relax
```

which is false when L_AM_S-T_EX has been loaded, but true when it hasn't been loaded, since the control sequence produced by `\csname... \endcsname` is given the value `\relax` if it hasn't already been defined.]

Next we redefine `\err@` from L_AM_S-T_EX to produce error messages saying 'LamS-TeX error:' instead of 'AmS-TeX error:'

```
\def\err@#1{\errmessage{LamS-TeX error: #1}}
```

`\Err@`, which is `\err@` with L_AM_S-T_EX's "default help message", will now also produce such error messages (see section 3.1).

As indicated in section 1.1, we redefine `^^L`, the `\new...` constructions, `\+`, and `\bye` from plain so that they are not `\outer`,

```
\def^^L{\par}
\let\+=\tabalign
\def\newcount{\alloc@0\count\countdef\insc@unt}
\def\newdimen{\alloc@1\dimen\dimendef\insc@unt}
\def\newskip{\alloc@2\skip\skipdef\insc@unt}
\def\newmuskip{\alloc@3\muskip\muskipdef\@cclvi}
\def\newbox{\alloc@4\box\chardef\insc@unt}
\let\newtoks=\relax
\def\newhelp#1#2{\newtoks#1#1\expandafter{\csname#2\endcsname}}
\def\newtoks{\alloc@5\toks\toksdef\@cclvi}
\def\newread{\alloc@6\read\chardef\sixt@@n}
\def\newwrite{\alloc@7\write\chardef\sixt@@n}
\def\newfam{\alloc@8\fam\chardef\sixt@@n}
\def\newlanguage{\alloc@9\language\chardef\@cclvi}
\def\newinsert#1{\global\advance\insc@unt by\m@ne
  \ch@ck0\insc@unt\count
  \ch@ck1\insc@unt\dimen
  \ch@ck2\insc@unt\skip
  \ch@ck4\insc@unt\box
  \allocationnumber=\insc@unt
  \global\chardef#1=\allocationnumber
  \wlog{\string#1=\string\insert\the\allocationnumber}}
```

```

\def\newif#1{\count@\escapechar \escapechar\m@ne
\expandafter\expandafter\expandafter
\edef\@if#1{true}{\let\noexpand#1=noexpand\iftrue}%
\expandafter\expandafter\expandafter
\edef\@if#1{false}{\let\noexpand#1=noexpand\iffalse}%
\@if#1{false}\escapechar\count@} % the condition starts out false

\def\bye{\par\vfill\supereject\end}

```

and then we

```
\let\beginsection=\undefined
```

to make `\beginsection` undefined (see `amstex.doc`).

bd In plain T_EX, the `\let\newtoks=\relax` is inserted before the definition of `\newhelp` so that `plain.tex` can be read in twice. Even though we are not allowing `lamstex.tex` to be read in twice, this is still required, since we read it in after `plain.tex`!

Chapter 3. Changes to $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$

The next part of $\mathcal{L}\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$ contains various changes to $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$. Some of the changes should, and probably eventually will, be made in $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$, though they didn't get included in $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$ version 2, while other changes are relevant only for $\mathcal{L}\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$. (Further changes to $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$ will be made later, at the relevant points.)

3.1. $\backslash\text{Err}\@$. $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$'s definition of $\backslash\text{Err}\@$,

```
 $\backslash\text{def}\backslash\text{Err}\@#1\{\backslash\text{errhelp}\backslash\text{default}\text{help}\@ \backslash\text{errmessage}\{\text{AmS-TeX error: }#1\}\}$ 
```

has been deleted in `amstex1.tex`, because it can obviously be shortened to

```
 $\backslash\text{def}\backslash\text{Err}\@#1\{\backslash\text{errhelp}\backslash\text{default}\text{help}\@ \backslash\text{err}\@ \{#1\}\}$ 
```

3.2. $\backslash\text{atdef}\@$. $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$'s original mechanism for defining the active $\@$ character can be improved considerably.

First of all, we want the active $\@$ to mean

```
 $\backslash\text{futurelet}\backslash\text{next}\backslash\text{at}\@$ 
```

the problem being that we need to make this definition while $\@$ is active

```
 $\{\backslash\text{catcode}\'\@=\backslash\text{active}\}$   
 $\backslash\text{def}\@ \{ \dots \}$ 
```

even though we want to allow $\@$ as part of the control sequence name $\backslash\text{at}\@$.

Now we can easily name $\backslash\text{at}\@$ even when $\@$ is active, as

```
 $\backslash\text{csname}\ \text{at}\backslash\text{string}\@ \backslash\text{endcsname}$ 
```

Of course, we can't simply say

```
 $\backslash\text{def}\@ \{ \backslash\text{futurelet}\backslash\text{next}\backslash\text{csname}\ \text{at}\backslash\text{string}\@ \backslash\text{endcsname} \}$ 
```

since @ would then simply \let\next=a—we need to have the combination \csname . . . \endcsname expanded out before the \futurelet\next takes effect.

We could do this with

```
\def@{\def\next{\futurelet\next}\expandafter\next
\csname at\string@\endcsname}
```

but that's somewhat unsatisfactory, since it requires the active @ to make a subsidiary definition each time it is used.

Another possibility is to use the triple \expandafter trick (see *The T_EXbook*, page 374), which we will be using later on. But for the present problem the simplest strategy is to use the code

```
\edef\next{\gdef\noexpand@{\futurelet\noexpand\next
\csname at\string@\endcsname}
\next
```

Here the \edef makes \next mean

```
\gdef|@|{\futurelet| \next| \at@}
```

where the boxed control sequences are not expanded out either because they are primitives or because they are preceded by \noexpand; the \csname . . . \endcsname is expanded out in the \edef, but the control sequence \at@ that it expands to is made equal to \relax, since \at@ hasn't been defined previously—so \at@ isn't expanded further in the \edef. (Here we are using the fact that lamstex.tex won't be read in twice [Chapter 2].)

Consequently, when we then call \next we get this \gdef. Thus, to get the desired definition of the active @ we just need

```
{\catcode'\@=\active
\edef\next{\gdef\noexpand@{\futurelet\noexpand\next
\csname at\string@\endcsname}}
\next
}
```

The definition of `\at@` itself is now easy, with `@` back as a letter. We will call the very same routine, `\at@@`, when the next token is a letter, other character, or control sequence (or active character); for any other type of token we will call `\at@@@`, which will be an error message:

```
\def\at@{%
  \ifcat\noexpand\next a\let\next@=\at@@\else
  \ifcat\noexpand\next0\let\next@=\at@@\else
  \ifcat\noexpand\next\relax\let\next@=\at@@\else
  \let\next@=\at@@@\fi\fi\fi\next@}
```

The error message `\at@@@` is simply

```
\def\at@@@{\errhelp\athelp@err@{Invalid use of @}}
```

using the help message `\athelp@` from *AMS-TEX*.

On the other hand, `\at@@` (token) will simply be the control sequence

`'\<token>@at'`

if it has been defined, or an error message otherwise. Here we put quotes around `\<token>@at` to emphasize that it is a single control word, even when `<token>` isn't a letter; in practice, of course, such control words have to be constructed using `\csname... \endcsname`:

```
\def\at@@#1{\expandafter
  \ifx\csname\string#1@at\endcsname\relax
  \let\next@=\at@@@
  \else
  \def\next@{\csname\string#1@at\endcsname}%
  \fi
  \next@}
```

Note that we use `\string#1` so that the token #1 can be a control sequence or active character, as well as a letter or other character.

Finally, `\atdef@`, the mechanism for defining the value of the active `@` on various tokens, is the same as in *AMS-TEX*, except that we add a `\string`:

```
\def\atdef@#1{\expandafter\def\csname\string#1@at\endcsname}
```

There are two noteworthy things about this redefinition:

- (1) The original definition of `\atdef@` remains in `amstex1.tex`, because it is used for

```
\atdef@;{...}      \atdef@,{...}
\atdef@:{...}      \atdef@!{...}
\atdef@?{...}      \atdef@.{...}
\atdef@-{...}
```

These `\atdef@`'s give the same results as the new `\atdef@` would give, since for these characters the `\string` is simply redundant—once `@` is active, `@;` and `@:` and so forth will work just as before. (The `\atdef@@\vert` is irrelevant: it is deleted in `amstex1.tex`, and not used in `LAMS-TEX`.)

- (2) Later in `LAMS-TEX` we are going to make `"` active. Nevertheless, `AMS-TEX`'s

```
\atdef@"
```

will still make the combination `@"` work correctly, because `\string"` for `"` active gives the same result as `AMS-TEX`'s `"` when `"` is not active. Actually, we are going to give a new `\atdef@"` (section 8), but that will be done before `"` is made active, so the same principle still applies.

3.3. Tests. `AMS-TEX` has the flag `\ifin@`, which is set only by the routine `\in@`, a test to determine whether a particular token appears in any sequence; this test, in turn, is used only by the routine `\tagin@` to check for the presence of `\tag` in a sequence. `LAMS-TEX` has numerous tests that are always performed independently, so it is economical to have a single flag that will be used by all of them; this flag will also replace `\ifin@` from `AMS-TEX`. So `amstex1.tex` deletes the line

```
\newif\ifin@
```

and in $\mathcal{L}\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$ we introduce the flag

```
\newif\iftest@
```

Moreover, in `amstex1.tex` we also delete

```
\def\in@#1#2{ . . . }
\def\tagin@#1{ . . . }
```

while in $\mathcal{L}\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$ we redefine the `\tagin@` routine so that instead of using `\ifin@` it merely reproduces (an equivalent of) the definition:

```
\def\tagin@#1{\tagin@false
\def\next@##1\tag##2##3\next@{\test@true
\ifx\tagin@##2\test@false\fi}%
\next@#1\tag\tagin@\next@
\tagin@false\iftest@\tagin@true\fi}
```

3.4. Spaces after control sequence names in error messages. Numerous error messages in $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$ use constructions of the form

```
. . . \string\controlseq\space . . .
```

to get a space after the control sequence `\controlseq` in the error message. However, it saves one token to instead use

```
. . . \noexpand\controlseq . . .
```

—the `\noexpand` prevents expansion of `\controlseq` in the error message, but we still get a space after `\controlseq`.

This device is used throughout $\mathcal{L}\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$, and the requisite changes were also made directly in `amstex1.tex`, since they were so minor.

bd The first change occurs in the definition of `\define@` where

```
\err@{\string\define\space must be . . . }
```

is replaced by

```
\err@{\noexpand\define must be . . . }
```

The next occurs in the definition

```
\define@@#1
```

which takes a control sequence #1 as its argument, where, for example

```
(A) \err@{\string#1 is already defined}
```

is changed to

```
\err@{\noexpand#1is already defined}
```

with no space before the 'is', since it will appear when the error message is given. [amstex.tex actually has

```
\err@{\string#1\space is already defined}
```

which is unnecessary complicated, though it has the same number of tokens as (A).]

Similar replacements are made in the definitions of

```
\vmodeerr@#1
\mathmodeerr@#1
\dmatherr@#1
\nondmatherr@#1
\onlydmatherr@#1
\nonmatherr@#1
\nonvmodeerr@#1
\textonlyfont@#1#2
```

Finally, in the definition of `\boldkey` (which is actually redefined in `lamstex.tex`—see page 31) the

```
\Err@{\string\boldkey\space can't ...}
```

is replaced by

```
\Err@{\noexpand\boldkey can't ...}
```

with a similar change for `\boldsymbol`.

Note, by the way, that these substitutions cannot be made in

```
\newhelp\athelp
\newhelp\defahelp
```

which end up putting things inside `\csname... \endcsname`.

3.5. Line breaking. The original $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$ definition of `\nolinebreak` had an extra element `\refskip@`, which was initially `\relax`, but which was changed for the bibliography. In version 2, that aspect of the bibliography macros (sections 29.3 and 29.11), as well as the indexing macros (section 38.3), has been improved. Consequently, the four control sequences

```
\nolinebreak
\allowlinebreak
\linebreak
\newline
```

will all have something added; it will suffice to add the same thing, which we will call `\lkerns@`, to the first three, and something that we will call `\nkerns@` to the fourth; like the old `\refskip@`, these are both initially `\relax`. The definitions of these four line-breaking macros are deleted in `amstex.tex`, and we now add the new definitions. They differ from the original definitions (see `amstex.doc`) in the inclusion of `\lkerns@` and `\nkerns@`; however, these do not occur in quite the place that `\refskip@` occurred in the original definition of `\nolinebreak`—that was, in fact, incorrect. We have also simplified the definition of `\newline`—as indicated in `amstex.doc`, the case of `\newline\par` really isn't worth worrying about.

```
\let\lkerns@=\relax

\def\nolinebreak{\relax
\ifmathmode@
\mathmodeerr@\nolinebreak\else
```

```

\ifhmode
\save skip@=\lastskip \unskip
\nobreak
\ifdim\save skip@ > Opt \hskip\save skip@\fi
\lkerns@
\else\vmodeerr@\nolinebreak\fi\fi}
\def\allowlinebreak{\relax
\ifmathmode@
\mathmodeerr@\allowlinebreak\else
\ifhmode
\save skip@=\lastskip \unskip
\allowbreak
\ifdim\save skip@ > Opt \hskip\save skip@\fi
\lkerns@
\else\vmodeerr@\allowlinebreak\fi\fi}
\def\linebreak{\relax
\ifmathmode
\mathmodeerr@\linebreak\else
\ifhmode
\unskip\unkern\break\lkerns@
\else\vmodeerr@\linebreak\fi\fi}

\let\nkerns@=\relax

\def\newline{\relax
\ifmathmode
\mathmodeerr@\newline\else
\ifhmode
\unskip\unkern\null\hfill\break\nkerns@
\else\vmodeerr@\newline\fi\fi}%

```

3.6. `\alloc@`, `\newcount@`, and `\newbox@`. Near the beginning of *AMS-TeX*, `\alloc@` is redefined so that it doesn't write anything to the `.log` file, while the original definition is reinstated at the end. This means that the `\new...` constructions used to create new counters, (dimen) registers, etc., within the file do not write anything to the `.log` file.

After redefining $\backslash\text{alloc@}$, $\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\mathcal{T}\mathcal{E}\mathcal{X}$ also uses

```
\let\alloc@@=\alloc@
```

to make $\backslash\text{alloc@@}$ *that* version of $\backslash\text{alloc@}$, even if called after $\backslash\text{alloc@}$ has been redefined at the end. In the definition of $\backslash\text{loadmsam}$, for example, the code

```
. . .
\alloc@@8\fam\chardef\sixt@@n\msafam
. . .
```

functions as a replacement for $\backslash\text{newfam}\backslash\text{msafam}$; this not only gets around the problem that $\backslash\text{newfam}$ is still $\backslash\text{outer}$ in $\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\mathcal{T}\mathcal{E}\mathcal{X}$, it also ensures that nothing gets written to the .log file even if $\backslash\text{loadmsam}$ is used after $\backslash\text{alloc@}$ is restored to its old definition.

In the definition of $\backslash\text{accentedsymbol}$, however, a non-outer $\backslash\text{newbox}$ is needed, because it appears in a construction like

```
\expandafter\newbox\csname ... \endcsname
```

where we can't simply use the code for $\backslash\text{newbox}$. So amstex.tex used $\backslash\text{newbox@}$ as a non-outer version of $\backslash\text{newbox}$. However, the $\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\mathcal{T}\mathcal{E}\mathcal{X}$ definition,

```
\def\newbox@{\alloc@4\box\chardef\insc@unt}
```

because it used $\backslash\text{alloc@}$ rather than $\backslash\text{alloc@@}$, wasn't really the right choice anyway.

In $\mathcal{L}\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\mathcal{T}\mathcal{E}\mathcal{X}$ we rectify this situation. First of all, the definitions of $\backslash\text{newbox@}$ and $\backslash\text{accentedsymbol}$ are deleted in amstexl.tex . Then in lamstex.tex , we

```
\def\newbox@{\alloc@@4\box\chardef\insc@unt}
```

and also, for later use,

```
\def\newcount@{\alloc@@@0\count\countdef\insc@unt}
```

Then we redefine `\accentedsymbol` using `\newbox@`. In addition, the combination

```
\expandafter\eat@\string
```

in the definition is replaced by `\exstring@`, since we will introduce this as an abbreviation for that combination later (section 17.1) (we also take the opportunity to eliminate two unnecessary `\expandafter`'s from the original code):

```
\def\accentedsymbol#1#2{\expandafter
\newbox@\csname\exstring@#1@box\endcsname
\setbox\csname\exstring@#1@box\endcsname
=\hbox{${\m@th#2$}}%
\define#1{\copy\csname\exstring@#1@box\endcsname{}}}
```

3.7. Lists. $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$ has two lists, `\alloclist@`, and `\fontlist@`, of the type introduced in *The $\mathcal{T}\mathcal{E}\mathcal{X}$ book*, page 378, and it defines `\rightappend@`, which is like `\rightappenditem` from that page.

`\alloclist@` is maintained for the `\showallocations` command, which is really only for the use of $\mathcal{T}\mathcal{E}\mathcal{X}$ nicians, and $\mathcal{L}\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$ dispenses with this feature, in order to save space; consequently, in `amstexl.tex` all material related to `\alloclist@` is deleted.

`\fontlist@` is used for the `\syntax` command, which $\mathcal{L}\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$ retains, but for this list of control sequence names it is more efficient to use a list of the type described on page 379 of *The $\mathcal{T}\mathcal{E}\mathcal{X}$ book*:

```
\\name_1\\name_2...
```

Several other lists of this sort will be used in $\mathcal{L}\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$; in some cases, we will even have a list of the form `_ _ _ _ _ _ ...`, where `_ _ _` are not control sequence names.

In $\mathcal{L}\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$ we will still be using `\rightappend@` on occasion, but we will also define the routine `\rightadd@#1\to#2` to add #1 to one of these simpler lists #2:

```
\def\rightadd@#1\to#2{\toks@={\#1}\toks@=\expandafter{#2}%
\xdef#2{\the\toks@\the\toks@}%
\toks@={}\toks@={}}
```

In `amstexl.tex`, the definition of `\fontlist@` is deleted, and in $\mathcal{L}\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\mathcal{T}\mathcal{E}\mathcal{X}$ we instead define `\fontlist@` to be a list of this simpler sort:

```
\def\fontlist@{\tenrm\sevenrm\fiverm\teni\seveni
  \fivei\tensy\sevensy\fivesy\tenex\tenbf
  \sevenbf\fivebf\tensl\tenit}
```

Similarly, the definition of `\font@` is deleted in `amstexl.tex` and replaced in $\mathcal{L}\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\mathcal{T}\mathcal{E}\mathcal{X}$ by

```
\def\font@#1=#2 {\rightadd@#1\to\fontlist@\font#1=#2 }
```

(Although `\font@` appears in `amstexl.tex`, it occurs only within other definitions, so the definition can be deferred to `lamstex.tex`.)

Although, as mentioned in section 3, the test `\ifin@` has been deleted in `amstexl.tex`, for a list #1 of control sequences,

```
\\name1\\name2...
```

we will need another (simpler) test to determine whether a control sequence #2 is in the list. Basically we want to use

```
\def\ismember@#1#2{\test@false\let\next@=#2%
  {\def\\#1{\let\nextii@=#1\ifx\nextii@\next@\global\test@true\fi}#1}%
```

But since we normally set the flag `\iftest@` only locally, we don't want to use a `\global\test@true` in this one situation (compare section 1.3). So instead we will use a new scratch token, `\Next@`, for which we will always use `\global` assignments. In addition, there are two further details that we will explore in a moment:

```
\def\ismember@#1#2{\global\let\Next@=F\let\next@=#2%
  {\def\\#1{\let\nextii@=#1\ifx\nextii@\next@
    \global\let\Next@=T\fi}#1}%
  \test@false\ifx\Next@ T\test@true\fi\let\next@=\relax}
```

This test may be compared with the test on page 379 of *The T_EXbook*. Using `\let\next@` instead of `\def\next@` allows the test

```
\ismember@#1\next
```

to be used after `\next` has been `\let` equal to some control sequence by a `\futurelet`, which will be important in section 7.2 (on some occasions we will also be using `\ismember@#1#2` when `#2` is an explicit argument). But two precautions are then in order:

- Although actual `lamstex.tex` code usually omits `=` signs after `\let`'s, in the above code we need both the boxed `=` sign and the space after it! Reason: Our `\futurelet` may have `\let\next@` be a space token, which is thus a `<space token>` in the notation of *The T_EXbook*, page 269. According to the syntax rule on page 277, one such space (but only one) will be ignored after the `=` sign; if we simply had

```
\let\next@=#2{\def\ \ ...
```

then the `<space token>` `#2` would be ignored, and `\next@` would end up being the `{`, which would then disappear, causing infinite confusion later on.

- Another important precaution is the `\let\next@=\relax` at the end. That is needed because the `\futurelet\next@` may have `\let\next@` equal something equivalent to `\iftrue` or `\iffalse`, so that `\next@` would then also be equal to `\iftrue` or `\iffalse`. If that situation were allowed to continue, havoc might ensue the next time we used a macro containing `\next@` within it.

As an example of this latter phenomenon, note that the original plain T_EX definitions

```
{\catcode'\'= \active \gdef'\{^\bgroup\prim@s}}
\def\prim@s{\prime\futurelet\next\prim@m@s}
\def\pr@m@s{\ifx'\next\let\next\pr@@@s \else\ifx^\next\let\next\pr@@@t
\else\let\next\egroup\fi\fi\next}
...

```

later had to be modified by changing `\pr@ms` to

```
\def\pr@ms{\ifx'\next\let\nxt\pr@@s \else\ifx'\next\let\nxt\pr@@t
\else\let\nxt\egroup\fi\fi\nxt}
```

For example, we might have

```
$a'\iffirstset x\else y \fi$
```

where `\iffirstset` is some user-defined construction, and then the `\futurelet\next` in `\prim@s` would `\let\next=\iffirstset`. Note that the appearance of `\next` after an `\ifx` test causes no problem, but its appearance within an `\if...` clause, even following a `\let` or `\def`, would make things go haywire. Similarly, in the definition of `\ismember@`, the `\next@` appears in safe places.

To avoid such problems in general, after any `\futurelet` we will use only the token `\next`, and otherwise `\next` will not appear in any macros except after `\ifx` tests (or `\ifcat` tests). One definition in `amstex.tex` requires modification to adhere to this rule: the definition

```
\gdef\comment@@@#1\comment@@@{\ifx\next\comment@@@\let\next\comment@
\else\def\next{\oldcodes@\endlinechar='\^M\relax}%
\fi\next}
```

has been changed in `amstex1.tex` to

```
\gdef\comment@@@#1\comment@@@{\ifx\next\comment@@@\let\next@\comment@
\else\def\next@{\oldcodes@\endlinechar='\^M\relax}%
\fi\next@}
```

3.8. Skipping spaces in `\futurelet`'s. On page 7 we mentioned $\mathcal{A}\mathcal{M}\mathcal{S}$ -TEX's device for skipping over space tokens in `\futurelet` constructions. Instead of using this device directly, which requires somewhat long definitions each time, $\mathcal{A}\mathcal{M}\mathcal{S}$ -TEX uses a special "futurelet-next-skipping-spaces" construction

```
\FNSS@\foo
```

which is like `\futurelet\next\foo`, except that any space tokens after `\foo` will be discarded, and `\foo` will be applied after `\next` has been `\let` equal to the first non-space token after `\foo`. `\FNSS@#1` begins by storing `#1` in `\FNSS@@`, and then applies a `\futurelet\next` construction, calling `\FNSS@@@`, which then does the checking for a space:

```
\def\FNSS@#1{\let\FNSS@@=#1\futurelet\next\FNSS@@@}
\def\FNSS@@@{\ifx\next\space@
\def\FNSS@@@. {\futurelet\next\FNSS@@@}\else
\def\FNSS@@@. {\FNSS@@}\fi
\FNSS@@@.}
```

Thus, when `\next` happens to be a space, we swallow the space and call the routine `\futurelet\next\FNSS@@@` again, to get the next non-space token; when we do get a non-space token, we simply apply `\FNSS@@`, the argument of `\FNSS@`.

The $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$ `\atdef@` (see `amstex.doc`) is deleted from `amstex1.tex`, because it can be shortened if we use this `\FNSS@` to get the first non-space token after `@`:

```
\atdef@{\unskip
\def\next@{\ifx\next'\def\next@'\{\futurelet\next\nextii@}%
\else\ifx\next\lq\def\next@\lq{\futurelet\next\nextii@}%
\else\def\next@###1{\futurelet\next\nextiii@}\fi\fi
\next@}%
\def\nextii@{\ifx\next'\def\next@'\{\sldl@' '%}
\else\ifx\next\lq\def\next@\lq{\sldl@' '%}
\else\def\next@{\dls1@'}\fi\fi\next@}%
\def\nextiii@{\ifx\next'\def\next@'\{\srd@'' '%}
\else\ifx\next\rq\def\next@\rq{\srd@'' '%}
\else\def\next@{\drs@'}\fi\fi\next@}%
\FNSS@\next@}
```

[In our definition of `\FNSS@@@` we used a new control sequence `\FNSS@@@` instead of using a scratch token like `\next@` to allow the use of `\FNSS@` in such “compressed format” definitions.]

There is only one other definition in $\mathcal{A}\mathcal{M}\mathcal{S}$ -TEX that is (deleted from `amstexl.tex` and) shortened using `\FNSS@`:

```

\def\root{%
  \def\next@{\ifx\next\uproot\let\next@=\nextii@\else
    \ifx\next\leftroot\let\next@=\nextiii@\else
    \let\next@=\plainroot@fi\next@}%
  \def\nextii@\uproot##1{\uproot@##1\relax\FNSS@\nextiv@}%
  \def\nextiv@\ifx\next\leftroot\let\next@=\nextv@\else
    \let\next@=\plainroot@fi\next@}%
  \def\nextv@\leftroot##1{\leftroot@##1\relax\plainroot@}%
  \def\nextiii@\leftroot##1{\leftroot@##1\relax
    \FNSS@\nextvi@}%
  \def\nextvi@\ifx\next\uproot\let\next@=\nextvii@\else
    \let\next@=\plainroot@fi\next@}%
  \def\nextvii@\uproot##1{\uproot@##1\relax\plainroot@}%
  \bgroup\uproot@z@\leftroot@z@
\FNSS@\next@}

```

However, there are numerous places in $\mathcal{L}\mathcal{A}\mathcal{M}\mathcal{S}$ -TEX where the use of `\FNSS@` will similarly save space, definitely make the extra tokens used for `\FNSS@` worth while. Note, moreover, that the above definition of `\root` not only saves space, but also avoids introducing the scratch tokens `\nextviii@` and `\nextix@` that occur in the original definition, but which occur nowhere else in $\mathcal{A}\mathcal{M}\mathcal{S}$ -TEX.

3.9. `\loop`. The same article that introduced the “K-method” (page 4) also introduced a new definition of plain TEX’s `\loop... \repeat` mechanism, which we will use in $\mathcal{L}\mathcal{A}\mathcal{M}\mathcal{S}$ -TEX:

```

\def\loop#1\repeat{%
  \def\iterate{#1\relax\expandafter\iteratefi}%
  \iterate\let\iterate=\relax}

```

This has the property that it allows constructions like

```
\loop --- \if... --- \else --- \repeat
```

where we repeat when the `\if...` test is *false* rather than true, which will turn out to be useful at several points in \LaTeX .¹

Corresponding to this redefinition of `\loop`, we add a new definition of `\gloop@` (which is used by the `\cfrac` construction):

```
\def\gloop@#1\repeat{%
  \gdef\iterate@{#1\relax\expandafter\iterate@\fi}%
  \iterate@\global\let\iterate@=\relax}
```

The main purpose of redefining `\gloop@` is so that it will use `\iterate@` rather than `\iterate`, so that `\iterate` will only be given a local definition, not both a local and a global one (compare section 1.3).

In `amstexl.tex` we delete the line

```
\newif\ifbadans@
```

and the definition of `\printoptions`, because we can define `\printoptions` in terms of `\iftest@` (section 3). We've deferred this redefinition until now because it is also a little bit more convenient to use this alternative `\loop` test:

```
\def\printoptions{W@{Do you want S(yntax check),
  G(alleys) or P(ages)?^^JType S, G or P, follow by
  <return>: }%
\loop
  \read -1 to\ans@
  \edef\next@{\def\noexpand\Ans@{\ans@}}%
  \uppercase\expandafter{\next@}%
  \ifx\Ans@\S@\global\test@true\syntax\else
  \ifx\Ans@\G@\global\test@true\galley\else
  \ifx\Ans@\P@\global\test@true\else
  \global\test@false\fi\fi\fi
  \iftest@
  \else
  \W@{Type S, G or P, follow by <return>: }%
  \repeat}
```

¹The alternative is to use `\if... \test@false \else \test@true \fi \iftest@ --- \repeat`.

3.10. *A la français.* Finally, certain changes are made to `amstex.tex` to accommodate French styles that make some or all of ; and : and ! and ? into active characters. In this case, various $\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\mathcal{T}\mathcal{E}\mathcal{X}$ macros that involve tests like

```
\ifx\next!
```

need to be changed.¹

Our goal is to allow all necessary changes to be indicated in a reasonably short file, say `french.tex`, which can be read in *after* `lamstex.tex` (so that it can be loaded on top of a $\mathcal{L}\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\mathcal{T}\mathcal{E}\mathcal{X}$ format file).

The most reasonable approach is to have certain control sequences that have been `\let` equal to the active punctuation symbols, so that we can use these in an `\ifx\next...` test. Of course, this can only be done after the active punctuation symbols have been defined, not in `amstex1.tex`. To get around this problem, we will insist that the definition of the active punctuation symbols in `french.tex` are not made with an ordinary `\def`, but with a special `\APdef`, which will manage things properly for us.

For the control sequences that will be `\let` equal to the active punctuation symbols we will use the following control words (which have to be created using `\csname... \endcsname`): `'\A@;`' and `'\A@:'` and `'\A@?'` and `'\A@!'`. We begin by assigning the non-active characters as default values:

```
\expandafter\let\csname A@;\endcsname=;
\expandafter\let\csname A@:\endcsname=:
\expandafter\let\csname A@?\endcsname=?
\expandafter\let\csname A@!\endcsname=!
```

When

```
\APdef:{ ... }
```

¹In addition, in `lamstex.tex` we have the problem that certain control sequences, notably those for commutative diagrams and tables, use certain punctuation as part of their *syntax*. For example, the `\ds` option for arrows in a commutative diagram is typed in the form `\ds(h;v)` (see page 155 of the $\mathcal{L}\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\mathcal{T}\mathcal{E}\mathcal{X}$ Manual). If we make a definition like `\def\ds(#1;#2){...}`, then $\mathcal{T}\mathcal{E}\mathcal{X}$ incorporates a type 12 ; as part of the syntax for `\ds`. So in a document where ; is active, the ; that the user types will not be recognized as the proper syntax element. The devices for handling this problem will be discussed at the appropriate time (in Volume 2).

appears in `french.tex`, we want to (1) `\def:{ ... }` and (2) `\let'\A@:'=:` (here, as on page 14, we put quotes around `\A@:` to emphasize that it is a single control word; in actual code something like

```
\expandafter\let\csname A@\string:\endcsname=:
```

will be needed).

To achieve this, we use

```
\def\APdef#1{\def\next@{\expandafter
\let\csname A@\string#1\endcsname=#1}%
\afterassignment\next@\def#1}
```

Thus, for example, `\APdef:` defines `\next@` to mean

```
\let'\A@:'=:
```

and after the assignment `\def:`, which swallows the following `{ ... }`, we perform `\next@`, so that `'\A@:'` has now been `\let` equal to the active `:`.

So, we can produce control sequences that have the value of each active punctuation symbol that may occur in a file, assuming that `\APdef` has always been used in `french.tex`.

Now consider the original $\mathcal{A}\mathcal{M}\mathcal{S}$ -TEX definition

```
\def\tdots@{\unskip
\def\next@{${\m@th\mathinner{\ldotp\ldotp\ldotp}}\,
\ifx\next,\,$\else\ifx\next.\,$\else
\ifx\next;\,$\else
\ifx\next:\,$\else
\ifx\next?,$\else
\ifx\next!,$\else
$ \fi\fi\fi\fi\fi\fi}%
\ \futurelet\next\next@}
```

We want to supplement this with `\ifx\next` tests that check whether `\next` is an active punctuation symbol, in case `\tdots@` gets used in a file where that is the case. We can do this with tests like

```
\expandafter\ifx\csname A@\string;\endcsname\next
```

For greater flexibility, when we encounter an active punctuation symbol we will not necessarily insert the extra `\`, that the non-active symbol gets; we will instead insert `\fextra@`, which by default is

```
\let\fextra@=\,
```

(but which `french.tex` can redefine, if desired).

The original definition of `\tdots@` is deleted from `amstexl.tex`, and a new definition is given in `lamstex.tex`:

```
\def\tdots@{\unskip
\def\next@{${\m@th\mathinner{\ldotp\ldotp\ldotp}}\,
\ifx\next,\,$\else\ifx\next.\,$\else
\ifx\next;\,$\else
\expandafter\ifx\csname A@\string;\endcsname\next
\fextra@$}\else
\ifx\next:\,$\else
\expandafter\ifx\csname A@\string;\endcsname\next
\fextra@$}\else
\ifx\next?,\,$\else
\expandafter\ifx\csname A@\string?\endcsname\next
\fextra@$}\else
\ifx\next!\,$\else
\expandafter\ifx\csname A@\string!\endcsname\next
\fextra@$}\else
$ \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi}%
\ \futurelet\next\next@}
```

Similarly, the definition of `\extrap@` is deleted from `amstexl.tex`, and in `lamstex.tex` we add

```
\def\extrap@#1{%
\ifx\next,\def\next@{#1\,}\else
\ifx\next;\def\next@{#1\,}\else
\expandafter\ifx\csname A@\string;\endcsname\next
\def\next@{#1\fextra@}\else
```



```

\ifx\next.\def\next@{#1\,}\else\extra@
\ifextra@\def\next@{#1\,}\else
\let\next@=#1\fi\fi\fi\fi\fi\next@}

```

The boxed = sign in this code is another case (compare page 23) where the = sign cannot be omitted, though now the reason is different: we might have #1 being an = sign! There are similar required =’s in the (original) definitions of `\boldsymbol@`, `\boldkeydots@`, and `\boldsymboldots@`, and in the new definition of `\boldkey`, to follow. These = signs and the one in the definition of `\ismember@` are the only ones that actually occur after a `\let` in `lamstex.tex`.

Similarly, the definitions of `\dotsc` and `\keybin@` are deleted from `amstexl.tex`, and in `lamstex.tex` we add

```

\def\dotsc{\def\next@{\ifx\next;\plainldots@\,}\else
\expandafter\ifx\csname A@\string;\endcsname\next
\plainldots@\fextra@\else
\ifx\next.\plainldots@\,}\else\extra@\plainldots@
\ifextra@\,}\fi\fi\fi\fi}%
\futurelet\next\next@}

\def\keybin@{\keybin@true
\ifx\next+\else\ifx\next=\else\ifx\next<\else
\ifx\next>\else\ifx\next-\else\ifx\next*\else
\ifx\next:\else
\expandafter\ifx\csname A@\string;\endcsname\next\else
\keybin@false\fi\fi\fi\fi\fi\fi\fi\fi}

```

The definition of `\boldkey#1` is also deleted from `amstexl.tex`. In this case, where we have a control sequence with an argument, rather than one that has picked up the next token with a `\futurelet\next`, and where our tests are of the form

```
\ifx#1!
```

we have to start with

```
\let\next=#1
```

so that we can then use things like

```
\expandafter\ifx\csname A@\string!\endcsname\next
```

to check for an active !. The new code is:

```
\def\boldkey#1{\ifcat\noexpand#1A%
  \ifcmmibloaded@{\fam\cmmibfam#1}\else
  \Err@{First bold symbol font not loaded}\fi
\else
\let\next=#1%
\ifx#!\mathchar"5\bfam@21 \else
\expandafter\ifx\csname A@\string!\endcsname\next
\mathchar"5\bfam@21 \else
\ifx#1(\mathchar"4\bfam@28 \else
\ifx#1)\mathchar"5\bfam@29 \else
\ifx#1+\mathchar"2\bfam@2B \else
\ifx#1:\mathchar"3\bfam@3A \else
\expandafter\ifx\csname A@\string:\endcsname\next
\mathchar"3\bfam@3A \else
\ifx#1;\mathchar"6\bfam@3B \else
\expandafter\ifx\csname A@\string;\endcsname\next
\mathchar"6\bfam@3B \else
\ifx#1=\mathchar"3\bfam@3D \else
\ifx#1?\mathchar"5\bfam@3F \else
\expandafter\ifx\csname A@\string?\endcsname\next
\mathchar"5\bfam@3F \else
\ifx#1[\mathchar"4\bfam@5B \else
\ifx#1]\mathchar"5\bfam@5D \else
\ifx#1,\mathchari@63B \else
\ifx#1-\mathchari@200 \else
\ifx#1.\mathchari@03A \else
\ifx#1/\mathchari@03D \else
\ifx#1<\mathchari@33C \else
\ifx#1>\mathchari@33E \else
\ifx#1*\mathchari@203 \else
```


Chapter 4. Numbering styles

Next we define the standard \LaTeX numbering styles: `\arabic`, `\alph`, `\Alph`, `\roman`, `\Roman`, and `\fnsymbol`.

These numbering styles are meant to be applied to a *number*, not to a counter (in the \LaTeX macros they are usually applied to `\number{counter}` for some `counter`). `\arabic` is consequently quite trivial:

```
\def\arabic#1{#1}
```

The definition of `\alph#1` uses the fact that the lower-case letters a–z have positions 97–122 (if some one were to design a perverse font for which this wasn't true, then `\alph` would have to be defined differently).

`\alph` begins by setting the scratch counter `\count@` to the value of `#1`:

```
\def\alph#1{\count@=#1\relax
```

We always add the `\relax` as a precaution in such situations, even if it may not be strictly necessary (*The TeXbook*, page 208, recommends a space, but `\relax` seems best, to avoid any anomalous situations where an unwanted space might somehow intrude itself into the token stream).

Then we add 96 to the value of `\count@` and we print the character `\char\count@`, but we give an error message if this would give us a character past 122 (this could easily happen if some one caused the counter to be augmented more than 25 times):

```
\def\alph#1{\count@=#1\relax\advance\count@ by 96
\ifnum\count@>122 \Err@{\noexpand\alph invalid for
numbers > 26}\else\char\count@\fi}
```

(See section 3.4 for the use of `\noexpand\alph` rather than `\string\alph`.)

We don't bother giving an error message if `\count@` ends up having a value less than 97, since reasonable macros will normally start the counter for a particular construction at 1. (So there is the possibility that some one will perversely `\Reset` some construction to 0, say, and then have `\alph` produce a left quotation mark '!' If this really seems bothersome, another check can easily be added to the code.)

The definition

```
\def\Alph#1{\count@=#1\relax\advance\count@ by 64
  \ifnum\count@>90 \Err@{\noexpand\Alph invalid for
    numbers > 26}\else\char\count@\fi}
```

is exactly analogous, using the fact that the letters A–Z should have the positions 65–90.

For `\roman` we just have to

```
\def\roman#1{\romannumeral#1\relax}
```

since TEX provides the `\romannumeral` primitive. (The `\relax` is essential here; otherwise something like `\roman{10}3` would be expanded into `\romannumeral103`, giving the result ‘cii’.)

But `\Roman` is more complicated,

```
\def\Roman#1{\uppercase\expandafter{\romannumeral#1}}
```

Here the `\expandafter` causes expansion after the left brace `{`. Without the `\expandafter`, something like `\Roman3` would become

```
\uppercase{\romannumeral3}
```

which is just `\romannumeral3`, and thus ‘iii’. With the `\expandafter`, we get

```
\uppercase{(expansion of \romannumeral3)}
```

i.e., `\uppercase{iii}`, or ‘III’.

Note that we don’t need to add `\relax` after the `#1` in his definition, because `\romannumeral#1` is expanded within the `\uppercase{...}`.

Finally, we have the `\fnsymbol` numbering style. As the code below shows, `\fnsymbol#1` successively sets

- (1) $\backslash\text{count@@} = \#1$
- (2) $\backslash\text{count@} = \left\lfloor \frac{\#1 - 1}{7} \right\rfloor$
- (3) $\backslash\text{count@@@} = \left\lfloor \frac{\#1 - 1}{7} \right\rfloor + 1$
- (4) $\backslash\text{count@@} = \#1 - 7 \cdot \left\lfloor \frac{\#1 - 1}{7} \right\rfloor$.

Hence

$$\backslash\text{count@@} = \begin{cases} 1 & \text{for } \#1 = 1, 8, 15, \dots \\ 2 & \text{for } \#1 = 2, 9, 16, \dots \\ \dots & \dots \\ 7 & \text{for } \#1 = 7, 14, 21, \dots \end{cases}$$

so `\count@@` tells which of the symbols `*`, `†`, `‡`, `¶`, `§`, `||`, `#` should be printed, namely, `*` for `\count@@ = 1`, `†` for `\count@@ = 2`, etc.

On the other hand,

$$\backslash\text{count@@@} = \begin{cases} 1 & \text{for } \#1 = 1, \dots, 7 \\ 2 & \text{for } \#1 = 8, \dots, 14 \\ \dots & \dots \end{cases}$$

so `\count@@@` is the number of times that this symbol has to be printed. To print the symbol this many times, we use a loop, initially setting the counter `\count@` to `\count@@@`:

```

\def\fnsymbol#1{\count@=#1\relax
\count@@=\count@
\advance\count@ by -1 \divide\count@ by 7
\count@@@=\count@ \advance\count@@@ by 1
\multiply\count@ by 7 \advance\count@@ by -\count@
\count@=\count@@@
{\loop
\ifcase\count@@\or*\or\dag\or\ddag\or\P\or\S\or
\text{\$|\$}\or\#\fi
\advance\count@ by -1 \ifnum\count@>0 \repeat}}

```

\LaTeX has already defined \dag , \ddag , \P , and \S , so that they can be used either in text or in math mode, where they will change size properly; $\text{\text{\$}\|\$}$ simply does the same for the $\|$ symbol (\# needs no special treatment). Modifications of \dag , ... may be necessary for fonts with different layouts.

In the above definition we put the \loop inside a group just in case \fnsymbol happens to be used inside some \loop itself, a somewhat finicky precaution, since a nested \loop involving \fnsymbol will never be produced by any \LaTeX construction. The same precaution will be used for any \LaTeX \loop that might conceivably occur within another \loop .

Chapter 5. Printing cardinal and ordinal numbers

This chapter is in some sense a companion to the previous one.

First we define `\cardnine@#1` for printing the final part of the name of a cardinal number; it will be applied to a counter with a value from 1, ..., 9:

```
\def\cardnine@#1{\ifcase#1\or one\or two\or  
three\or four\or five\or six\or seven\or eight\or nine\fi}
```

The number 10 will be used so often in the macros of this chapter that we want to use `\newcount` to introduce a counter having that value.

As we mentioned in section 3.6, `amstex.tex` (and `amstex1.tex`) redefine `\alloc@` so that it doesn't write anything to the `.log` file; and

```
\let\alloc@@=\alloc@
```

is used so that `\alloc@@` is *that* version of `\alloc@`, even if called after `\alloc@` has been restored to the old definition from plain `TEX` at the end. We now want to reinstate the new definition until the end of `lamstex.tex`. Instead of redefining it directly, we just have to

```
\let\alloc@=\alloc@@
```

since `\alloc@@` was permanently given the new definition.

Since `\newcount` is defined in terms of `\alloc@`, which we have just made equivalent to `\alloc@@`, the next code

```
\newcount\ten@  
\ten@=10
```

does not produce anything in the `.log` file of a file that has `\input lamstex` (nor will any of the other `\new...` constructions to follow).

Although `\cardinal{...}` will normally be applied when `...` is a number, to be on the safe side, we first safely store the value in a counter,

```
\def\cardinal#1{\count@=#1\relax
```


and then proceed by cases.

```

\def\cardinal#1{\count@=#1\relax
\ifnum\count@>99 \number\count@
\else
\ifnum\count@=0 zero%
\else
\ifnum\count@<\ten@ \cardnine@\count@
\else
\ifnum\count@<20
\advance\count@ by -\ten@
\ifcase\count@ ten\or eleven\or twelve\or
thirteen\or fourteen\or fifteen\or sixteen\or
seventeen\or eighteen\or nineteen\fi
\else
\count@@=\count@ \count@@@=\count@@
\divide\count@ by \ten@ \multiply\count@ by \ten@
\advance\count@@@ by -\count@
\divide\count@ by \ten@
\ifcase\count@\or\or twenty\or thirty\or forty
\or fifty\or sixty\or seventy\or eighty\or ninety\fi
\ifnum\count@@@=0 \else-\cardnine@\count@@@\fi
\fi
\fi
\fi
\fi}

```

Thus, if $\#1 > 99$, we simply typeset this number. If $\#1 = 0$, we simply typeset 'zero'. If $0 < \#1 < 10$, we just use `\cardnine@` to print the number. For $10 < \#1 < 20$, we must explicitly specify 'ten', ..., 'nineteen'.

Finally, for $20 < \#1 < 99$, we have to do a little calculation. The next code successively sets

$$(1) \quad \text{\count@@@} = \text{\count@@} = \#1$$

$$(2) \quad \text{\count@} = 10 \cdot \left\lfloor \frac{\#1}{10} \right\rfloor$$

$$(3) \quad \begin{aligned} \backslash\text{count}@@@ &= \#1 - 10 \cdot \left[\frac{\#1}{10} \right] \\ &= \#1 \pmod{10} \end{aligned}$$

$$(4) \quad \backslash\text{count}@ = \left[\frac{\#1}{10} \right]$$

The first part of the word for #1 is ‘twenty’ if #1 = 20, ..., 29, and hence $\backslash\text{count}@=2$; it is ‘thirty’ if #1 = 30, ..., 39, and hence $\backslash\text{count}@=3$; etc. If $\backslash\text{count}@@@ = 0$ the name is complete; otherwise we must add ‘-one’, ‘-two’, ..., depending on the value of $\backslash\text{count}@@@$.

$\backslash\text{ordnine}@$, for ordinal numbers, is exactly analogous to $\backslash\text{cardnine}@$:

```
\def\ordnine@#1{\ifcase#1\or first\or second\or
third\or fourth\or fifth\or sixth\or seventh\or
eighth\or ninth\fi}
```

The ordinal numbers < 100 can be treated in a manner exactly like the cardinal numbers. But a problem arises for ordinal numbers like ‘100th’, ‘101st’, ‘102nd’, ‘103rd’, ..., ‘109th’, ‘110th’, ‘111th’, ‘112th’, ...—now the proper suffix depends not only on the last digit of the number, but also on whether the next-to-last digit is a 1.

The routine $\backslash\text{ordsuffix}@$ selects the right suffix in these cases, assuming that the number in question has been stored in $\backslash\text{count}@$. The first part of the routine below divides $\backslash\text{count}@$ by 10, and then calculates $\backslash\text{count}@@@$ to be $\backslash\text{count}@ \pmod{10}$ [compare the previous calculations for $\backslash\text{cardinal}$]. If this is 1, so that the next-to-last digit of the number in $\backslash\text{count}@$ was 1, ‘th’ is selected. Because the original value of $\backslash\text{count}@$ is needed for the second part of the routine, we need to store it in yet another counter, $\backslash\text{count}@@@$, which we have to declare.

This second part simply computes $\backslash\text{count}@@@$ as $\backslash\text{count}@ \pmod{10}$ [for the original value of $\backslash\text{count}@$]; if this second part of the routine ends up being invoked, the correct choice of the suffix depends only on this value of $\backslash\text{count}@@@$:

```
\newcount\count@@@
```

```

\def\ordsuffix@{\count@@@=\count@
\divide\count@ by \ten@
\count@@=\count@ \count@@=\count@
\divide\count@@ by \ten@ \multiply\count@@ by \ten@
\advance\count@@@ by -\count@@
\ifnum\count@@@=1 th%
\else
\count@@@=\count@@@
\count@@=\count@@@
\divide\count@@ by \ten@ \multiply\count@@ by \ten@
\advance\count@@@ by -\count@@
\ifcase\count@@@ th\or st\or nd\or rd\else th\fi
\fi}

```

`\nordinal` and `\spordinal` are easy, since they are simply a number followed by the proper suffix, or the suffix superscripted:

```

\def\nordinal#1{\count@=#1\relax\number\count@\ordsuffix@}
\def\spordinal#1{\count@=#1\relax\number\count@
  $\text{\ordsuffix@}$}

```

And `\ordinal` itself simply mimics `\cardinal`, using `\ordsuffix@` for numbers ≥ 100 :

```

\def\ordinal#1{\count@=#1\relax
\ifnum\count@>99 \number\count@\ordsuffix@
\else
\ifnum\count@=0 zeroth%
\else
\ifnum\count@<\ten@ \ordnine@\count@
\else
\ifnum\count@<20 \advance\count@ by -\ten@
\ifcase\count@ tenth\or eleventh\or twelfth\or
thirteenth\or fourteenth\or fifteenth\or sixteenth\or
seventeenth\or eighteenth\or nineteenth\fi
\else

```

```
\count@=\count@
\divide\count@ by \ten@ \multiply\count@ by \ten@
\count@@=\count@ \advance\count@@ by -\count@
\divide\count@ by \ten@
\ifcase\count@\or\or twent\or thirt\or fort\or fift\or
sixt\or sevent\or eight\or ninet\fi
\ifnum\count@@=0 ieth\else y-\ordnine@\count@@\fi
\fi
\fi
\fi
\fi}
```

Chapter 6. Inhibiting expansion

There are numerous situations where we want to suppress expansion, during both `\write`'s and `\edef`'s or `\xdef`'s, of the numbering control sequences `\arabic`, `\alph`, ... as well as the font change control sequences `\rm`, `\it`, `\bf`, ... The latter includes `\smc`, which may not have been defined yet, so we take care of that first:

```
\font@\tensmc=cmcsc10
\textonlyfont@\smc\tensmc
```

[For the details of `\font@` and `\textonlyfont@` see `amstex.doc` (and also page 22); basically, this is like saying `\font\tensmc=cmcsc10` and `\def\smc{\tensmc}`.]

We introduce a new token list

```
\newtoks\noexpandtoks@
```

which will be a list of commands, and then `\noexpands@` will issue these commands by inserting this token list:

```
\noexpandtoks@={\let\arabic=\relax\let\alph=\relax
\let\Alpha=\relax\let\roman=\relax\let\Roman=\relax
\let\fnsymbol=\relax\let\rm=\relax\let\it=\relax
\let\bf=\relax\let\sl=\relax\let\smc=\relax
\let\/=\relax\let\null=\relax}
\def\noexpands@{\the\noexpandtoks@}
```

It is easy to define the construction `\Nonexpanding#1`, which is supposed to make `\let#1=\relax` also be executed:¹

```
\def\Nonexpanding#1{\global\noexpandtoks@
=\expandafter{\the\noexpandtoks@\let#1=\relax}}
```

¹In version 1 of `LATEX`, this was called `\Noexpand`, but that seems too close to `\noexpand` for comfort.

`\let\/= \relax` was added to `\noexpandtoks@` because `\/` will often occur in “style” commands (pages 50, 74, 104, et al.) that appear in `\edef`'s or `\xdef`'s and `\write`'s (pages 71, 76, et al.) and `\/` is no longer a primitive in $\mathcal{A}\mathcal{M}\mathcal{S}$ - \TeX or $\mathcal{L}\mathcal{M}\mathcal{S}$ - \TeX (page 5). Similarly `\let\null= \relax` was added because `\null` sometimes occurs in “style” commands (see page 208, for example).

Chapter 7. Invisibility

7.1. Invisible constructions. All constructions that are supposed to be “invisible” (the most important examples being `\label` and `\pagelabel`) begin with `\prevanish@`. If we are in horizontal mode, this sets `\saveskip@` (a glue register declared in $\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\mathcal{T}\mathcal{E}\mathcal{X}$) to be the previous glue, and then removes that glue. Otherwise, it simply sets `\saveskip@` to `0pt`.

```
\def\prevanish@{\saveskip@=0pt
\ifhmode\saveskip@=\lastskip\unskip\fi}
```

“Invisible” constructions end with `\postvanish@`, which puts back the `\saveskip@` glue, if greater than zero;¹ it must also look ahead to see if the next token is a space, and swallow up that space if the `\saveskip@` glue was greater than zero:

```
\def\postvanish@{\ifdim\saveskip@>0pt\hskip\saveskip@\fi
\futurelet\next\postvanish@@}
\def\postvanish@@{\def\next@.{}%
\ifx\next\space\ifdim\saveskip@>0pt\def\next@.{} \fi\fi
\next@.}
```

Here we use the method of page 7. Note that we *don't* want to use `\FNSS@` (section 3.8) in this special situation, since we want to eliminate the space only when `\saveskip@` is greater than zero.

[It is important to note that `\saveskip@` appears only in these definitions, except for certain $\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\mathcal{T}\mathcal{E}\mathcal{X}$ constructions that will not ever appear within an “invisible” construction. So we don't have to worry about the value of `\saveskip@` being clobbered before `\postvanish@` is applied.]

These constructions allow us to define the general construction to make anything “invisible”:

¹We don't want to add `\hskip\saveskip@` when `\saveskip@` is zero, because that case can occur when there is *no* previous glue: adding `\hskip\saveskip@` might then allow a break where none was allowed before.

```
\def\invisible#1{\prevanish@\ignorespaces#1\unskip
\postvanish@}
```

(Notice that, because of the `\ignorespaces` and `\unskip`, the remark in the \LaTeX Manual on page 32, lines 6–8, is incorrect.)

We will need a list, `\vanishlist@`, of all invisible constructions; this will be a list of the sort discussed in section 3.7. We initialize it as:

```
\def\vanishlist@{\\\invisible}
```

7.2. Special considerations for invisible constructions. If we have a construction like

```
\par
\invisible{...}\_Some text
```

then the `\prevanish@` in `\invisible` sets `\saveskip@` to `Opt`. Consequently, the `\postvanish@` will *not* delete the following space token that precedes ‘Some text’. But this space token is ignored in vertical mode (*The TeXbook*, page 282), so we don’t get an extra space before “Some text”.

The situation is quite different, however, if we have

```
\par
\noindent \invisible{...}\_Some text
```

because now the space token is not ignored, since it is encountered in horizontal mode; consequently, we will get an extra space before “Some text”.

In $\mathcal{A}\mathcal{M}\mathcal{S}$ - \TeX and $\mathcal{L}\mathcal{A}\mathcal{M}\mathcal{S}$ - \TeX , the combination `\par\noindent` has the special abbreviation `\flushpar`. We can avoid this difficulty with invisible constructions by redefining `\flushpar` as

```
\def\flushpar{\par\noindent\futurelet\next\pretendspace@}
```

where `\pretendspace@` simply inserts `\hskip-1pt\hskip1pt` (preceded by `\nobreak` just as a precaution) when `\next` is something in `\vanishlist@`:

```
\def\pretendspace@{\ismember@\vanishlist@\next
\iftest@\nobreak\hskip-1pt\hskip1pt\fi}
```


[A precautionary `\relax` after the `\hskip1pt` is not needed here—the `\fi` will stop the scanning of `\hskip`.]

As a result of this definition,

```
\flushpar\invisible{...}\u
```

becomes

```
\par\noindent\nobreak\hskip-1pt\hskip1pt\invisible{...}\u
```

(and similarly for `\flushpar\label{...}`, and any other “invisible” constructions that we will eventually define, and add to `\vanishlist@`).

Consequently, the `\prevanish@` removes the `\hskip1pt` and then sets `\saveskip@` to `1pt`. Then the `\postvanish@` adds back the `\hskip1pt` once again, canceling out the `\hskip-1pt`. Moreover, since `\saveskip@` is now positive, the space token after the `\invisible` will be thrown away, so that `\invisible{...}` will really be invisible.¹

Note, by the way, that `\ismember@` was defined in such a way that the test `\ismember@\vanishlist@\next` sets `\iftest@` to be true when `\next` has been `\let` equal to a control sequence in `\vanishlist@` (page 23).

[In the above definitions, it might seem that we could simply add the `\hskip-1pt\hskip1pt` in all cases. But that wouldn’t quite work, because the next construction might begin with an `\unskip` (e.g., `\linebreak` or `\dots`). Aside from such a case, however, the `\hskip-1pt\hskip1pt` doesn’t do any harm.]

Since some users might type `\noindent` instead of `\flushpar`, we might as well add the `\futurelet\next\pretendspace@` to `\noindent` also.

```
\let\noindent@=\noindent
\def\noindent{\par\noindent@\futurelet\next\pretendspace@}
\def\pretendspace@{\ismember@\vanishlist@\next
\iftest@\nobreak\hskip-1pt\hskip1pt\fi}
```

¹If some one has typed `\define\foo{\invisible{...}}` and then used `\flushpar\foo`, this presents no problem, since in this case no space after `\foo` will appear. (But, of course, `\define\foo/{\invisible{...}}` would make `\flushpar\foo/\u` behave incorrectly; this didn’t seem worth worrying about!)

```
\let\flushpar=\noindent
```

`\pretendspace@` will be needed at several other points in \LaTeX -TEX (sections 16.1, 18.5, et al.).

In `amstex1.tex` we delete the definition of `\flushpar`, since it will be replaced with this new definition.

Chapter 8. Special considerations for `\everypar`

Numerous \LaTeX constructions use `\noindent@` to start an unindented paragraph. If some

```
\everypar={...}
```

has been specified in the document, these unindented paragraphs would also start with the `\everypar` tokens, which is normally not desired.

Although this might be regarded as a rather paranoid concern, since `\everypar`'s should presumably be used only within some region of the document that contains only text, \LaTeX contains a special construction to deal with this problem.

First we introduce a new token list

```
\newtoks\everypartoks@
```

and then we define

```
\def\noindent@{\par\everypartoks@=\expandafter{\the\everypar}%  
\everypar={}%  
\noindent@\everypar=\expandafter{\the\everypartoks@}}
```

Thus, `\noindent@@`

- (1) ends the previous paragraph,
- (2) stores the current value of `\everypar` in `\everypartoks@`,
- (3) sets `\everypar` to be empty, and
- (4) starts an unindented paragraph, which
- (5) resets `\everypar` to its original value for the *next* paragraph.

Note that the original value of `\everypar` will *not* be inserted before the `\noindent@`ed paragraph, because it gets the value `{}` that was current when the `\noindent@` was encountered.

In \LaTeX , `\noindent@@` will usually be used instead of `\noindent`, with a `\futurelet\next\pretendspace@` added when an invisible construction might follow.

Chapter 9. \page

In \LaTeX the control sequence `\page` can be manipulated like `\tag`, `\claim`, etc. Thus, we can use `\Reset\page`, `\newpre\page`, But we want `\page` by itself to give an error message:

```
\def\page{\Err@{\noexpand\page has no meaning by itself}}
```

(Again, see section 3.4 for the use of `\noexpand`.)

As we will see in Chapter 11, associated with the \LaTeX construction `\tag` we have

```
\tag@C  the counter associated with \tag
\tag@P  the "pre" material for \tag
\tag@Q  the "post" material for \tag
\tag@S  the style for \tag
\tag@N  the numbering style for \tag
\tag@F  the font for \tag
```

Likewise, `\claim` and all other \LaTeX constructions that can be given a (label) have similar counters and control sequences associated with them.

At present, we simply want to consider the counter and control sequences associated to `\page`. For `\page@C` we just use plain \TeX 's `\pageno`

```
\let\page@C=\pageno
```

and then we introduce default values (`\empty` is defined in `plain.tex` by `\def\empty{}`, so `\let\page@P=\empty` is just a briefer way of saying `\def\page@P{}`):

```
\let\page@P=\empty
\let\page@Q=\empty
\def\page@S#1{#1//}
\def\page@F{\rm}
\def\page@N{\arabic} % cannot be \let
```

The `\` in `\page@S` might be useful if `\page@F` is ever chosen to be a slanted font.

We want to have `\def\page@F{\rm}` rather than `\let\page@F=\rm`, because `\rm` may actually change definitions. For example,

```
\fontstyle\page{...}
```

expands out (page 227) to

```
{\page@F...}
```

and if we are in 9-point type at the time, we would expect to get 9-point roman type, not the 10-point roman type that is in effect at the time that `\page@F` is specified.

And, as we will see later (page 59), it is even more critical that we have `\def\page@N{\arabic}` rather than `\let\page@N=\arabic`.

Chapter 10. Indexing

The indexing macros were placed next, because they use the fact that " is active in \LaTeX , and we would like to get this declared soon, so that any " appearing in other macros will refer to this active ". Some of the methods used here will also be crucial in Chapters 23 and 32.

It should be noted that in version 1 of \LaTeX , the index entries were written to one file, with the extension `.ndx`, while the corresponding page numbers were written to another file, with the extension `.npg`. That has all changed, however, and now the entry and the page number are written together to the `.ndx` file. Similarly, as we will see in Chapter 23, heading levels will be written together with their page numbers in the `.toc` file, and as we'll see in Chapter 32, Figures, Tables, etc., will be written together with their page numbers in one file.

10.1. The `.ndx` file. We will need a flag,

```
\newif\ifindexing@
```

to tell whether an index file is being made.

`\indexfile` will (globally) set the flag `\ifindexing@` to be true the first time it is used; it will also test this flag when called, so that if it is called twice it will do nothing at all the second time. The first time `\indexfile` is called, it should create a new output stream,

```
\newwrite\ndx@
```

associated with the file `\jobname.ndx` (where `\jobname` will be 'foo' when \TeX is processing `foo.tex`).

Instead of using `\newwrite`, we will just write the code for it instead, substituting `\alloc@@` (page 38) for `\alloc@`:

```
\def\indexfile{\ifindexing@\else  
  \alloc@@7\write\chardef\sixt@@n\ndx@  
  \immediate\openout\ndx@=\jobname.ndx  
  \global\indexing@true\fi}
```

Then (compare section 3.6), since we used `\alloc@` rather than `\alloc@`, nothing will be written to the `.log` file, even though `\indexfile` is used after `\alloc@` itself has been redefined at the end of `lamstex.tex`.

10.2. \indexproofing. We will need an insertion class, called `\margin@`, for index entries that are to appear in the margin if `\indexproofing` has been specified. So we would like to say `\newinsert\margin@`. But `\newinsert` is defined differently than all other `\new...` constructions in plain, and it will write something to the `.log` file, despite our redefinition of `\alloc@`. So we instead simply restate everything from plain in the definition of `\newinsert` except for the `\wlog` part:

```
\global\advance\insc@unt\m@ne
\ch@ck0\insc@unt\count
\ch@ck1\insc@unt\dimen
\ch@ck2\insc@unt\skip
\ch@ck4\insc@unt\box
\allocationnumber\insc@unt
\global\chardef\margin@\allocationnumber
```

Notice that although this takes up a lot of space in the file, it takes up hardly any space within TeX itself, just like `\newinsert\margin@`.

We put no limit on the number of marginal notes on a page, and they take up no space (compare *The TeXbook*, page 415):

```
\dimen\margin@=\maxdimen
\count\margin@=0
\skip\margin@=0pt
```

The flag `\ifindexproofing@` will tell us whether `\indexproofing` (and/or `\noindexproofing`) appears:

```
\newif\ifindexproofing@
\def\indexproofing{\indexproofing@true}
\def\noindexproofing{\indexproofing@false}
```

10.3. Converting tokens to type 12. If a control sequence `\controlseq` has been defined by

```
\def\controlseq⟨parameter text⟩{⟨replacement text⟩}
```

(see *The T_EXbook*, page 203, for terminology), then the T_EX primitive

```
\meaning\controlseq
```

expands to

```
macro : ⟨parameter text⟩ -> ⟨replacement text⟩
```

where all non-space tokens are of type 12.

The construction `\unmacro@` is used to store the `⟨parameter text⟩` in `\macpar@` and the `⟨replacement text⟩` in `\macdef@`:

```
\def\unmacro@#1:#2->#3\unmacro@{\def\macpar@{#2}%
\def\macdef@{#3}}
```

In particular, if we

```
\def\foo{#1}
```

where `#1` is any text with balanced braces, and we do

```
\expandafter\unmacro@\meaning\foo\unmacro@
```

then¹ `\macdef@` will consist of `#1` *with all non-space tokens converted to type 12*.

Some information is lost in the process: multiple spaces in `#1` coalesce to single spaces in `\macdef@`, control words in `#1` are followed by spaces in `\macdef@` even if no spaces appear after them in `#1`, and line breaks in `#1` simply become spaces in `\macdef@`. So this method is not particularly useful for literal mode, especially since it cannot be applied at all unless `#1` has balanced

¹There is no problem with `:` being part of the syntax of `\unmacro@` even in a file where `:` has been made active (compare section 3.10), because `\unmacro@` will always be used like this, to work on some value of `\meaning`.

braces. Nevertheless, it alleviates considerably the problems that arise when we want to `\write` the string `#1` to a file without having control sequences expanded. We simply have to write `\macdef@` instead!

More precisely, for some output stream, like `\ndx@`, instead of using an `\immediate\write` like

```
\immediate\write\ndx@{#1}
```

we can

```
\def\next@{#1}
\expandafter\unmacro@meaning\next@\unmacro@
\immediate\write\ndx@{\macdef@}
```

For delayed `\write`'s we have to be more careful, since `\macdef@` may have been redefined by the time the `\write` occurs. Instead of

```
\write\ndx@{\macdef@}
```

we must use

```
\edef\next@{\write\ndx@{\macdef@}}
\next@
```

The `\nxd@` is not expanded in this `\edef`, since it was created with `\chardef`; such control sequences aren't expanded in `\edef`'s. Consequently, the `\edef` simply makes `\next@` mean

```
\write\ndx@{(expansion of \macdef@)}
```

so that `\next@` then produces this `\write`.

Notice that it is irrelevant that we are writing type 12 tokens to the `.ndx` file: once they are written to that file their category codes are completely irrelevant—if T_EX reads this file later, they will simply be given the category codes that are in force at the time.

10.4. *The `\starparts@` and `\windex@` routines.* In version 1 of L^AT_EX, only invisible entries could have * optional entries, but now even visible entries can have them.

We will use a construction `\starparts@#1` that determines if #1 contains a * and defines

```
\stari@    to be all of #1
\starii@   to be the part of #1 before the first * (or all of #1 if there is none)
\starii@   to be the part of #1 after the first * (or empty if there is none)
```

We begin by choosing the values that will hold when no * appears:

```
\def\starparts@#1{\def\stari@{#1}\def\starii@{#1}\let\starii@=\empty
. . .
```

Then we perform a test that sets `\iftest@` to be true if * appears in #1 and false if it doesn't (compare the definition of `\tagin@` in section 3.3):

```
\test@false
\def\next@##1*##2##3\next@{\ifx\starparts@##2\test@false
\else\test@true\fi}
\next@#1*\starparts@\next@
```

If no * appears we are done; otherwise we will have to call another routine that separates the two parts:

```
\def\starparts@#1{\def\stari@{#1}\def\starii@{#1}%
\let\starii@=\empty
\test@false
\def\next@##1*##2##3\next@{\ifx\starparts@##2\test@false
\else\test@true\fi}%
\next@#1*\starparts@\next@
\iftest@\def\next@{\starparts@@#1\starparts@@}%
\else\let\next@=\relax\fi\next@}

\def\starparts@@#1*#2\starparts@@{\def\starii@{#1}%
\def\starii@{*#2}}
```

Once our "... " and "... " constructions, to be defined in section 5, have used `\starparts@` to determine `\stari@`, `\starii@`, and `\stariii@`, we will use `\stariii@` to typeset '...' in the case of a visible index entry, and then we will use the "write-index" routine `\windex@`.

When we are making an `.ndx` file, this routine will first

```
\expandafter\unmacro@meaning\stari@\unmacro@
```

to convert '...' to type 12 tokens. Then

```
\edef\macdef@{\string"\macdef@\string"}
```

will add " at each end, for the sake of the index program; `\string`" is needed since " will be active.

Then we will use the `\edef` of page 55 to write these tokens to the `.ndx` file. This will be followed by the page number. Actually, instead of writing just the page number, we write four groups, the first containing the page number, and the next three containing the page numbering style, the "pre-page" material, and the "post-page" material (Chapter 9),

```
\write\ndx@{\number\pageno}{\page@N}{\page@P}{\page@Q}}
```

This allows the index program to deal with all sorts of special page numbering possibilities.

In addition, when `\ifindexproofing@` has been set true, we want to

```
\insert\margin@{\hbox{\rm\vrule \height9pt \depth2pt
\width0pt ...
```

where the `\hbox` begins with a "strut", designed to keep baselines of successive entries 11 points apart (see page 7 for the use of `\height`, ...).¹

¹Perhaps this is a good place to mention something that tends to be obscured in discussions about struts. Most TeXnicians are familiar with struts as the device that allows one to place one `\vbox` above another and still have the proper space between the bottom baseline of the top `\vbox` and top baseline of the bottom `\vbox`. The problem here is that TeX will normally insert only `\lineskip` space between the two boxes, since the baseline of the second box is so far from the baseline of the first. But the situation with regard to `\footnote`'s, or members of other insertion classes, is really quite different: TeX inserts *no interline glue whatsoever* between two different members of an insertion class (*The TeXbook*, page 125). Thus, even single line footnotes will be spaced incorrectly without struts! (Struts are discussed further in section 25.2.)

This `\hbox` should contain all material before any `*` in `'...'` typeset in `\rm`, but all material after the first `*` should be converted to type 12 tokens, and typeset in the `\tt` font,¹ since it contains things like `*e\it`, which are not supposed to be acted upon, but merely convey information to the index program.

This is all accomplished with the following code:

```

\def\windex@{\ifindexing@
  \expandafter\unmacro@\meaning\stari@\unmacro@
  \edef\macdef@{\string"\macdef@\string"}%
  \edef\next@{\write\ndx@\macdef@}\next@
  \write\ndx@{\number\pageno}{\page@N}{\page@P}{\page@Q}}%
\fi
\ifindexproofing@
  \ifx\stariii@\empty\else
    \expandafter\unmacro@\meaning\stariii@\unmacro@\fi
    \insert\margin@{\hbox{\rm\vrule \height9pt \depth2pt
      \width0pt \starii@
      \ifx\stariii@\empty\tt\macdef@\fi}}%
  \ifx\stariii@\empty\else\tt\macdef@\fi}}%
\fi}

```

At the time that our `\write` is performed, we will want `\noexpands@` to be in effect, partly to prevent expansion of any font control sequences that might appear in `\page@P` and `\page@Q`, but mainly because we want to be sure that `\page@N` isn't expanded during the `\write`, since the index program expects to see a numbering control sequence in this second group.

But there's no point putting

```

{\noexpands@
  \write\npg@{\number\pageno}{\page@N}{\page@P}{\page@Q}}

```

in our definition, because this delayed `\write` is simply added to the main vertical list and does not take place until a `\shipout`. Instead, we will have to be careful to specify `\noexpands@` during any `\shipout` (section 36.1).

Nevertheless, this is an appropriate time to discuss the problems that would be encountered if expansion were not prohibited. Expansion would clearly

¹ Other styles (compare Part VII) may use smaller print for these side notes.

cause problems if `\page@N` is defined as `\alph` or `\Alph`, but, in fact, for this particular `\write`, expansion would be a problem even if `\page@N` is defined as `\arabic`, because `\page@N` appears in a group by itself—TeX would complain during the `\write` that

```
! Argument of \page@N has an extra }.
```

And here comes the most subtle point of all: If we `\let\page@N=\arabic`, then our `\write` would not put `\arabic` in place of `\page@N`, even if `\noexpands@` is in effect when it takes place, and we would get the very same error message. That is because `\page@N` would have the *original* meaning of `\arabic`—although `\noexpands@` says `\let\arabic=\relax`, our `\page@N` would not be this `\arabic`! On the other hand, when we `\def\page@N{\arabic}`, the `\write` first expands `\page@N` to (the current) `\arabic`, and then doesn't expand *this* `\arabic` further.

10.5. Indexing. Now we are finally ready to make " active,

```
\catcode'\="=\active
```

and define the action of ". As we have already noted (page 15), the combination @" will still work when " is active.

First of all, " will have to look ahead to see if it is followed by another ", because this indicates an invisible entry:

```
\def"\{\futurelet\next\quote@}
\def\quote@{\ifx\next"\expandafter\quote@@\else
\expandafter\quote@@@\fi}
```

Note that we are using the "K-method" here (see section 1.1).

`\quote@@@`, the result when " isn't followed by another ", so that we have a visible index entry, is simply defined as

```
\def\quote@@@#1"{\starparts@{#1}\starii@\windex@}
```

Thus, after `\starparts@` defines `\stari@`, `\starii@`, and `\stariii@`, we typeset `\starii@`—the part before any *—and then apply the "write-index" routine `\windex@`.

We should note that as a consequence of the definition of `\quote@@@#1`, the indexed word `#1` may be followed by an `\insert` and/or a “whatsit”, namely, the `\write` produced by `\windex@`. But either an `\insert` or a “whatsit” can appear *after* a word without suppressing hyphenation—see *The T_EXbook*, third paragraph from the bottom on page 454. (This should be compared to the `\makexref` macro on page 424 of *The T_EXbook*, where the `\insert` appears *before* the word, and therefore suppresses hyphenation of the word.) Similarly, as we will see in a moment, an invisible index entry simply supplies an `\insert` and/or a `\write`. The warning on page 100 of the L_A_M_S-T_EX Manual is therefore inaccurate: an invisible entry will suppress hyphenation of a word only if it immediately precedes it, not if it follows it. Similarly, the warnings on pages 31 and 33 are inaccurate; only a `\label` or `\pagelabel` immediately preceding a word will interfere with its hyphenation—in particular, the example given on page 33 won’t interfere with hyphenation.

`\quote@@` is not that much different from `\quote@`, except that we want it to swallow the next `"`, and begin with `\prevanish@`, so that it will be invisible:

```
\def\quote@@#1"{\prevanish@\starparts@{#1}\windex@
\futurelet\next\quote@@@}
```

The `\futurelet\next\quote@@@` is needed to see whether yet another `"` occurs after the third `"` that caused all this to happen. If a fourth `"` didn’t occur, we insert the `\postvanish@`, and if a fourth `"` did occur, we simply swallow it up, and then insert the `\postvanish@`:

```
\def\quote@@@{\ifx\next"\def\next@{"\postvanish@}\else
\let\next@=\postvanish@\fi\next@}
```

10.6. Changes to the L_A_M_S-T_EX Manual. Because of changes in the indexing macros, almost every caveat on page 101 of the L_A_M_S-T_EX Manual is wrong.

The first paragraph is wrong: index entries within heading levels *will* show up in the margins when `\indexproofing` has been specified. (Of course, one had better not type something like

```
\HL1 "Disappearing" words\endHL
```

since "Disappearing" would then be interpreted as a "quoted" number for \HL1! Something like \HL1{"Disappearing" is needed.)

The third paragraph is wrong, because only the parts after the first * will be typeset in the typewrite font, with characters of type 12.

The fourth paragraph is wrong: \" can be used within an invisible entry.

The fifth paragraph is wrong: invisible entries can now appear anywhere.

It is true, as the final paragraph claims, that index entries in \footnote's won't appear in the margin (they are \insert's within an \insert, and won't migrate out). However, no special efforts are required in the \footnote macros to get indexing to work within \footnote.

10.7. Invisibility. After all this, we want to add " to \vanishlist@:

```
\rightadd@"\to\vanishlist@
```

This probably looks wrong, since it is only the double mark "" that indicates an invisible entry, but

```
\nobreak\hskip-1pt\hskip1pt"..."
```

and

```
\nobreak\hskip-1pt\hskip1pt""..."
```

will both work out just right: before the visible index entry "... " the \hskip-1pt\hskip1pt will simply be irrelevant, while before an invisible index entry ""..." it provides the right clues for the \prevanish@ called by \quote@@ (compare page 47).

10.8. Other delimiters for index entries. The use of " as a delimiter for index entries conflicts with its use in German styles (this will probably remain true even when the international font layouts are in use, although then the " will presumably no longer be active for German). However, it is not very hard to set up other delimiters for this purpose.

For example, suppose we want to use <...> delimiters, so that

```
Beauty<<beauty>> is <truth>.
```

will produce a index entry for truth, and an invisible index entry for beauty.
For this, we could

```

\catcode'\<=\active
\let<=" % we might as well continue using \quote@, ...
% \def"{...} if we have new definitions for German, or
% \catcode'"=12 if " should no longer be active

\def\windex@{\ifindexing@
\expandafter\unmacro@\meaning\nextii@\unmacro@
\xdef\nextii@{\string\macdef>}%
. . .
\fi}

\def\quote@{\ifx\next<\expandafter\quote@@\else
\expandafter\quote@@@\fi}
\def\quote@@@#1>{\starparts@{#1}\starii@\windex@}
\def\quote@@<#1>{\prevanish\starparts@{#1}\windex@
\futurelet\next\quote@@@}
\def\quote@@@{\ifx\next>\def\next@>{\postvanish@}\else
\let\next@=\postvanish@\fi\next@}

```

The new version of the index program (see Chapter 39) now accepts any delimiters in the .ndx file. However, as Chapter 39 points out, for German alphabetization we would probably want some modifications to deal with words with umlauts.

We should probably also remove " from \vanishlist@. There is no general mechanism for removing something from \vanishlist@. However, since we know that \vanishlist@ will be of the form

```

\\invisible\\" ...


```

we can

```

\def\next@\\invisible\\"#1\next@{\def\vanishlist@{#1}}
\expandafter\next@\vanishlist@\next@

```

 For consistency, it would probably be better to choose, once and for all, index entry delimiters that could be used in all cases; <...> don't satisfy that requirement,

since Scandinavian keyboards have letters instead of < and >. There aren't too many possibilities left, however! The only reasonable candidates are _ and | (although + and = would also be possible, if we insisted that people never used them outside of math mode), and

```
Beauty||beauty|| is |truth|, truth beauty
```

doesn't look too bad. If only | weren't used for something else in German styles!

10.9. \define and \iabbrev. A construction like

```
\define\cs{parameter text}{(replacement text)}
```

has to

```
\define\cs{parameter text}{(replacement text)}
```

and also send this definition off to the .ndx file.

\define first stores its argument, \cs say, in \next@, and also stores \noexpand\cs in \nextii@, for later use:

```
\def\next@{#1}\def\nextii@{\noexpand#1}%
```

Then we will apply the construction \define@ once we have suitably swallowed up the (parameter text) and (replacement text):

```
\def\define#1{\def\next@{#1}\def\nextii@{\noexpand#1}%
\afterassignment\define@\def\nextiii@}
```

Here the \def\nextiii@ will cause the following (parameter text) and (replacement text) to be digested into a definition of \nextiii@, after which assignment we will apply \define@.

Since \nextiii@ now has the definition that we want for \cs, the first thing \define@ must do is to

```
\let\cs=\nextiii@
```

Since `\next@` was `\def`'ed to be `\cs`, we can do this with

```
\expandafter\let\next@=\nextiii@
```

Then, if we are indexing, we need to recover the `<parameter text>` and `<replacement text>` for `\cs`, which is now that for `\nextiii@`. So we use

```
\expandafter\unmacro@\meaning\nextiii@\unmacro@
```

[This construction doesn't work if `'->'` appears in the `<parameter text>` of a definition, so let's hope no one ever makes such a definition.]

Now we want to write

```
\define\cs<parameter text>{<replacement text>}
```

to the `.ndx` file. Since `\nextii@` was defined as `\noexpand\cs`, we can do this with

```
\immediate\write\ndx@{\noexpand\define
\nextii@\macpar@{\macdef@}}
```

—note that the `\write` will first expand `\nextii@` to `\noexpand\cs`, and then replace this with `\cs`, unexpanded. Since all tokens in `\macpar@` and `\macdef@` are type 12, we don't have to worry about their expansion (and note the remark on page 55).

Thus, the definition of `\idefine@` reads:

```
\def\idefine@{\ifindexing@
\expandafter\let\next@=\nextiii@
\expandafter\unmacro@\meaning\nextiii@\unmacro@
\immediate\write\ndx@{\noexpand\define\nextii@
\macpar@{\macdef@}}\fi}
```

`\iabbrev` is simpler. Recall that `\iabbrev` must be used in the form

```
\iabbrev*\cs{...}
```

so we define

```
\def\iabbrev*#1#2{\ifindexing@  
\toks@={#2}%  
\immediate\write\ndx@  
{\noexpand\abbrev*\noexpand#1{\the\toks@}}\fi}
```

Part II

*Labels and
Cross References*



Chapter 11. The `\label` mechanism

\LaTeX 's `\label` mechanism—one of its most crucial features—involves several different parts of \LaTeX , and will occupy the next few chapters. The present chapter merely explains the basic strategy, without presenting any explicit code. (Section 4 also explains about an important new construction that has been added to \LaTeX .)

11.1. Constructions that can be given `(label)`'s. For any \LaTeX construction `\foo` that can be given a `(label)`, \LaTeX uses

<code>\foo@C</code>	for the counter associated with <code>\foo</code>
<code>\foo@P</code>	for the “pre” material associated with <code>\foo</code>
<code>\foo@Q</code>	for the “post” material associated with <code>\foo</code>
<code>\foo@S</code>	for the style associated with <code>\foo</code>
<code>\foo@N</code>	for the numbering style associated with <code>\foo</code>
<code>\foo@F</code>	for the font associated with <code>\foo</code>

So, for example, `\tag` has an associated counter `\tag@C`, and associated control sequences `\tag@P ...`, `\tag@F`; `\claim` has an associated counter `\claim@C`, and associated control sequences `\claim@P, ...`, `\claim@F`; etc. The values of the `...@C` counters can be manipulated directly, or indirectly through `\Reset` and `\Offset`, and the `...@P`, `...@Q`, `...@S`, `...@N`, and `...@F` control sequences can be redefined directly, or indirectly using `\newpre, ...`, `\newfontstyle`. (This is covered in detail in Chapter 24.)

Moreover, every `\tag` in a displayed formula, every `\claim... \endclaim`, etc., locally defines four quantities, `\thelabel@, ...`, `\thelabel@@@`, to be used if the construction is ever given a `(label)`:

<code>\thelabel@</code>	will be the value of <code>\ref{(label)}</code>
<code>\thelabel@@</code>	will be the value of <code>\Ref{(label)}</code>
<code>\thelabel@@@</code>	will be the value of <code>\nref{(label)}</code>
<code>\thelabel@@@@</code>	will be the value of <code>\pref{(label)}</code>

All \LaTeX constructions that can be given a `(label)` implicitly provide grouping¹ and outside of the group `\thelabel@@, ...`, `\thelabel@@@@` are

¹`\claim... \endclaim` and all similar \LaTeX constructions provide grouping; in the case of `\tag` the grouping in question is provided by the display `$$...$$` in which the `\tag` lies.

undefined (unless the construction happens to lie within another construction that can be given a `\label`). However, we will initially

```
\let\thelabel@=\relax
```

Then the simple test

```
\ifx\thelabel@\relax
```

will be true unless we are in a construction where `\label` is legitimate.

As a (somewhat contrived) example of how this works, suppose that in the third section of some document, `\claim` numbers are being printed as `(3.i)`, `(3.ii)`, `(3.iii)`, ... , and that before the tenth `\claim` we state

```
\Offset\claim0
\newpost\claim{-A}
```

so that the number of the tenth `\claim` will be printed as `'(3.ix-A)'`.

Then within the tenth `\claim`

the value of <code>\claim@C</code>	will be	9
<code>\claim@P</code>	will be	3.
<code>\claim@Q</code>	will be	-A
<code>\claim@S#1</code>	will be	(#1)
<code>\claim@N</code>	will be	<code>\roman</code>
<code>\claim@F</code>	will be	<code>\bf</code>

and, correspondingly

<code>\thelabel@</code>	will be	<code>\roman{9}</code>
<code>\thelabel@@</code>	will be	<code>(3.\roman{9}-A)</code>
<code>\thelabel@@@</code>	will be	9
<code>\thelabel@@@@</code>	will be	<code>3.\roman{9}-A</code>

The value of the counter for `\claim`, the numbering style, the pre- and post-material, and the style for `\claim` are all involved in the values of `\thelabel@`, ... , `\thelabel@@@@`; see section 5 regarding the font style.

11.2. Restrictions. The first concrete example of defining `\thelabel@, \dots, \thelabel@@@` occurs in Chapter 16. For the moment, we simply want to note that these control sequences will essentially be created using `\edef`'s, to insure that they will contain the current values of the counter, numbering style, etc. Moreover, they will often occur in `\xdef`'s and `\write`'s (see page 76).

This means that

Any control sequences appearing in
`\dots@P, \dots@Q, \dots@S, or \dots@N`
 must be ones that can appear in `\xdef`'s and `\write`'s.

Actually, we will use `\noexpands@` (Chapter 6) to inhibit expansion during the `\edef`'s, `\xdef`'s, and `\write`'s (pages 82, 84, et al.), so the proper strategy is to allow only control sequences for which `\noexpands@` prevents expansion.

We've already noted (page 44) that `\let\/= \relax` was added to the token list `\noexpandtoks@` because `/` often appears in `\dots@S` definitions; and `\let\rm= \relax, \dots` were added because font change control sequences often appear also—see, for example, pages 74 and 104.

Similarly, page 35 of the \LaTeX Manual indicates that if a new font selection command `\TimesRoman` is ever going to be used in defining the style for anything that can be labelled, then `\Nonexpanding\TimesRoman` ought to be added, and if a new numbering command `\Babylonain` is ever going to be used to number anything that can be labelled, then we should add `\Nonexpanding\Babylonian` to the file.¹

11.3. Consequences of these restrictions. Since constructions like `\newpre` simply define various `\dots@P` control sequences, the restrictions of the previous section also mean that in constructions like

```
\newpre\tag{...}
```

any control sequences in '`...`' must be ones for which `\noexpands@` prevents expansion. And this means that some of the things stated, and implied, in the \LaTeX Manual are false.

¹The \LaTeX Manual also indicates that the style file for that manual (and for this manual, as well) includes `\Nonexpanding\FC`, because the "font complement" `\FC` command occurs in `\footmark@S` and `\foottext@S`.

While the illustrations of `\newpost` given for `\tag` and the illustrations of `\newstyle` and `\newnumstyle` given in connection with `\list` were quite legitimate, the small print on page 77 gives the example

```
\newpre\exno{\value\HL1.\value\h11.}
```

for numbering a user-created construction. But this example is completely wrong! Things like `\value` can't be used for `\newpre`.

Instead, we need something like¹

```
(A)      \Evaluate\HL1
         \edef\HLvalue{\number\Value}
         \Evaluate\h11
         \newpre\exno{\HLvalue.\number\Value.}
```

And the situation is even more complicated, because we need to restate this after every `\h11`.

11.4. `\Initialize`. To handle such situations, \LaTeX now has the construction `\Initialize` (the details of which are described in sections 23.4 and 23.15). If you type

```
(B)      \Initialize\h11{\Evaluate\HL1
         \edef\HLvalue{\number\Value}%
         \Evaluate\h11
         \newpre\exno{\HLvalue.\number\Value.}}
```

then the set of commands (A) will be executed each time an `\h11` occurs.

Similarly,

```
\Initialize\HL1{\newpre\h11{}}
```

would make the pre-material for `\h11` be empty even for the default style, which normally makes the pre-material for `\h11` be '1.' in the first `\HL1`, and '2.' in the second, etc.

¹To be on the safe side, we might prefer `\Evaluatepref\HL1 \edef\HLvalue{\Pref}, ...`, in case `"..."` has been used to quote a heading level, but `\Evaluate` will illustrate the point well enough.

If `\chapter` and `\section` have been introduced as names for `\HL1` and `\h11` (see Chapter 23 for details), then we can substitute `\chapter` for `\HL1` and/or `\section` for `\h11` in the `\Initialize` command, e.g.,

```
\Initialize\chapter{\newpre\section{}}
```

[Different `\Initialize` commands don't accumulate, so if you later want to add `\newpre\h12{}`, then you must use

```
\Initialize\HL1{\newpre\h11{}}\newpre\h12{}}
```

This shouldn't be much of a problem, since such commands are rather special, and would probably go near the beginning of a document; moreover, if they did accumulate, it would quite dicey to *cancel* any such command.]

As an extra bonus, within the `\Initialize` construction, `\pref` may be used with a special significance. For example, in something like

```
\Initialize\h11{\newpre\exno{\pref.}}
```

the '`\pref`' will give the value that `\pref` would have for a `(label)` in the `\h11` that has just been executed. Consequently, in the default style this will have the same effect as `(B)`.¹

11.5. The question of fonts. Note that the font for `\claim` is not recorded anywhere in `\thelabel@`, ..., `\thelabel@@@`—it is relevant only when the `\claim` number is actually printed.

Thus, as we'll see in Chapter 24, `\Ref{(label)}` will not print '`(3.ix-A)`', but simply '`(3.ix-A)`' [or '`(3.ix-A)`' if we are using italic type, etc.]. But we can get '`(3.ix-A)`' by typing `\fontstyle\claim{\Ref{(label)}}`, which expands out (page 227) to

```
{\claim@F\Ref{(label)}}
```

This seems like the optimal arrangement, giving the option of using the same font or not.

¹More precisely, it will have the same effect as if we had used `\Evaluatepref`, as suggested in footnote 1 on page 72.

Note, by the way, that to print `\claim` numbers as **(3.i)**, **(3.ii)**, **(3.iii)**, . . . , with bold numbers and letters, but with *roman* parentheses, we would define `\claim@S` (either directly, or indirectly through `\newstyle`), to be

```
\def\claim@S#1{\rm(#1\/\rm)}
```

and `\fontstyle\claim{\Ref{label}}` will then give the claim number printed in exactly this way. (The `\/` is useful in case `\claim@F` is ever chosen to be a slanted font.)

11.6. Storing `\label`'s. Every time a valid `\label{label}` or `\pagelabel` appears, so that `label` can be given associated values `\ref{label}`, . . . , \LaTeX has to record this information in two places:

- (1) The information must be kept internally, in a new control sequence that we will consider in a moment.
- (2) The information must also be written to the auxiliary `.lax` file.¹

The reason for writing to the auxiliary file, of course, is so that the information from that auxiliary file can be read in again the next time the document is \TeX 'ed, thus allowing for forward references.

It will be important to distinguish information read in from the auxiliary file from information created on the current run, and also to distinguish labels created by `\label` from those created by `\pagelabel`.

The basic datum that we will be recording in the `.lax` file will be a term of the form

$$@\langle label \rangle \sim V_1 \sim V_2 \sim V_3 \sim V_4 \sim \langle \text{type indicator} \rangle$$

where

$$\begin{aligned} V_1 &= \text{value of } \langle \text{ref}\{label\} \rangle \\ V_2 &= \text{value of } \langle \text{Ref}\{label\} \rangle \\ V_3 &= \text{value of } \langle \text{nref}\{label\} \rangle \\ V_4 &= \text{value of } \langle \text{pref}\{label\} \rangle \end{aligned}$$

¹ In version 1 of \LaTeX , the auxiliary file had the extension `.aux`; however, this has been changed to `.lax` (" \LaTeX auxiliary file"), not only to avoid conflict with `.aux` files produced by \LaTeX , but also because \LaTeX can now write `.aux` files also (Chapter 30); these have the structure of \LaTeX auxiliary files, but contain only entries relevant to \BibTeX , so that \BibTeX can be used with \LaTeX files also. In conformity with this change, the old `\readaux` has been changed to `\readlax` (section 15.1).

and the $\langle \text{type indicator} \rangle$ is

- 0 if $\langle \text{label} \rangle$ was created by a $\backslash \text{label}$ on the current run
- 1 if $\langle \text{label} \rangle$ was created by a $\backslash \text{label}$ on the previous run
- 2 if $\langle \text{label} \rangle$ was created by a $\backslash \text{pagelabel}$ on the current run
- 3 if $\langle \text{label} \rangle$ was created by a $\backslash \text{pagelabel}$ on the previous run

Each $\backslash \text{label}\{\langle \text{label} \rangle\}$ will create a new control sequence ' $\backslash \langle \text{label} \rangle @L$ ' with the value

$$V_1 \sim V_2 \sim V_3 \sim V_4 \sim \langle \text{type indicator} \rangle$$

(we use quotes around $\backslash \langle \text{label} \rangle @L$ as on pages 14 and 29), and it will also write

$$@\langle \text{label} \rangle \sim V_1 \sim V_2 \sim V_3 \sim V_4 \sim \langle \text{type indicator} \rangle$$

to the `.lax` file.

$@$ and \sim are simply two conveniently chosen tokens that should not appear within any $\langle \text{label} \rangle$. Actually, the $@$ and \sim appearing in the definition of ' $\backslash \langle \text{label} \rangle @L$ ' and in the `.lax` file will *not* be active characters, but will have category code 11 (so that normally they won't even appear in the input file). For $@$ this is easy to arrange (indeed it would be quite difficult to avoid), since all our definitions are going to be made while $@$ has category code 11; for \sim we will simply declare $\backslash \text{catcode}'\sim'=11$ at the beginning of our definitions, and return to $\backslash \text{catcode}'\sim'=\text{active}$ at the end. Because these \sim 's are not active, we do not have to worry about their being expanded in any $\backslash \text{xdef}$'s or $\backslash \text{write}$'s, which will turn out to be quite a convenience.

When we begin our document, with the $\backslash \text{document}$ command, if the `.lax` file already exists (from a previous run), it will be read in, line by line, and each line

$$@\langle \text{label} \rangle \sim V_1 \sim V_2 \sim V_3 \sim V_4 \sim \langle \text{type indicator} \rangle$$

will be used to define ' $\backslash \langle \text{label} \rangle @L$ ' to have the value

$$V_1 \sim V_2 \sim V_3 \sim V_4 \sim \langle \text{type indicator} \rangle$$

Thus, whenever a `.lax` file already exists, we will start with all the information obtained on the previous run. Of course, we will have to make sure that `~` has category code 11 while reading in the lines of the `.lax` file so that it will have that category code when it is used in defining the corresponding control sequence (compare the remark on page 55).

11.7. `\ref` and its relatives. It should be pretty obvious how `\ref#1`, `\Ref#1`, and other cross-referencing commands will work: The test

```
\expandafter\ifx\csname#1@L\endcsname\relax
```

is true precisely when `#1` has not yet been used as a `\label`. If the test is true, we simply give an error or warning message. Otherwise, the value of `\csname#1@L\endcsname` will be

```
V1~V2~V3~V4~<type indicator>
```

`\ref` will return `V1`, `\Ref` will return `V2`, etc.

The really interesting question is how we are going to keep the `\label` and `\pagelabel` information updated.

11.8. `\label`. Whenever we encounter a `\label{<label>}`, we will first check that we are in a construction that allows `\label`'s, using the test (page 70)

```
\ifx\thelabel@\relax
```

If the result is true, so that we are in a construction not allowing a `\label`, we will simply give an error message. Otherwise, we will use the test

```
\expandafter\ifx\csname#1@L\endcsname\relax
```

which is true if and only if `#1` has not already been used as a `\label`.

If `#1` has not already been used as `\label`, we will use

```
\expandafter\xdef\csname#1@L\endcsname{V1~V2~V3~V4~0}
```

the 0 at the end indicating that `\label` comes from a `\label` added on the current run. In addition, we will

```
\immediate\write\laxwrite@{0#1~V1~V2~V3~V4~1}
```

with a 1 at the end. When this line of the .lax file is read by \document on the next run, ‘\<label>@L’ will then be defined so that its value has a 1 at the end, indicating that it represents “previous” information.

On the other hand, if <label> has already been used, we need to examine the whole sequence

```
@<label>~...~...~...~...~<type indicator>
```

and determine the value of the <type indicator>.

If this value is 0 or 2, then <label> has already been used on the current run, and we will just give an error message saying that <label> has already been used.

But if the value is 1 or 3, then the information for <label> was compiled during the previous run. Since we want to allow the previous value to be changed, we will change the definition of ‘\<label>@L’ so that the sequence

```
...~...~...~...~(1 or 3)
```

that currently appears in the definition is replaced by the appropriate

```
V1~V2~V3~V4~0
```

the 0 once again indicating that <label> now comes from a \label created on the current run. And, once again, we will write appropriate information to the .lax file, but with 1 replacing 0.

11.9. \pagelabel. The \pagelabel construction works rather differently. First of all, we don’t use the

```
\ifx\thelabel@\relax
```

test, because \pagelabel’s are allowed anywhere.

So we simply start with the test

```
\expandafter\ifx\csname#1@L\endcsname\relax
```

If this test is false, then <label> has never been used, and now we define ‘\<label>@L’ to be

```
V1~V2~V3~V4~2
```

where the 2 at the end indicates that `\label` comes from a `\pagelabel` added on the current run, and where

V_1 = value of `\page@N{\number\page@C}`

V_2 = value of `\page@S{\page@P\page@N{\number\page@C}\page@Q}`

V_3 = value of `\number\page@C`

V_4 = value of `\page@P\page@N{\number\page@C}\page@Q`

As in the case of `\label`, we also want to write appropriate information to the `.lax` file, except that the 2 at the end should be replaced by 3, so that when this line is read by `\document` in the next run, '`\label@L`' will be defined with a 3 at the end, so that it will appear as "previous" information.

But now there is a big difference: for `\label` we will use an `\immediate\write`, but for `\pagelabel` we will use an ordinary `\write`. That is because only a (delayed) `\write` is certain to give the proper value of `\page@C`—the information that we have recorded in '`\label@L`' may actually be incorrect, because the value of `\page@C` may not be the number of the page on which the `\pagelabel` eventually appears. And this means, of course, that only `\pagelabel`'s read in from the *previous* run can be assured of being correct.

The other case, when `\label` has already been used, also requires changes. As before, we must examine the value

...~...~...~...~(type indicator)

of '`\label@L`' and determine the value of the (type indicator).


If this value is 0 or 2, then `\label` has already been used on the current run, and we will just give an error message saying that `\label` has already been used.

On the other hand, if the value is 1 or 3, then the information for `\label` was compiled during the previous run.

Now a 1 indicates that in the previous run `\label` was used for a `\label`, rather than a `\pagelabel`—presumably because the user has now decided to use `\label` for a `\pagelabel` that was used for a `\label` on the previous run. In this case, we will simply replace the definition of '`\label@L`' with the appropriate information for this new `\label`, replacing the 1 with a 2. We will also write corresponding new information to the `.lax` file, changing the final

1 to a 3, however, so that when it is read in again on the next run, it will be recognized as information coming from a previous `\pagelabel`.

On the other hand, a 3 indicates that in the previous run `\label` was used for a `\pagelabel`. In this case, as indicated above, this previously obtained information is more likely to be the correct one. Consequently, we will *not* change the definition of `'\label@L'`. But we will still write this appropriate new information (using a delayed `\write`) to the `.lax` file, keeping the final 3, so that it will appear once again as “previous” information on the next run.

 The label mechanism in version 2 of `LAMS-TEX` is entirely different, and much more efficient than, the one used in version 1 of `LAMS-TEX`, which was designed using a `TEX` that allowed 3,000 control sequence names. Since `LAMS-TEX` itself uses up about 2,700 names, not counting additional control sequences introduced by style files, it seemed imprudent at that time to use a labelling method that introduces a new control sequence name for each `\label`. But with that old mechanism, main memory was usually exhausted after only about 40 or 50 `\label`'s had been created (necessitating the stratagems explained in section 4.8 of the `LAMS-TEX` Manual), so this excessive wariness was clearly counterproductive. Moreover, more generous `TEX`s, allowing at least 3,500 control sequence names, are now quite common (and there's always `tinylams.tex` for the truly parsimonious).

Chapter 12. Beginning the document

12.1. Preliminaries. Since we have to read an existing .lax file at the beginning of the document, and then write to a new .lax file during the document, we first declare

```
\newread\laxread@
\newwrite\laxwrite@
```

\LaTeX makes use of a special sort of list, initially defined to be empty,

```
\let\fnpages@=\empty
```

which is used for fancy footnote numbering. As we will see in Chapter 25, when `\fancyfootnotes` is in effect, each `\footnote` will cause \LaTeX to write a special line to the .lax file: If the first footnote occurs on page 7, say, then

```
F7
```

will be written to the .lax file. If the second and third footnotes occur on pages 12 and 14, then the lines

```
F12
```

and

```
F14
```

will each be written (sometime later on). These lines are always written with a (delayed) `\write`, and the corresponding information is *not* recorded internally. On the other hand, when we read in an existing .lax file, any such `F(number)` lines will be incorporated into `\fnpages@`, which will end up looking like

```
\\7\\12\\14 . . .
```

(As we will see in section 25.3, we will be able to extract information from \fnpages@ in an extremely efficient way.) For this purpose, we will be using

```
\def\Finit@#1#2\Finit@{\let\nextii@=#1\def\nextiii@{#2}}
```

so that \Finit@... \Finit@ will \let\nextii@ be the first token in ‘...’ and define \nextiii@ to be the remainder. (Since \Finit@ will always be applied to a line we read in from the .lax file, \nextii@ will always be either F or @.)

At this point, lamstex.tex changes ~ to a letter, and introduces the routine \getparts@:

```
\catcode'\~=11
\def\getparts@ @#1~#2~#3~#4~#5~#6{\def\nextiv@{#1}%
\def\nextiii@{#2~#3~#4~#5~}\count@=#6\relax}
```

Thus, when applied to something of the form

$$\langle \text{label} \rangle \sim V_1 \sim V_2 \sim V_3 \sim V_4 \sim \langle \text{type indicator} \rangle$$

\getparts@ stores \langle label \rangle in \nextiv@ and $V_1 \sim V_2 \sim V_3 \sim V_4 \sim$ in \nextiii@, and sets \count@ to the value of \langle type indicator \rangle.

12.2. \document. As in $\mathcal{A}_M\mathcal{S}\text{-TEX}$, \document first defines \fontlist@ to be empty. It then opens the file \jobname.lax for reading. It processes the contents of this file one line at a time, reading each line into \next@. This is done with a \loop, where we \repeat until \ifeof detects the end of the file (see section 3.9).

First we will see if \next@ is a special line starting with F by using the test

```
\expandafter\Finit@\next@\Finit@
\ifx\nextii@ F
```

If this test is true, then \next@ is ‘F<number>’, and \nextiii@ has been defined to be the <number>. We want to add ‘\<number>’ to \fnpages@,

```
\expandafter\rightadd@\nextiii@\to\fnpages@
```

If the test is false, so that `\next@` is not one of the special lines for fancy footnote numbering, then it will be of the form

```
(label)~V1~V2~V3~V4~(type indicator)
```

and we want to make the appropriate definition

```
\def \<label>@L' {V1~V2~V3~V4~(type indicator)}
```

To do this we use

```
\expandafter\getparts@\next@
```

and then

```
\edef\next@{\gdef\csname\nextiv@ @L\endcsname
  {\nextiii@\number\count@}
\next@
```

Here the `\edef` makes `\next@` mean

```
\gdef \<label>@L' {V1~V2~V3~V4~(type indicator)}
```

The control sequence '`\<label>@L'`' is not expanded further because it has been made equal to `\relax`. As noted on page 71, any control sequences appearing in `\next@` should be ones whose expansion is inhibited by `\noexpands@`; so `V1~V2~V3~V4~` will not be expanded further, because we will be doing all this within a group with `\noexpands@`. In addition, as we mentioned on page 76, we will want to make `@` and `~` have category code 11 within this group. Finally, we will set

```
\endlinechar=-1
```

within this group, so that `\next@` will not contribute a blank space at the end because of the `<carriage-return>` at the end of the line.

There is one further detail that we have to worry about. Many files, especially files that have been written by T_EX itself, have a `<carriage-return>` at the

end. In this case, the last \next@ before the end of the file will be empty (if we hadn't set \endlinechar=-1 it would be \par). Consequently,

```
\expandafter\Finit@\next@\Finit@
```

and

```
\expandafter\getparts@\next@
```

would give error messages. So, before any of our tests, we will first make sure that \next@ isn't empty.

After having read \jobname.lax, and thus obtaining all the information from the last run of the file, we re-open the file, to record information produced with this run:

```
\def\document{\let\fontlist@=\empty
\immediate\openin\laxread@=\jobname.lax\relax
{\endlinechar=-1 \noexpands@
\catcode'\@=11 \catcode'\~=11
\loop \ifeof\laxread@ \else
\read\laxread@ to\next@
\ifx\next@\empty
\else
\expandafter\Finit@\next@\Finit@
\ifx\nextii@ F%
\expandafter\rightadd@\nextiii@\to\fnpages@
\else
\expandafter\getparts@\next@
\edef\next@{\gdef\csname\nextiv@ @L\endcsname
{\nextiii@\number\count@}}%
\next@
\fi
\fi
\repeat}%
\immediate\closein\laxread@
\immediate\openout\laxwrite@=\jobname.lax\relax}
```

Chapter 13. Labels

13.1. `\label`. We begin by setting

```
\let\thelabel@=\relax
```

(see pages 70 and 76). Since the combination

```
\thelabel@ ~\thelabel@@ ~\thelabel@@@ ~\thelabel@@@~ ~
```

occurs several times in our constructions, it will save time and space to introduce an abbreviation for it:

```
\def\thelabels@{\thelabel@ ~\thelabel@@ ~\thelabel@@@ ~\thelabel@@@~ ~}
```

The definition of `\label#1` begins with `\prevanish@`, since `\label`'s are supposed to be invisible, and then gives an error message if `\thelabel@` is `\relax`, so that we are not in a construction that allows `\label`'s. Otherwise we first use the test

```
\expandafter\ifx\csname#1@L\endcsname\relax
```

which is true precisely when `#1` has not already been used as a `(label)`. In this case, we simply use

```
\expandafter\xdef\csname#1@L\endcsname{\thelabels@0}  
\immediate\write\laxwrite@{#1~\thelabels@1}
```

to define '`#1@L`' and to write to the `.lax` file. Of course, this must be done within a group with `\noexpands@`.

If `#1` has already been used as a `(label)`, then we need to look at the `(type indicator)` of '`#1@L`'. To do this we use

```
\edef\next@{~\csname#1@L\endcsname}  
\expandafter\getparts@\next@
```

so that `\count@` will have the value of `(type indicator)`. We need the `\edef\next@` so that `\next@` will contain the actual value that the control sequence `\csname#1@L\endcsname` expands out to; naturally, we also need to perform this step in a group with `\noexpands@`.

If, as a result of our test, `\count@` is even, we issue an error message that label #1 has already been used. Otherwise, we simply use

```
\expandafter\xdef\csname#1@L\endcsname{\thelabels@0}
\immediate\write\laxwrite@{#@1~\thelabels@1}
```

to define the control sequence and write to the `.lax` file:

```
\def\label#1{\prevanish@
\ifx\thelabel@\relax
\Err@{There's nothing here to be labelled}%
\else
{\noexpands@
\expandafter\ifx\csname#1@L\endcsname\relax
\expandafter\xdef\csname#1@L\endcsname{\thelabels@0}%
\immediate\write\laxwrite@{#@1~\thelabels@1}%
\else
\edef\next@{~\csname#1@L\endcsname}%
\expandafter\getparts@\next@
\ifodd\count@
\expandafter\xdef\csname#1@L\endcsname{\thelabels@0}%
\immediate\write\laxwrite@{#@1~\thelabels@1}%
\else
\Err@{Label #1 already used}%
\fi
\fi
}%
\fi
\postvanish@}
```

For simplicity, we used a single group to enclose all the constructions that require a `\noexpands@`.

Finally, having defined `\label` we now

```
\rightadd@\label\to\vanishlist@
```

13.2. \pagelabel. There are several differences between the definition of `\pagelabel` and that of `\label`.

(1) First of all, we don't have to check the value of `\thelabel@`, since `\pagelabel` is allowed anywhere.

(2) Instead of `\thelabel@`, ..., `\thelabel@@@`, which are defined by constructions that can be given a `<label>`, we use the values we want for a page label. Again, it will save time and space to introduce an abbreviation:

```
\def\thepages@{\page@N{\number\page@C}~%
\page@S{\page@P\page@N{\number\page@C}\page@Q}~%
\number\page@C ~\page@P\page@N{\number\page@C}\page@Q ~}
```

(As with `\thelabels@`, we will be using `\thepages@` within a group where we have stated `\noexpands@`.)

(3) Instead of an `\immediate\write\laxwrite@`, we must use a (delayed) `\write\laxwrite@`. This is necessary to be sure of getting the proper value of `\page@C` into the `.lax` file.

When we encounter a `\pagelabel#1`, we again first use the test

```
\expandafter\ifx\csname#1@L\endcsname\relax
```

which is true if `#1` has not already been used as a `<label>`; in this case, we simply use

```
\expandafter\def\csname#1@L\endcsname{\thepages@2}
\write\laxwrite@{#@#1~\thepages@3}
```

If the test is not true, we again need to look at the type indicator with

```
\edef\next@{~\csname#1@L\endcsname}
\expandafter\getparts@\next@
```


If this test sets \count@ to be even, we issue an error message. But if \count@ is odd we always

```
\write\laxwrite@{#@1~\thepages@3}
```

moreover, we also

```
\expandafter\xdef\csname#1@L\endcsname{\thelabels@2}
```

if \count@ is 1 (but not if it is 3):

```
\def\pagelabel#1{\prevanish@
\expandafter\ifx\csname#1@L\endcsname\relax
{\noexpands@
\expandafter\xdef\csname#1@L\endcsname{\thepages@2}}%
\write\laxwrite@{#@1~\thepages@3}%
\else
{\noexpands@
\edef\next@{~\csname#1@L\endcsname}%
\expandafter\getparts@\next@
\ifodd\count@
\ifnum\count@=1
\expandafter\xdef\csname#1@L\endcsname{\thelabels@2}%
\fi
\write\laxwrite@{#@1~\thepages@3}%
\else
\Err@{Label #1 already used}%
\fi
}%
\fi
\postvanish@}
```

For simplicity, we have again used a single group for all constructions that require the \noexpands@. The \write happens to appear in this group, but as before (page 58), that is irrelevant, since the \write only happens during a \shipout.

Finally, we add \pagelabel to \vanishlist@:

```
\rightadd@pagelabel\to\vanishlist@
```

Page 39 of the \LaTeX Manual mentions that for a counter, say '`\somecounter`', we want to allow such things as

```
\pagelabel{thispage\number\somecounter}
```

In version 1 of \LaTeX , special manipulations were required for this, but now both `\label` and `\pagelabel` automatically allow this, because in both cases the argument #1 is expanded out in all parts of the definition.

Chapter 14. Cross-Referencing

14.1. Preliminaries. We need a flag `\ifreferr@` to tell whether we want error messages or merely warning messages when a `\ref` isn't found:

```
\newif\ifreferr@
\referr@true
\def\RefErrors{\global\referr@true}
\def\RefWarnings{\global\referr@false}
```

And we want to define a routine that prints either the desired error message or the desired warning message. For `TEX` version 2, the best we can do is the following, where `\W@` from `AMS-TEX` stands for `'\immediate\write16 '`:

```
\ifreferr@\Err@{No \noexpand\label found for #1}\else
\W@{Warning: No \noexpand\label found for #1.}\fi}
```

(As before, compare section 3.4 for the use of `\noexpand`.)

But in `TEX` version 3, we can mimic an error message more closely by printing the line number after the warning, using `\inputlineno`:

```
\W@{Warning: No \noexpand\label found for #1.}
\W@{1.\number\inputlineno\space ... #1}
```

(We have to settle for `'...'` since we can't actually capture the contents of the input line.)

It looks even better to print something like

```
1.3 ... \ref{#1}
```

when a `\ref` isn't found, and

```
1.3 ... \Ref{#1}
```

when a `\Ref` isn't found, etc. So we will actually be creating a control sequence with two arguments, the first corresponding to the `\ref` or

`\Ref`, and the second to the `\label`, so that we will print

```
\W@{Warning: No \noexpand\label found for #2.}
\W@{1.\number\inputlineno\space ... \string#1{#2}}}
```

Since we can't be sure when people will be switching to version 3, it seems best to use different code for the two versions. We can check for the version of `TEX` with

```
\setbox0=\hbox{\global\count@='^^30}    (here 0 is 'zero', not 'oh')
```


In `TEX` version 3, `\count@` will then have the value 48, but in `TEX` version 2, it will have the value 115 (and `\box0` will also contain the character 0). Instead of running this test each time we have to print a warning, we will simply let `'\versionthree@'` be undefined in version 2 and `\relax` in version 3:

```
\setbox0=\hbox{\global\count@='^^30}
\ifnum\count@=48 \let\versionthree@=\relax\fi
```

Then we can use a message of the form

```
\ifreferr@Err@{No \noexpand\label found for #2}\else
\W@{Warning: No \noexpand\label found for #2.}%
\ifx\versionthree@\relax
\W@{1.\number\inputlineno\space ... \string#1{#2}}\fi
\fi
```

(The `\global\count@` assignment here doesn't really contradict the policy of section 1.3 because this `\global` assignment is made just once, at the top level, not within a macro.)

 The simpler looking test

```
\setbox0=\hbox{\global\count@='^^00}
```

has all sorts of insidious complications, because in version 3 of `TEX`, `^^00` stands for the ASCII NUL, which is usually an *ignored* character in `TEX`! Since `^^30` is the code for the number 0, it is unlikely to be special.

14.2. \ref and its relatives. For \ref#1 we simply have to use the test

```
\expandafter\ifx\csname#1@L\endcsname\relax
```

to check whether #1 is a label, and then pick out the relevant portion of the value of \csname#1@L\endcsname.

With this in mind, we might first define \nolabel@ by

```
\def\nolabel@#1#2{\expandafter\ifx\csname#2@L\endcsname\relax
\ifreferr@{\Err@{No \noexpand\label found for #2}}\else
\W@{Warning: No \noexpand\label found for #2.}%
\ifx\versionthree@\relax
\W@{1.\number\inputlineno\space ... \string#1{#2}}\fi
\fi
\else}
```

so that \nolabel@#1#2 would give an error or warning message if #2 hasn't been used as a label, and otherwise do whatever follows. However, it will be a little more convenient to define \nolabel@#1#2#3, with an extra argument, #3, that is often given the value \relax:

```
\def\nolabel@#1#2#3{%
\expandafter\ifx\csname#2@L\endcsname\relax
\ifreferr@{\Err@{No \noexpand\label found for #2}}\else
\W@{Warning: No \noexpand\label found for #2.}%
\ifx\versionthree@\relax
\W@{1.\number\inputlineno\space ... \string#1{#2}}\fi
\fi
#3\else}
```

We also want a routine that sets a scratch token to the actual value of the control sequence \csname#1@L\endcsname; because this will have to be defined within a group, we use the scratch token \Next@, reserved for \global assignments (page 22):

```
\def\csL@#1{{\noexpands@\xdef\Next@{\csname#1@L\endcsname}}}
```

Now `\ref#1` should print an error or warning message if #1 is not a `\label`,

```
\def\ref#1{\nolabel@\ref{#1}\relax
. . .
```

Otherwise, it should print everything up to the first `~` in the value of the control sequence `\csname#1@L\endcsname`. If we define

```
\def\nextii@#1~#2\nextii@{#1}
```

and also use

```
\csL@{#1}
```

to set `\Next@` to the value of `\csname#1@L\endcsname`, then we just have to use

```
\expandafter\nextii@\Next@\nextii@
```

So the definition of `\ref` is

```
\def\ref#1{\nolabel@\ref{#1}\relax
\def\nextii@##1~##2\nextii@{##1}%
\csL@{#1}\expandafter\nextii@\Next@\nextii@\fi}
```

Similarly for

```
\def\Ref#1{\nolabel@\Ref{#1}\relax
\def\nextii@##1~##2~##3\nextii@{##2}%
\csL@{#1}\expandafter\nextii@\Next@\nextii@\fi}
```

and

```
\def\nref#1{\nolabel@\nref{#1}\relax
\def\nextii@##1~##2~##3~##4\nextii@{##3}%
\csL@{#1}\expandafter\nextii@\Next@\nextii@\fi}
```

and

```
\def\pref#1{\nolabel@\pref{#1}\relax
\def\nextii@##1~##2~##3~##4~##5\nextii@{##4}%
\csL@{#1}\expandafter\nextii@\Next@\nextii@\fi}
```

For later purposes, we will

```
\let\pref@=\pref
```

so that \pref@ may be used to reinstate the usual meaning of \pref if it has been redefined.


The definition of \Evaluatenref is somewhat different—we use the generality built into \nolabel@ to \gdef\Nref{-10000 } when #1 isn't a label:

```
\def\Evaluatenref#1{%
\nolabel@\Evaluatenref{#1}{\gdef\Nref{-10000 }}%
\def\nextii@##1~##2~##3~##4~##5\nextii@{\def\nextii@{##3}}%
\csL@{#1}\expandafter\nextii@\Next@\nextii@
\xdef\Nref{\nextii@}\fi}
```

\Evaluatepref is analogous:

```
\def\Evaluatepref#1{%
\nolabel@\Evaluatepref{#1}{\global\let\Pref=\empty}%
\def\nextii@##1~##2~##3~##4~##5\nextii@{\def\nextii@{##4}}%
\csL@{#1}\expandafter\nextii@\Next@\nextii@
\xdef\Pref{\nextii@}\fi}
```

Note that the <label> is expanded out in both \nolabel@ and \csL@. Consequently (compare page 88), \Evaluatenref and \Evaluatepref can have arguments containing \number\somcounter for a counter ‘\somcounter’; in fact, even \ref, \Ref, \pref and \nref can contain such arguments.

 Applications of \Evaluatenref and \Evaluatepref are given on pages 38–39 of the L^AT_EX Manual. ‘\Evaluaterref’ and ‘\EvaluateRef’ haven't been provided, on the grounds that in similar situations one would be able to determine the value of \ref from that of \nref, and the value of \Ref from that of \pref.

Chapter 15. Reading auxiliary files

Although `\readlax` is similar to `\document`, there are a few significant differences.

15.1. \readlax. `\readlax#1` first opens the file `#1.lax` for reading, giving a conspicuous message if the file isn't found, i.e., if the test `\ifeof` is true at the beginning. (Note that `\document` *doesn't* give a message if `\jobname.lax` isn't found.) Like `\document`, it then processes the contents of this file one line at a time, except that it will read the line into the control sequence `\nextv@`. Again, this is done with a `\loop` that repeats until `\ifeof` detects the end of the file.

As with `\document`, we want to ignore the case where `\nextv@` is empty. Now, however, we also want to ignore the lines beginning with `F`, because information about the numbering of footnotes from another file would conflict with information gathered for the current file.

So the definition of `\readlax#1` begins

```
\def\readlax#1{\immediate\openin\xaxread@=#1.lax\relax
\ifeof\xaxread@\W@{ }\W@{File #1.lax not found.}\W@{ }\fi
{\endlinechar=-1 \noexpands@
\catcode'\@=11 \catcode'\~=11
\loop \ifeof\xaxread@ \else
\read\xaxread@ to\nextv@
\ifx\nextv@\empty
\else
\expandafter\Finit@\nextv@\Finit@
\ifx\nextii@ F%
\else
```

When a suitable line `\nextv@` has been found, we use

```
\expandafter\getparts@\nextv@
```

to stores the `<label>` part of `\nextv@` in `\nextiii@`, the value of the `<type indicator>` in the counter `\count@`, and the rest of `\nextv@`, except for the initial `~`, in `\nextiii@`.

Then we have to use the test

```
\expandafter\ifx\csname\nextiv@ @L\endcsname\relax
```

to see if the `<label>` part of `\nextiv@` has already been used, because this represents a conflicting use of `<label>`, so that we need to issue an error message (remember that `\readlax` can be used at any time, possibly after some `<label>`'s have already been made).

If this test is true, so that the `<label>` does not appear, we will define `'\<label>@L'` to be the remainder of `\nextiv@`, *except that we will change the* (type indicator) to 0 if it is 1 and to 2 if it is 3. For this we can use

```
\edef\next@{\gdef\csname\nextiv@ @L\endcsname
  {\nextiii@\ifnum\count@=1 0\else 2\fi}}
\next@
```

(within a group with `\noexpands@`).

As a consequence of this arrangement, any `<label>` from the auxiliary file that we read in with `\readlax` will count as a “current” label, so that if `\label{<label>}` occurs later in the file we will get an error message, rather than changing the data for this `<label>`. That seems like the reasonable arrangement, since we use `\readlax` to get information from what is presumably a different part of the same document, and the same `\label{<label>}` shouldn't appear in two different parts of the document (at least, not if we intend to combine the labels in the two parts with `\readlax`).

Our whole definition is

```
\def\readlax#1{\immediate\openin\xaxread@=#1.lax\relax
\ifeof\xaxread@\W@{\}\W@{File #1.lax not found.}\W@{\}\fi
{\endlinechar=-1 \noexpands@
\catcode'\@=11 \catcode'\~=11
\loop \ifeof\xaxread@ \else
\read\xaxread@ to\nextiv@
\ifx\nextiv@\empty
\else
\expandafter\Finit@\nextiv@\Finit@
\ifx\nextii@ F%
```

```

\else
\expandafter\getparts@\nextv@
\expandafter\ifx\csname\nextiv@ @L\endcsname\relax
\edef\next@{\gdef\csname\nextiv@ @L\endcsname
{\nextiii@\ifnum\count@=1 0\else 2\fi}}%
\next@
\else
\Err@{Label \nextiv@\space in #1.lax already used}%
\fi
\fi
\fi
\repeat}%
\immediate\closein\laxread@}

```

At this point we are finished with all definitions that involve ~ with category code 11:

```
\catcode'\~=\active
```

15.2. Style files. Control sequences to read in style files are simple:

```

\def\docstyle#1{\input #1.st\relax}
\def\predocstyle#1{\input #1.stf\relax}
\def\postdocstyle#1{\input #1.stb\relax}

```

Part III

***Particular
Constructions
Allowing Labels
and their associates***



Chapter 16. Displayed formulas

The next part of \LaTeX is concerned with displayed formulas and the `\tag` mechanism; it is the first place where we define `\thelabel@ . . .`. However, numerous special considerations for `\tag` are also required, and sections 2 and 3 are the only ones specifically related to the `\label` mechanism.

16.1. Invisibility. An “invisible” construction following a display, in a case like

```
$$  
.  
.  
.  
$$\label{...}\more text
```

or even

```
$$  
.  
.  
.  
$$\label{...}\more text
```

presents the same problem as an invisible construction following `\noindent` (section 7.2): The `\prevanish@` sets `\saveskip@` to 0pt (even in the second case, because \TeX ignores a space after the `$$` that end a display). Consequently, the `\postvanish@` does not skip the space following the `\label{ . . . }`, and we end up with an extra space before ‘more text’.

There doesn’t seem to be any correction that we can add to `\prevanish@` to address this problem, because there is apparently no way to tell when a \TeX construction happens to appear immediately after a display. To get around this problem, whenever \LaTeX encounters the `$$` that begin a display, it calls a control sequence that reads in everything up to the closing `$$` as its argument, and then puts back both this argument and the closing `$$`, together with the proper compensating mechanism:

```
\everydisplay{\csname displaymath \endcsname}  
\expandafter\def\csname displaymath \endcsname#1$$\{%  
#1$$\FNSS@\pretendspace@}
```


Note that here we need `\FNSS@` (section 3.8), not just `\futurelet\next`, since we have to skip over any space after the final `$$`.

We will introduce the abbreviation

```
\def\FNSSP@{\FNSS@\pretendspace@}
```

not only because it will save numerous tokens, but also because at one point (section 25.2), it will be essential to have it. So we use

```
\everydisplay{\csname displaymath \endcsname}
\expandafter\def\csname displaymath \endcsname#1$${%
#1$$\FNSSP@}
```

 The control sequence

```
\csname displaymath \endcsname
```

(compare `\csname align \endcsname` etc., as explained in `amstex.doc`) shows up on the screen as

```
\displaymath_
```


so if a blank line occurs before the closing `$$`, we will get the error message

```
! Paragraph ended before \displaymath_ was complete.
```

[If we defined `\displaymath_` to be `\long`, we would get basically the same error message that T_EX normally gives,

```
! Missing $ inserted.
```

but it would be presented in a much more confusing way, since “context lines”, involving the argument of `\displaymath_`, would also be presented.]

 Notice that a construction like

```
$$ . . . $$
\bye
```

will eventually `\let\next=\bye` and then call `\pretendspace@`, which uses the test `\ismember@\vanishlist@\next`. If `\bye` were `\outer` we would get an error message

```
! Forbidden control sequence found while scanning use of \ismember@.
```

Unlike the situation for `\noindent` (section 7.2), which is presumably used only when some text is going to follow, if we have something like

```
A line of text.
$$
. . .
$$
\label{...}
```

```
Another line of text.
```

the `\hskip-1pt\hskip1pt` added by the `\prevanish@` in `\label` will have a dire effect: it will cause a *blank line* to be typeset after the display. Since there will be `\baselineskip` glue before this blank line, the next line, ‘Another line of text.’ will be separated from the display by too much space.

But there’s really nothing that can be done about this. Even in plain \TeX ,

```
A line of text.
$$
. . .
$$
\writen{...}
```

```
Another line of text.
```


will create a spurious blank line after the display. So users just have to be warned against using “invisible” constructions after a display that ends a paragraph.

Because of this (admittedly convoluted) approach to this (admittedly rather special) problem, constructions that change category codes won’t work within a displayed formula. If that needs to be allowed (as it sometimes was for the \LaTeX - \TeX Manual), one can

```
\def\Math{\begingroup\everydisplay{}}
\def\endMath{$$\endgroup\futurelet\next\pretendspace@}
```

and then use `\Math... \endMath` instead of `$$... $$`. (We don't need `\FNSS@` here, since there won't be a space token after the control sequence `\endMath`.)

Some people, of course, might argue that this is really the "logical" way to do things anyway. If such a definition were to be instituted permanently, then the basic $\mathcal{A}_M\mathcal{S}$ - $\mathcal{T}_E\mathcal{X}$ definition of `\tag` (see (A) below) would have to be changed, so that the argument of `\tag` is delimited by `\endMath` rather than by `$$`. (When the $\mathcal{L}_M\mathcal{S}$ - $\mathcal{T}_E\mathcal{X}$ Manual required literal mode in a displayed formula with a `\tag`, the literal mode material was first set in a box, which was then used within the usual `$$... \tag$$` construction.) It should also be noted that the $\mathcal{A}_M\mathcal{S}$ - $\mathcal{T}_E\mathcal{X}$ construction `\align... \endalign` must read in ... as an argument, in which case category code changes within ... will still be ignored. The same is true of the $\mathcal{L}_M\mathcal{S}$ - $\mathcal{T}_E\mathcal{X}$ `\CD... \endCD` construction. So, all in all, `\Math... \endMath` should be reserved for special effects.

 There is a way of allowing category changes within a display `$$... $$` if we are willing to change the category code of `$` itself:

```

\let\dollar@=$
\newif\ifindisplay@
\catcode'\$=\active
\def$ {\futurelet\next\dollar@@}
\def\dollar@@{%
\ifx\next$%
\ifindisplay@
\def\next@${\dollar@\dollar@\FNSSP@}%
\else
\def\next@${\dollar@\dollar@\indisplay@true}%
\fi
\else
\let\next@=\dollar@
\fi
\next@}

```

(We never need to set `\indisplay@false`, since our `\indisplay@true` occurs within the display, which $\mathcal{T}_E\mathcal{X}$ implicitly encloses within a group.)

I wanted to avoid additional category changes as much as possible, but perhaps this approach is really preferable.

If such definitions were made, we would also have to restate any definitions that involved `$` as part of their syntax. For example, we would have to restate $\mathcal{A}_M\mathcal{S}$ - $\mathcal{T}_E\mathcal{X}$'s

definition of `\tag`,

```
(A)      \def\tag#1${\iftagsleft@ \leqno \else \eqno \fi
          \maektag@#1 \maketag}}}
```

at some point after `$` has been made active.

It would naturally also be sensible to replace any `$. . . $` combinations in macros by `\dollar@. . . \dollar@` (many such appear in the $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$ macros, or their replacements in $\mathcal{L}\mathcal{A}\mathcal{T}\mathcal{E}\mathcal{X}$, e.g., those in sections 4 and 5).

But a `$` in an `\xdef` (e.g., as on page 106) would instead have to be replaced by `\noexpand$`. Moreover, we would have to add `'\let$=\relax'` to `\noexpandtoks@` (or, equivalently, state `\Nonexpanding$`).

bd One advantage of this approach, by the way, is that one could modify the definition of `\dollar@@` to be:

```
\def\dollar@@{%
  \ifx\next$%
    . . .
  \else
    \ifhmode
      \def\next@{\saveskip@=\lastskip\unskip\/%
        \ifdim\saveskip@>0pt \hskip\saveskip@\fi\dollar@}%
    \else
      \let\next@=\dollar@
    \fi
  \fi
  \next@}
```

With such a definition, something like

(1) If $|x| < 3$, then ...

would be treated as if it had been typed

(2) If $\sloppy |x| < 3$, then ...

(On the other hand, input (2) would remain as is, since `\sloppy` has no effect except after a character or ligature.)

This arrangement is quite useful if the current font is slanted or italic, as in the statement of a `\claim`, because

If $|x| < 3$, then ...

looks much better if the italic correction is added:

If $|x| < 3$, then ...

(See page 185 for other examples where the italic correction seems called for.) I used this approach in the style file for the Publish or Perish *Mathematics Lecture Series*.

As far as I can tell, there is no other way of achieving this: `\everymath` doesn't do any good, because a `$` puts a 'mathon' on the horizontal list, and one can't remove it, in order to `\unskip`.

16.2. Localizing labels. As already mentioned in section 11.1, \LaTeX uses `\thelabel@`, `\ref`, `\pref` associated to a `\langle label \rangle`, and any \LaTeX construction that can be given a `\langle label \rangle` must define these control sequences. It turns out that these four components will always have to be defined *globally*, because they have to be made within a group beginning with `\noexpands@`—compare page 71—even though we want `\thelabel@`, `\ref`, to be defined only locally. So \LaTeX uses `\TheLabel@`, `\TheLabel@@@` for the four components that it defines globally, and then uses the constructions

```
\let\thelabel@=\TheLabel@, ...
```

to create locally defined ones. Since this is used so often, we abbreviate it:

```
\def\locallabel@{\let\thelabel@=\TheLabel@
\let\thelabel@@=\TheLabel@@\let\thelabel@@@=\TheLabel@@@
\let\thelabel@@@@=\TheLabel@@@@}
```

16.3. `\tag`. We introduce the counter `\tag@C` for `\tag`, and the initial values for the other associated things for labelling:

```
\newcount\tag@C
\tag@C=0
\let\tag@P=\empty
\let\tag@Q=\empty
\def\tag@S#1{\rm(#1/\rm)}
\let\tag@N=\arabic
\def\tag@F{\rm}
```

As in the case of \page@F (page 51), we use \def\tag@F{\rm} rather than \let\tag@F=\rm, so that \fontstyle\tag will work correctly. In \tag@S we specified roman parentheses even if \tag@F is changed (compare page 74); of course, \tag@S could be changed if we didn't want this.

Unlike the situation for \page@N, we can \let\tag@N=\arabic, because \tag@N appears only in certain \xdef's (page 106), and then this value of \tag@N will simply disappear, resulting in a shorter string than if we kept \arabic unexpanded.

NOTE: Nevertheless, we must use \def instead of \let for other numbering styles. Note that \newnumstyle does this (Chapter 24).

$\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$ already defines the combination \tag#1\$\$ to be \leqno or \eqno followed by \maketag@#1\maketag\$\$, and we just have to redefine \maketag@ for $\mathcal{L}\mathcal{A}\mathcal{T}\mathcal{E}\mathcal{X}$, although the argument #1 will now have a rather different significance: in $\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$, #1 would be the tag number that we want to use, but now #1 could instead involve things like \label or \pagelabel, or even \Reset\tag (to affect the next \tag), as well as a "quoted" tag number "...", again followed by things like \label or \pagelabel.

The action of \maketag@ will depend on whether or not it is followed by a " for a "quoted" \tag.

```
\def\maketag@{\futurelet\next\maketag@@}
```

However, for reasons that will be explained in section 4, if \maketag@ is followed by \relax"... " we will want to get the same result as if it were followed simply by "...". So we will call \maketag@@@ if \maketag@ is followed by " and \maketag@@@ if it is followed by anything else other than \relax. But if \maketag@ is followed by \relax, we call a control sequence that swallows this \relax and then reiterates the process:

```
\def\maketag@@{\ifx\next\relax
\def\next@\relax{\futurelet\next\maketag@@}\else
\ifx\next"\let\next@=\maketag@@@\else
\let\next@=\maketag@@@\fi\fi\next@}
```

The definition of \maketag@@@ (when \tag is not followed by a ") illustrates the general scheme by which $\mathcal{L}\mathcal{A}\mathcal{T}\mathcal{E}\mathcal{X}$ deals with labels.

- (1) First we globally advance the `\tag@C` counter, `\tag@C`, by one.
- (2) Then we `\xdef` the values of `\TheLabel@`, `\TheLabel@@@` appropriately, using the current values of `\tag@C`, `\tag@P`, etc.
- (3) Then we use `\locallabel@` to (locally) set the values of `\thelabel@`, `\thelabel@@@`.
- (4) Then we actually typeset the tag, using `\tag@S` for the style, and in the font `\tag@F` (which will be irrelevant if we have `\TagsAsMath`).

In step 2 we will be doing something like

```
{\noexpands@
\xdef\TheLabel@@@{\number\tag@C}
\xdef\TheLabel@{\tag@N{\TheLabel@@@}}
\xdef\TheLabel@@@{\ifmathtags@$\tag@P\TheLabel@\tag@Q$\else
\tag@P\TheLabel@\tag@Q\fi}
\xdef\TheLabel@@{\tag@S{\TheLabel@@@}}
}
```

All of the \LaTeX constructions that can be given a `\label` will define `\TheLabel@`, `\TheLabel@@@` in much the same way (with `\tag@C` replaced by the suitable `\...@C` counter, etc.). Consequently, it is worth introducing an abbreviation for the steps defining `\TheLabel@` and `\TheLabel@@@`:

```
\def\xdefTheLabel@#1{\xdef\TheLabel@{#1{\TheLabel@@@}}
\def\xdefTheLabel@@#1{\xdef\TheLabel@@{#1{\TheLabel@@@}}}
```

Thus, we define `\maketag@@@` by

```
\def\maketag@@@#1\maketag@{\global\advance\tag@C by 1
{\noexpands@
\xdef\TheLabel@@@{\number\tag@C}%
\xdef\TheLabel@{\tag@N
\xdef\TheLabel@@@{\ifmathtags@
$\tag@P\TheLabel@\tag@Q$\else
\tag@P\TheLabel@\tag@Q\fi}%
\xdef\TheLabel@@{\tag@S
}%
}
```

```
\locallabel@
\hbox{\tag@F\thelabel@@}%
#1}
```

When \tag is followed by ", it might occur in constructions like

```
\tag "\style{...}" or \tag "\style{\pre 3}" etc.,
```

involving any of \pre, \post, \style, or \numstyle. In such situations, \pre must be interpreted as \tag@P, etc., when the tag is printed,

```
{\let\pre=\tag@P \let\post=\tag@Q
 \let\style=\tag@S \let\numstyle=\tag@N
 \hbox{\tag@F...}}
```

Moreover, \TheLabel@, ..., \TheLabel@@@ must be suitably interpreted.

For this we use a routine that is also used by many other constructions:

```
\def\Qlabel@#1-{\noexpands@\xdef\TheLabel@@{#1}%
 \let\style=\empty\xdef\TheLabel@@@{#1}%
 \let\pre=\empty\let\post=\empty\xdef\TheLabel@{#1}%
 \let\numstyle=\empty\xdef\TheLabel@@@{#1}}}
```


For example, \Qlabel@{#1} for \tag"#1" will be used after we have \let\pre=\tag@P, etc. So

- (1) \TheLabel@@ (which is what \Ref is supposed to produce), will give #1, with \pre interpreted as \tag@P, etc.
- (2) \TheLabel@@@ (what \pref is supposed to produce), will be the same, except \style will be ignored if it appears, since \pref is supposed to produce what \Ref produces, but without any of the \style formatting.
- (3) \TheLabel@ (what \ref is supposed to produce), also ignores \pre and \post if they appear.
- (4) \TheLabel@@@ (what \nref is supposed to produce), also ignores \numstyle if it appears.

```


\def\maketag@@@#1#2\maketag@{%
  {\let\pre=\tag@P \let\post=\tag@Q
   \let\style=\tag@S \let\numstyle=\tag@N
   \hbox{\tag@F#1}%
   \noexpands@
   \Qlabel@{#1}%
   }%
 \locallabel@
 #2}

```

 It might seem unlikely that any one would use a `\label` for a “quoted” `\tag`, since then the tag is already known. But the label mechanism is provided nevertheless, and might even be of some use. For example,

```
$$ ... \tag"\style{\pre A}" \label{tagA}$$
```

might be used to produce the tag ‘(3.A)’, where the user wouldn’t know what precedes the ‘A’. In this case, `\Ref{tagA}` or `\pref{tagA}` would be needed to print ‘(3.A)’ or ‘3.A’ in the text.

 Because of the `\xdef`’s in `\Qlabel`, any “quoted” number “...” following `\tag` (or any other construction that allows a (label)) must contain only things that can safely be used in `\xdef`’s when `\noexpands@` is in force.

As a far-fetched example, in this manual Chapter 11 was labelled ‘STARTLABEL’ and Chapter 15 was labelled ‘ENDLABEL’. If the next chapter were commentary on these chapters, and for some reason we wanted this commentary to be called

Chapter 11’–15’. Remarks on Labels and Cross References

we wouldn’t be able to type

```
\chapter "\nref{STARTLABEL}$’$--\nref{ENDLABEL}$’$” Remarks ...
```

Instead, we would have to use something like

```

\Evaluatenref{STARTLABEL}
\edef\startlabel{\Nref}
\Evaluatenref{ENDLABEL}
\edef\endlabel{\Nref}
\chapter "\startlabel$’$--\endlabel$’$” Alternate ...

```

In such a case, we might even use a \label with this “quoted” number,

```
\chapter "... " . . . \label{...} \endchapter
```

in order to refer to this new chapter later on.

16.4. \align. This section presumes familiarity with $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$'s \align construction (see `amstex.doc`). Before considering this construction in detail, we need to consider one aspect of the general scheme.

When $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$ sees something like

```
\align
<formula1> & <formula2> \tag <formula number> \\
. . .
```

the \align is an \halign whose preamble contains three &'s, of the form

```
\halign{ ...#... & ...#... & ... \maketag@#\maketag@ ... \cr
. . .
```

Within the \align, a \tag is simply interpreted as an &. But this means that after a \tag, $\mathcal{T}\mathcal{E}\mathcal{X}$ will examine the next token, *expanding if necessary*, to see if a \noalign or \omit follows (*The $\mathcal{T}\mathcal{E}\mathcal{X}$ book*, page 240). So if

```
<formula1> & <formula2> \tag "... " \\
```

appears, the first " *will already be expanded out to \futurelet\next\quote@* before being submitted to \maketag@. Consequently, \maketag@ will think that the next token is \futurelet, rather than "!

To get around this, we will instead have \tag interpreted as &\relax, to insure that the " is not expanded out. Thus, \tag"... " will lead to

```
\maketag@\relax"... "\maketag@
```

and we have already arranged for this to give the same result as if the \relax weren't there (section 3).

The definition of \tag for \align is made by $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$'s control sequence \align@, which is now redefined to include the \relax, together with other changes that will be explained later.

```

\def\align@{\inalign@true\inany@true
\vspace@\allowdisplaybreak@\displaybreak@\intertext@
\def\tag{\global\tag@true\ifnum\and@=0
\def\next@{\&\omit|\global\rwidth@=0pt&\relax}}\else
\def\next@{\&\relax}\fi\next@}%
\iftagsleft@\def\next@{\csname align \endcsname}\else
\def\next@{\csname align \space\endcsname}\fi\next@}

```

Still further changes to the rest of the $\mathcal{A}_{\mathcal{M}}\mathcal{S}$ -TEX `\align` construction itself are also required, because the remainder of this construction first calls `\measure@` to measure all the formulas, before actually setting them. Even in $\mathcal{A}_{\mathcal{M}}\mathcal{S}$ -TEX, this can produce problems: if a user puts a constructed `\box` within an `\align`, it will be emptied out by `\measure@` and hence produce nothing at all when the typesetting is done (this means that the user needs to use `\copy` rather than `\box`).

In $\mathcal{L}_{\mathcal{M}}\mathcal{S}$ -TEX, the problem is that we want to ignore any `\Reset\tag` and `\Offset\tag` constructions during the `\measure@`, since these globally change the `\tag` counter, and we don't want to do that twice (`\Reset` and `\Offset` are described in Chapter 24).

We introduce a special abbreviation for a more general construction that disables `\Offset` and `\Reset` (but keeps them invisible):

```

\def\noset@{\def\Offset##1##2{\prevanish@\postvanish@}}%
\def\Reset##1##2{\prevanish@\postvanish@}

```

Now we redefine `\measure@` so that `\noset@` is used:

```

\def\measure@#1\endalign{%
\global\lwidth@=0pt \global\rwidth@=0pt
\global\maxlwidth@=0pt \global\maxrwidth@=0pt
\global\and@=0
\setbox0=\vbox
{\noset@}\everycr{\noalign{\global\tag@false
\global\and@=0}}\Let@

```

```

\halign{\setbox0=\hbox{\m@th\displaystyle{\@lign##}\$}%
\global\lwidth@=\wd0
\ifdim\lwidth@>\maxlwidth@ \global\maxlwidth@=\lwidth@\fi
\global\advance\and@ by 1
&\setbox0=\hbox{\m@th\displaystyle{\@lign##}\$}%
\global\rwidth@=\wd0
\ifdim\rwidth@>\maxrwidth@ \global\maxrwidth@=\rwidth@\fi
\global\advance\and@ by 1
&\Tag@{\eat@{##}\crrcr#1\crrcr}}%
\totwidth@=\maxlwidth@ \advance\totwidth@ by \maxrwidth@}

```

(Here `\eat@` from *AMS-TeX* is defined by `\def\eat@#1{}`, see section 1.2.) The assignments of `\lwidth@`, ... at the beginning of the definition don't really have to be `\global`, but they do have to be `\global` later in the definition, so we make them `\global` at all times (compare section 1.3).

Now, in the definition of '`\alignl`' and '`\alignll`' (called by `\align`), after we `\measure@` we can allow `\Offset` and `\Reset` to have their usual meanings, but there is a different problem. When we have something like

$$\langle \text{formula}_1 \rangle \ \& \ \langle \text{formula}_2 \rangle \ \backslash \text{Offset} \backslash \text{tag} 0 \ \backslash \text{newpost} \backslash \text{tag} \{ \$ ' \$ \} \ \backslash \text{tag} \ \backslash \backslash$$

the new values of `\tag@C` and `\tag@P` created by `\Offset` and `\newpost` are processed as part of `\langle \text{formula}_2 \rangle` (since the `&` that ends this formula is supplied by the `\tag`). But the new value of `\tag@P` is created only *locally*, so it will not be known when `\tag` has to do its work. To get around this problem, we need to pass such information through by means of two new globally defined control sequences, `\tag@P@` and `\tag@Q@`, and we make a special abbreviation for this:

```

\def\prepost@{\global\let\tag@P@=\tag@P
\global\let\tag@Q@=\tag@Q}

```

We also have an abbreviation for the construction that (locally) sets `\tag@P` and `\tag@Q` to the globally defined `\tag@P@` and `\tag@Q@`:

```

\def\reprepost@{\let\tag@P=\tag@P@\let\tag@Q=\tag@Q@}

```

Now we redefine `\alignl` and `\alignlll` from $\mathcal{A}_{\mathcal{M}\mathcal{S}}\text{-TEX}$ using `\prepost@` within the formulas, and using `\reprepost@` when the formula number is being printed; note that we use `\prepost@` at the *end* of each formula, so that any `\newpre` or `\newpost` will have been discovered, but `\reprepost@` at the *beginning* of each formula number. (These maneuvers aren't needed for `\measure@`, since we simply `\eat@` each formula number.)

First comes '`\alignlll`':

```

\expandafter\def\csname align \space\endcsname#1\endalign
{\measure@#1\endalign\global\and@=0
\ifingather@\everycr{\noalign{\global\and@=0}}\else
\display@{\fi
\let@ \tabskip\centering@
\halign to\displaywidth
{\hfil\strut@\setbox0=\hbox{${m@th\displaystyle
{\@lign##\prepost@}}}$}%
\box0 \global\advance\and@ by 1
\tabskip\z@skip&
\setbox0=\hbox{${m@th\displaystyle{\{}{\@lign##\prepost@}}}$}%
\global\rwidth@=\wd0
\box0 \hfil \global\advance\and@ by 1
\tabskip\centering@ &
\setbox0=\hbox{\@lign\strut@\reprepost@
\maketag@##\maketag@}}%
\dimen@=\displaywidth \advance\dimen@ by -\totwidth@
\divide\dimen@ by 2 \advance\dimen@ by \maxrwidth@
\advance\dimen@ by -\rwidth@
\ifdim\dimen@<2\wd0
\llap{\vtop{\normalbaselines\null\box0}}}%
\else\llap{\box0}\fi
\tabskip\z@skip
\crr#1\crr
\black@\totwidth@}}

```

Note that in a formula like

```
\align
. . .
<formula> \newpre\tag{a}\tag ... \\\
. . .
```

where there is a line without any &, the definition of \tag in \align@ (page 110) means that this line will be interpreted as

```
<formula> \newpre\tag{a}& \omit\global\rwidth@=0pt &\relax
\reprepost@ \maketag@... \maketag@ \cr
```

The \prepost@ in the first part of the preamble for \align globally defines \tag@P@ to be 'a', and the \reprepost@ then makes \tag@P properly defined for \maketag@. But if the \omit were not inserted, then the \prepost@ for the *second* part of the preamble would globally redefine \tag@P@ to be the default value of \tag@P, so that the new value would not be properly propagated to the \maketag@ part of the preamble.

Since we've added the \omit, we also have to add \global\rwidth@=0pt, which would normally be set by an empty formula. The extra clause

```
\ifdim\rwidth@>\maxrwidth@ \global\maxrwidth@=\rwidth@\fi
```

appearing in the definition of \measure@ doesn't have to be duplicated, since it would be inoperative for an empty formula.

The \tabskip\centering@ also appears after the first & in the preamble, but we don't have to worry about that: once the preamble has been read, the various \tabskip glues are determined, and they remain the same for any row, even those that have \omit's in them.

(The '\global\advance\and@ by 1' will also be omitted because of the \omit, but this is irrelevant, since the current value of \and@ has already been used to determine properly the meaning of \tag.)

The re-definition of '\align₁' is exactly analogous:

```
\expandafter\def\cename align \endcename#1\endalign
{\measure@#1\endalign\global\and@=0
```

```

\ifdim\totwidth@>\displaywidth\let\displaywidth@=\totwidth@
\else\let\displaywidth@=\displaywidth\fi
\ifingather@\everycr{\noalign{\global\and@=0}}\else
\display@ \fi
\let@ \tabskip \centering@
\halign to \displaywidth
{\hfil \strut@ \setbox0=\hbox{\$ \m@th \displaystyle
  {\@lign## \prepost@} }$}%
\global \linewidth@=\wd0 \global \lineht@=\ht0
\box0 \global \advance \and@ by 1
\tabskip \z@skip & \setbox0=\hbox{\$ \m@th \displaystyle {} \@lign
  ## \prepost@} }$}%
\ifdim\ht0>\lineht@ \global \lineht@=\ht0 \fi
\box0 \hfil \global \advance \and@ by 1
\tabskip \centering@ & \kern-\displaywidth@
\setbox0=\hbox{\@lign \strut@ \reprepost@
  \maketag@## \maketag@}%
\dimen@=\displaywidth \advance \dimen@ by -\totwidth@
\divide \dimen@ by 2 \advance \dimen@ by \maxlinewidth@
\advance \dimen@ by -\linewidth@
\ifdim \dimen@<2\wd0
\rlap{\vbox{\normalbaselines \box0 \vbox to \lineht@{}}}%
\else \rlap{\box0} \fi
\tabskip \displaywidth@ \crrc#1 \crrc
\black@ \totwidth@}

```

The extra clause

```
\ifdim\ht0>\lineht@ \global \lineht@=\ht0 \fi
```

appearing in the second part of the preamble doesn't have to be added to the definition of `\tag` in `\align@`, along with the `\omit`, since it would always be inoperative for a blank formula.

16.5. `\alignat` and `\xalignat`. This section presumes familiarity with *AMS-TeX's* `\alignat` and `\xalignat` constructions, which also require changes (no changes are required for `\xxalignat`, which doesn't allow `\tag's`).

\alignat and \xalignat process their arguments twice, like \align, although the initial processing provides much less information. In these constructions a \tag might even end up overlapping a formula, but at least a black box is produced at the end, to indicate such overlapping (which is easy to miss when proofreading). The initial processing is done with empty \tag's, but as if the \tag's were set at their minimum distance from the formula (i.e., at a distance equal to their widths), and the result is saved in \box\savealignat@. During the second processing the only role played by this box is in the

```
\black@{\wd\savealignat@}
```

that is appended after the \halign, to give a black box if the whole construction turns out to be too wide.

Unlike the situation for \align, where we specify the preamble for the first process in \measure@, the $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$ routine \attag@ is used to produce the preambles for both processes of \alignat and \xalignat, using a flag \ifmeasuring@ to determine whether we are making a preamble for the first process or for the second, as well as the flag \ifxat@ to determine whether we are making preambles for \alignat or for \xalignat.

The definition of \attag@ from $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$ has to be modified in the same way as the definitions of \measure@, '\align_{\l}' and '\align_{\r}', by inserting \prepost@ and \reprepost@ in the appropriate places:

```
\def\attag@#1{\let\Maketag@=\maketag@\let\TAG@=\Tag@
\let\Prepost@=\prepost@|\let\Reprepost@=\reprepost@
\let\Tag@=\relax\let\maketag@=\relax
\let\prepost@=\relax|\let\reprepost@=\relax
\ifmeasuring@
\def\llap@##1{\setbox0=\hbox{##1}\hbox to2\wd0{}}%
\def\rlap@##1{\setbox0=\hbox{##1}\hbox to2\wd0{}}%
\else\let\llap@=\llap\let\rlap@=\rlap\fi
\toks@={\hfil\strut@
$\m@th\displaystyle{\@lign\the\hashtoks@prepost@}}$%
\tabskip\z@skip\global\advance\and@ by 1 &
$\m@th\displaystyle{\@lign\the\hashtoks@\prepost@}}$\hfil
\ifxat@\tabskip\centering@\fi\global\advance\and@ by 1}%
```

```

\iftagsleft@
\toks@={\tabskip\centering@&\Tag@\kern-\displaywidth
\rlap@{\@lign\reprepost@
\maketag@\the\hashtoks@\maketag@}%
\global\advance\and@ by 1 \tabskip\displaywidth}\else
\toks@={\tabskip\centering@&\Tag@
\llap@{\@lign\reprepost@\maketag@
\the\hashtoks@\maketag@}\global\advance\and@ by 1
\tabskip\z@skip}\fi
\atcount@#1\relax\advance\atcount@ by -1
\loop\ifnum\atcount@>0
\toks@=\expandafter{\the\toks@&\hfil
$m@th\displaystyle{\@lign\the\hashtoks@\prepost@}$%
\global\advance\and@ by 1 \tabskip\z@skip&
$m@th\displaystyle{\}\@lign\the\hashtoks@
\prepost@}$\hfil
\ifxat@\tabskip\centering@\fi\global\advance\and@ by 1}%
\advance\atcount@ by -1
\repeat
\xdef\preamble@{\the\toks@\the\toks@}%
\xdef\preamble@{\preamble@}%
\let\maketag@=\Maketag@\let\Tag@=\TAG@
\let\prepost@=\Prepost@|\let\reprepost@=\Reprepost@}

```

Now we must modify the definition of ‘`\alignatn`’, which is called by `\alignat`, in several ways:

- (1) `\tag` must mean `&\relax`;
- (2) More generally, `\tag` must mean `&\omit&\relax` if used when one `&` has been omitted, or `&\omit&\omit&\relax` if used when two `&`'s have been omitted, etc.

Fortunately, when we define `\tag` for `\alignat` and `\xalignat` we don't have to worry about additional things appearing in the preamble, as was necessary for `\align`. On the other hand, whereas the first pass for `\align` simply had `\eat@{#}` in the preamble for the `\tag` part, `\alignat` and `\xalignat` have `\maketag@#\maketag@`.

- (3) This means that the tag counter \tag@C will be incremented during the first pass. So we will first store the current value of \tag@C in a new counter \tag@CC, and then restore \tag@C to this value before doing the second pass.
- (4) Moreover, we want to disable \label during the first pass (or we will be using the same <label> twice during the second pass, and get an error message).

We could use \let\label=\eat@ for this purpose, but we will instead call \unlabel@, defined by

```
\def\unlabel@{\def\label##1{\prevanish@postvanish@}%
\def\pagelabel##1{\prevanish@postvanish@}}
```

since this construction will be needed later anyway (section 32.4).

```
\def\unlabel@{\def\label##1{\prevanish@postvanish@}%
\def\pagelabel##1{\prevanish@postvanish@}}
\newcount\tag@CC

\expandafter\def\csmname alignat \endcsmname#1#2\endalignat
{\inany@true\xat@false
\gdef\tag{\global\tag@true
\count@=#1\relax\multiply\count@ by 2
\advance\count@ by -1
\gdef\tag@{&}\loop\ifnum\count@>\and@
\edef\tag@{&\omit\tag@}\advance\count@ by -1
\repeat\tag@\relax}%
\vspace@\allowdisplaybreak@\displaybreak@\intertext@
\display@\measuring@true\tag@CC=\tag@C
\setbox\savealignat=\hbox{\noset@\unlabel@%
$m@th\displaystyle\Let@
\attag@{#1}\vbox{\halign{\span\preamble@\crcr#2\crcr}}}$}%
\measuring@false
\Let@\attag@{#1}\tag@C=\tag@CC
\tabskip\centering\halign to\displaywidth
{\span\preamble@\crcr#2\crcr\black@{\wd\savealignat}}}
```

Similar changes are made for ‘`\xalignatL`’:

```

\expandafter\def\csname xalignat \endcsname#1#2\endxalignat
{\inany@true\xat@true
\def\tag{\global\tag@true
\count@=#1\relax\multiply\count@ by 2
\advance\count@ by -1
\def\tag@{&}\loop\ifnum\count@>\and@
\xdef\tag@{&\omit\tag@}\advance\count@ by -1
\repeat\tag@\relax}%
\vspace@\allowdisplaybreak@\displaybreak@\intertext@
\display@\measuring@true\tag@CC=\tag@C
\setbox\savealignat@=\hbox{\noset@\unlabel@%
$m@th\displaystyle\Let@
\attag@{#1}\vbox{\halign{\span\preamble@@\crrc#2\crrc}}$}%
\measuring@false
\Let@\attag@{#1}\tag@C=\tag@CC
\tabskip\centering@\halign to\displaywidth
{\span\preamble@@\crrc#2\crrc\black@\wd\savealignat@}}

```

16.6. `\gather`. Finally, the $\mathcal{A}\mathcal{M}\mathcal{S}$ -TEX construction `\gather` also has a `\def\tag{&}` clause, which must be changed to `\def\tag{&\relax}`:

```

\def\gather{\relax\ifmmode\ifinner
\def\next@{\onlydmatherr@\gather}\else
\ingather@true\inany@true\def\tag{&\relax}%
\vspace@\allowdisplaybreak@\displaybreak@\intertext@
\display\Let@
\iftagsleft\def\next@{\csname gather \endcsname}\else
\def\next@{\csname gather \space\endcsname}\fi\fi
\else\def\next@{\onlydmatherr@\gather}\fi\next@}

```

Chapter 17. New counters

Having seen how `\tag` deals with `\label`, we are now in a position to consider \LaTeX 's `\newcounter` construction for creating a new counter.¹

17.1. \newcounter. We introduce a very frequently used abbreviation

```
\def\exstring@{\expandafter\eat@\string}
```

where `\eat@` from $\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\mathcal{T}\mathcal{E}\mathcal{X}$ is simply defined (section 1.2) as

```
\def\eat@#1{}
```

Note that if `#1` is a control sequence, say `\tag`, then something like

```
\csname\exstring@#1C\endcsname
```

just becomes `\tag@C` (the `\eat@` eats the `\` at the beginning of `\string\tag`).

The first thing `\newcounter\foo` will do is to

```
\define\foo{}
```

to give an error message if `\foo` has already been defined.

Since `\foo\label{<label>}` is supposed to work, albeit not by the general mechanism for `\label`'s, `\foo` needs to be defined in terms of a `\futurelet`:

```
\def\foo{\futurelet\next\foo@Z}
```

We use `\foo@Z` because no control sequence from $\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\mathcal{T}\mathcal{E}\mathcal{X}$ or $\mathcal{L}\mathcal{A}\mathcal{T}\mathcal{E}\mathcal{X}$ ends with `@Z`.

The name `'\foo@Z'` will be created with

```
\csname\exstring@\foo @Z\endcsname
```

¹In version 1 of $\mathcal{L}\mathcal{A}\mathcal{T}\mathcal{E}\mathcal{X}$, this was called `\counter`, because `\newcounter` seemed too close to $\mathcal{T}\mathcal{E}\mathcal{X}$'s `\newcount` primitive for comfort, but it actually seems unlikely that anyone (except perhaps $\mathcal{T}\mathcal{E}\mathcal{X}$ perts!) would mistakenly type `\newcount` instead of `\newcounter`, and `\newcounter` fits much better with all the other $\mathcal{L}\mathcal{A}\mathcal{T}\mathcal{E}\mathcal{X}$ names.

and to define `\foo` we use the same strategy that was used in section 3.2:

```
\edef\next@{\def\noexpand\foo{\futurelet\noexpand\next
\csname\exstring@\foo @Z\endcsname}}
\next@
```

Now `\foo@Z` should increase the counter `\foo@C` by 1, and print the properly formatted number, in the proper font:

```
\def\foo@Z{\global\advance\foo@C by 1
{\foo@F\foo@S{\foo@P{\foo@N{\number\foo@C}}\foo@Q}}
```

(for the moment, let's not worry about how `\foo@C`, ... are to be defined).

Moreover, if `\foo` is followed by `\label{...}`, then we need a way of getting this `\label{...}` properly recorded. To do this, we can simply use the code

```
{\noexpands@\xdef\Thelabel@{...}...}
{\locallabel@\label{...}}
```

The `\label` in the second group will make sense, because the `\locallabel@` will define `\thelabel@`; this `\label` will thus write to the `.lax` file, and append to `\laxlist@`. After the group containing `\locallabel@` is finished, however, any values of `\thelabel@`, ... will be restored to their previous values, so another `\label` will give an error message (unless `\foo` happens to appear within some other construction that itself allows a `\label`).

```
\def\foo@Z{\global\advance\foo@C by 1
{\foo@F\foo@S{\foo@P{\foo@N{\number\foo@C}}\foo@Q}}%
\ifx\next\label
\def\next@\label##1{%
{\noexpands@
\xdef\Thelabel@{...}\xdef\Thelabel@@@{...}}%
{\locallabel@\label{##1}}}%
\else
\let\next@=\relax
\fi
\next@}
```

Of course, these ##'s will all have to be ####'s, since this whole definition appears within the definition of \newcounter. But that's only a minor problem. We are also going to have to treat this whole thing with an \edef\next@ (compare page 82), so it is going to look pretty complicated (in particular, now our ####'s will all have to be #####'s, since an \edef changes ## to #).

The control sequence \foo@C will be specified within this \edef\next@ as

```
\csname\exstring@\foo @C\endcsname
```

and similarly for \foo@P, In all cases, T_EX will make these equivalent to \relax, so they will not be expanded further.

Finally, after all this, we can create \foo@C, ... with their proper meanings,

```
\expandafter\newcount@\csname\exstring@\foo@ C\endcsname
\expandafter\let\csname\exstring@\foo @N\endcsname=\arabic
. . .
```

Note that here we use \newcount@ (section 3.6), since it appears within a definition, and thus might be used after \alloc@ has been returned to its old definition. Note also that we use \let rather than \def, just as with \tag (page 105), but \def would be required for anything other than \arabic (and is supplied whenever \newnumstyle is used).

In the following code for \newcounter, note that \noexpand is required before \number, \ifx, \else and \fi because these primitives are "expanded" in an \edef:

```
\def\newcounter#1{\define#1{}}%
\edef\next@{\def\noexpand#1{\futurelet
\noexpand\next\csname\exstring@#1@Z\endcsname}}\next@
\edef\next@{\def\csname\exstring@#1@Z\endcsname
{\global\advance\csname\exstring@#1@C\endcsname by 1
f\csname\exstring@#1@F\endcsname
\csname\exstring@#1@S\endcsname{\csname\exstring@
#1@P\endcsname\csname\exstring@#1@N\endcsname
{\noexpand\number\csname\exstring@#1@C\endcsname}}%
\csname\exstring@#1@Q\endcsname}}}%
\noexpand\ifx\noexpand\next\noexpand\label
```

```

\def\noexpand\next@\noexpand\label#####1{%
  {\noexpand\noexpands@
  \xdef\noexpand\Thelabel@
    {\csname\exstring@#1@N\endcsname
    {\noexpand\number\csname\exstring@#1@C\endcsname}}}%
  \xdef\noexpand\Thelabel@@@
    {\noexpand\number\csname\exstring@#1@C\endcsname}%
  \xdef\noexpand\Thelabel@@
    {\csname\exstring@#1@S\endcsname
    {\csname\exstring@#1@P\endcsname
    \csname\exstring@#1@N\endcsname
    {\noexpand\number\csname\exstring@#1@C\endcsname}}}%
    {\csname\exstring@#1@Q\endcsname}}}%
  \xdef\noexpand\Thelabel@@@@
    {\csname\exstring@#1@P\endcsname
    \csname\exstring@#1@N\endcsname
    {\noexpand\number\csname\exstring@#1@C\endcsname}}}%
    {\csname\exstring@#1@Q\endcsname}}}%
  {\noexpand\locallabel@\noexpand\label{#####1}}}%
\noexpand\else\let\noexpand\next@=\relax\noexpand\fi
\noexpand\next@}}}%

\next@
\expandafter\newcount@\csname\exstring@#1@C\endcsname
\expandafter\let\csname\exstring@#1@N\endcsname=\arabic
\expandafter\def\csname\exstring@#1@S\endcsname##1{##1\}/}%
\expandafter\let\csname\exstring@#1@P\endcsname=\empty
\expandafter\let\csname\exstring@#1@Q\endcsname=\empty
\expandafter\def\csname\exstring@#1@F\endcsname{\rm}%
}

```

17.2. `\usecounter`. To follow through the definition of `\usecounter`, it helps to consider a specific case, like the following simplification of an example from the *L_AT_EX* Manual (page 78):

(*) `\usecounter\exno\example#1{\it Example #1.}_}`

Here we want

```
\example           to give {\it Example \exno.}\_
\example"... "    to give {\it Example {\exno@F_...}.}\_
\example\label{...} to give {\it Example \exno\label{...}.}\_
```

where `_ _ _` stands for

```
\let\pre=\exno@P ... \let\numstyle=\exno@N
```

In the third case, the combination `\exno\label{...}` will itself take care of printing the right number, and labelling it.

Page 79 of the *L_AT_EX* Manual suggest that the user should modify (*) to read

```
\usecounter\exno\example#1{{\it Example #1.}\_ignorespaces}
```

But that's silly—it's clearly better to have `\usecounter` automatically add the `\ignorespaces`. Actually, we really need to add `\FNSSP@` (page 100) for cases where `_` doesn't appear at the end of the definition, but an invisible construction, like `\pagelabel`, occurs after the use of the construction being defined.

So we really want

```
(E1) \example
      to give {\it Example \exno.}\_ \FNSSP@
(E2) \example"... "
      to give {\it Example {\exno@F_...}.}\_ \FNSSP@
(E3) \example\label{...}
      to give{\it Example \exno\label{...}.}\_ \FNSSP@
```

Clearly, `\example` is going to involve a `\futurelet`, so we expect that `\usecounter\exno\example` should expand out to something like

```
\def\example{\futurelet\next\exno@@Z}
. . .
. . .
```

[Note that the subsidiary control sequence `\exno@@Z` is written entirely in terms of `\exno`. So if we later type

```
\usecounter\exno\otherexample
```

`\otherexample` will then be defined exactly the same way, as

```
\futurelet\next\exno@@Z
```

—though of course `\exno@@Z` will now end up being defined differently. This is a reasonable arrangement, since the same counter `\exno` shouldn't be allowed for two different constructions.]

In the use

```
\usecounter\exno\example#1{\it Example #1.}\_
```

we also have to somehow capture the `<parameter text>` and `<replacement text>`. This suggests that `\usecounter\exno\example` should expand out as

```
\def\example{\futurelet\next\exno@@Z}
. . .
. . .
\def\exno@@Z@
```

where the `\def\exno@@Z@` at the end will then swallow up the subsequent `<parameter text>` and `<replacement text>`, and thus, in our case,

```
\def\exno@@Z@#1{\it Example #1.}\_
```

Once we have `\exno@@Z@`, it is easy to say what `\exno@@Z` should do:

(1) If `\next` is neither `\label` nor `"`, then (E1) we want

```
{\it Example \exno.}\_ \FNSSP@
```

which can be obtained as

```
\exno@@Z@{\exno}\FNSSP@
```

(2) If \next is "", so that we have \example"...", then (E2) we want

```
{\it Example {\exno@F \let\pre=\exno@P ...
\let\numstyle=\exno@N ... }.}\FNSSP@
```

which can be obtained as

```
\exno@@Z@{\exno@F \let\pre=\exno@P ...
... \let\numstyle=\exno@N ... }\FNSSP@
```

(3) If \next is \label, so that we have \example\label{...}, then (E3) we want

```
{\it Example \exno\label{...}.\}\FNSSP@
```

which can be obtained as

```
\exno@@Z@{\exno\label{...}}\FNSSP@
```

Thus, we want \usecounter\exno\example to expand as

```
\def\example{\futurelet\next\exno@@Z}
\def\exno@@Z{\ifx\next\label
\def\next@\label##1{\exno@@Z@
{\exno\label{##1}}\FNSSP@}
\else
\ifx\next"\def\next@"##1"{\exno@@Z@
{\exno@F_ _ _##1}}\FNSSP@}
\else
\def\next@{\exno@@Z@{\exno}\FNSSP@}
\fi\fi\next@}
\def\exno@@Z@
```

The general definition of \usecounter has the same features as that for \newcounter, with several \edef\next@s. But there are some complications.

First, we need a way of inserting the

```
\exno@F
```

which we will specify as

```
\csname\exstring@#1@F\endcsname
```

except that after this is expanded to `\exno@F`, which is now *not* `\relax`, we need to inhibit further expansion of this `\exno@F`. This can be done with

```
\expandafter\noexpand\csname\exstring@#1@F\endcsname
```

because the primitive `\expandafter` is “expanded” in an `\edef`: the `\csname ... \endcsname` is first expanded to `\exno@F`, and then this expansion is placed in front of `\noexpand`, and consequently *not* expanded in the `\edef`!

Similar maneuvers are needed for `\exno@P ...`; our `\edef\next@` will

```
\let\noexpand\pre=
\expandafter\noexpand\csname\exstring@#1@P\endcsname
. . .
```

In addition, the control sequences `\exno@@Z` and `\exno@@Z@` might already exist, if ‘`\usecounter\exno`’ has already appeared (compare page 124). In this case, the control sequences

```
\csname\exstring@#1@@Z\endcsname
\csname\exstring@#1@@Z@\endcsname
```

would also *not* be `\relax`, and therefore they would be expanded in `\edef`’s, probably with disastrous results. So we will simply `\let` them be `\relax` to begin with.

Finally, we want to add an error message at the beginning if the first argument of `\usecounter` hasn’t already been created by `\newcounter`.

```
\def\usecounter#1#2{\expandafter
\ifx\csname\exstring@#1@Z\endcsname\relax
\Err@{\noexpand#1not created with \string\newcounter}\fi
\expandafter\let\csname\exstring@#1@@Z\endcsname=\relax
\expandafter\let\csname\exstring@#1@@Z@\endcsname=\relax
```



```

\edef\next@{\def\noexpand#2{\futurelet\noexpand\next
\csname\exstring@#1@Z\endcsname}}%
\next@
\edef\next@{\def\csname\exstring@#1@Z\endcsname
{\noexpand\ifx\noexpand\next\noexpand\label
\def\noexpand\next@\noexpand\label
#####1{\csname\exstring@#1@Z\endcsname
{\noexpand#1\noexpand\label{#####1}}\noexpand\FNSSP@}%
\noexpand\else
\noexpand\ifx\noexpand\next\noexpand"%
\def\noexpand\next@\noexpand"#####1\noexpand"%
{\csname\exstring@#1@Z\endcsname{\expandafter\noexpand
\csname\exstring@#1@F\endcsname
\let\noexpand\pre=
\expandafter\noexpand\csname\exstring@#1@P\endcsname
\let\noexpand\post=
\expandafter\noexpand\csname\exstring@#1@Q\endcsname
\let\noexpand\style=
\expandafter\noexpand\csname\exstring@#1@S\endcsname
\let\noexpand\numstyle=
\expandafter\noexpand\csname\exstring@#1@N\endcsname
#####1}}\noexpand\FNSSP@}%
\noexpand\else
\def\noexpand\next@{\csname\exstring@#1@Z@\endcsname
{\noexpand#1}\noexpand\FNSSP@}%
\noexpand\fi\noexpand\fi\noexpand\next@}}%
\next@
\expandafter\def\csname\exstring@#1@Z@\endcsname}

```

Chapter 18. Lists

Now that we've discussed `\tag`, with just one counter `\tag@C`, it seems appropriate to discuss `\list`, because it has five counters `'\list@C1'`, ..., `'\list@C5'`, corresponding to the five levels of a list (Chapter 24 explains how this is handled by the `\pre`, `\newpre`, ..., constructions). As on pages 14, 29 and 75, we will always use quotation marks around control sequence names like `'\list@C1'`, which really have to be named in the file by means of `\csname ... \endcsname`.

18.1. Style choices. Lists require four different style decisions, each in five versions, for the five different levels of a list.

A. First,

- (A.1) `\listbi@` will be the material (usually vertical spacing and/or penalties) that goes before the first `\item` at the first level of a list.
- (A.2) `\listbii@` will be the material that goes before the first `\item` at the second level.

\LaTeX begins by setting the default values

```
\def\listbi@{\penalty50 \medskip}
\def\listbii@{\penalty100 \smallskip}
\let\listbiii@=\relax
\let\listbiv@=\relax
\let\listbv@=\relax
```

Thus,

- (A.1) `\listbi@` will produce a `\penalty50 \medskip` before the very first `\item` at the top level of a `\list` (when we define `\item` we will use `\listbi@` only for the first `\item` at the top level [and after the previous paragraph has been ended], so this penalty and space will not apply to later `\item`'s at the top level).
- (A.2) Similarly, `\listbii@` produces `\penalty100 \smallskip` before the first item at the second level of a list.
- (A.3) In the default style, nothing is added before the first item at the third through fifth levels.

B. Next,

- (B.1) `\listmi@` will be the changes to be made in the midst of `\item's` at the first level (usually changing the left and/or right indentations).
- (B.2) `\listmii@` will be the changes to be made in the midst of `\item's` at the second level.

...

L^AT_EX sets the default values

```
\def\listmi@{\advance\leftskip by 30pt}
\let\listmii@=\listmi@
\let\listmiii@=\listmi@
\let\listmiv@=\listmi@
\let\listmv@=\listmi@
```

Thus,

- (B.1) `\listmi@` will indent the text for `\item's` at the top level by 30pt;
- (B.2) `\listmii@` will indent the text for `\item's` at the second level by yet another 30pt;

...

As we will see later (page 133), although `\listbiii@`, ... can be left undefined, it is important to have definitions for `\listmii@`, ..., even if they are just the same as `\listmi@`.

C. Next,

- (C.1) `\itemi@` will specify the formatting of the `\item` numbers at the first level;
- (C.2) `\itemii@` will specify the formatting of the `\item` numbers at the second level;

...

L^AT_EX sets the default values

```
\def\itemi@#1{\noindent@\llap{#1\hskip5pt}}
\let\itemii@=\itemi@
\let\itemiii@=\itemi@
\let\itemiv@=\itemi@
\let\itemv@=\itemi@
```

For the use of `\noindent@@`, see Chapter 8. Notice that if `\everypar` is non-empty, then new paragraphs within an `\item` (like the one shown on page 17 of the *L_AT_EX* Manual), will have this `\everypar` material before them. (If we wanted to prohibit that, we could simply set `\everypar={}` right after the `\begingroup` in the definition of `\list` to follow, and use `\noindent@` instead of `\noindent@@` at this point.)

Thus,

(C.1) `\itemi@` will format an `\item` number #1 as

```
\noindent@@\llap{#1\hskip5pt}
```

The `\llap{#1\hskip5pt}` simply causes the `\item` number to appear 5pt to the left of the rest of the text;

(C.2) Similarly, `\itemii@` formats an `\item` number at the second level in exactly the same way;

...

D. Finally, we have some control sequences that are numbered differently:

(D.1) `\liste@` will be the formatting (usually vertical spacing and/or penalties) that goes at the end of the last `\item` at the *first* level;

(D.2) `\listei@` will be the formatting that goes at the end of the last `\item` at the *second* level;

...

(D.5) `\listeiv@` will be the formatting that goes at the end of the last `\item` at the *fifth* level.

Thus, for reasons that will become clear (page 137) these control sequences are “numbered” one less than the level to which they apply.

L_AT_EX sets the default values

```
\def\liste@{\penalty-50 \medskip}
\def\listei@{\penalty-100 \smallskip}
\let\listeii@=\relax
\let\listeiii@=\relax
\let\listeiv@=\relax
```

Thus,

- (D.1) `\listei@` will produce `\penalty-50 \medskip` at the end of the list;
 (D.2) `\listei@` will produce `\penalty-100 \smallskip` when we go from the end of the second level of the list back to the first level;

but nothing will be added when we go from the end of the third level back to the second level, etc.

18.2. Counters, etc. Next we must create the counters `'\list@C1'`, ..., `'\list@C5'`, which we initialize to 0:

```
\expandafter\newcount\csname list@C1\endcsname
\csname list@C1\endcsname=0
. . .
```

We want `'\list@C1'`, `'\list@C2'`, ..., in conformity with a general \LaTeX -principle for handling constructions with more than one counter (see Chapter 24), but we use `\listbi@`, `\listbii@`, ..., because there are a fixed number of such control sequences, which we will usually be mentioning explicitly, so there's no need to complicate matters by using names that combine letters and numbers.

Just as we use `'\...@C1'`, ..., `'\...@C5'` to indicate counters at various levels, we also use `'\...@P1'`, ... for the pre-material at the various levels, and `'\...@Q1'`, ... for the post-material at the various levels. We initialize all of these to be empty:

```
\expandafter\let\csname list@P1\endcsname=\empty
. . .
\expandafter\let\csname list@Q1\endcsname=\empty
. . .
```

Then come the styles at each level (compare page 74):

```
\expandafter\def\csname list@S1\endcsname#1{\rm(#1)\/\rm}}
. . .
```

Note that these styles determine the formatting of an item number, but the spacing after the formatted number is determined by `\itemi@`, ... (page 130).

In conformity with this, style control sequences in \LaTeX never address the question of the spacing after the formatted number, which has to be handled separately.

Then come the numbering styles at each level:

```
\expandafter\let\csname list@N1\endcsname=\arabic
. . .
```

Note that here we once again use `\let` rather than `\def`, just as with `\tag` (page 105), but `\def` would be required for anything other than `\arabic`.

Finally, we also need the font styles at each level:

```
\expandafter\def\csname list@F1\endcsname{\rm}
. . .
```

There will be occasions when we want to refer to the list counter, etc., for the current level, without having to know or to specify this level explicitly. For this purpose, we first create a counter,

```
\newcount\listlevel@
\listlevel@=0
```

which will always hold the current list level, and then we

```
\def\list@@C{\csname list@C\number\listlevel@\endcsname}
\def\list@@P{\csname list@P\number\listlevel@\endcsname}
\def\list@@Q{\csname list@Q\number\listlevel@\endcsname}
\def\list@@S{\csname list@S\number\listlevel@\endcsname}
\def\list@@N{\csname list@N\number\listlevel@\endcsname}
\def\list@@F{\csname list@F\number\listlevel@\endcsname}
```

so that, for example, `\list@@C` will be `'\list@C1'` if we are at the first level, `'\list@C2'` if we are at the second level, etc.

18.3. Other preliminaries. Since, as we've already indicated in section 1, the first `\item` at each level needs to be treated specially, we need flags

```

\newif\iffirstitemi@
\newif\iffirstitemii@
\newif\iffirstitemiii@
\newif\iffirstitemiv@
\newif\iffirstitemv@

```

Moreover, we need ways of setting the flag for each level true or false without explicitly mentioning the level:

```

\def\Firstitem@true{\csname
  firstitem\romannumeral\listlevel@ @true\endcsname}
\def\Firstitem@false{\csname
  firstitem\romannumeral\listlevel@ @false\endcsname}

```

We will also need to refer to `\listm...@`, `\item...@`, and `\liste...@` without having to specify the level ‘...’. So we define

```

\def\Listm@{\csname
  listm\romannumeral\listlevel@ @\endcsname}
\def\Item@{\csname
  listformatt\romannumeral\listlevel@ @\endcsname}
\def\Liste@{\csname
  listformate\romannumeral\listlevel@ @\endcsname}

```

(For this to work, we must have `\listm...@`, `\item...@`, defined for all levels; `\Liste@` will be applied only for $1 \leq \text{\listlevel@} < 5$.)

Version 1 of *L_AT_EX* had `\continuelist`, which was meant to be used as

```

\continuelist
\list
. . .
\endlist

```

although the L_AM_S-T_EX Manual mistakenly indicated the usage

```
\continuelist
\item
. . .
\endlist
```

(which would conflict with the general ‘\foo ... \endfoo’ convention for L_AM_S-T_EX constructions). This was a natural mistake to make, however, so now ‘\continuelist’ has been replaced by ‘\keepitem’.

\keepitem itself will simply set a flag,

```
\newif\iflistcontinue@
\def\keepitem{\listcontinue@true}
```

while \endlist will always reset \listcontinue@false.

18.4. \list. Unlike the case of \tag, whenever we start a \list we want to reset the list counters ‘\list@C1’, ... to 0, except if \keeplisting is in force, in which case ‘\list@C1’ will not be changed. Then we want to begin a group, set \firstitemi@true, set the list level counter to 1, and define \item in terms of a \futurelet, since it needs to see if a “quoted” number follows:¹

```
\def\list{%
\iflistcontinue@\else
\global\csname list@C1\endcsname=0
\fi
\global\csname list@C2\endcsname=0
\global\csname list@C3\endcsname=0
\global\csname list@C4\endcsname=0
\global\csname list@C5\endcsname=0
\begingroup
\firstitemi@true
\listlevel@=1
```

¹An \item outside a \list will continue to have its usual meaning from plain T_EX, though it might be preferable to specify \Invalid@item (see section 1.1) and to \let\itemitem=\undefined, since that usage more or less conflicts with L_AM_S-T_EX usage.


```
\def\item{\futurelet\next\item@}
```

At this point it might seem like it's time to end the previous paragraph, and get down to work. But we need a slight diversion, because we are also going to allow the possibility that \list is followed by \runinitem instead of \item, as in $\mathcal{A}\mathcal{M}\mathcal{S}$ -TEX. So we also need a \futurelet for that:

```
\def\list{%
  \iflistcontinue@\else
    \global\csname list@C1\endcsname=0
  \fi
  \global\csname list@C2\endcsname=0
  \global\csname list@C3\endcsname=0
  \global\csname list@C4\endcsname=0
  \global\csname list@C5\endcsname=0
  \begingroup
  \firstitemi@true
  \listlevel@=1
  \def\item{\futurelet\next\item@}%
  \futurelet\next\list@}
```

\runinitem has no role except to serve as an indicator after \list [or after \inlevel, since we also allow \runinitem to be used instead of the first \item at each level], so (compare section 1.1) we first state

```
\Invalid@\runinitem
```

In the definition of \list@, the test \ifx\next\runinitem can be used to detect if \list is followed by \runinitem. If \list@ detects a \runinitem, it will then swallow this \runinitem and do yet another \futurelet (to see if \runinitem is followed by a quoted number).

Otherwise, the first thing \list@ should do is to end the current paragraph. At this point, however, the default style adds some adjustment for the space between paragraphs, since paragraphs are allowed within each \item. If our style does leave some space, say 1pt, between paragraphs, we probably want to do the same for paragraphs within each \item. Normally, however, \parskip

is something like ‘Opt plus 1pt’, with only stretchable space. In this situation, it is inadvisable to leave the stretchability, for, on a page requiring a fair amount of vertical stretching, this interparagraph stretch might easily end up looking too big compared to the other spacing that the style selects for `\list`’s (I speak from experience!). This stretchability can be eliminated with the code

```
\dimen@=\parskip \parskip=\dimen@
```

(since `\dimen@` is a dimension, the first assignment sets `\dimen@` to the non-stretchable part of `\parskip`, and the second assignment resets `\parskip` to this non-stretchable part).

So the definition of `\list@` might be

```
\def\list@{\ifx\next\runinitem
  \def\next@\runinitem{\futurelet\next\runinitem@}\else
  \def\next@{\par \dimen@=\parskip \parskip=\dimen@}\fi\next@}
```

That’s not quite good enough however, because we also want to allow a blank line before the `\runinitem`, since blank lines are generally allowed before `\item`’s.¹

So if `\next` happens to be `\par`, we will call a construction that swallows this `\par` and then repeats the `\futurelet\next\list@`:

```
\def\list@{\ifx\next\par
  \def\next@\par{\futurelet\next\list@}\else
  \ifx\next\runinitem
    \def\next@\runinitem{\futurelet\next\runinitem@}\else
    \def\next@{\par \dimen@=\parskip \parskip=\dimen@}%
  \fi\fi\next@}
```

Leaving aside the definition of `\runinitem@` for the moment, we consider the case where `\item` occurs next.

¹ On the other hand, there’s no way we can allow a blank line to occur before `\list` in a ‘`\list\runinitem`’ combination; when a run-in `\item` is required, the `\list` must appear in the same paragraph as the previous text.

18.5. \item. \item has already been set to \futurelet\next\item@. Before worrying about whether a quoted \item number follows, \item@ will take care of any needed formatting. This will involve two new flags

```
\newif\ifoutlevel@
\newif\ifrunin@
```

The first will be true if the \item was preceded by \outlevel (so that \item's at a higher level have just been completed). The second will be true if the present \item follows a \runinitem at the same level. In either of these cases, the appropriate flag for first \item's at this level (\iffirstitemi@ or \iffirstitemii@ or ...) will be *false*.

The first thing \item@ adds is

```
\ifoutlevel@\Liste@\outlevel@false\fi
```

So, for example, if our \item occurs at the top level (\listlevel@ = 1), and we have just completed \item's at the second level, we will add \listei@—recall (page 130) that this is the formatting that goes at the end of the last \item at the *second* level.

The reason for this approach is that in a situation like

```
\list
\item ...
  \inlevel
    \item ...
      \inlevel
        \item ...
          \item ...
            \outlevel
          \outlevel
        \item ...
```

where we go from third level \item's right back to first level \item's, the spacing before that next \item at the first level should be the spacing that goes after *second* level items, not the spacing that goes after third level items (and certainly not the sum of the spacing that goes after the second and third

levels). So we don't want the spacing to be put in by the `\outlevel's`; instead `\outlevel` will just set `\outlevel@true`, for use by `\item`.

Next, we consider the case where `\ifrunin@` is true. In this case, we simply want to set `\runin@false`, end the current paragraph (which contains the previous `\runinitem`, which has not been indented any extra amount), add the same adjustments that were made for `\list@`, in case we are at the first level, and then add `\Listm@` (the `\listm...@` for the current level) to apply to the remaining `\item's` at the current level:

```
\ifrunin@\runin@false\par
\dimen@=\parskip \parskip=\dimen@
\Listm@\fi
```

If neither of these cases occurs, we have to consider the possibility that the `\item` was the first at its level. At the first level, this means that we will add

```
\listbi@ \listmi@
```

if `\iffirstitemi@` is true, also setting `\firstitemi@false`, but simply add a `\par` for other items:

```
\iffirstitemi@
\listbi@\listmi@\firstitemi@false
\else\par\fi
```

Note that `\listbi@` will be occurring after a `\par` supplied by `\list`, via `\list@`, or by `\outlevel` (section 8).

Analogous code is added for the situation where we are at the second level (`\iffirstitemii@... \fi`); in this case, `\listbii@` will be occurring after the `\par` supplied by the previous code. And similarly for the third through fifth levels.

Each of these `\iffirstitem...@` tests has to be made separately, and `\listbi@, ...` appear only in such constructions; that is why there is no point having a '`\Listb@`' construction.

We will use compressed format, as well as the K-method, for `\item@`, so that the definition ends

```
\ifx\next"\expandafter\next@\else\expandafter\nextii@\fi
```

with \next@ and \nextii@ defined first.

The definitions of \next@ and \nextii@ are quite similar to the definitions of \maketag@@@ and \maketag@@@, using \Qlabel@ from section 16.3 for “quoted” \item numbers, but with some additions:

```

\def\next@##1"{\let\pre=\list@P \let\post=\list@Q
\let\style=\list@S \let\numstyle=\list@N
\vskip-\parskip
\Item@{\list@F##1}%
\noexpands@
\Qlabel@{##1}}%
\locallabel@
\FNSSP@}%

\def\nextii@{\global\advance\list@C by 1
\noexpands@
\edef\Thelabel@@@{\number\list@C}%
\edef\Thelabel@\list@N
\edef\Thelabel@@@@{\list@P\Thelabel@\list@Q}}%
\edef\Thelabel@@\list@S
}%
\locallabel@
\vskip-\parskip
\Item@{\list@F\thelabel@@}%
\futurelet\next\pretendspace@}%

```

We add the \vskip-\parskip because \Item@ will normally start a new paragraph, and we want the spacing before the \item to be explicitly specified by \listbi@, ..., and not involve any \parskip, which is easy to forget about.

And we add the \FNSSP@ and \futurelet\next\pretendspace@'s because \Item@ puts us into horizontal mode (in the default style it also produces some space after the \item number—the \hskip5pt at the end of the \llap in the definition of \item...@—but this space is “hidden” inside the \llap, and will not be discovered by \lastskip). So (compare section 7.2)

```
\item \label{...}\Text ...
```

would leave an extra space before the “Text ...”. The `\pretendspaces`'s take care of this. In the case of `\nextii` we don't need `FNSSP`, since a space token won't appear after `\item` (compare `\endMath`, page 102).

It should perhaps also be noted that something like `\let\pre=\list@P` does not actually make `\pre` have the value of the appropriate '`\list@P1`' or '`\list@P2`' or ... , but simply makes `\pre` expand out to the definition of `\list@P`, i.e., to

```
\csname list@P\number\listlevel@\endcsname
```

This is adequate, however, since we are not storing this value of `\pre` for later use: when this `\pre` gets used, either in printing the number,

```
{\list@F##1}
```

or in the `\xdef`'s involved in

```
\Qlabel@{##1}
```

the current value of '`\list@P1`' or '`\list@P2`' or ... , will be inserted.

Thus, the definition of `\item` is:

```
\def\item@{%
  \ifoutlevel@\Liste@\outlevel@false\fi
  \ifrunin@\runin@false\par
  \dimen@=\parskip \parskip=\dimen@ \Listm@\fi
  \iffirstitemi@
    \listbi@\listmi@\firstitemi@false
  \else\par\fi
  \iffirstitemii@
    \listbii@\listmii@\firstitemii@false
  \else\par\fi
  \iffirstitemiii@
    \listbiii@\listmiii@\firstitemiii@false
  \else\par\fi
  \iffirstitemiv@
    \listbiv@\listmiv@\firstitemiv@false
  \else\par\fi
```

```

\iffirstitemv@
  \listbv@\listmv@\firstitemv@false
\else\par\fi
\def\next@"##1"{\let\pre=\list@P \let\post=\list@Q
\let\style=\list@S \let\numstyle=\list@N
\vskip-\parskip
\Item@\list@F##1}
\noexpands@
\Qlabel@{##1}}%
\locallabel@
\FNSSP@}%
\def\nextii@{\global\advance\list@C by 1
{\noexpands@
  \xdef\Thelabel@@@{\number\list@C}%
  \xdef\Thelabel@\list@N
  \xdef\Thelabel@@@{\list@P\Thelabel@\list@Q}}%
  \xdef\Thelabel@@\list@S
}%
\locallabel@
\vskip-\parskip
\Item@\list@F\thelabel@@}%
\futurelet\next\pretendspace@}%
\ifx\next"\expandafter\next@\else\expandafter\nextii@\fi}

```

18.6. \runitem@. \runitem@ is similar to, but simpler than, \item@.

First, \runitem@ sets \ifrunin@true and \Firstitem@false (to be used by the next \item [page 137]). The preliminary formatting of \item@ isn't necessary, so \runitem@ then immediately defines \next@ and \nextii@ for the compressed format:

```

\def\next@"##1"{\let\pre=\list@P \let\post=\list@Q
\let\style=\list@S \let\numstyle=\list@N
\unskip\space{\list@F##1}\quad}%
\noexpands@
\Qlabel@{##1}}%
\locallable@
\ignorespaces}

```

```

\def\nextii@{\global\advance\list@@C by 1
  {\noexpands@
   \xdef\Thelabel@@@{\number\list@@C}%
   \xdef\Thelabel@\list@@N
   \xdef\Thelabel@@@@{\list@@P\Thelabel@\list@@Q}}}%
  \xdef\Thelabel@@\list@@S
}%
\locallabel@
\unskip\space{\list@@F\thelabel@@}\square}

```

In other words, after suitably defining `\thelabel@`, ... , we leave a space after the preceding text, and then print the `\item` number, either as explicitly quoted, or as supplied automatically, and then add a space. In the case of `\runitem"..."` we have to ignore any space that follows the "...". Notice, however, that in neither case do we have to worry about invisible constructions that follow, since now a real space has been inserted.

Thus, the definition of `\runitem@` reads:

```

\def\runitem@{%
  \runin@true
  \Firstitem@false
  \def\next@##1{\let\pre=\list@@P \let\post=\list@@Q
    \let\style=\list@@S \let\numstyle=\list@@N
    \unskip\space{\list@@F##1}\square}%
  \noexpands@
  \Qlabel@{##1}}}%
  \locallabel@
  \ignorespaces}%
\def\nextii@{\global\advance\list@@C by 1
  {\noexpands@
   \xdef\Thelabel@@@{\number\list@@C}%
   \xdef\Thelabel@\list@@N
   \xdef\Thelabel@@@@{\list@@P\Thelabel@\list@@Q}}}%
  \xdef\Thelabel@@\list@@S
}%

```



```

\locallabel@
\unskip\space{\list@F\thelabel@}\l}%
\ifx\next"\expandafter\next@\else\expandafter\nextii@\fi}

```

18.7. \inlevel. We will keep each level of a list within its own group, to localize \listlevel@, etc.

\inlevel will simply produce an error message if \listlevel@ is already 5. Otherwise it will provide a \begingroup and advance \listlevel@ by 1; notice that this is *not* a \global\advance: the value of \listlevel@ will be different within the different groups provided by different levels. Then we will set \Firstitem@true (i.e., set \firstitemii@true if we are at now at the second level, \firstitemiii@true if we are now at the third level, etc.). No special formatting has to be done by \inlevel, because, once \listlevel@ has been correctly set, each \item will take care of all necessary formatting. But we are not quite done, because we need a \futurelet to see if a \runitem follows:

```

\def\inlevel{\ifnum\listlevel@=5
\def\next@{\Err@{Already 5 levels down}}\else
\def\next@{\begingroup\advance\listlevel@ by 1
\Firstitem@true\futurelet\next\inlevel@}\fi\next@}

```

If \inlevel@ detects a \runitem, it just has to swallow this \runitem and call \futurelet\next\runitem@, exactly like \list; otherwise nothing remains to be done at all. However, just as in the case of \list@, we must also allow the possibility that a \par precedes a \runitem¹:

```

\def\inlevel@{\ifx\next\par
\def\next@\par{\futurelet\next\inlevel@}\else
\ifx\next\runitem
\def\next@\runitem{\futurelet\next\runitem@}\else
\let\next@=\relax\fi\fi\next@}

```

¹As in the case of \list, there's no way we can allow a blank line to occur before \inlevel in an '\inlevel\runitem' combination; when a run-in \item is required at a new level, the \inlevel must appear in the same paragraph as the previous text.

18.8. `\outlevel`. Similarly, `\outlevel` gives an error message if we are at level 1. Otherwise, we want to end the paragraph and provide an `\endgroup` to match the `\begingroup` provided by the previous `\inlevel`. Nothing has to be done to `\listlevel@`, since it will simply return to the value it was already given before this new group had been entered. Note that it is important to end the paragraph before the `\endgroup`; otherwise, the current value of `\leftskip` provided by `\Listm@` would no longer be in force when the paragraph ended. In addition, before the `\endgroup` we need to globally reset the counter for the current level back to 0 (in case we go down to this level by another `\inlevel`). Finally, we want to set `\outlevel@true`, for use by the next `\item` (page 137 ff.).

```
\def\outlevel{\ifnum\listlevel@=1
  \Err@{At top level}\else
  \par\global\list@@C=0 \endgroup\outlevel@true\fi}
```

18.9. `\endlist`. `\endlist` first ends the current paragraph:

```
\def\endlist{\par . . .
```

Note that it's quite possible for `\endlist` to occur after several consecutive `\inlevel`'s—there may not be `\outlevel`'s to match all these `\inlevel`'s. Consequently, `\endlist` must not only supply an `\endgroup` to match the `\begingroup` supplied by `\list`, but it must also supply enough `\endgroup`'s to match any `\inlevel`'s that do not have matching `\outlevel`'s; this is accomplished by the following code:

```
\global\toks1={}%
\count@=\listlevel@
{\loop
  \ifnum\count@>0 \global\toks1=\expandafter{\the\toks1 \endgroup}%
  \advance\count@ by -1
  \repeat}
\the\toks1
```

(The possibility that an `\inlevel` does not have a matching `\outlevel` is the reason why we reset the counters for all levels at the beginning of a `\list`

[page 134], even though `\outlevel` resets the counter for its level [see the previous page.] The `\loop` is enclosed within a group for the unlikely eventuality that some `\list... \endlist` occurs within a `\loop` construction (compare page 37). Because of this, we need a `\global` assignment of the token list, so we use `\toks1` (compare section 1.3); and for consistency, we begin with the `\global` assignment `\global \toks1={}`. We don't make an abbreviation for `\toks1` because it is used so infrequently.

The `\endgroup`'s are followed by

```
\liste@
```

(page 130), and then by

```
\listcontinue@false
```

since `\listcontinue@true` is set by `\keeplisting`, which appears before a `\list`.

The final step is to take care of the fact that a `\list... \endlist` is not supposed to start a new paragraph at the end, unless a new paragraph actually appears in the file. For this we add

```
\vskip-\parskip
\noindent@@
```

If text follows immediately after the `\endlist`, it will start an unindented paragraph, with no extra space, except that provided by `\liste` (and so it will appear that the `\list... \endlist` has merely "interrupted" the paragraph).

On the other hand, when `\endlist` is followed by a `\par` or blank line before new text, so that we have

```
\vskip-\parskip
\noindent@@
\par
. . .
```

the "empty paragraph" `\noindent@@... \par` doesn't produce a blank line, but we do get `\parskip` glue inserted before the `\noindent@@` and *also* before the text following the `\par`. Together with the `\vskip-\parskip`, this means

that the following text, which will start a new paragraph, will have the usual `\parskip` glue before it.


There is just one further detail: We need to add

```
\futurelet\next\pretendspace@
```

in case an invisible construction like `\pagelabel` happens to appear after the `\endlist` (compare page 139).

Thus, the definition of `\endlist` is:

```
\def\endlist{\par
\global\toks1={}%
\count@=\listlevel@
{\loop
\ifnum\count@>0
\global\toks1=\expandafter{\the\toks1 \endgroup}%
\advance\count@ by -1
\repeat}%
\the\toks1
\liste@
\listcontinue@false
\vskip-\parskip
\noindent@@
\futurelet\next\pretendspace@}
```

 As on page 100, note that if `\bye` were `\outer`, then `\endlist\bye` would give an error message.

Notice that a `\par` after `\endlist` doesn't have to appear explicitly for all this to work. For example, something like

```
\list
. . .
\endlist
\section{...}
```

where `\section` starts a new paragraph, will behave correctly. Consequently, this approach is preferable to one that would use a `\futurelet` to see if a

\par comes next (such uses of \futurelet in sections 4 and 7 were quite different—they were meant only to *skip over* any \par's that might appear). If somewhat different design decisions are required for the spacing after the \endlist, we could, for example, use

```
\edef\parskip@{\parskip=\the\parskip}
\parskip=(dimen1)
\noindent@@
\futurelet\next\pretendspace@
```

[The construction

```
\edef\parskip@{\parskip=\the\parskip}
```

is similar to the construction

```
\edef\@sf{\spacefactor\the\spacefactor}
```

used in plain T_EX: the primitive \parskip is not expanded in the \edef, but '\the' is expanded, so \parskip@ means

```
'\parskip=(current value of \parskip)'
```

after the \edef is finished.]

Chapter 19. `\describe` and `\margins`

Although `\describe` and `\margins` don't really come next by any logical imperative, they come next in \LaTeX because they are so similar to `\list`.

19.1. `\describe`. Since `\describe` has only one level, it is simplest to incorporate all necessary style decisions directly into the definition, without using subsidiary control sequences like `\listbi@`, etc. In addition, `\describe` is much less complicated than `\list` because nothing gets numbered, `\describe` doesn't have to check for a `\runitem`, and `\item`'s in a `\describe` don't have to check to see if a " follows.

We require just one flag, for the first `\item` in a `\describe`:

```
\newif\iffirstdescribe@
```

`\describe` can immediately end the previous paragraph (unlike `\list`, which has to worry about a `\runitem` following); then, like `\list`, it begins a group and sets `\firstdescribe@true`. (The default style doesn't bother adding `'\dimen@=\parskip \parskip=\dimen@'`, but other styles might want to add that here.) Then it simply has to define `\item` within `\describe`, which is simply a control sequence with an argument:

```
\def\describe{\par
  \begingroup
  \firstdescribe@true
  \def\item##1{%
    \iffirstdescribe@
    \penalty50 \medskip \vskip-\parskip
    \firstdescribe@false\else\par\fi
    \hangindent2pc \hangafter1
    \noindent@{\bf##1}\hskip.5em}
```

(compare page 130 for the use of `\noindent@`).

In the definition of `\item`, the

```
\penalty50 \medskip \vskip-\parskip
\hangindent2pc \hangafter1 \noindent
\noindent@{\bf##1}\hskip.5em
```

represent style decisions, which might be changed for other styles.

`\enddescribe` is also much simpler than `\endlist`: we simply end the previous paragraph, add spacing and penalties (style decisions) and end the group started by `\describe`:

```
\def\enddescribe{\par
  \penalty-50 \medskip\vskip-\parskip
  \endgroup}
```

Since `\describe... \enddescribe` is supposed to start a new paragraph at the end (at least in the default style), we don't need the special machinations that were used for `\endlist` (page 145); of course, they could always be added for a style that wants to handle this question differently.

19.2. \margins. The `\margins` construction uses the commands `\pullin` and `\pullinmore`, rather than `\item`. We might as well have these give error messages outside of a `\margins... \endmargins` construction (see section 1.1),

```
\Invalid@\pullin
\Invalid@\pullinmore
```

There is no special formatting before the first paragraph of a `\margins` construction. Nevertheless, we still need a flag

```
\newif\iffirstpull@
```

but this flag will play quite a different role than the analogous flag in `\describe`: Each `\pullin` command is going to start a new group, within which `\leftskip` and `\rightskip` will be determined by the arguments of this command; since a `\pullin` is usually followed by yet another `\pullin`, this means that each `\pullin` will also have to provide the `\endgroup` that matches the `\begingroup` from the previous `\pullin`, except that the *first* `\pullin` should not provide this extra `\endgroup`.

`\margins`, like `\describe`, will end the previous paragraph, begin a group, set `\firstpull@true`, and then define `\pullin` and `\pullinmore`:

```
\def\margins{\par\begingroup\firstpull@true
\def\pullin##1##2{...}%
\def\pullinmore##1##2{...}}
```

`\pullin` will end the previous paragraph, and supply an `\endgroup`, except for the first `\pullin`, as already indicated, and then start yet another group:

```
\def\pullin##1##2{\par
\iffirstpull@\firstpull@false\else\endgroup\fi
\begingroup
```

(Notice that it is important to have the `\par` before the `\endgroup`, compare page 144.)

In this new group we want to set `\leftskip=##1` and `\rightskip=##2`. But we want

```
\pullin{...} or \pullin{L}{...}
```

to yield `\leftskip=0pt`, and similarly for the second argument, so we explicitly have to check for these possibilities:

```
\def\next@{##1}%
\ifx\next@\empty\leftskip=0pt\else
\ifx\next@\space\leftskip=0pt\else
\leftskip=##1\fi\fi
\def\next@{##2}%
\ifx\next@\empty\rightskip=0pt\else
\ifx\next@\space\rightskip=0pt\else
\rightskip=##1\fi\fi\ignorespaces}
```

The final `\ignorespaces` is needed to get rid of any spaces following the `\pullin{...}{...}`.

Normally, \margins is meant to be used as

```
\margins
  \pullin{...}{...}
  . . .
\endmargins
```

but our definition allows text to intervene between the ‘\margins’ command and the first ‘\pullin’; such text will just be treated as a paragraph with no special indentations.

Note that this definition of \pullin regards the arguments as ‘absolute’ dimensions, rather than as dimensions relative to values of \leftskip and \rightskip that may have already been set. Indeed, when one of the arguments is {} or { }, we explicitly set the value of \leftskip or \rightskip to 0pt, instead of simply leaving it alone.

Since no other \LaTeX macros fool with \leftskip and \rightskip, this seems like a reasonable design decision; a sophisticated user who knows about \leftskip and \rightskip will presumably have the sense either to adjust the arguments of \pullin appropriately (or to use \pullinmore), or to first set \leftskip and \rightskip to 0pt before using \margins.

To make the arguments of \pullin relative dimensions, it would suffice to replace the ‘\leftskip’ and ‘\rightskip’ with ‘\advance\leftskip’ and ‘\advance\rightskip’, respectively. In this case, we could simply omit the ‘\leftskip=0pt\relax’ and ‘\rightskip=0pt\relax’ both times.

The definition of \pullinmore follows just such a scheme, except that we must store the current values of \leftskip and \rightskip before ending the previous group:

```
\xdef\Next@{\leftskip=\the\leftskip
\rightskip=\the\rightskip}
```

The effect of this \xdef (compare page 147) is to make \Next@ mean

```
\leftskip=<current value of \leftskip>
\rightskip=<current value of \rightskip>
```

We need \xdef rather than \edef, because this will be followed by an \endgroup; then, after the following \begingroup we can reinstate these

values, before using the arguments of `\pullinmore` to decide how much to `\advance\leftskip` or `\advance\rightskip`:

```

\def\pullinmore##1##2{\par
  \xdef\Next@{\leftskip=\the\leftskip
    \rightskip=\the\rightskip}%
  \iffirstpull@\firstpull@false\else\endgroup\fi
  \begingroup
  \Next@
  \def\next@{##1}%
  \ifx\next@\empty\else\ifx\next@\space\else
    \advance\leftskip by ##1\fi\fi
  \def\next@{##2}%
  \ifx\next@\empty\else\ifx\next@\space\else
    \advance\rightskip by ##1\fi\fi
  \ignorespaces}

```

We use the scratch token `\Next@` here, because it has been reserved for `\global` assignments (page 22).

Thus, the definition of `\margins` is

```

\def\margins{\par\begingroup\firstpull@true
  \def\pullin##1##2{\par
    \iffirstpull@\firstpull@false\else\endgroup\fi
    \begingroup
    \def\next{##1}%
    \ifx\next@\empty\leftskip=Opt\else
    \ifx\next@\space\leftskip=Opt\else
    \leftskip=##1\fi
    \def\next{##2}%
    \ifx\next@\empty\rightskip=Opt\else
    \ifx\next@\space\rightskip=Opt\else
    \rightskip=##2\fi\fi
    \ignorespaces}%
  \def\pullinmore##1##2{\par
    \xdef\Next@{\leftskip=\the\leftskip
      \rightskip=\the\rightskip}%

```

```

\iffirstpull@\firstpull@false\else\endgroup\fi
\beginngroup
\Next@
\def\next@{##1}%
\ifx\next@\empty\else\ifx\next@\space\else
\advance\leftskip by ##1\fi\fi
\def\next@{##2}%
\ifx\next@\empty\else\ifx\next@\space\else
\advance\rightskip by ##2\fi\fi
\ignorespaces}}

```

And `\endmargins` simply has to end the current paragraph, and supply two `\endgroup`'s (one to match the `\beginngroup` from the previous `\pullin` or `\pullinmore`, and one to match the initial `\beginngroup` supplied by `\margins`):

```

\def\endmargins{\par\endgroup\endgroup}

```

Chapter 20. `\nopunct`, `\nospace`, and `\overlong`

In the next chapter we will consider `\demo`, because it uses some preliminary constructions for `\claim`, the subject of the chapter after that. Since both `\demo` and `\claim` involve punctuation and spacing that are normally supplied by a style, but which a user might want to override, this chapter is devoted to such considerations, which version 1 of \LaTeX handled with the `\nofrills` construction.

20.1. `\nopunct`, `\nospace`, and `\overlong`. In $\mathcal{A}\mathcal{M}\mathcal{S}$ - \TeX 's `amspt` style, `\nofrills` was used in several, not entirely consistent, ways (unfortunately extended yet further by the AMS in their additions to the style), and this inconsistent usage was brought over to version 1 of \LaTeX (see the small print on page 209).

In version 2 of \LaTeX , `\nofrills` has been changed to `\nopunct`, so that it affects only punctuation. It then seems silly to allow `\nopunct` to also delete the spacing after the punctuation, with `\usualspace` required to put this spacing back. Instead, it seems more consistent to have '`\nospace`' to delete the space, so that removal of the punctuation and removal of the spacing are handled separately.

In addition to these two new control sequences, `\overlong` has been retained. Although no construction in the default \LaTeX style happens to allow both `\overlong` and `\nopunct` or `\nospace`, other style might, so our macros will allow for the possibility that any combination of `\nopunct`, `\nospace`, and `\overlong` precedes some construction (however, we will assume that each of these is used just once).

In \LaTeX , `\nopunct`, `\nospace`, and `\overlong` all work in the same way, by checking whether the next control sequence after any of these is in an appropriate list, and setting a flag to be true if it is; it is then the prerogative of that next control sequence to deal with this information (and to reset the flags to false at the end).

We need three flags

```
\newif\ifnopunct@
\newif\ifnospace@
\newif\ifoverlong@
```

a list, initialized as

```
\let\nofrillslist@=\empty
```

of constructions to which both \nopunct and \nospace can apply, and a list, initialized as

```
\let\overlonglist@=\empty
```

of constructions to which \overlong can apply.

Because each of \nopunct, \nospace, and \overlong has to allow the possibility that it is followed by one or both of the others, the macros are complicated, though in no way interesting; basically, each will set the corresponding flag to be true, although the flag may be reset to false if we eventually find that an appropriate control sequence doesn't follow.

First of all, each of these construction begins with a \futurelet\next:

```
\def\nopunct{\nopunct@true\futurelet\next\nopunct@}
\def\nospace{\nospace@true\futurelet\next\nospace@}
\def\overlong{\overlong@true\futurelet\next\overlong@}
```

If \nopunct is followed by \nospace or \overlong, it will swallow these control sequences, set the corresponding flags true, and then use yet another \futurelet:

```
\def\nopunct@{%
  \ifx\next\nospace
    \def\next@\nospace{\nospace@true\futurelet\next\nopnos@}%
  \else
    \ifx\next\overlong
      \def\next@\overlong{\overlong@true\futurelet\next\nopol@}%
    \else
      \let\next@=\nopunct@@
    \fi\fi\next@}

```

We reach \nopunct@@ when neither \nospace nor \overlong follows our original \nopunct, so now we have to check whether the control sequence

that follows is in \nofrillslist@. If so, we simply execute this control sequence (the flag \ifnopunct@ has already been set true); otherwise we reset \ifnopunct@ to be false and give an error message, and still execute the control sequence:

```
\def\nopunct@@#1{\ismember@\nofrillslist@#1%
\iftest@
\let\next@=#1%
\else
\def\next@{\nopunct@false
\Err@{\noexpand\nopunct can't be used with
\string#1}#1}$%
\fi\next@}
```

(We use an argument #1 for \nopunct@@, rather than picking up the next control sequence with a \futurelet\next, so that we can properly include #1 in the error message.) For the use of \noexpand in this, any future, error messages, compare section 3.4.

Temporarily leaving aside \nopnos@ and \nopol@, the other possible outcomes of \nopunct@, we use basically the same procedures for \nospace@ and \overlong@:

```
\def\nospace@{%
\ifx\next\nopunct
\def\next\nopunct{\nopunct@true\futurelet\next\nopnos@}%
\else
\ifx\next\overlong
\def\next@\overlong{\overlong@true\futurelet\next\nosol@}%
\else
\let\next@=\nospace@@
\fi\fi\next@}
```

(notice that we use the same \nopnos@ that appeared in \nopunct@)

```
\def\nospace@@#1{\ismember@\nofrillslist@#1%
\iftest@
\let\next@=#1%
```

```

\else
\def\next@{\nospace@false
\Err@{\noexpand\nospace can't be used with
\string#1}#1}%
\fi\next@}
\def\overlong@{%
\ifx\next\nopunct
\def\next@\nopunct{\nopunct@true\futurelet\next\nopol@}%
\else
\ifx\next\nospace
\def\next@\nospace{\nospace@true\futurelet\next\nosol@}%
\else
\let\next@=\overlong@@
\fi\fi\next@}

```

(notice that we use the same \opol@ and \osol@ that appeared in \nopunct@ and \nospace@)

```

\def\overlong@@#1{\ismember@\overlonglist@#1%
\iftest@
\let\next@=#1%
\else
\def\next@{\overlong@false
\Err@{\noexpand\overlong can't be used with
\string#1}#1}%
\fi\next@}

```

Now each of \nopnos@, \opol@, and \osol@ must look for the third of the triumvirate:

```

\def\nopnos@{\ifx\next\overlong
\def\next@\overlong{\overlong@true\nopnosol@}\else
\let\next@=\nopnos@@\fi\next@}
\def\nopol@{\ifx\next\nospace
\def\next@\nospace{\nospace@true\nopnosol@}\else
\let\next@=\opol@@\fi\next@}

```

```

\def\nosol@{\ifx\next\nopunct
\def\next@\nopunct{\nopunct@true\nopnosol@}\else
\let\next@=\nosol@@\fi\next@}

```

The first of the newly created possibilities, `\nopnos@@` is easy:

```

\def\nopnos@@#1{\ismember@\nofrillslist@#1%
\iftest@
\let\next@=#1%
\else
\def\next@{\nopunct@false\nospace@false
\Err@{\noexpand\nopunct\noexpand\nospace
can't be used with \string#1#1}%
\fi\next@}

```

Notice that we may be giving the error message

```
\nopunct \nospace can't be used with ...
```

not worrying about niceties like commas between `\nopunct` and `\nospace`!

For the next two, which require testing for both `\nofrillslist@` and `\overlonglist@`, we will `\let\nextiii@=T` or `F`, depending on whether or not the argument is in `\nofrillslist@`, and `\let\nextiv@=T` or `F`, depending on whether or not the argument is in `\overlonglist@`; we use `\nextiii@` and `\nextiv@` because `\ismember@` redefines `\next@` and `\nextii@`. It helps to define a routine that makes these tests, and then also sets `\iftest@` to be true precisely when both of the tests were positive:

```

\def\testii@#1{\ismember@\nofrillslist@#1%
\iftest@\let\nextiii@=T\else\let\nextiii@=F\fi
\ismember@\overlonglist@#1%
\iftest@\let\nextiv@=T\else\let\nextiv@=F\fi
\test@false
\if\nextiii@ T\if\nextiv@ T\test@true\fi\fi}

```

Then we define `\nopol@@` and `\nosol@@`:

```

\def\nopol@@#1{\testii@{#1}%
\iftest@
\let\next@=#1%
\else\def\next@{\if\nextiii@ T\else\nopunct@false\fi
\if\nextiv@ T\else\overlong@false\fi
\Err@{\if\nextiii@ T\else\noexpand\nopunct\fi
\if\nextiv@ T\else\noexpand\overlong\fi
can't be used with \string#1}#1}%
\fi\next@}
\def\nosol@@#1{\testii@{#1}%
\iftest@\let\next@=#1%
\else\def\next@{\if\nextiii@ T\else\nospace@false\fi
\if\nextiv@ T\else\overlong@false\fi
\Err@{\if\nextiii@ T\else\noexpand\nospace\fi
\if\nextiv@ T\else\noexpand\overlong\fi
can't be used with \string#1}#1}%
\fi\next@}

```

Finally, `\nopnosol@`—when `\nopunct`, `\nospace`, and `\overlong` have all appeared—works almost the same:

```

\def\nopnosol@#1{\testii@{#1}%
\iftest@\let\next@=#1%
\else\def\next@{%
\if\nextiii@ T\else\nopunct@false\nospace@false\fi
\if\nextiv@ T\else\overlong@false\fi
\Err@{%
\if\nextiii@ T\else\noexpand\nopunct\noexpand\nospace\fi
\if\nextiv@ T\else\noexpand\overlong\fi
can't be used with \string#1}#1}%
\fi\next@}

```

20.2. Using the flags. The additional punctuation for constructions that allow `\nopunct@` will be added precisely when `\ifnopunct@` is false, so we introduce an abbreviation for that:

```
\def\punct@#1{\ifnopunct@\else#1\fi}
```

Similarly, we have

```
\def\addspace@#1{\ifnospace@\else#1\fi}
```

for adding the space.

Control sequences that allow `\overlong` are ones that normally might contain an `\hss`, like

```
\def\centerline#1{\line{\hss#1\hss}}
```

Any such candidates will have the `\hss` replaced by `\hss@`, which is defined by

```
\def\hss@{\ifoverlong@ 0pt plus1000pt minus1000pt
\else 0pt plus1000pt\fi}
```

So both stretch and shrink will be allowed when `\ifoverlong@` has been set true, but only stretch will be allowed otherwise.

Chapter 21. \demo

Since `\demo` is one of the constructions to which `\nopunct` and `\nospace` should apply, we put it in `\nofrillslist@`:

```
\rightadd@\demo\to\nofrillslist@
```

As in $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$, we will introduce a new flag,

```
\newif\ifclaim@
```

so that each `\claim` can set `\ifclaim@` to be true (within the group that that `\claim` will begin). Then `\demo` can give an error message when `\ifclaim@` is true, since we shouldn't be giving a proof within the statement of a `\claim`. (On the other hand, `\claim` will not make a similar check regarding `\demo`, since we *could* be stating subsidiary `\claim`'s within a `\demo`. In fact, the flag `\ifclaim@` is used only by `\demo`, since virtually anything other than a `\demo` can come within a `\claim`.)

As we will see in the next chapter, whenever we have started a `\claim`, or something like `\Thm` that has been constructed using `\newclaim` or `\shortenclaim`, not only will `\ifclaim@` be true, but also `\claimtype@` will be defined to be `\claim` or `\Thm`, etc. So we can give an error message that mentions `\claim` when we are in the middle of a general `\claim`, but will instead state

```
Previous \Thm has no matching \endThm
```

when we are in the middle of a `\Thm` produced with `\newclaim`, etc. We can do this with

```
\Err@{Previous \expandafter\noexpand\claimtype@ has  
no matching \string\end  
\expandafter\expandafter\expandafter\eat@\expandafter  
\string\claimtype@}
```

Here (compare page 126) the `\expandafter` is expanded in the error message, so `\claimtype@` is expanded—to `\claim`, `\Thm`, or whatever—before

having `\noexpand` placed in front of it, which then prevents further expansion. See *The T_EXbook*, page 374 for the triple `\expandafter`; here `\claimtype@` is expanded first, to `\claim` or `\Thm`, etc., then `\string` is applied to this, and then `\eat@` (section 1.2) is applied to the result of this `\string`, thus eating the `\` at the beginning of the `\string\claim` or `\string\Thm` or whatever. (Finally, see section 3.4 for the spacing after the `\noexpand`'ed control sequence.)

The combination

```
\expandafter\expandafter\expandafter\eat@\expandafter\string
```

will occur quite frequently, so we abbreviate it:

```
\def\exxx@{\expandafter\expandafter\expandafter
\eat@\expandafter\string}
```

Normally (i.e., when `\ifclaim@` is false), `\demo#1` will end the previous paragraph and add a `\smallskip` (deleting the previous skip if smaller, or using the previous skip instead, if it is larger).

Then it will begin a group and start an unindented paragraph with `#1` in `\smc` (with spaces before and after `#1` ignored, since they are presumably typing errors); we use `\noindent@@` for this (Chapter 8). This will be followed by a colon, unless `\nopunct@` preceded the demo,

```
\punct@:
```

and an `\enspace`, unless `\nospace` preceded the `\demo`,

```
\addspace@\enspace
```

Instead of typing a (type 12) `:`, we will

```
\let\colon@=:
```

and use `\colon@` instead, so that a French style that makes `:` active can change `\colon@` (compare 3.10). Moreover, we will use

```
\punct@{\null\colon@}
```

just in case #1 happens to end with an upper-case letter (although this will usually be irrelevant, since an \enspace, rather than a space, follows).

At this point, after we have determined whether or not the colon and \enspace should be added, we will reset \ifnopunct@ and \ifnospace@ to be false immediately, even though \enddemo will also do this, just to minimize problems if the user forgets the \enddemo.

Finally, we want to switch to \rm. However, we must also be careful to add a \FNSSP@, in case an invisible construction follows the \demo{...} (this will also throw away any extraneous space after the }). We can't simply say

```
\def\demo#1{\ifclaim@ ...
  \else
  . . .
  \rm\FNSSP@\fi}
```

because \next will be \let equal to the \fi, rather than to the next non-space token after \demo{...}. So instead we have to use the definition

```
\def\demo#1{\ifclaim@
  \Err@{Previous \expandafter\noexpand\claimtype@ has
    no matching \string\end\exxx@\claimtype@}%
  \let\next@=\relax
\else
  \par
  \ifdim\lastskip<\smallskipamount
    \remove\lastskip\smallskip\fi
  \begingroup
  \noindent@{\smc\ignorespaces#1\unskip
    \punct@{\null\colon@}\addspace@\enspace}%
  \nopunct@false\nospace@false
  \rm
  \def\next@{\FNSSP@}%
\fi
\next@}
```

\enddemo simply has to end the current paragraph, supply the \endgroup to match the \begingroup from \demo, reset the flags \ifnopunct@ and \ifnospace@, and insert a \smallskip:

```
\def\enddemo{\par\endgroup  
\nopunct@false\nospace@false\smallskip}
```

Chapter 22. `\claim's`

Now we come to `\claim` and related constructions, one of the more complicated complexes in \LaTeX .

22.1. Preliminaries. First of all, `\claim` is another thing that can follow `\nopunct`, so we add it to `\nofrillslist@`:

```
\rightadd@{\claim}\to\nofrillslist@
```

The fontstyle for `\claim's`,

```
\def\claim@F{\smc}
```

will be needed right away, but, in fact, we will need something more general.

In Chapter 21, we pointed out that `\claimtype@` will be defined to be `\claim` when we have started an ordinary `\claim`, but `\Thm` when we have started a construction like `\Thm` that has been constructed with `\newclaim` or `\shortenclaim`.

Each construction like `\Thm` will have an associated font style `\Thm@F`. This can be named (see page 119) as

```
\csname\exstring@\Thm @F\endcsname
```

or, more generally (recall the definition of `\exxx@` on page 162), as

```
\csname\exxx@\claimtype@ F\endcsname
```

Since we often need to refer to this general font style, we introduce the special construction

```
\def\claim@@@F{\csname\exxx@\claimtype@ @F\endcsname}
```

[We use a triple `@@@`, rather than the double `@@` used in `\list@@F` to emphasize the distinction: `\list@@F` depends on the value of the counter `\listlevel@`, the level of a `\list`, while `\claim@@@F` depends on `\claimtype@`, the name of the `\claim`.]

Now we can introduce `\claimformat@` to indicate the general format of a `\claim`:

```
\def\claimformat@#1#2#3{\medbreak
\noindent@@{\smc#1 {\claim@@@F#2} #3%
\punct@{\null.}\addspace@\enspace}\sl}
```

(Compare page 162 for the `\null.` and Chapter 8 for the `\noindent@@`.) Arguments #1 and #3 for `\claimformat@` are the two arguments that a user types in

```
\claim{...}{...}
```

while argument #2 is the `\claim` number, either produced automatically, or specifically “quoted” after `\claim`.

Section 11 explains how a style file can modify `\claimformat@` to deal with numerous possibilities for formatting different sorts of `\claims` in different ways.

22.2. `\claimformat@@`. Since `\claimformat@` is meant to be easily modified by a style designer, it omits several messy details:

- (1) Any extraneous spaces at the beginning and end of arguments #1 and #3 should be removed.
- (2) If #3 is empty, or a space (which occurs if the user types `{ }` instead of `{}`), then the space before it must be removed.
- (3) #2 should be the properly formatted `\claim` number. Moreover, if this is empty (because the user typed `\claim""`), then the space before it should be removed.
- (4) A space following the `\claim{...}{...}` should be ignored; more generally, we need `\FNSSP@`, in case an invisible construction follows.

So instead of using `\claimformat@` directly, we will use `\claimformat@@`, which calls `\claimformat@` with all these details added. When we are using `\claimformat@@`, the control sequence `\thelabel@@` will contain the properly formatted claim number.¹ In the definition below, argument #2 corresponds to argument #3 for `\claimformat@`. Only the next-to-last line of code needs further amplification.

¹The formatting of the `\claim` number is specified by `\claim@S`—it should not be specified directly in `\claimformat@`.

```

\def\claimformat@@#1#2{%
  \claimformat@{\ignorespaces#1\unskip}%
  {\ifx\thelabel@@\empty\unskip\else\thelabel@@\fi}%
  {\ignorespaces#2\unskip}%
  \let\Claimformat@@=\claimformat@@
  \FNSSP@}

```

Note that it wasn't necessary to add any special clause for the cases where #2 is empty or a space—in either case the space preceding #2 will end up being removed. (Section 11 illustrates how modifications may be made to the definition of \claimformat@. In some cases, \claimformat@@ might need some tinkering also.)

To explain the mysterious next-to-last line of code, we have to confess to a little white lie. \claim, and all related constructions, never actually use \claimformat@@. Instead they use \Claimformat@@, which we will initially set to be the same as \claimformat@@:

```

\let\Claimformat@@=\claimformat@@

```

This indirect approach has been implemented to deal with constructions produced with \newclaim and \shortenclaim. Suppose, for example, that we produce \Thm with

```

\newclaim\Thm\c{thm}{Theorem}

```

Roughly speaking, this defines \Thm as

```

\def\Thm{ ...
  \def\Claimformat@@{\claimformat@@{Theorem}} ...
  \claim }

```

The \claim in this definition will call \Claimformat@@, and hence

```

\claimformat@@{Theorem}

```

so that the ‘Theorem’ label will automatically be inserted. Although the final `\endclaim` will return `\Claimformat@@` to its original state, so that any subsequent `\claim` will just be using `\claimformat@@` (unless it happens to be called by another construction that redefines `\Claimformat@@`), we add

```
\let\Claimformat@@=\claimformat@@
```

at this point, just to minimize problems if the user forgets the `\endclaim`.

22.3. Further preliminaries. Although we won’t consider all the complications of `\newclaim` until later, there are several other aspects that we need to mention before moving on.

We’ve already used `\claimtype@` to define `\claim@@@F`. More generally, we will

```
\def\claim@@@P{\csname\exxx@\claimtype@ @P\endcsname}
\def\claim@@@Q{\csname\exxx@\claimtype@ @Q\endcsname}
\def\claim@@@S{\csname\exxx@\claimtype@ @S\endcsname}
\def\claim@@@N{\csname\exxx@\claimtype@ @N\endcsname}
```

The counter has to be treated differently, however, for the following reason. If the user directly types something like

```
\claim\c{thm}
```

then the style, numbering style, pre- and post-material for this `\claim` are just the standard ones. *But a new counter must be involved*, since such `\claim's` are to be numbered independently from other `\claim's`; this new counter is required even if no `\newclaim` construction has been used.

To keep track of these different numbering classes, we will introduce `\claimclass@` in addition to `\claimtype@`. A `\claim` with a particular “class”, produced by

```
\claim\c{(claim class)}
```

defines `\claimclass@` to be this `(claim class)` (ordinary `\claim's`, which are equivalent to `\claim\c{}`, define `\claimclass@` to be empty). And the `\newclaim` construction

```
\newclaim\Thm\c{thm}{Theorem}
```

makes `\Thm` define `\claimclass@` to be ‘`thm`’ also.

Now, when `\claim\c{thm}` is typed, we will use

```
\claim@Cthm
```

for the counter. More generally, we will use

```
\csname claim@C\claimclass@\endcsname
```

(As we will see later, the construction

```
\newclaim\Thm\c{thm}{Theorem}
```

creates `\Thm@P`, `\Thm@Q`, etc., directly, but it creates `\Thm@C` indirectly: first the counter `\claim@Cthm` is created, if it doesn’t already exist, and then `\Thm@C` is made equivalent to this counter.)

Consequently, the counter `\claim@@@C` is simply defined by:

```
\def\claim@@@C{\csname claim@C\claimclass@\endcsname}
```

It is the `\claimclass@` of a construction like `\Thm`, etc., that determines its numbering. Consequently,

```
\newclaim\Thm\c{thm}{Theorem}
\newclaim\Lem\c{lem}{Lemma}
```

produces `\Thm`’s and `\Lem`’s that are numbered independently, while

```
\newclaim\Thm\c{thm}{Theorem}
\newclaim\Lem\c{thm}{Lemma}
```

makes `\Thm`’s and `\Lem`’s share the same numbering.

22.4. Starting a `\claim`. First we introduce the other components of a printed `\claim` number:

```

\newcount\claim@C
\claim@C=0
\let\claim@P=\empty
\let\claim@Q=\empty
\def\claim@S#1{#1\}/}
\let\claim@N=\arabic

```

Compare page 105 for the ‘\let\claim@N=\arabic’.

\claim initializes \ifclaim@ to be true, \claimclass@ to be empty, and \claimtype@ to be \claim, and then uses a \futurelet to see if we are “quoting” the claim number with a " or specifying a claim class with \c:

```

\def\claim{\claim@true
\let\claimclass@=\empty\def\claimtype@{\claim}%
\futurelet\next\claim@}

```

If \claim is followed by \c, we will use \claim@c, and if it is followed by " we will use \claim@q,

```

\def\claim@{%
\ifx\next\c
\let\next@=\claim@c
\else
\ifx\next"%
\let\next@=\claim@q
\else
. . .

```

Otherwise, we will use

```
\begingroup
```

to begin a group. [The definitions of \claim@c (section 5) and \claim@q (section 6) will each begin with \begingroup also; but we don't simply put the \begingroup into the definition of \claim because, as we will see later (page 179), constructions produced by \newclaim or \shortenclaim lead us directly to \claim@c, without passing through \claim.]

Then we will advance the \claim counter by 1, define \thelabel@, ... in the usual way, and then call \Claimformat@. In the following, we can use \claim@c, ... rather than the more general \claim@@@C, ... because, as we have just mentioned, this clause of \claim@ will only be called directly by \claim, never indirectly by something created by \newclaim:

```

\def\claim@{%
  \ifx\next\c
    \let\next@=\claim@c
  \else
    \ifx\next"%
      \let\next@=\claim@q
    \else
      \begingroup
      \global\advance\claim@c by 1
      {\noexpands@
        \xdef\Thelabel@@@{\number\claim@c}%
        \xdef\Thelabel@\claim@N
        \xdef\Thelabel@@@{\claim@P\Thelabel@\claim@Q}}%
        \xdef\Thelabel@@\claim@S
      }%
      \locallabel@
      \let\next@=\Claimformat@@
    \fi
  \fi
\next@}

```

22.5. Starting a \claim@c. The definition of \claim@c begins

```
\def\claim@c\c#1{\claim@true\begingroup
```

so that \claim@c swallows up the succeeding \c, sets \ifclaim@ to be true, and then begins a group. We add the \claim@true even though this appears in \claim, because constructions created by \newclaim and \shortenclaim call \claim@c directly (pages 179 ff. and 182 ff.).

Next we have to find out if the counter

```
\csname claim@c#1\endcsname
```

already exists (which will happen if \claim\c{#1} has already appeared); this is easily done with the test

```
\expandafter\ifx\csname claim@C#1\endcsname\relax
```

which is true when the counter hasn't been defined yet. If the counter *has* already been defined, we simply advance it by 1. If it hasn't been defined, we create it, with \newcount@ (compare page 121), and set it to 1.

Then we define \claimclass@ to be #1, and define \thelabel@, ...; in this case, for the preliminary construction, defining \Thelabel@, ... , we do use the more general constructions \claim@@@C, ... (page 168); fortunately, these definitions can all be used within \xdef's. Finally, we need a \futurelet to see if our \claim@c\c{...} is followed by a quoted number "...", which can happen when \claim@c is called by a construction created by \newclaim or \shortenclaim. We have to allow the possibility that a space might intervene between the \c{...} and a ", so we use \FNSS@ (section 3.8):

```
\def\claim@c\c#1{\claim@true
\beginngroup
\expandafter\ifx\csname claim@C#1\endcsname\relax
\expandafter\newcount@\csname claim@C#1\endcsname
\global\csname claim@C#1\endcsname=1
\else
\global\advance\csname claim@C#1\endcsname by 1
\fi
\def\claimclass@{#1}%
{\noexpands@
\xdef\Thelabel@@@{\number\claim@@@C}%
\xdef\Thelabel@\claim@@@N
\xdef\Thelabel@@@{\claim@@@P\Thelabel@\claim@@@Q}}%
\xdef\Thelabel@@\claim@@@S
}%
\locallabel@
\FNSS@\claim@c}
```

We're not really done with \claim@c yet, but we will return to \claim@c@ in a moment.

22.6. Starting a `\claim@q`. For our definition of `\claim@q` we use `\qlabel@` from section 16.3 for defining `\thelabel@`, ..., and we add a `\FNSS@`, since we have to see whether our `\claim@q` is followed by `\c{...}` (possibly after a space):

```
\def\claim@q"#1"{\begingroup
  {\let\pre=\claim@@@P \let\post=\claim@@@Q
   \let\style=\claim@@@S \let\numstyle=\claim@@@N
   \noexpands@
   \qlabel@{#1}}%
  \locallabel@
  \FNSS@\claim@q}
```

Unlike the situations for `\maketag@` and `\item@`, we are not yet ready to actually typeset the quoted `\claim` number; however, the number that we want to typeset has been safely stored in `\thelabel@@`, which eventually finds its way into `\claimformat@@`.

In regard to the `\let\pre=\claim@@@P`, ..., compare page 140.

22.7. Finishing off. Let's return to `\claim@c`, which ended with

```
\FNSS@\claim@c@
```

Here `\claim@c@` must check to see whether `\next` is ". If `\next` is not ", we just call `\Claimformat@@`. If `\next` is ", we will call yet another routine `\claim@cq`. But we will also have to make an adjustment: Remember that `\claim@c` has already increased the appropriate counter `\csname claim@c\claimclass@\endcsname` by 1. If `\claim@c@` finds a " next, so that the claim number is actually being quoted, then it must counteract this change:

```
\def\claim@c@{\ifx\next"%
  \global\advance\claim@@@C by -1
  \let\next@=\claim@cq
  \else\let\next@=\Claimformat@@
  \fi
  \next@}
```

The definition of `\claim@c` is now fairly straightforward, using only devices already encountered; instead of `\Claimformat@@` at the end, we use `\FNSS@\Claimformat@@`, just to get rid of a possible space following the second ":

```
\def\claim@cq"#1"{\let\pre=\claim@@P \let\post=\claim@@Q
\let\style=\claim@@S \let\numstyle=\claim@@N
\noexpands@
\Qlabel@{#1}}%
\locallabel@
\FNSS@\Claimformat@@}
```

Similarly, our definition of `\claim@q` ended with

```
\FNSS@\claim@q@
```

where `\claim@q@` can simply be defined by

```
\def\claim@q@{\ifx\next\c\expandafter\claim@qc
\else\expandafter\Claimformat@@\fi}
```

(the "K-method" again, see section 1.1).

We reach `\claim@qc` only when we have the combination

```
\claim"..."\c{...}
```

(never via a construction that has been created with `\newclaim`), which is actually pretty unlikely, since there's not much point indicating the class of a `\claim` if the number is being quoted (unless different classes of `\claim's` are going to be formatted differently, in which case the style designer has presumably already used `\newclaim` to introduce a new name), but we might as well carry it through.

Before calling `\Claimformat@@`, we just have to use the `\c{...}` part to define `\claimclass@`, and, just in case this particular class `\c{...}` of `\claim's` has never been used before, create the new counter '`\claim@...`', if necessary, and set it to 0 (not to 1, as in `\claim@c`, since the counter isn't going

to be used now). And finally, we must again use `\FNSS@Claimformat@@` to skip over any space after the `\c{...}`:

```
\def\claim@qc#c#1{\expandafter
\ifx\csname claim@C#1\endcsname\relax
\expandafter\newcount@\csname claim@C#1\endcsname
\global\csname claim@C#1\endcsname=0 \fi
\FNSS@Claimformat@@}
```

22.8. \endclaim. Finally, `\endclaim` simply ends the group begun by `\claim` (or `\claim@c` if called by something created by `\newclaim` or `\shortenclaim`), sets `\ifclaim@` and `\ifnopunct@` to be false, resets `\Claimformat@@` to be `\claimformat@@`, and adds a `\medbreak`.

```
\def\endclaim{\endgroup\claim@false
\nopunct@false\nospace@false
\let\Claimformat@@=\claimformat@@\medbreak}
```

22.9. \newclaim. To finish off the entire `\claim` complex, we have to define `\newclaim` and `\shortenclaim`.

`\newclaim` allows `\claimclause` as optional syntax, and we do the standard thing to prevent its being used at inappropriate times (see section 1.1):

```
\Invalid@\claimclause
```

`\newclaim` itself will require a `\futurelet` to see if `\claimclause` comes next:

```
\def\newclaim{\futurelet\next\newclaim@}
```

If `\claimclause` does come next, we will swallow the `\claimclause` and incorporate the following argument into

```
\newclaim@@{#1}
```

otherwise, the result will be the same as if we had typed

```
\newclaim\claimclause\relax
```

so we will simply call \newclaim@\relax directly:

```
\def\newclaim@{\ifx\next\claimclause
\def\next@\claimclause##1{\newclaim@{##1}}\else
\def\next@{\newclaim@\relax}\fi\next@}
```

Thus, we have reduced everything to the definition of \newclaim@.

Several aspects of \newclaim@ are included for eventual use by the \shortenclaim construction. For example, something like

```
\shortenclaim\Thm\thm
```

should be allowed only if \Thm has already been created with \newclaim; for this reason, we are going to keep a list, \claimlist@, of all such possibilities, and we initialize it as:

```
\def\claimlist@{\}\claim}
```

We will also need two new token lists,

```
\newtoks\claim@i
\newtoks\claim@v
```

and, as we'll see, one other auxiliary control sequence is going to be defined in the process of defining \newclaim@. Finally, there will be a situation where we *don't* want the \claimclause to be inserted, even if it has been specified. For this purpose, we initially

```
\let\nocclaimclause@=F
```

In the special case where we want to suppress the claim clause, we will \let\nocclaimclause@=T (this happens only once, and is more efficient than declaring \ifnocclaimclause@, which actually creates three control sequence names).

If $\langle _ \rangle$ denotes an optional space, then \newclaim will appear in something of the form

```
\newclaim \claimclause {(claim clause)}\langle \_ \rangle\Thm \c{thm}\langle \_ \rangle{Theorem}
```

so \newclaim@@ will appear in something of the form

(A) \newclaim@@ {<claim clause>}_␣\Thm \c{thm}_␣{Theorem}

Our definition of \newclaim@@ will be of the form

```
\def\newclaim@@#1#2#3\c#4#5{ . . .
```

In case (A), argument #1 will be <claim clause> (which may be empty), argument #2 will be ‘\Thm’, and argument #3 will be empty. In fact, argument #3 will *always* be empty, but by adding the #3 before the \c we make argument #2 an undelimited argument, and consequently argument #2 will simply be ‘\Thm’ even in the situation where the space occurs,

```
\newclaim@@{...}_␣\Thm\c{thm}
```

Similarly argument #4 will be ‘thm’, and argument #5 will be ‘Theorem’ (even if a space precedes ‘{Theorem}’, since #5 is an undelimited argument).

The definition of \newclaim@@ begins

```
\def\newclaim@@#1#2#3\c#4#5{\define#2{}}%
\rightadd@#2\to\claimlist@\rightadd@#2\to\nofrillslist@
```

Illustrating with case (A) again, the \define\Thm{} is inserted simply to allow \define to give an error message if \Thm is already defined—for in that case we certainly don’t want \Thm to be given another meaning by \newclaim. Then we add \Thm to \claimlist@, for use by \shortenclaim, as explained above, and to \nofrillslist@, since we will want \nopunct\Thm to be allowed.

Next we have to create default values for \Thm@P, \Thm@Q, \Thm@S, \Thm@N, \Thm@F, which will simply be the default values for \claim@P, ... :

```
\expandafter\def\csname\exstring@#2@P\endcsname{\claim@P}
\expandafter\def\csname\exstring@#2@Q\endcsname{\claim@Q}
\expandafter\def\csname\exstring@#2@S\endcsname{\claim@S}
\expandafter\def\csname\exstring@#2@N\endcsname{\claim@N}
\expandafter\def\csname\exstring@#2@F\endcsname{\claim@F}
```

Notice that we use `\def`, rather than `\let`, so that when `\newclaim\Thm` is used, `\Thm@P, ...` will be assigned the current values of `\claim@P, ...`. These may have been modified by some

```
\newpre\claim
\newpost\claim
\newstyle\claim
\newnumstyle\claim
\newfontstyle\claim
```

That was done because style files may quite well define `\claim@P` to be something like

```
(chapter number).(section number).
```

and we would presumably want such a numbering scheme to be carried through for our `\Thm`.

[A `\newpost` will usually be made only locally, so that `\claim@Q` probably won't change, except for individual `\claim's`. As for `\claim@S` and `\claim@N`, if the user changes them, presumably the change should be brought along also for `\Thm`. One could always change some of this code, to

```
\expandafter\let\csname\exstring@\Thm@S\endcsname=\claim@S
```

for example, if this doesn't seem like the best arrangement.]

We also make `\endThm` mean `\endclaim`:

```
\expandafter\def\csname end\exstring@#2\endcsname{\endclaim}
```

The counter `\claim@Cthm` may already exist (because of a previous `\claim\c{thm}`); if not, we must create it, with `\newcount@` (again, see page 121), and, for safety's sake, initialize it to 0:

```
\expandafter\ifx\csname claim@C#4\endcsname\relax
\expandafter\newcount@\csname claim@C#4\endcsname
\global\csname claim@C#4\endcsname=0 \fi
```

Then we have to `\let\Thm@C=\claim@Cthm`, which is accomplished by the code

```
\edef\next@{\let
\csname\exstring@#2@C\endcsname
=\csname claim@C#4\endcsname\endcsname}
\next@
```

Here the first `\csname... \endcsname` will be expanded to `\Thm@C`, which will be made equivalent to `\relax`, since `\Thm@C` isn't already defined; on the other hand, the second `\csname... \endcsname` will expand to `\claim@Cthm`, which has been created with `\newcount@`, and hence with a `\countdef`; such control sequences aren't expanded further in an `\edef` (compare page 55).

After all this, we will want to define `\Thm`:

```
\def#2{ . . .
\def\claimtype@{#2}%
\def\Claimformat@@{\claimformat@@{#5}}\claim@c\c{#4}}}
```

Thus, `\Thm` will not call `\claim` directly, but instead call

```
\claim@c\c{thm}
```

This is where `\claim\c{thm}` usually gets us. The difference is that we first take the opportunity to define `\Claimformat@@` as

```
\claimformat@@{Theorem}
```

so that the argument 'Theorem' is automatically supplied, and we also take the opportunity to

```
\def\claimtype@{\Thm}
```

In addition, for the sake of `\shortenclaim` we want to add

```
\global\claim@i={#1}\gdef\claim@iv{#4}\global\claim@v={#5}
```

In situation (A), \Thm will thus store the (claim clause) in the token list \claim@i, the class 'thm' in \claim@iv, and 'Theorem' in the token list \claim@v:

```
\def#2{ . . .
  \global\claim@i={#1}\gdef\claim@iv{#4}\global\claim@v={#5}
  \def\claimtype@{#2}%
  \def\Claimformat@{\claimformat@{#5}}\claim@c{c{#4}}}
```

In the next section we will see why we need \global assignments.

Finally, we want the claim clause, argument #1, to be executed, except in the special case where we have \let\nocclaimclause@=T:

```
\def#1{\ifx\nocclaimclause@ T\else#1\fi
  \global\claim@i={#1}\gdef\claim@iv{#4}\global\claim@v={#5}
  \def\claimtype@{#2}%
  \def\Claimformat@{\claimformat@{#5}}\claim@c{c{#4}}}
```


So the full definition of \newclaim@@ reads:

```
\def\newclaim@@#1#2#3\c#4#5{\define#2{}}%
  \rightadd@#2\to\claimlist@\rightadd@#2\to\nofrillslist@%
  \expandafter\def\csname\exstring@#2@P\endcsname{\claim@P}%
  \expandafter\def\csname\exstring@#2@Q\endcsname{\claim@Q}%
  \expandafter\def\csname\exstring@#2@S\endcsname{\claim@S}%
  \expandafter\def\csname\exstring@#2@N\endcsname{\claim@N}%
  \expandafter\def\csname\exstring@#2@F\endcsname{\claim@F}%
  \expandafter\def\csname end\exstring@#2\endcsname{\endclaim}%
  \expandafter\ifx\csname claim@C#4\endcsname\relax
  \expandafter\newcount@\csname claim@C#4\endcsname
  \global\csname claim@C#4\endcsname=0 \fi
  \edef\next@{\let
  \csname\exstring@#2@C\endcsname
  =\csname claim@C#4\endcsname}%
  \next@
```

```

\def#2{\ifx\noclaimclause@ T\else#1\fi
\global\claim@i{#1}\gdef\claim@iv{#4}\global\claim@v{#5}%
\def\claimtype@{#2}%
\def\Claimformat@@{\claimformat@{#5}}\claim@c{c{#4}}}}

```

 We `\def\endThm{\endclaim}` rather `\let\endThm=\endclaim` for situations where a style file changes `\endclaim`, and also uses `\newclaim` to create special claims, like `\Thm`. If we used `\let`, then the `\newclaim\Thm` *would have to come after* the redefinition of `\endclaim`, or `\endThm` would have the wrong meaning.

On the other hand, even if `\endclaim` isn't changed, but the particular case of `\endThm` is supposed to be different (leaving extra space perhaps, or possibly even something more extreme, like an `\hrule` across the page), then one might have to

```

\redefine\endThm{\endgroup\claim@false\nopunct@false
(special formatting after \Thm)}

```

But special formatting at the beginning of `\Thm` should be handled as in the case of `\Conj` in section 11.

22.10. \shortenclaim. Finally, we come to `\shortenclaim#1#2`, a typical usage of which is

```
\shortenclaim\Thm\thm
```

The first thing `\shortenclaim` will do, when called this way, is

```
\define\thm{}
```

to get an error message if `\thm` is already defined.

Next, `\shortenclaim` will try the test

```
\ismember@\claimlist@\Thm
```

If this is false, we will just give an error message

```
\Thm not yet created by \newclaim.
```

Otherwise, we will first add \thm to \nofrillslist@, and then make \thm@S be \Thm@S, etc:

```
\rightadd@#2\to\nofrillslist@
\expandafter\def\csname\exstring@#2@S\endcsname
  {\csname\exstring@#1@S\endcsname}
. . .
```

(The *L_AT_EX* Manual incorrectly implies, on page 45, that \thm@S, etc., will actually be \claim@S, etc., on the grounds that these constructions for \Thm might be changed later on. That would indeed be a problem if we used \let instead of \def, but with the \def, any use of \newstyle\Thm, etc., will automatically carry over to \thm. A problem arises only if something like \newstyle\thm is ever used; in that case, any succeeding \newstyle\Thm's will no longer carry over to \thm.)

Then, as with \newclaim, we must make \endthm mean \endclaim, and let \thm@C be \Thm@C.

Finally, \shortenclaim will have to define \thm. The problem now is that we would like to

```
\def\thm{(claim clause)
\def\claaintype@{\thm}%
\def\Claimformat@@{\claimformat@@{Theorem}-{}}%
\claim@c\c{thm}}
```

where (claim clause) is the claim clause for \Thm.

So we need a way of getting this (claim clause), as well as the 'thm' and 'Theorem' that actually go with \Thm. The strategy for this is to set a box

```
(A) \setbox0=\vbox{\Thm""\relax\endgroup}
```

because the globally defined token list \claim@i will then contain our desired (claim clause), \claim@iv will be defined to be 'thm', and the token list \claim@v will contain 'Theorem'. We use an empty label "" for the \Thm so that the counter won't be increased, and instead of \endThm, we just use \endgroup, since the other parts of \endThm will be irrelevant in this situation.

Once we have recovered these quantities, we will be in a position to globally `\def\thm`, but again we will need a somewhat indirect route:

```
\xdef#2{\the\claim@i
\def\noexpand\claimtype@{\noexpand#2}%
\def\noexpand\Claimformat@@
{\noexpand\claimformat@@{\the\claim@v}{}}%
\noexpand\claim@c\noexpand\c{\claim@iv}}
```

This works as before, noting that for a token list like `\claim@i`, the expansion of `\the\claim@i` is simply that token list. (We need token lists to store the `\claim` clause) and the part of the `\claim` exemplified by ‘Theorem’ because both of these might contain control sequences, which should remain unexpanded in the `\xdef`; on the other hand, the claim class, like a `\label`, is not supposed to have expandable tokens in it.)

But there is still one little fillip that we need to add: Instead of (A), we really want to use

```
\setbox0=\vbox{\let\noclaimclause@=T
\Thm""\relax\endgroup}
```

To see why, imagine a situation (compare section 11), where we have

```
\newclaim\Cor\c{cor}{Corollary}
\newclaim\claimclause{\Reset\Cor1}\Thm\c{thm}{Theorem}
\shortenclaim\Thm\thm
```

so that each `\Thm` and `\thm` resets the numbering of `\Cor`’s to 1. If we were to declare these in a different order,

```
\newclaim\claimclause{\Reset\Cor1}\Thm\c{thm}{Theorem}
\shortenclaim\Thm\thm
\newclaim\Cor\c{cor}{Corollary}
```

then the `\shortenclaim\Thm\thm` will cause us to

```
\setbox0=\vbox{\Thm""\relax\endgroup}
```

and if the <claim clause> \Reset\Cor1 were executed, we would get an error message, since \Cor hasn't been created yet! So we add the

```
\let\noclaimclause@=T
```

which prevents the <claim clause> from being executed.

Summing up, the definition of \shortenclaim reads:

```
\def\shortenclaim#1#2{\define#2{}}%
\ismember@\claimlist@#1%
\iftest@
\rightadd@#2\nofrillslist@%
\expandafter\def\csname\exstring@#2@S\endcsname
{\csname\exstring@#1@S\endcsname}%
\expandafter\def\csname\exstring@#2@N\endcsname
{\csname\exstring@#1@N\endcsname}%
\expandafter\def\csname\exstring@#2@P\endcsname
{\csname\exstring@#1@P\endcsname}%
\expandafter\def\csname\exstring@#2@Q\endcsname
{\csname\exstring@#1@Q\endcsname}%
\expandafter\def\csname\exstring@#2@F\endcsname
{\csname\exstring@#1@F\endcsname}%
\expandafter\def
\csname end\exstring@#2\endcsname{\endclaim}%
\edef\next@{\let
\csname\exstring@#2@C\endcsname
=\csname claim\exstring@#1@C\endcsname}%
\next@
\setbox0=\vbox{\let\noclaimclause@=T#1""\relax\endgroup}%
\edef#2{\the\claim@i
\def\noexpand\claimtype@{\noexpand#2}%
\def\noexpand\Claimformat@@
{\noexpand\claimformat@@{\the\claim@v}\relax}%
\noexpand\claim@c\noexpand\c{\claim@iv}}%
\else
\Err@{\string#1 not yet created by \string\newclaim}%
\fi}
```

22.11. Customizing \claim's. To assist in printing different sorts of \claim's in different ways, L^AT_EX provides two tests,

```
\def\classtest@#1{\def\next@{#1}%
\ifx\next@\claimclass@\test@true\else\test@false\fi}
\def\typetest@#1{\def\next@{#1}%
\ifx\next@\claimtype@\test@true\else\test@false\fi}
```

Thus, for example, \classtest@{thm} will set \iftest@ to be true precisely when \claimclass@ is 'thm', and \typetest@\Thm will set \iftest@ to be true precisely when \claimtype@ is \Thm.

To see how these are used, let us suppose that we want our Theorems, Lemmas, Corollaries, Definitions and Conjectures to be printed as follows (temporarily switching to Computer Modern fonts):

LEMMA 6. *If n is odd, then n^2 is odd; and if n is even, then n^2 is even.*

COROLLARY 1 (CONVERSE). *If n^2 is odd, then n is odd; and if n^2 is even, then n is even.*

THEOREM 7 (PYTHAGORAS). *$\sqrt{2}$ is irrational.*

DEFINITION 8. An integer x is a *perfect square* if $x = y^2$ for some integer y .

? CONJECTURE A. *If x is not a perfect square, then \sqrt{x} is irrational.*

Thus, Theorems, Lemmas, and Definitions are all numbered together, but Corollaries begin at 1 after each Theorem or Lemma. In addition, Definitions are in \rm type. Finally, Conjectures are numbered completely independently, as A, B, C, ... , and they have a ? in the margin.

To set things up, we first use

```
\newclaim\claimclause{\Reset\Cor1}\Thm\c{thm}{Theorem}
\shortenclaim\Thm\thm

\newclaim\claimclause{\Reset\Cor1}\Lem\c{thm}{Lemma}
\shortenclaim\Lem\lem

\newclaim\Cor\c{cor}{Corollary}
\shortenclaim\Cor\cor

\newclaim\Defn\c{thm}{Definition}
\shortenclaim\Defn\defn
```

```

\newclaim\Conj\c{conj}{Conjecture}
\shortenclaim\Conj\conj
\newnumstyle\Conj\Alph

```

Since \Thm, \Lem and \Defn all have the same claim class 'thm', they will be numbered together (and the numbering for \thm, \lem, and \defn follow those for \Thm, \Lem, and \Defn, respectively). The \claimclause's in the \newclaim's for \Thm and \Lem ensure that Corollary numbers start at 1 after each one; these claim clauses carry over to \thm and \lem.

Finally, we just have to redefine \claimformat@ as follows:

```

\catcode'\@=11
\def\claimformat@#1#2#3{\medbreak\noindent@
\classtest@{conj}\iftest@{\llap{\bf?\hskip5pt}\fi
{\smc#1 {\claim@@@F#2} #3\punct@{\null.}\addspace@\enspace}%
\typetest@\Defn
\iftest@\rm
\else
\typetest@\defn
\iftest@\rm
\else\sl\fi\fi}
\catcode'\@=\active

```

Then the output on page 185 will be produced by

```

\lem If  $n$  is odd, ... \endlem

\Cor {(Converse)} If  $n^2$  is odd ... \endCor

\Thm{(Pythagoras)}  $\sqrt{2}$  is irrational.\endThm

\defn An integer  $x$  is a perfect square ... \enddefn

\conj If  $x$  is not a perfect square, ... \endconj

```

Chapter 23. Heading levels

The “heading levels” `\HL` and `\hl` exhibit a strange mixture of the features of `\list` and `\claim`. They have levels, like `\list`, but these levels indicate separate entities, rather than parts of a single construction. And we can create new names, like `\chapter` and `\section` for, say, `\HL1` and `\hl1`, but this is somewhat different than creating new `\claim`'s with `\newclaim`, because `\chapter` is essentially just a synonym for `\HL1`, rather than a new class of heading levels.

Moreover, heading levels introduce yet another complication, since they are constructions that get written to the table of contents file, which we need to consider first.

23.1. The .toc file. In version 1 of *L_AT_EX*, headings were written to the `.toc` file, while the corresponding page numbers were written to a separate `.tpg` file. Now, however, the page numbers are written together with the headings in the `.toc` file.

On the other hand, `\island`'s, and their page numbers, will be written to a separate file, which I can't resist calling the `.tic` file, even though it's not really a proper acronym. Corresponding to `\indexfile` (page 52), we have

```
\newif\iftoc@
\def\tocfile{\iftoc@\else
  \alloc@7\write\chardef\sixt@@n\toc@
  \immediate\openout\toc@=\jobname.toc
  \alloc@7\write\chardef\sixt@@n\tic@
  \immediate\openout\tic@=\jobname.tic
  \global\toc@true\fi}
```

23.2. Preliminaries. To begin with, we want to add `\hl` to `\nofrillslist@`:

```
\rightadd@\hl\to\nofrillslist@
```

We don't need to add `\HL` to `\nofrillslist@`, since these heading levels don't have any punctuation in the default style (compare the small print section on page 209). But `\HL` can be preceded by `\overlong`, so we need to

```
\rightadd@HL\to\overlonglist@
```

23.3. Different levels of \HL. Just as the `\list` construction used `\listlevel@` to keep track of the `\list` level, we will use `\HLlevel@` to keep track of the `\HL` level. But `\listlevel@` was a counter, whereas `\HLlevel@` will simply be a control sequence that will be defined to have the proper value.¹

Analogous to `\list@@C, ...`, we have

```
\def\HL@@C{\csname HL@C\HLlevel@\endcsname}
\def\HL@@P{\csname HL@P\HLlevel@\endcsname}
\def\HL@@Q{\csname HL@Q\HLlevel@\endcsname}
\def\HL@@S{\csname HL@S\HLlevel@\endcsname}
\def\HL@@N{\csname HL@N\HLlevel@\endcsname}
\def\HL@@F{\csname HL@F\HLlevel@\endcsname}
```

In addition, as with `\claim`, we will have `\HLtype@`, which will be defined to be `\chapter` when we are using `\chapter` instead of `\HL1`, etc. However, `\HL` itself will simply `\let\HLtype@=\relax`.

Corresponding to `\claim@@@C, etc.`, we define

```
\def\HL@@@C{\csname\exxx@\HLtype@ @C\endcsname}
\def\HL@@@P{\csname\exxx@\HLtype@ @P\endcsname}
\def\HL@@@Q{\csname\exxx@\HLtype@ @Q\endcsname}
\def\HL@@@S{\csname\exxx@\HLtype@ @S\endcsname}
\def\HL@@@N{\csname\exxx@\HLtype@ @N\endcsname}
```

all of which will only be used when `\HLtype@` is not `\relax`; `\HL@@@F` won't be needed at all.

23.4. The \HL construction. Roughly speaking, `\HL1` will translate to `'\HL@1'` and `\HL2` will translate to `'\HL@2'`, ..., while an error message will be given if the corresponding control sequence `'\HL@n'` does not exist. The default style defines `'\HL@1'` (section 10), and it is up to other style files to define any

¹This has the advantage that in various `\edef's, \csname ... \endcsname's, etc.`, we can use `\HLlevel@` alone instead of `\number\HLlevel@`. On the other hand, it was a little more efficient to have `\listlevel@` be a counter, since at one point we `\advance\listlevel@`.

further such constructions. We use quotation marks around \HL@*n* as before (see page 128).

Assuming that \HL#1 can be used, we will not call '\HL@#1' directly, but instead store #1 in \HLlevel@ and call \HL@, where \HL@ will then take care of calling '\HL@*n*' where *n* is the value \HLlevel@. We do this so that something like \chapter can simply define \HLlevel@ to be 1 and then call \HL@ directly.

Actually, \HL#1 will call \FNSS@\HL@, because we need to see if a quoted \HL number "... " follows, and we also have to skip over any space between the argument #1 of \HL#1 and the next token; moreover, we will define \HLname@ to be \HL{#1}, for use when writing to the .toc file (section 8), and we initialize \HLtype@ to be \relax:

```
\def\HL#1{\expandafter
\ifx\csname HL@C#1\endcsname\relax
\def\next@{\Err@{\string\HL#1 not defined in this style}}%
\else
\def\next@{\gdef\HLlevel@{#1}\def\HLname@\HL{#1}}%
\let\HLtype@=\relax\FNSS@\HL@}%
\fi
\next@}
```

We use a \gdef\HLlevel@ just in case \HL happens to be used within a group, because the value of \HLlevel might be of interest after the \HL has been printed (compare section 13).

The action of \HL@ will depend on whether a " follows next, so \HL@ will essentially be defined as

```
\def\HL@{%
\def\next@"##1"##2\endHL{...}
\def\nextii@##1\endHL{...}
\ifx\next"\expandafter\next@\else\expandafter\nextii@\fi}
```

Both \next@ and \nextii@ will be calling

```
\csname HL@\HLlevel@\endcsname##1\endHL
```

where things like

```
\expandafter\def\csname HL@1\endcsname#1\endHL{...}
\expandafter\def\csname HL@2\endcsname#1\endHL{...}
```

are the basic data that a style file will provide.

(1) `\next@` first stores `##2` in `\entry@`, for eventually writing to the `.toc` file.

```
\def\next@"##1"##2\endHL{\def\entry@{##2} . . .
```

Then `\next@` must create `\TheLabel@`, `...`, `\TheLabel@@@`. The preliminary step

```
\let\pre=... \let\post=...
```

must be divided into two cases, depending on whether we are using an ordinary `\HL<number>`, so that `\HLtype@` is `\relax`, or a substituted name, like `\chapter`, in which case `\HLtype@` will be defined to be `\chapter`:

```
\ifx\HLtype@\relax
\let\pre=\HL@@P \let\post=\HL@@Q
\let\style=\HL@@S \let\numstyle=\HL@@N
\else
\let\pre=\HL@@@P \let\post=\HL@@@Q
\let\style=\HL@@@S \let\numstyle=\HL@@@N
\fi
\Qlabel@{##1}
```

Note that this would be much harder to state if we hadn't introduced `\HL@@@P`, `...`, and compare page 140.

After defining `\TheLabel@`, `...`, `\TheLabel@@@`, we will want to store the value of `\TheLabel@@@` in `\Thepref@`. The reason for this is that after the appropriate

```
'\HL<number>' ... \endHL
```


construction, we may perform other steps using the value of \TheLabel@@@, and it is possible (though unlikely) that the \HL... \endHL construction contains some other construction allowing (label)'s, which would then create new values for \TheLabel@@@.¹

But this is one other detail that we need to worry about, in connection with writing information to the .toc file (section 8). Normally, we will be using \TheLabel@@@@ for the properly formatted heading number that we will write to the .toc file. Thus, we will be writing the number, together with any pre- and post- material, but without extra formatting determined by the style, so that we can use a different formatting style in the Contents if we desire. For example, in the text we might print \h11 numbers as '\$1', '\$2', ... (compare the small print section on page 209), but we want the option of including or omitting the § in the Contents.

If we do decide to print the § in the Contents, then we will have a bit of quandary when we are dealing with a "quoted" number, like

```
\HL1 "B" ... \endHL
```

In this case, the printed number will appear as 'B' rather than '\$B', so it should presumably also appear that way in the contents; on the other hand, "\style B" would appear as '\$B', so should continue to appear that way in the Contents.

What this means is that quoted numbers *should also appear quoted in the .toc file*, and that any occurrence of \style in a quoted number *should also appear in the .toc file*, although occurrences of \pre and \post should simply be expanded out to their proper values.

(This is admittedly a rather piddling point; compare the small print section on page 206.)

To handle this, we introduce a new flag

```
\newif\ifquoted@
```

¹\footnote is really the only plausible candidate for this, aside from user-constructed counters. (\tag might seem like a candidate, except that displayed formulas aren't allowed in headings; actually they can easily be simulated by setting \$\dsizex...\$ on a separate line, but it would be quite strange to expect such a formula to have a \tag over at the margin).

which we will set true right after defining `\entry@`, and after the `\Qlabel@{##1}` we will add

```
\let\style=\relax\xdef\Qlabel@{##1}
```

so that `\style` will remain unexpanded in `\Qlabel@`; on the other hand, `\nextii@` will set `\ifquoted@` to be false and won't define `\Qlabel@`.

After all this, we use

```
\csname HL\HLlevel@\endcsname##2\endHL
```

to actual typeset the heading.

[We use `##2` explicitly, rather than `\entry@`, because certain style files, like the book style, store the argument of `'\HL@1'... \endHL` for use in running heads. In such cases, we want to store the actual heading that was originally typed, not simply `\entry@`, since `\entry@` may have changed its meaning by the time the running head is typeset.]

Then we will use

```
\csname HL@I\HLlevel@\endcsname
```

to perform the proper “initializations”, determined by the style file. For example, the default style file will basically make `'\HL@I1'` mean

```
\Reset\h1{1}
\newpre\h1{\Thepref@.}
```

(recall [page 190] that `\Thepref@` holds the value of `TheLabel@`), so that in the first `\HL1`, the `\h1` numbers will be 1.1, 1.2, ...; in the second `\HL1`, they will be 2.1, 2.2, ...; etc. As we will see in Chapter 24, a `\newpre` will essentially do an `\edef`, so that the current value of `\Thepref@` will be stored in the pre-material for `\h1`.

And after that we will use

```
\csname HL@J\HLlevel@\endcsname
```

to perform “initializations” prescribed by the user, via `\Initialize` (section 11.4); as we will see in section 15, `\Initialize` simply defines things like `'\HL@J1'`.

Before performing these two routines we will

```
\let\pref=\Thepref@
```

so that any \pref that the user places in an \Initialize will be interpreted properly (this even allows the style file designer to use \pref in defining ‘\HL@I1’), and we will use \pref@ to restore the original definition of \pref:

```
\let\pref=\Thepref@
\csname HL@I\HLlevel@\endcsname
\csname HL@J\HLlevel@\endcsname}
\let\pref=\pref@
```

Then we will use

```
\HLtoc@
```

to write to the .toc file, if necessary. \HLtoc@ won’t be defined until section 8.

After writing to the .toc file, yet other steps may be required, which we call \aftertoc@. Initially we

```
\let\aftertoc@=\relax
```

but various heading level definitions may change this (compare section 12).

Finally, after that we simply want to reset values that may have been set before the \HL@:

```
\let\aftertoc@=\relax \overlong@false
```

(2) \nextii@ is exactly analogous, except that for defining \TheLabel@, ..., we will need

```
\ifx\HLtype@\relax
\global\advance\HL@@C by 1
\xdef\TheLabel@@@{\number\HL@@C}%
\xdef\TheLabel@{\HL@@N}%
```

```

\edef\TheLabel@@@{\HL@@P\TheLabel@\HL@@Q}%
\edef\TheLabel@@{\HL@@S}%
\else
\global\advance\HL@@@C by 1
\edef\TheLabel@@@{\number\HL@@@C}%
\edef\TheLabel@{\HL@@@N}%
\edef\TheLabel@@@{\HL@@@P\TheLabel@\HL@@@Q}%
\edef\TheLabel@@{\HL@@@S}%
\fi

```

(and, as already mentioned, we set `\ifquoted@` to be false, and do not define `\Qlabel@@@`).

The whole definition of `\HL@` is:

```

\let\aftertoc@=\relax

\def\HL@{%
\def\next@"##1"##2\endHL{\def\entry@{##2}\quoted@true
\noexpands@
\ifx\HLtype@\relax
\let\pre=\HL@@P \let\post=\HL@@Q
\let\style=\HL@@S \let\numstyle=\HL@@N
\else
\let\pre=\HL@@@P \let\post=\HL@@@Q
\let\style=\HL@@@S \let\numstyle=\HL@@@N
\fi
\Qlabel@{##1}\let\style=\relax\edef\Qlabel@@@{##1}%
\edef\Thepref@{\TheLabel@@@}%
\csname HL@\HLlevel@\endcsname##2\endHL
\let\pref=\Thepref@
\csname HL@I\HLlevel@\endcsname
\csname HL@J\HLlevel@\endcsname
\let\pref=\pref@
\HLtoc@
\aftertoc@
\let\aftertoc@=\relax \overlong@false}%

```

```

\def\nextii@##1\endHL{\def\entry@{##1}\quoted@false
  {\noexpands@
  \ifx\HLtype@\relax
    \global\advance\HL@@C by 1
    \xdef\Thelabel@@@{\number\HL@@C}%
    \xdef\Thelabel@{\HL@@N}%
    \xdef\Thelabel@@@@{\HL@@P\Thelabel@\HL@@Q}%
    \xdef\Thelabel@@{\HL@@S}%
  \else
    \global\advance\HL@@@C by 1
    \xdef\Thelabel@@@{\number\HL@@@C}%
    \xdef\Thelabel@{\HL@@@N}%
    \xdef\Thelabel@@@@{\HL@@@P\Thelabel@\HL@@@Q}%
    \xdef\Thelabel@@{\HL@@@S}%
  \fi
  \xdef\Thepref@{\Thelabel@@@}%
  \csname HL@\HLlevel@\endcsname##1\endHL
  \let\pref=\Thepref@
  \csname HL@I\HLlevel@\endcsname
  \csname HL@J\HLlevel@\endcsname
  \let\pref=\pref@
  \HLtoc@
  \aftertoc@
  \let\aftertoc@=\relax \overlong@false}%
\ifx\next"\expandafter\next@\else\expandafter\nextii@\fi}

```

In all these definitions, \endHL simply functions as syntax, so we want to declare (see section 1.1)

```
\Invalid@\endHL
```

As of yet, no heading level \HL has actually been defined. Before considering this, however, we first tend to \hl.

23.5. The \hl construction. Everything for \hl is almost exactly analogous to that for \HL:

We start with

```

\def\hl@@C{\csname hl@C\hllevel@\endcsname}
\def\hl@@P{\csname hl@P\hllevel@\endcsname}
\def\hl@@Q{\csname hl@Q\hllevel@\endcsname}
\def\hl@@S{\csname hl@S\hllevel@\endcsname}
\def\hl@@N{\csname hl@N\hllevel@\endcsname}
\def\hl@@F{\csname hl@F\hllevel@\endcsname}

```

Then we define

```

\def\hl@@@C{\csname\exxx@\hltype@ @C\endcsname}
\def\hl@@@P{\csname\exxx@\hltype@ @P\endcsname}
\def\hl@@@Q{\csname\exxx@\hltype@ @Q\endcsname}
\def\hl@@@S{\csname\exxx@\hltype@ @S\endcsname}
\def\hl@@@N{\csname\exxx@\hltype@ @N\endcsname}

```

Similarly, we define

```

\def\hl#1{\expandafter
\ifx\csname hl@C#1\endcsname\relax
\def\next@{\Err@{\string\hl#1 not defined in this style}}%
\else
\def\next@{\gdef\hllevel@{#1}\def\hlname@{\hl{#1}}%
\let\hltype@=\relax\FNSS@\hl@}%
\fi
\next@}

```

Then we define `\hl@` as follows (see the remarks at the end regarding the `\FNSSP@s`):

```

\def\hl@{%
\def\next@"##1"##2{\def\entry@{##2}\quoted@true
{\noexpands@
\ifx\hltype@\relax
\let\pre=\hl@@P \let\post=\hl@@Q
\let\style=\hl@@S \let\numstyle=\hl@@N

```

```

\else
  \let\pre=\hl@@P \let\post=\hl@@Q
  \let\style=\hl@@S \let\numstyle=\hl@@N
\fi
\Qlabel@{##1}\let\style=\relax\xdef\Qlabel@@@{##1}%
\xdef\Thepref@{\Thelabel@@@}%
\csname hl@hllevel@\endcsname{##2}%
  \let\pref=\Thepref@
  \csname hl@I\hllevel@\endcsname
  \csname hl@J\hllevel@\endcsname
  \let\pref=\pref@
\hltoc@
\aftertoc@
\let\aftertoc@=\relax
\nopunct@false \nospace@false \FNSSP@}%
\def\nextii@##1{\def\entry@{##1}\quoted@false
f\noexpands@
\ifx\hltype@\relax
  \global\advance\hl@@C by 1
  \xdef\Thelabel@@{\number\hl@@C}%
  \xdef\Thelabel@@@{\hl@@P\Thelabel@\hl@@Q}%
  \xdef\Thelabel@@{\hl@@S}%
\else
  \global\advance\hl@@@C by 1
  \xdef\Thelabel@@{\number\hl@@@C}%
  \xdef\Thelabel@{\hl@@@N}%
  \xdef\Thelabel@@@{\hl@@@P\Thelabel@\hl@@@Q}%
  \xdef\Thelabel@@{\hl@@@S}%
\fi
\xdef\Thepref@{\Thelabel@@@}%
\csname hl@hllevel@\endcsname{##1}%
  \let\pref=\Thepref@
  \csname hl@I\hllevel@\endcsname
  \csname hl@J\hllevel@\endcsname
  \let\pref=\pref@

```

```

\hltoc@
\aftertoc@
\let\aftertoc@=\relax
\nopunct@false \nospace@false \FNSSP@}%
\ifx\next"\expandafter\next@\else\expandafter\nextii@\fi}

```

The `\FNSSP@`'s are added to deal with spaces and invisible constructions that might occur after the whole `\hl{number}{...}` combination. (It wouldn't do any good to add the `\FNSSP@`'s to the definitions of `'\hl@1'`, ..., since other material comes after them in the definition of `\hl@`.)

23.6. Other elements of heading levels. In the default style, `\newstyle` can be used to change the style for printing `\HL` and `\hl` numbers, and `\nopunct` and `\nospace` can be used to control the punctuation and spacing that follows an `\hl` heading level (compare page 208).

On the other hand, most styles will have one further element of a heading level that we might want to control, namely, a word like 'Chapter' that is printed before the `\HL` number.

Someone writing in German would naturally want to replace 'Chapter' with 'Kapitel', etc. Although such a replacement could easily be made directly within the style file, there is considerable objection to this, on the grounds that additional style files shouldn't have to be made for such minor changes.

So, to the pantheon of constructions like `\...@C`, `\...@P`, ..., we admit one more candidate, `\...@W`, the "word" associated with a construction. This will be complemented by a construction `\newword`, analogous to `\newpre`, ..., (Chapter 24), allowing us to change its values, as well as `\word`, to print its value. From the point of view of the user, `\word` and `\newword` will work just like `\pre` and `\newpre`, etc., but they will not be applicable to all constructions allowing labels, only to a select few (and also one or two constructions that do not allow labels—see section 29.4).

The idea of this device is that a style file can define `'\HL@1'` in terms of `'\HL@W1'`, which might initially be defined to mean Chapter. Then if the user types

```
\newword\HL1{Kapitel}
```

thereby redefining `'\HL@W1'`, the word 'Kapitel' will be substituted for 'Chapter' in the `\HL1` headings.

This all fails spectacularly, however, if the \langle material to be split \rangle is empty, or happens to begin with a space! So we will have to be careful about that.

Since we are going to need a similar routine to `\write` to the `.tic` file in Chapter 32, the `'\toc@'` in the above definition will be replaced by an argument `#1`, so that we define

```
\def\six@#1#2 #3 #4 #5 #6 #7 {\def\next@{#2}%
\ifx\next@\empty
\def\next@##1\six@{}%
\else
\write#1{ #2 #3 #4 #5 #6 #7}\def\next@{\six@#1}%
\fi
\next@}
```

We are going to be applying this when the \langle material to be split \rangle is stored in `\macdef@`. If `\macdef@` happens to be empty (because of an empty heading level), we will do nothing. Otherwise, we will first eliminate any initial space in `\macdef@` with

```
\def\next@#1#2\next@{\def\macdef@{#1#2}}
\expandafter\next@\macdef@\next@
```

If `\macdef@` originally expands to $_s_1s_2 \dots$, then `#1` will be s_1 , since TeX ignores spaces in looking for undelimited arguments, and `#2` will be $s_2 \dots$, so the new `\macdef@` will be the old, with any initial space deleted.

Then we will use

```
\edef\next@
{\noexpand\six@\toc@\macdef@
\space\space\space\space\space\space
\space\space\space\space\space\space\noexpand\six@}
\next@
```

The `\edef` not only inserts 12 spaces, but it also expands out `\macdef@` to its current value, so that the resulting series of (delayed) `\write`'s doesn't depend on what value `\macdef@` may have by the time the `\write`'s are done.

Finally, we will use

```
\let\macdef@=\relax
```

to clear up memory space, since \macdef@ might be quite long.

The whole definition is:

```
\def\Sixtoc@{\ifx\macdef@\empty\else
\def\next@##1##2\next@{\def\macdef@{##1##2}}%
\expandafter\next@\macdef@\next@
\edef\next@
{\noexpand\six@\toc@\macdef@
\space\space\space\space\space\space
\space\space\space\space\space\space
\noexpand\six@}%
\next@\let\macdef@=\relax\fi}
```

bd Even if some argument for \six@ is of the form {...}, TEX *won't* remove the braces, since these are now type 12 characters, not real braces. (In version 1 of L_AT_EX, where this wasn't the case, a much more complicated routine was required.) Similarly, we don't have to worry about braces being removed during the first stage of deleting any initial space.

bd Some remarks near the end of page 87 of the L_AT_EX Manual are now wrong. Now spaces after control words *are* counted in this break-up process (and will be inserted even if they weren't typed in the input file). Spaces within curly braces will also count (since these type 12 braces won't introduce any grouping for the arguments). This has the added advantage that one doesn't have to worry about a long group within a heading.

bd If the prospect of short lines in the .tic file is too unappealing, a more complicated routine could be used, in which we selected the pieces delimited by spaces one at a time, and stored them until we obtained a sufficiently large aggregate; the "size" of a collection #1 of type 12 tokens is easily computed by examining the width of \hbox{\tt#1}.

23.8. \HLtoc@ and \hltoc@. For \HLtoc@, we want to write either something

like

```
\HL {<heading level>}{<word>}{<formatted \HL number>}
  The heading, with line breaks after
  every sixth space.
\Page{<page number>}{\arabic }{< >}
```

or something like

```
\chapter {<word>}{<formatted \chapter number>} . . .
  The heading, with line breaks after
  every sixth space.
\Page{}{\arabic }{< >}
```

if we have created `\chapter` as a name for `\HL1`, say, using `\NameHL` (section 14). [If the heading is empty, then we will simply have the `\Page...` line follow the `\HL` line.]

Here `<word>` is the value of `'\HL@Wn'` (section 6). In the default style `'\HL@W1'` is empty, while for the book style `book.st` `\HL1` (*alias* `\chapter`) has `\HL@W1` initially defined as `Chapter`. Notice that the `<word>` is treated as a separate argument from the `<formatted number>`. So, even if the heading level itself prints something like

Chapter 3. . . .

the table of contents will be able to combine these elements in different ways, if desired.

This whole arrangement is a big change from version 1 of \LaTeX ; in addition to the `<word>`, the page number is included, and all the other information appears without extraneous matter that made the `.toc` file harder to read.

To handle the special case of quoted numbers, we define

```
\def\QorTheLabel@@@@{\ifquoted@
  \noexpand\noexpand\noexpand"\QLabel@@@@
  \noexpand\noexpand\noexpand"\else
  \TheLabel@@@@\fi}
```

so that \QorTheLabel@@@ will expand in an \edef to either

```
\noexpand"<expansion of \Qlabel@@@>\noexpand"
```

or to

```
<expansion of \TheLabel@@@>
```

Our first \write will involve \HLname@, which is either \HL {<heading level>} or something like \chapter; the <word> for the heading level; and \QorTheLabel@@@, the actual <formatted number> (or \Qlabel@@@). However, this will be a delayed \write, and the values of \HLname@ and \TheLabel@@@ or \Qlabel@@@ may be completely different by the time the \write is executed, so everything has to be expanded out.

For \QorTheLabel@@@ we need to be in a group with \noexpands@; moreover, within this group we need to \let\style=\relax, in case \style appears in \Qlabel@@@.

For \HLname@ we need something like

```
\edef\next@{\write\toc@{\noexpand\noexpand
\expandafter\noexpand\HLname@}}
\next@
```

In this \edef, the \toc@ is not expanded, since it was created with a \chardef (compare page 179), the \noexpand\noexpand collapses to a single \noexpand, and the \expandafter\noexpand\HLname@ (compare pages 126 and 161) inhibits expansion of a control sequence occurring at the beginning of the expansion of \HLname@ (i.e., either \HL or \chapter, etc.) Thus, our \edef makes \next@ mean

```
\write\toc@{\noexpand(\HL1 or \chapter, etc.)}
```

so that \next@ then gives us what we want.

The <word>, stored in '\HL@W1', etc., presents more problems, because it may contain control sequences that shouldn't be expanded out in the \write.

So we will use the `\unmacro\meaning` trick of section 7, within the triple `\expandafter` trick (page 162): the code

```
\expandafter\expandafter\expandafter\unmacro@
\expandafter\meaning\csname HL@W\HLlevel@\endcsname\unmacro@
```

will make `\macdef` contain the tokens of the `<word>`, but with all non-space tokens converted to type 12.

So, altogether, we can use

```
\expandafter\expandafter\expandafter\unmacro@
\expandafter\meaning\csname HL@W\HLlevel@\endcsname\unmacro@
{\noexpands@\let\style=\relax
\edef\next@{\write\toc@{\noexpand\noexpand\expandafter\noexpand
\HLname@{\macdef@}{\QorTheLabel@}}}}
\next@}
```

Note that, just as we will need to declare `\noexpands@` before a `\shipout`, when the `\write` is actually done (pages 58 and 87), we will also need to declare `\let\style=\relax` before the `\shipout`.

Then we use

```
\expandafter\unmacro@\meaning\entry@\unmacro@
\Sixtoc@
```

to write the heading, with all non-space characters of type 12, and line breaks after every sixth space.

Finally, we add

```
\write\toc@{\noexpand\Page{\number\pageno}{\page@N}}%
{\page@P}{\page@Q}^^J}
```

where the extra blank line at the end is added to make the `.toc` file more readable.¹

¹ Blank lines after the `\Page...` line will usually create no problem, since the table of contents will be usually be set as a series of lines each contributed in vertical mode. But it's conceivable that a style file will format things in a way that puts us into horizontal mode, and then it might

(During the `\write` we will have `\noexpands@` in effect, so that the numbering control sequence `\page@N` won't be expanded, nor will any font change control sequences in `\page@P` or `\page@Q`.)

So our final definition is:

```

\def\HLtoc@{%
  \iftoc@
  \expandafter\expandafter\expandafter\unmacro@
  \expandafter\meaning
  \csname HL@W\HLlevel@\endcsname\unmacro@
  {\noexpands@\let\style=\relax
  \edef\next@{\write\toc@{%
    \noexpand\noexpand\expandafter\noexpand\HLname@
    {\macdef@}{\QorTheLabel@@@@}}}%
  \next@}%
  \expandafter\unmacro@\meaning\entry@\unmacro@
  \Sixtoc@
  \write\toc@{\noexpand\Page{\number\pageno}{\page@N}%
    {\page@P}{\page@Q}^^J}%
  \fi}

```

For `\hltoc@` we want to write something like

```

\hl {<heading level>}{<word>}{<formatted \hl number>}
  The heading, with line breaks after
  every sixth space
  \Page{<page number>}{\arabic }{-}{-}

```

be important that we not leave horizontal mode after the page number is printed; in such cases, the macros must scan for `\Page#1#2#3#4\par`, to eliminate the blank line. Although this is an added pain, it is unlikely to occur often, so it seems worth while adding the extra readability to the `.toc` file.

In the case of the index, where an unknown number of `\Page`'s can follow each `\Entry`, we will be careful to eliminate such `\pars`'s (section 38.1).

or something like

```
\section {<word>}{<formatted \section number>} .
  The heading, with line breaks after every
  sixth space
\Page{<page number>}{\arabic }{-{}}
```

In addition, we want `\nopunct` and `\nospace` to appear before the `\h1` or `\section` in the `\write` if they appeared in the input file:

```
\def\hltoc@{%
\iftoc@
\expandafter\expandafter\expandafter\unmacro@
\expandafter\meaning
\csname hl@W\hllevel@\endcsname\unmacro@
{\noexpands@\let\style=\relax
\edef\next@{\write\toc@{%
\ifnopunct@\noexpand\noexpand\noexpand\nopunct\fi
\ifnospace@\noexpand\noexpand\noexpand\nospace\fi
\noexpand\noexpand\expandafter\noexpand\hlname@
{\macdef@}{\QorTheLabel@@@@}}}%
\next@}%
\expandafter\unmacro@\meaning\entry@\unmacro@
\Sixtoc@
\write\toc@{\noexpand\Page{\number\pageno}{\page@N}%
{\page@P}{\page@Q}^^J}%
\fi}
```

(We didn't add the analogous clause

```
\ifoverlong@\noexpand\noexpand\noexpand\overlong\fi
```

to the definition of `\HLtoc@`, because `\overlong` will usually be irrelevant to the treatment of the heading level in the Contents; but for some styles this addition might be reasonable.)

wd It's conceivable that a style file might need different information in the `.toc` file that we have provided. For example, perhaps the sections 3.1, 3.2, 3.3, ... of

Chapter 3 are simply going to be numbered 1, 2, 3, ... under the Chapter 3 entry in the Contents. To handle such possibilities, the .toc file should probably get the whole set of information

```
{\TheLabel@}{\TheLabel@@}{\TheLabel@@@}{\TheLabel@@@@}
```

instead of simply {\TheLabel@@@@}. Then front matter style files (Chapter 37) could decide which information to use. Since I've never actually seen a book in which formatting of this sort was used, it seemed silly to encumber L^AT_EX with this extra baggage. But if such a treatment were needed, it should be clear how to modify the definitions of \HLtoc@ and \hltoc@ accordingly.

23.9. \mainfile. When we have specified \tocfile in our main file, say paper.tex, the file paper.toc is written, but this file is not meant to be T_EX'ed directly. Instead, we will be making a "front matter" file, say paperfm.tex, and since this file will need to know the name of the original file, in order to \input the proper .toc file, paperfm.tex will begin with a line like

```
\mainfile{paper}
```

We define

```
\def\mainfile#1{\def\mainfile@{#1}}
```

so that, at the appropriate time the macros for the front matter style can

```
\input mainfile@.toc
```

If the \mainfile command is omitted from paperfm.tex, we would like to produce an error message when \maketoc is encountered. So we add

```
\def\checkmainfile@{\ifx\mainfile@\undefined
  \Err@{No \noexpand\mainfile specified}\fi}
```

(We put these definitions directly into lamstex.tex so that front matter style files don't have to worry about them.) (See section 3.4 for the use of \noexpand.)

23.10. Creating heading levels. Although we now have the general mechanism for handling heading levels, no heading levels have actually been defined.

The default style makes `\HL1` and `\hl1` be defined, which will indicate the general procedure required.

We begin by adding the necessary counters and control sequences.

```

\expandafter\newcount@\csname HL@C1\endcsname
\csname HL@C1\endcsname=0
\expandafter\def\csname HL@S1\endcsname#1{#1\null.}
\expandafter\let\csname HL@N1\endcsname=\arabic
\expandafter\let\csname HL@P1\endcsname=\empty
\expandafter\let\csname HL@Q1\endcsname=\empty
\expandafter\def\csname HL@F1\endcsname{\bf}
\expandafter\let\csname HL@W1\endcsname=\empty


\expandafter\newcount@\csname hl@C1\endcsname
\csname hl@C1\endcsname=0
\expandafter\def\csname hl@S1\endcsname#1{#1\/}
\expandafter\let\csname hl@N1\endcsname=\arabic
\expandafter\let\csname hl@P1\endcsname=\empty
\expandafter\let\csname hl@Q1\endcsname=\empty
\expandafter\def\csname hl@F1\endcsname{\bf}
\expandafter\let\csname hl@W1\endcsname=\empty


```

Using `\newcount@` (see page 121) rather than `\newcount` isn't necessary here, since these definitions are made within the file `lamstex.tex`, but they serve as a reminder that auxiliary files should use `\newcount@` (unless they have stated `\let\alloc@=\alloc@@` at the beginning). We added the `\null` in '`\HL@S1`' in case #1 ends with an upper-case letter (compare page 162). (The proficient *AMS-TEX* or *L^AS-TEX* user would type '@.' in such a case, but @ can't be allowed in a style command, since that would essentially be as bad as allowing it in a (label).)

Note, by the way, that the period after the heading number in '`\HL@S1`' is not considered to be punctuation that can be eliminated with `\nopunct`. There is no need for that, since a quoted heading level could always be used instead (and `\newstyle\HL1` or `\newstyle\hl1` could be used to make a permanent change). Basically, `\nopunct` only affects punctuation that the

user normally wouldn't be able to change at all, like the period after an `\h1` title in the default style (see page 211).

 In version 1 of *L^AT_EX*, I didn't have this principle clearly formulated, and the period after the heading number in `\HL1` was considered a "frill". Moreover, the default style used to print a § before the heading number, and this "\S" was "hard-wired" into the definition of `\HL@1` (except that we `\let\Ssymbol@=\S` and used `\Ssymbol@`, just in case some one decided to redefine `\S`). If we wanted to retain this §, then it should really be placed within the definition of '`\HL@S1`'.

 In that case, however, it would also be necessary to add `\Nonexpanding\S` (or `\Nonexpanding\Ssymbol@`). In view of all this, it seemed better just to leave the damn thing out.

To allow `\HL1` to be defined, we must create '`\HL@1`' with

```
\expandafter\def\csname HL@1\endcsname#1\endHL{...}
```

Although that's quite a mouthful, such definitions are meant to be supplied only by style file designers, and numerous subtleties are involved in the definitions anyway, so there's no point trying to make this part easier.

Roughly speaking, *L^AT_EX*'s definition of the default style '`\HL@1`' is

```
\expandafter\def\csname HL@1\endcsname#1\endHL{\bigbreak
\locallabel@
\vbox{\Let@\tabskip\hss@
\halign to\hsize{\bf\hfil##\hfil\cr
\csname HL@W1\endcsname\space
{\HL@@F\thelabel@@}\_#1\crr}}
\nobreak\medskip}
```

where

- (1) We put the whole construction in a group with `\locallabel@`, so that a `\label` within the

```
\HL1 ... \endHL
```

will be given the values relevant to this `\HL1`.

- (2) We set a `\vbox` consisting of centered lines. `\let@`, from *AMS-TEX*, is the device for letting `\\` be the same as `\cr` within this `\vbox`. Normally we would use `\tabskip\hfil` or `\tabskip\hss` to produce the centered lines; `\tabskip\hss@` is used in case `\overlong` precedes the `\HL` (page 160).
- (3) At the beginning of the argument #1 we print `\thelabel@@` (the number, together with any pre- and post-material, plus anything produced by the style), in the proper font. Notice that we explicitly leave the space after `\thelabel@@`, since the spacing is not made part of '`\HL@S1`' (compare page 132). Moreover, before `\thelabel@@` we print the `\langle word \rangle` for '`\HL@1`' (empty in the default style).

However, there are a few details that we need to change:

- (1) The preamble should really contain

```
\halign to\hsize{\bf\hfil\ignorespaces##\unskip\hfil\cr
```

to discard extraneous spaces at the beginning or end of each line of the heading.

- (2) In addition, we need to replace the

```
{\HL@@F\thelabel@@} #1
```

by

```
{\HL@@F\thelabel@@} \ignorespaces#1
```

to get rid of an extraneous space at the very beginning of the heading.

- (3) Moreover, if `\thelabel@@` is empty (from a `\HL 1 ""`), then the space before the `\ignorespaces` should be eliminated. And if the `\langle word \rangle` is empty, then the space after it should be eliminated.
- (4) We don't really want our heading to be a `\vbox`, because `\insert`'s (like `\footnote`) won't migrate out. So we will instead globally `\setbox1` to be the `\vbox`, and then `\unvbox1`:

```
\expandafter\def\cename HL@1\endcename#1\endHL{\bigbreak
{\locallabel@
```

```

\global\setbox1=\vbox{\Let@\tabskip\hss@
\halign to\hsize{\bf\hfil\ignorespaces##\unskip\hfil\cr
\expandafter\ifx\csname HL@W1\endcsname\empty\else
\csname HL@W1\endcsname\space\fi
{\HL@@F\ifx\thelabel@\empty\else\thelabel@\space\fi}%
\ignorespaces#1\crr}%
}%
\unvbox1 \nobreak\medskip}

```

NOTE: The `\nobreak` here is absolutely essential—see section 12.

The definition of `\hl1` for the default style is quite straightforward, using `\punct@` and `\addspace@`, since `\hl` is on `\nofrillslist@`:

```

\expandafter\def\csname hl@1\endcsname#1{\medbreak\noindent@
{\locallabel@
\bf{\hl@@F\ifx\thelabel@\empty\else\thelabel@\space\fi}%
\ignorespaces#1\unskip
\punct@{\null.}\addspace@\enspace}}

```

See Chapter 8 for the use of `\noindent@`. Notice that here we don't even print the `<word>`, even if it has been changed by `\newword`. (Generally speaking, `\HL` heading levels will have words printed as part of them, while `\hl` levels won't. Of course a style file could redefine '`\hl@1`' to include the `<word>`, if necessary.)

23.11. Initializations. In addition to defining '`\HL@1`', we want a definition like

```

\expandafter\def\csname HL@I1\endcsname{\Reset\hl1{1}%
\newpre\hl1{\pref.}}

```

so that these "initializations" are always performed after each `\HL1` (though they may be overridden if `\csname HL@J1\endcsname` has been defined by an `\Initialize` [section 15]). Recall that `\pref` will mean `\Thepref@` during this initialization.

But there is a little subtlety involved here, since `\HL1` might be used with an empty label `\HL1""`. For example,

```

\HL1""Introduction\endHL

```

might be used to get an unnumbered Introduction preceding the other \HL1's. In that case \pref would be empty, and we don't want a lonely period preceding the \h11 numbers! So instead we use

```
\expandafter\def\csname HL@I1\endcsname{\Reset\h11{1}%
\ifx\pref\empty\newpre\h11{}}\else\newpre\h11{\pref.}\fi
```

No initializations are required at each \h11 in the default style, so we simply leave 'HL@I1' undefined.

bd In the situation on page 72, where we used

```
\Initialize\h11{%
\Evaluate\HL1\edef\HLvalue{\number\Value}
\Evaluate\h11
\newpre\exno{\HLvalue.\the\Value.}}
```

to set the numbering for \exno, subtleties about empty heading level numbers were not taken into account. Actually, there's no way that they can be handled at this point, since there's no way of determining, once we get to an \h11, whether or not the \HL1 before it had an empty heading number. To deal with these possibilities, we could, for example, have \HL1 store the current value of \TheLabel@@@ in another control sequence, say \HLpref, in addition to storing it in \Thepref@; then \HLpref wouldn't change at an \h11, so it would give information about the \HL1 that comes before the current \h11. If we were designing a style file in which something like \exno were a standard feature, then we would probably need to do this.

A clever user could always emulate this with

```
\Initialize\HL1{\Evaluate\HL1\xdef\HLpref{\number\Value}}
\Initialize\h11{ . . .
\ifx\HLpref\empty\else
\Evaluate\HL1\edef\HLvalue{\number\Value}\fi
. . . }
```

(Or a new \Initialize\HL1 could simply be given before any \HL1"" is stated.)

23.12. \aftertoc@. Remember that \HL1 calls \HL@, and \HL@ in turn will then call '\HL@1'... \endHL followed by

```
\HLtoc@
```

for writing to the .toc file.

IT IS THEREFORE ESSENTIAL that no breaks be allowed by any material at the end of the definition of ‘\HL@1’... \endHL, to insure that the proper page number will be written; thus, in the definition for the default style (page 210), the \nobreak before the \medskip would be necessary even if the style file design weren’t particularly concerned about prohibiting a page break here.

It is quite possible that a style file won’t try to prohibit a page break after the heading is printed. In fact, a style file might even *demand* a page break at this point. For example, the style file for the L^AT_EX Manual uses \HLO for the Part pages, and in the definition of ‘\HL@0’, we want a page break after typesetting the Part page; and then we want to \Offset\page2, since the next page is supposed to be blank. It would certainly not do to

```
\expandafter\def\csname HL@0\endcsname#1\endHL{%
  <instructions for typesetting the Part page>
  \vfil\break\Offset\page2}
```

because in the .toc file the page number for this Part page would then be 1 more than it should be! Instead, the definition has the additional clause

```
\def\aftertoc@{\vfil\break\Offset\page2}
```

—then the proper page is written by the \HLtoc@ command before we increase the page numbering by 1.

23.13. Order of heading levels. The default style allows an \h11 to appear before an \HL1, but we could easily disallow this. For example, we could first

```
\def\HLlevel@{0}
```

and then use

```
\expandafter\def\csname hl@1\endcsname{%
  \ifnum\HLlevel@=0
  \Err@{\string\h11 not allowed before some \string\HL}
```

Similarly, if we created \h12, we could

```
\def\h1level@{0}
```

and then

```
\expandafter\def\csname hl@2\endcsname#1{%
  \ifnum\hllevel@=0
    \Err@{\string\hl2 not allowed before some \string\hl1}
  . . . }
```

For this to work properly, we should also add

```
\def\hllevel@{0}
```

to our definition of ‘\HL@I1’.

Note, by the way, that there’s really no necessity for “higher” heading levels to have smaller numbers.¹

23.14. Naming header levels. Now we come to the \NameHL and \Namehl mechanisms, by which, for example,

```
\NameHL1\chapter
```

makes \chapter... \endchapter function like \HL1... \endHL, and

```
\Namehl1\section
```

makes \section{...} function like \hl1{...}. As the uppercase ‘N’ suggests, \NameHL and \Namehl will act globally.¹

As a special case of

```
\NameHL#1#2
```

let us consider \NameHL1 \chapter.

First we will want to

```
\define\chapter{}
```

¹ Actually, a heading level doesn’t even have to be a number (but it’s probably best not to exploit that fact).

¹ \NameHL and \Namehl replace the \newHL and \newhl constructions from version 1 of L^AT_EX.

to get an error message if `\chapter` is already defined.

We will also want to

```
\rightadd@{\chapter\to\overlonglist@
```

And we will want to let `\chapter@C` be `'\HL@C1'`, and `\chapter@P` be `'\HL@P1'`, etc. For this we will need the old `\edef` trick (see, for example, page 82 and Chapter 17, and see pages 126 and 161 for the use of the `'\expandafter\noexpand'`):

```
\edef\next@{\let\csname\exstring@#2@C\endcsname
=\expandafter\noexpand\csname HL@C#1\endcsname}\next@
```

NOTE: Since we `\let\chapter@C=''\HL@C1'`, etc., it is therefore essential that `\NameHLn` be used *only after* `'\HL@C1'`, ..., `'\HL@F1'` have been defined.

Then we want to

```
\def\chapter##1\endchapter
{\def\HLtype@{\chapter}
\def\HLname@{\chapter}
\gdef\HLlevel@{#1}
\FNSS@\HL@##1\endHL}
```

This definition will have to be done with an `\edef` also:

```
\edef\next@{\def\noexpand#2####1\expandafter\noexpand
\csname end\exstring@#2\endcsname
{\def\noexpand\HLtype@{\noexpand#2}
\def\noexpand\HLname@{\noexpand#2}
\gdef\noexpand\HLlevel@{#1}
\noexpand\FNSS@\noexpand\HL@####1\noexpand\endHL}}
\next@
```

[We used `\FNSS@` rather than `\futurelet\next` simply because it takes fewer tokens, especially with the `\noexpand` required before it.]

We also want to state ‘\Invalid@\endchapter’ so that an improper use of \endchapter will give an error message (see section 1.1),

```
\edef\next@{\noexpand\Invalid@\expandafter\noexpand
\csname end\exstring@#2\endcsname}
\next@
```

That is the main outline of the definition, but additional details are needed. With this much of the definition, \chapter would still not function just like \HL1, because, for example, \newpre\chapter and \newpre\HL1 would have entirely different effects: one would change \chapter@P, while the other would change ‘\HL@P1’—if a user typed \newpre\HL1 this would not affect the pre-material for \chapter. That might not seem like such a terrible problem—users could just be warned not to use \newpre\HL1 instead of \chapter—until we realize that our definition of ‘\HL@I1’ has a \newpre\h11 clause: if the user of the default style decided to

```
\NameHL 1 \chapter
\Namehl 1 \section
```

then \section’s would not have the proper pre-material determined by the \chapter in which they were used. (Similarly, \newfontstyle\chapter and \newfontstyle\section would have no effect.)

To get around this problem, we need a way of passing special information to the \new... constructions of Chapter 24, so that, for example, \newpre\chapter and \newpre\HL1 will each change *both* \chapter@P and ‘\HL@P1’. This will be done by means of two Replacement control sequences:

```
‘\HL@R1’      will be defined to have the value \chapter
\chapter@R    will be defined to have as value the pair {HL}{1}
```

So we will add

```
\expandafter\gdef\csname HL@R#1\endcsname{#2}
\expandafter\gdef\csname\exstring@#2@R\endcsname{{HL}{#1}}
```

In some cases, some of this information must also be passed to the .toc file. For example, the default front matter style file lamstex.stf contains instructions for typesetting entries that begin

```
\HL {1}
```

but if \NameHL1\chapter appears in the main file, then \chapter will produce entries in the .toc file beginning

```
\chapter
```

In order for the lamstex.stf file to be able to deal with these lines, the information

```
\NameHL1\chapter
```

must be passed to it:

```
\iftoc@
\immediate\write\toc@{\noexpand\NameHL#1\noexpand#2^^J}
\fi
```

with a ^^J to give a blank line afterwards, to separate this from an entry line that might come next.

For all this to work properly when we get to front matter style files (Chapter 37), it will be essential, of course, that the user has placed the \tocfile command *before* any \NameHL commands. Actually, in addition to its invocation by a user who wants names for heading levels that a style hasn't given names to, \NameHL can also be used by a style file designer. In the latter case, we really don't want any extra information written to the .toc file, since the style file designer will presumably also write special code for the front matter style files to deal with lines that begin with something like

```
\chapter
```

As we will see in Part VII, special precautions will be taken to insure that nothing is written to the .toc file even if a file contains \tocfile before the \docstyle command.

A style file might well want to `\NameHL1` twice. For example, in the book style file, the control sequence

```
\appendices
```

calls `\NameHL1\appendix` to make

```
\appendix ... \endappendix
```

work just like `\chapter... \endchapter`.¹

Once `\appendices` has used `\NameHL1\appendix`, we want a previous name for `\HL1`, say `\chapter`, to become undefined, and we also want `\endchapter` to become undefined, and we want `\Offset\chapter`, etc., to be disallowed. To test whether `\NameHL1` has already appeared, we simply use the test

```
\expandafter\ifx\csname HL@R1\endcsname\relax
```

If this test is false, we want to `\let`

```
\...@C, \...@P, \...@Q, \...@S, \...@N, \...@F,, \...@W,
```

all be `\relax`, where `...` is the value of the control sequence `\csname HL@R#1\endcsname`, since, as we'll see in Chapter 24, this will disallow the corresponding `\Reset,...`, constructions.

Unfortunately, `\csname... \endcsname` is quite unsuitable for use in `\expandafter` constructions, which is just how we will need it, so we will first use the code

```
\def\nextiv@{\let\nextiii@=}
\expandafter\nextiv@\csname HL@R#1\endcsname
```

¹In this style '`\HL@1`' uses '`\HL@W1`' (section 6), which is initially defined to be 'Chapter', but which is then changed to 'Appendix' by the `\appendices` command. Similarly, `\appendices` also uses `\Reset\HL1{1}`, to start the numbering of Appendices at 1.

which makes the “nameable” control sequence `\nextiii@` have the same value as `'\HL@R#1'`. Then we can use

```
\expandafter\let\nextiii@\undefined
\expandafter\let\csname\exxx@\nextiii@ @C\endcsname=\relax
. . .
\expandafter\let\csname end\exxx@\nextiii@\endcsname=\undefined
```

So our whole definition is:

```
\def\NameHL#1#2{\define#2}%
\expandafter\ifx\csname HL@R#1\endcsname\relax
\else
\def\nextiv@{\let\nextiii@}%
\expandafter\nextiv@\csname HL@R#1\endcsname
\expandafter\let\nextiii@\undefined
\expandafter\let\csname\exxx@\nextiii@ @C\endcsname=\relax
\expandafter\let\csname\exxx@\nextiii@ @P\endcsname=\relax
\expandafter\let\csname\exxx@\nextiii@ @Q\endcsname=\relax
\expandafter\let\csname\exxx@\nextiii@ @S\endcsname=\relax
\expandafter\let\csname\exxx@\nextiii@ @N\endcsname=\relax
\expandafter\let\csname\exxx@\nextiii@ @F\endcsname=\relax
\expandafter\let\csname\exxx@\nextiii@ @W\endcsname=\relax
\expandafter\let\csname
end\exxx@\nextiii@\endcsname=\undefined
\fi
\expandafter\gdef\csname HL@R#1\endcsname{#2}%
\expandafter\gdef\csname\exstring@#2@R\endcsname{{HL}-{#1}}%
\iftoc@
\immediate\write\toc@{\noexpand\NameHL#1\noexpand#2^^J}%
\fi
\rightadd@#2\to\overlonglist@
\edef\next@{\let\csname\exstring@#2@C\endcsname
=\expandafter\noexpand\csname HL@C#1\endcsname}\next@
\edef\next@{\let\csname\exstring@#2@P\endcsname
=\expandafter\noexpand\csname HL@P#1\endcsname}\next@
\edef\next@{\let\csname\exstring@#2@Q\endcsname
=\expandafter\noexpand\csname HL@Q#1\endcsname}\next@
```

```

\edef\next@{\let\csname\exstring@#2@S\endcsname
=\expandafter\noexpand\csname HL@S#1\endcsname}\next@
\edef\next@{\let\csname\exstring@#2@N\endcsname
=\expandafter\noexpand\csname HL@N#1\endcsname}\next@
\edef\next@{\let\csname\exstring@#2@F\endcsname
=\expandafter\noexpand\csname HL@F#1\endcsname}\next@
\edef\next@{\let\csname\exstring@#2@W\endcsname
=\expandafter\noexpand\csname HL@W#1\endcsname}\next@
\edef\next@{\def\noexpand#2####1\expandafter\noexpand
\csname end\exstring@#2\endcsname
{\def\noexpand\HLtype@{\noexpand#2}%
\def\noexpand\HLname@{\noexpand#2}%
\gdef\noexpand\HLlevel@{#1}%
\noexpand\FNSS@\noexpand\HL@####1\noexpand\endHL}}%
\next@
\edef\next@{\noexpand\Invalid@\expandafter\noexpand
\csname end\exstring@#2\endcsname}%
\next@
}

```

For `\Namehl` we don't have to worry about an `\end...` construction. Moreover, `\Namehl` doesn't worry about `...@W` constructions, since words are seldom added to such heading levels.

```

\def\Namehl#1#2{\define#2{
\expandafter\ifx\csname hl@R#1\endcsname\relax
\else
\def\nextiv@{\let\nextiii@}%
\expandafter\nextiv@\csname hl@R#1\endcsname
\expandafter\let\nextiii@=\undefined
\expandafter\let\csname\exxx@\nextiii@ @C\endcsname=\relax
\expandafter\let\csname\exxx@\nextiii@ @P\endcsname=\relax
\expandafter\let\csname\exxx@\nextiii@ @Q\endcsname=\relax
\expandafter\let\csname\exxx@\nextiii@ @S\endcsname=\relax
\expandafter\let\csname\exxx@\nextiii@ @N\endcsname=\relax
\expandafter\let\csname\exxx@\nextiii@ @F\endcsname=\relax
\fi
}

```

```

\expandafter\gdef\csname hl@R#1\endcsname{#2}%
\expandafter\gdef\csname\exstring@#2@R\endcsname{{hl}-{#1}}
\iftoc@
  \immediate\write\toc@{\noexpand\Namehl#1\noexpand#2^^J}%
\fi
\rightadd@#2\to\nopunctlist@%
\edef\next@{\let\csname\exstring@#2@C\endcsname=
  \expandafter\noexpand\csname hl@C#1\endcsname}\next@
\edef\next@{\let\csname\exstring@#2@P\endcsname=
  \expandafter\noexpand\csname hl@P#1\endcsname}\next@
\edef\next@{\let\csname\exstring@#2@Q\endcsname=
  \expandafter\noexpand\csname hl@Q#1\endcsname}\next@
\edef\next@{\let\csname\exstring@#2@S\endcsname=
  \expandafter\noexpand\csname hl@S#1\endcsname}\next@
\edef\next@{\let\csname\exstring@#2@N\endcsname=
  \expandafter\noexpand\csname hl@N#1\endcsname}\next@
\edef\next@{\let\csname\exstring@#2@F\endcsname=
  \expandafter\noexpand\csname hl@F#1\endcsname}\next@
\edef\next@{\def\noexpand#2{%
  \def\noexpand\hltype@{\noexpand#2}%
  \def\noexpand\hlname@{\noexpand#2}%
  \gdef\noexpand\hllevel@{#1}%
  \noexpand\FNSS@\noexpand\hl@}}%
\next@}

```

23.15. \Initialize. Finally, we want to consider \Initialize, which allows the user to prescribe additional “initializations” that are done at a heading level.

We want \Initialize\HL(number) and \Initialize\hl(number) to be allowed, and also \Initialize\chapter if \chapter has been created using \NameHL. So we will use one routine, \InitH@, if \Initialize is followed by a header level \HL or \hl, and another routine, \InitS@, if it is followed by a single name like \chapter:

```

\def\Initialize{\futurelet\next\Init@}

```

```
\def\Init@{\ifx\next\HL\let\next@=\InitH@
\else\ifx\next\hl\let\next@=\InitH@
\else\let\next@=\InitS@\fi\fi\next@}
```

The definition of `\InitH@` is quite straightforward. For example,

```
\InitH@ \HL 1 {...}
```

is supposed to (globally) define ‘`\HL@J1`’ to be ‘...’. But we want to give an error message if the corresponding heading level hasn’t been defined:

```
\def\InitH@#1#2{\expandafter
\ifx\curname\exstring@#1@C#2\endcurname\relax
\def\next@{\Err@{\noexpand#1level #2 not defined
in this style}}%
\else
\def\next@{\expandafter
\gdef\curname\exstring@#1@J#2\endcurname}%
\fi
\next@}
```

Now we have to reduce `\InitS@` to `\InitH@`. Let’s suppose the argument of `\InitS@` is `\chapter`, which has been created by

```
\NameHL1\chapter
```

so that `\chapter@R` has the value `{HL}{1}`. We introduce the combining construction

```
\def\InitC@#1#2{\edef\nextii@{\expandafter\noexpand
\curname\curname#1\endcurname#2}}
```

(again compare pages 126 and 161 for the `\expandafter\noexpand`), so that

```
(A) \InitC@{HL}{1}
```

makes `\nextii@` mean `\HL1`; the point of this is that we can then use

```
\expandafter\InitH@\nextii@
```


Actually, things are not that direct. To get (A) we need

```
\expandafter\InitC@\chapter@R
```

except that ‘\chapter@R’ will actually have to be specified as

```
\csname\exstring@#1@R\endcsname
```

which cannot be used directly in this \expandafter. As on page 218, we will actually use the combination

```
\def\next@{\let\next@=}
\expandafter\next@\csname\exstring@#1@R\endcsname
\expandafter\InitC@\next@
```

The first two lines make the “nameable” control sequence \next@ have the same value as \chapter@R, and then the third line makes \nextii@ mean \HL1.

Putting this all together, our final definition is

```
\def\InitS@#1#2{\expandafter
\ifx\csname\exstring@#1@R\endcsname\relax
\Err@{\noexpand#1not defined in this style}%
\let\next@=\relax
\else
\def\next@{\let\next@=}%
\expandafter\next@\csname\exstring@#1@R\endcsname
\expandafter\InitC@\next@
\def\next@{\expandafter\InitH@\nextii@}%
\fi
\next@}
```

Chapter 24. Accessing and controlling counters, styles, etc.

We are finally ready to examine the various constructions that allow us to access and manipulate the values that `\ref` and its relatives give us.

First we will consider the constructions that allow us to access the values: `\value`, `\Evaluate`, `\pre`, `\post`, `\style`, `\numstyle`, and `\fontstyle`.

The definition of `\value` illustrates the basic strategy required: `\value\tag` should make sense, and if we `\NameHL1\chapter`, then `\value\chapter` should make sense; on the other hand, `\value\list` and `value\HL` should make sense only when followed by an appropriate number. These two different cases are easily distinguished, because `\tag@C` and `\chapter@C` are defined, but `\list@C` and `\HL@C` are not defined, while things like `'\list@C1'` and `'\HL@C1'` are defined.

The definition of `\value#1` first uses the test

```
\ifx\csname\exstring@#1@C\endcsname\relax
```

to see whether `#1` is a construction like `\tag` with a single counter. If this `\ifx` test is false, so that the counter in question does exist, it simply prints

```
\number\csname\exstring@#1@C\endcsname
```

If the `\ifx` test gives a positive result, we want to use the test

```
\ifx\csname\exstring#@#1@C1\endcsname\relax
```

to see if `#1` is a construction like `\list` with several counters. If this test also gives a positive result, we want to give an error message,

```
\noexpand\value can't be used with \string#1
```

(see section 3.4 for the use of `\noexpand`) but if this second test is false, we want to call `\value@#1`, where `\value@#1#2` tests whether

```
\csname\exstring@#1@C#2\endcsname
```

exists, giving an error message if it doesn't, and typesetting the value of this counter if it does:

```

\def\value#1{\expandafter
\ifx\csname\exstring@#1C\endcsname\relax
\expandafter\ifx\csname\exstring@#1C1\endcsname\relax
\def\next@{\Err@{\noexpand\value can't be used with
\string#1}}%
\else
\def\next@{\value@#1}%
\fi
\else
\def\next@{\number\csname\exstring@#1C\endcsname\relax}%
\fi
\next@}

\def\value@#1#2{\expandafter
\ifx\csname\exstring@#1C#2\endcsname\relax
\def\next@{\Err@{\string\value\string#1 can't be followed
by \string#2}}%
\else
\def\next@{\number\csname\exstring@#1C#2\endcsname\relax}%
\fi
\next@}

```

NOTE: In order for this to work properly, it is thus essential that any construction having any ‘`\dotsC n` ’ counter should always have at least the counter ‘`\dotsC1`’. In particular, `\HL1` should always be defined if any `\HL` level is defined, and similarly for `\hl`.

In the error message for `\value@`, we used `\string\value` since it will generally look better not to have a space after `\value`, and `\string#1` rather than `\noexpand#1`, since `#1` may not be (and usually won't be) a control sequence.

We introduce the counter `\Value`, which holds the value for `\Evaluate`,

```

\newcount\Value

```

but we won't bother printing the code for `\Evaluate`, because it is strictly analogous, except that it globally sets the value of the counter `\Value`, instead of printing a number.

The definition of `\pre#1` is quite similar, except that instead of typesetting

```
\number\csname\exstring@#1@C\endcsname
```

we typeset

```
{\csname\exstring@#1@P\endcsname}
```

(we enclose this material inside braces, in case a font change instruction is involved).

Only the first part of the definition will be given:

```
\def\pre#1{\expandafter
\ifx\csname\exstring@#1@P\endcsname\relax
\expandafter\ifx\csname\exstring@#1@P1\endcsname\relax
\def\next@{\Err@{\noexpand\pre can't be used
with \string#1}}%
\else
\def\next@{\pre@#1}%
\fi
\else
\def\next@{\csname\exstring@#1@P\endcsname}}%
\fi
\next@}
```

Notice that we now use `\foo@P` and `'\foo@P1'` rather than `\foo@C` and `'\foo@C1'` to determine whether `\pre` can be used with `\foo`. So it is possible for `\Reset\foo` to be allowed but not `\pre\foo` and vice versa. This generality will extend to all our constructions, and will even be of some importance in Chapter 25.

The definition of `\post` is strictly analogous to that for `\pre`, but with 'Q' replacing 'P' in the tests.

The definition of `\style` is also analogous, except that we don't have the extra set of braces, i.e., we have the clause

```
\def\next@{\csname\exstring@#1S\endcsname}
```

Nevertheless, this definition functions quite differently, because the `\next@` that we call will actually be a control sequence that will process its argument (the following input).

And exactly the same remarks hold for `\numstyle`.

On the other hand, `\fontstyle` has to be handled differently, because we want something like

```
\fontstyle\claim{...}
```

to expand to

```
{\claim@F...}
```

so `\fontstyle#1` must itself be a control sequence with an argument:

```
\def\fontstyle#1{\expandafter
\ifx\csname\exstring@#1F\endcsname\relax
\expandafter\ifx\csname\exstring@#1F1\endcsname\relax
\def\next@{\Err@{\noexpand\fontstyle can't be used
with \string#1}}%
\else
\def\next@{\fontstyle@#1}%
\fi
\else
\def\next@##1{{\csname\exstring@#1F\endcsname##1}}%
\fi
\next@}
```

```

\def\fontstyle@#1#2{\expandafter
\ifx\csname\exstring@#1@F#2\endcsname\relax
\def\next@{\Err@{\string\fontstyle\string#1 can't be
followed by \string#2}}%
\else
\def\next@##1{{\csname\exstring@#1@F#2\endcsname##1}}%
\fi
\next@}

```

Next we come to the constructions that allow us to manipulate the values, namely `\Reset`, `\Offset`, and the `\new...` constructions.

`\Reset` is similar to `\fontstyle`, in that `\Reset\tag`, for example, must be a control sequence with an argument, namely the number following `\Reset\tag`.

There is also one new wrinkle. If we `\Reset\tag5`, for example, then we want to set the `\tag` counter, `\tag@C`, to 4 (since the next use of `\tag` increases this counter by 1 before printing the `\tag`). But if we `\Reset\page5`, then we simply want to set `\pageno=5`;¹ this special case is handled by the boxed code:

```

\def\Reset#1{\expandafter
\ifx\csname\exstring@#1@C\endcsname\relax
\expandafter\ifx\csname\exstring@#1@C1\endcsname\relax
\def\next@{\Err@{\noexpand\Reset can't be used
with \string#1}}%
\else
\def\next@{\Reset@#1}%
\fi
\else
\def\next@##1{\count@=#1\relax
\ifx#1\page\else\advance\count@ by -1 \fi
\global\csname\exstring@#1@C\endcsname=\count@}%
\fi
\next@}

```

¹Of course, `\Reset\page` should only be used at a reasonable place, e.g., after a construction, like `\chapter`, that has started a new page.

```

\def\Reset@#1#2{\expandafter
\ifx\cscname\exstring@#1@C#2\endcscname\relax
\def\next@{\Err@{\string\Reset\string#1 can't be
followed by \string#2}}%
\else
\def\next@##1{\count@=#1\relax\advance\count@ by -1
\global\cscname\exstring@#1@C#2\endcscname=\count@}%
\fi
\next@}

```

We won't repeat the code for `\Offset`, which is analogous to that for `\Reset`, except that we don't need any special compensation for `\page`. However, there is an important point to be made about both `\Reset` and `\Offset`. Suppose that we have used

```
\NameHL1\chapter
```

(see Chapter 23). Then `\Reset\chapter` sets the value of the counter `\chapter@C`, while `\Reset\HL1` sets the value of the counter `'\HL@C1'`. But when we look at the definition of

```
\NameHL1\chapter
```

we see (page 219) that it basically involves

```
\let\chapter@C='\HL@C1'
```

and this means that `\chapter@C` and `'\HL@C1'` are simply two different names for the *same counter* (e.g., `\count47`). Consequently, `\Reset\chapter` and `\Reset\HL1` have exactly the same significance, and the same is true of `\Offset`.

This fortunate circumstance will not extend to `\newpre`, which introduces numerous new problems. First of all, `\newpre` is used in a construction like

```
\newpre\tag{\new pre material}
```

So `\newpre\tag` should essentially define `\next@` to be

```
\def\tag@P
```

and then call `\next@`, so that we will obtain

```
\def\tag@P{<new pre material>}
```

allowing the `\def` to do the work of scanning the `{<new pre material>}`.

As a first attempt we might use

```
\def\newpre#1{\expandafter
\ifx\csname\exstring@#1@P\endcsname\relax
\expandafter\ifx\csname\exstring@#1@P1\endcsname\relax
\def\next@{\Err@{\noexpand\newpre can't be used
with \string#1}}}%
\else
\def\next@{\newpre@#1}%
\fi
\else
\def\next@{\expandafter
\def\csname\exstring@#1@P\endcsname}%
\fi
\next@}
```

However, there are two features that need to be added to this definition:

- (1) As mentioned on page 192, we really want an `\edef` rather than a `\def`.
- (2) If we have used `\NameHL1\chapter`, then, unlike the situation with `\Offset` and `\Reset` discussed above, `\chapter@P` and `'\HL@P1'` are two independent control sequences, and special efforts are required to insure that `\newpre\chapter` and `\newpre\HL1` will each change *both* `\chapter@P` and `'\HL@P1'`.

For (1) it would appear that we simply have to replace the

```
\expandafter\def\csname\exstring@#1@P\endcsname
```


with

```
\expandafter\edef\csname\exstring@#1@P\endcsname
```

Unfortunately, things are not quite that simple, because, although `\newpre` should involve an `\edef`, we still want this `\edef` to be carried out while `\noexpands@` is in force!

This would seem to require something like

```
\begingroup\noexpands@
\expandafter\edef\csname\exstring@#1@P\endcsname
```

But that means

- (a) We need to supply an `\endgroup` after the `\edef` is completed.
- (b) This `\edef` must therefore be an `\xdef` in order to survive the grouping.

We can attack these two problems at once by saying

```
(A) \def\next@{%
      \def\nextii@{\endgroup
        \expandafter\let\csname\exstring@#1@P\endcsname
        =\Next@}
      \begingroup\noexpands@\afterassignment\nextii@
      \xdef\Next@}
      \next@
```

Thus, `\next@`

- (i) first supplies `\begingroup\noexpands@`,
- (ii) and then it `\xdef`'s `\Next@` to be the following text;
- (iii) after that assignment is completed, it supplies an `\endgroup`, and
- (iv) then it (locally) lets `\dots@P` be this (globally defined) `\Next@`.

We use `\Next` for this globally assigned scratch token (page 22).

For (2) we have to worry about the fact that a single control sequence to which `\newpre` applies, like `\chapter`, might have an associated pair `\HL1` to which `\newpre` should apply, and conversely, a pair like `\HL1` might have an associated control sequence `\chapter`.

In section 23.14, we already noted that if `\chapter` is created by

```
\NameHL1\chapter
```

then `'\HL@R1'` will be defined, with `\chapter` as its value, and `\chapter@R` will be defined and have as value the pair `{HL}{1}`.

On page 218 of that section, one of the problems associated with this `\...@R...` mechanism was handled with the code

```
\def\nextiv@{\let\nextiii@=}
\expandafter\nextiv@\csname HL@R1\endcsname
```

which makes the “nameable” control sequence `\nextiii@` have the same value as `'\HL@R1'`.

More generally, we will define

```
\def\getR@#1#2{\def\nextiv@{\let\nextiii@=}
\expandafter\nextiv@\csname\exstring@#1@R#2\endcsname}
```

which can be used in two ways.

1. First of all

```
\getR@\chapter{}
```

will make `\nextiii@` be `\chapter@R`. We will also define

```
\def\letR@#1#2#3{\expandafter
\let\csname#1@#3#2\endcsname=\Next@}
```

Then the effect of

```
\getR@\chapter{ }
\expandafter\letR@\nextiii@ P
```

is

```
\letR<expansion of \chapter@R>P i.e., \letR@{HL}{1}P
```

hence

```
\let\HL@P1=\Next@
```

2. On the other hand,

```
\getR@HL1
```

will make `\nextiii@` be `\HL@R1`. We will also define

```
\def\letR@#1#2{\expandafter
\let\csname\exstring@#1@#2\endcsname=\Next@}
```

Then the effect of

```
\getR@HL1
\expandafter\letR@\nextiii@ P
```

is

```
\letR@<expansion of '\HL@R1'>P i.e., \letR@\chapter P
```

hence

```
\let\chapter@P=\Next@
```

Putting this all together, we can define `\newpre` as follows:

```
\def\newpre#1{\expandafter
\ifx\csname\exstring@#1@P\endcsname\relax
\expandafter\ifx\csname\exstring@#1@P1\endcsname\relax
\def\next@{\Err@{\noexpand\newpre can't be used
with \string#1}}%
\else
\def\next@{\newpre@#1}%
\fi
```

```

\else
\def\next@{%
\def\nextii@{%
\endgroup
\expandafter\let\csname\exstring@#1@P\endcsname
=\Next@
\expandafter\ifx\csname\exstring@#1@R\endcsname\relax
\else
\getR@#1{\}\expandafter\letR@\nextiii@ P\fi}%
\begin\group\noexpands@\afterassignment\nextii@
\edef\Next@}%
\fi
\next@}

\def\newpre@#1#2{\expandafter
\ifx\csname\exstring@#1@P#2\endcsname\relax
\def\next@{\Err@{\string\newpre\string#1 can't be
followed by \string#2}}%
\else
\def\next@{%
\def\nextii@{%
\endgroup
\expandafter\let\csname\exstring@#1@P#2\endcsname
=\Next@
\expandafter\ifx\csname\exstring@#1@R#2\endcsname\relax
\else
\getR@#1{#2}\expandafter\letR@\nextiii@ P\fi}%
\begin\group\noexpands@\afterassignment\nextii@
\edef\Next@}%
\fi
\next@}

```

Of course, `\newpost` is strictly analogous, so we won't write out any of the code.

The `\newstyle` command is used in slightly more complicated situations like

```
\newstyle\tag{parameter text}{(replacement text)}
```

but this can be handled in much the same way as `\newpre`. For example, we want `\newstyle\tag` to define `\next@` to be

```
\def\tag@S
```

and then call `\next@`, so that we will obtain .

```
\def\tag@S(parameter text){(replacement text)}
```

In this case, we want a `\def`, rather than an `\edef`, and we don't need to enter a group with `\noexpands@`. Aside from these differences, however, we simply copy the code for `\newpre`:

```
\def\newstyle#1{\expandafter
\ifx\csname\exstring@#1@S\endcsname\relax
\expandafter\ifx\csname\exstring@#1@S1\endcsname\relax
\def\next@{\Err@{\noexpand\newstyle can't be used
with \string#1}}%
\else
\def\next@{\newstyle@#1}%
\fi
\else
\def\next@{%
\def\nextii@{%
\expandafter\let\csname\exstring@#1@S\endcsname
=\Next@
\expandafter\ifx\csname\exstring@#1@R\endcsname\relax
\else
\getR@#1{\expandafter\letR@\nextiii@ S\fi}%
\afterassignment\nextii@\gdef\Next@}%
\fi
\next@}

\def\newstyle@#1#2{\expandafter
\ifx\csname\exstring@#1@S#2\endcsname\relax
\def\next@{\Err@{\string\newstyle\string#1 can't be
followed by \string#2}}%
```

```

\else
\def\next@{%
\def\nextii@{%
\expandafter\let\csname\exstring@#1@S#2\endcsname
=\Next@
\expandafter\ifx\csname\exstring@#1@R#2\endcsname\relax
\else
\getR@#1{#2}\expandafter\letR@@\nextiii@ S\fi}%
\afterassignment\nextii@\gdef\Next@}%
\fi
\next@}

```

`\newnumstyle` will be handled a bit differently from `\newstyle` because `\newnumstyle` will usually be used with a single numbering style control sequence, like

```
\newnumstyle\tag{\roman}
```

and it is natural for the user to think that `\roman` is simply the second argument to `\newnumstyle`, and thus doesn't need the braces (I kept making this mistake all the time in version 1 of \LaTeX , with disastrous consequences). So we will make `\next@` be a control sequence with an argument—this works whether or not the user has added the braces. In this case, instead of having `\def\next@` at the end of our definition of `\next@`, we have it at the beginning:

```

\def\newnumstyle#1{\expandafter
\ifx\csname\exstring@#1@N\endcsname\relax
\expandafter\ifx\csname\exstring@#1@N1\endcsname\relax
\def\next@{\Err@{\noexpand\newnumstyle can't be used
with \string#1}}%
\else
\def\next@{\newnumstyle@#1}%
\fi

```

```

\else
\def\next@##1{%
\gdef\Next@{##1}%
\expandafter\let\csname\exstring@#1@N\endcsname
=\Next@
\expandafter\ifx\csname\exstring@#1@R\endcsname\relax
\else
\getR@#1{\}\expandafter\letR@\nextiii@ N\fi}%
\fi
\next@}

\def\newnumstyle@#1#2{\expandafter
\ifx\csname\exstring@#1@N#2\endcsname\relax
\def\next@{\Err@{\string\newnumstyle\string#1 can't be
followed by \string#2}}%
\else
\def\next@##1{%
\gdef\Next@{##1}%
\expandafter\let\csname\exstring@#1@N#2\endcsname
=\Next@
\expandafter\ifx\csname\exstring@#1@R#2\endcsname\relax
\else
\getR@#1{#2}\expandafter\letR@@\nextiii@ N\fi}%
\fi
\next@}

```

`\newfontstyle` is exactly analogous to `\newnumstyle`, so we won't bother repeating the code.

The definition of `\word` is strictly analogous to that of `\pre`, etc., with `W` substituted for `P` everywhere, and `word` substituted for `pre` everywhere.

The definition of `\newword` follows that for `\newstyle`, where we want a `\def`, rather than an `\edef`, and don't need to enter a group with `\noexpands@`.

```

\def\newword#1{\expandafter
\ifx\csname\exstring@#1@W\endcsname\relax
\expandafter\ifx\csname\exstring@#1@W1\endcsname\relax
\def\next@{\Err@{\noexpand\newword can't be used
with \string#1}}%
\else
\def\next@{\newword@#1}%
\fi
\else
\def\next@{%
\def\nextii@{%
\expandafter\let\csname\exstring@#1@W\endcsname
=\Next@
\expandafter\ifx\csname\exstring@#1@R\endcsname\relax
\else
\getR@#1{\}\expandafter\letR@\nextiii@ W\fi}%
\afterassignment\nextii@\gdef\Next@}%
\fi
\next@}

\def\newword@#1#2{\expandafter
\ifx\csname\exstring@#1@W#2\endcsname\relax
\def\next@{\Err@{\string\newword\noexpand#1can't
be followed by \string#2}}%
\else
\def\next@{%
\def\nextii@{%
\expandafter\let\csname\exstring@#1@W#2\endcsname
=\Next@
\expandafter\ifx\csname\exstring@#1@R#2\endcsname\relax

```



```
\else
  \getR@#1{#2}\expandafter\letR@@\nextiii@ W\fi
}%
\afterassignment\nextii@\gdef\Next@}%
\fi
\next@}
```

Chapter 25. Footnotes

25.1. Preliminaries. In addition to `\footnote`, \LaTeX has `\footmark` and `\foottext`, for special situations like the one indicated on page 52 of the \LaTeX Manual. Because of the ways that these three control sequences interact (page 246), we will need a flag

```
\newif\iffn@
```

which will be set true at the beginning of `\footnote` and then false again at the end.

Next we declare certain constructions associated with `\footmark`:

```
\newcount\footmark@C
\footmark@C=0
\def\footmark@S#1{${}^{\#1}$}
\let\footmark@N=\arabic
\def\footmark@F{\rm}
```

No definitions were given for `\footmark@P` and `\footmark@Q` because footnote markers almost never have pre- or post- material in their numbers. Consequently, `\pre\footmark`, `\newpre\footmark`, etc., will give error messages. Of course, a style file could always change this arrangement if it were necessary.

With this default definition of `\footmark@S`, the value of `\footmark@F` is actually quite irrelevant, but it might be significant if `\newstyle\footmark` were used.

We will also define

```
\def\foottext@S#1{${}^{\#1}$}
\def\foottext@F{\rm}
```

This will allow `\newstyle\foottext` and `\newfontstyle\foottext` to change things about the numbers in the `\foottext`; as mentioned on page 54 of the \LaTeX Manual, it is possible to have a different style for printing the marks within the footnote marker and within the footnote itself. On the other

hand, we don't want to have a separate counter '\foottext@C', since the numbers in the marker and in the footnote certainly have to be the same. (Since \foottext@N is undefined, we also can't have different numbering styles. If, for some weird reason, a style wanted something like a footnote mark of 3 referring to a footnote beginning with iii, the necessary modifications should be pretty obvious.)

25.2. \vfootnote@. In plain tex, the \footnote macro is defined in terms of \vfootnote (footnote in vertical mode), which does the main work of producing an \insert. L^AT_EX uses \vfootnote@ for this purpose, to avoid any possible conflict (\vfootnote@ will also work rather differently from \vfootnote).

In L^AT_EX,

```
\vfootnote@#1{_ _ _}
```

will be called when #1 represents something like the footnote marker, which has already been determined by other L^AT_EX constructions, while '_ _ _' is the text that we want to appear at the bottom of the page.

A straightforward definition of \vfootnote@ would be

```
\def\vfootnote@#1#2{\insert\footins
  {\floatingpenalty=20000
   \interlinepenalty=\interfootnotelinepenalty
   \leftskip=0pt \rightskip=0pt
   \spaceskip=0pt \xspaceskip=0pt
   \rm \splittopskip=\ht\strutbox \splitmaxdepth=\dp\strutbox
   \locallabel@{\noindent@{\footmark@F#1}
   \strut#2\strut}%
  }
```

(using \noindent@, see Chapter 8).

The \locallabel@ defines \thelabel@, . . . , in terms of \TheLabel@, . . . , which have been set by the constructions that will call \vfootnote@ (either \footnote or \foottext); this is for the use of any \label in #2. And #1 will be something like \foottext@S{\thelabel@@}, which we set at the beginning of an unindented paragraph (in the font \footmark@F, if that's relevant). Then the rest of the footnote is typeset in \rm.

The large value of `\floatingpenalty`, which is the penalty added to a page when there is a split `\insert` on it, discourages footnotes from breaking across a page. `\interlinepenalty` is the penalty for page breaks between two arbitrary lines of a paragraph. It is normally 0, but is here temporarily set equal to `\interfootnotelinepenalty`, which plain TeX gives the value 200; so if a footnote must split across a page, it will be more likely to split between paragraphs. Other styles might set other values for `\interfootnotelinepenalty`.

`\leftskip` and `\rightskip` must be set to 0pt within the `\insert` in case it occurs within text where other values are in force (paragraphs indented on the left or right), and similarly for `\spaceskip` and `\xspaceskip` (paragraphs set `\raggedright`, for example).

If a style had a construction that did something else strange, like changing the value of `\parfillskip`, then this would normally also have to be changed back within the `\footnote@`.

The `\strut#2\strut` adds a strut to the first and last line of the footnote (compare the footnote on page 57). `\strut` is defined by plain TeX in terms of `\strutbox`, which contains a vertical rule of zero width and the desired height and depth of a `\strut`; in the default style a `\strut` has height 8.5pt and depth 3.5pt. And `\splittopskip` is made the height of `\strutbox` so that the space before the first line of a split footnote will be just the same as if there were a `\strut` on this line also; similarly, the value of `\splitmaxdepth` allows footnotes to go below the bottom of the page only by the depth of `\strutbox`. Normally any point-size command, like `\tenpoint`, `\ninepoint`, etc., will redefine `\strutbox`; in the paper and book styles, `\rm` is replaced by `\eightpoint`, so that the `\strutbox` then has a smaller height appropriate for 8 point type; the `\rm` in this definition has been placed before the assignments of `\splittopskip` and `\splitmaxdepth`, to emphasize that in general, font size commands must precede them.

Actually, plain TeX uses

```
\footstrut#2\strut
```

where `\footstrut` is defined as `\vbox to\splittopskip{}`, which thus has height `\ht\strutbox`, but no depth. This will be important for a two-line footnote, whenever `\lineskiplimit` happens to have been chosen to be greater than 0pt: If a full `\strut` appeared on both lines, so that the first line has depth 3.5pt while the second line has height 8.5pt, then these two lines could not be placed together with 0pt glue between them because they would then be closer together than `\lineskiplimit`; consequently, additional `\lineskip` glue would intrude.

L^AT_EX also replaces the second `\strut` by

```
\lower\dp\strutbox\vbox to\dp\strutbox{}
```

which has the proper depth, but zero height.

(By the way, Appendix E of *The T_EXbook*, page 416, illustrates a more complicated arrangement; the `\strut` for the eight-point footnote has a height of 7pt, but a `\smallskip` (= `\vskip 3pt plus 1pt minus 1pt`) precedes each footnote, so `\splittopskip` is set to the sum, 10pt plus 1pt minus 1pt.)

Aside from being aesthetically unpleasing, struts don't actually give the right results. For example, plain T_EX, with `\baselineskip=12pt`, defines a `\strut` to have height 8.5pt and depth 3.5pt. If the last line of one footnote actually has depth 0pt, while the first line of the next footnote contains a tall symbol having a height of 9pt, there should still be 12pt between the baselines (assuming a reasonably small value of `\lineskiplimit`). But the last line of the first footnote will be artificially increased to 3.5pt by the strut, so the baselines will be separated by a total of 3.5pt + 9pt = 12.5pt. Of course, when there are large symbols of this sort, the extra space probably won't be considered too extravagant.

A more disquieting disadvantage of struts is the fact that the proper definition of a `\strut` for any particular point size is *font-dependent*. In the Computer Modern fonts, the parentheses are usually the tallest and deepest characters; in `cmr8`, for example, their height is 6pt and their depth is 2pt. Thus, for eight point type and a `\baselineskip` of 9pt, a `\strut` of height 7pt and depth 2pt is appropriate (compare *The T_EXbook*, page 415). For this manual, which uses a `\baselineskip` of 10pt for the `\eightpoint` footnotes, I originally chose a `\strut` of height 8pt and depth 2pt. But this sometimes gave strange inconsistent line spacing; it turns out that some characters (not parentheses) were deeper than 2pt, and the dimensions had to be changed to 7.5pt and 2.5pt to work with the Baskerville fonts used here.

Instead of using the straightforward definition, we will follow plain T_EX and use a more complicated construction, which doesn't consider the group `{_ _ _}` as an argument to `\vfootnote@`, thus allowing category changes within `'_ _ _'`. In plain T_EX this virtuoso performance was provided so that literal mode constructions could appear in footnotes. In version 1 of L^AT_EX, this was needed to allow invisible index entries within a footnote, since such entries involved category changes. That's no longer necessary, but we might as well allow category changes anyway, since plain T_EX already supplies most of the necessary machinery, and it is a nice feature to have.¹

¹It sure made this manual easier!

So the actual definition is more sophisticated, using the “implicit” left brace `\bgroup`. Basically, we would like to

```
\def\vfootnote@#1{\insert\footins
  \bgroup . . .
  \noindent@{\foottext@F#1}\footstrut
  \aftergroup\@foot
  \bgroup
  \let\next@=}
\def\@foot{\lower\dp\strutbox\vbox to\dp\strutbox{\egroup}}
```

Then when we have

```
\vfootnote@#1{ _ _ }
```

the `\let\next@=` is followed by `{`, so that we `\let\next@={`, thereby *removing* the `{`. Consequently, we now have

```
\insert\footins
  \bgroup . . .
  \noindent@{\foottext@F#1}\footstrut \bgroup _ _ }
```

with the `}` matching the second `\bgroup`. And after that `}`, T_EX inserts the `\aftergroup` token, `\@foot`, which gives the `\strut` and the `\egroup` that matches the remaining first `\bgroup`.

Notice that the `{ _ _ }` ends up staying in a group—the `\bgroup _ _ }`—which wouldn’t happen if we had read in it as an argument. As we will see later (page 246), this is *not* irrelevant: it introduces complications, which are most easily dealt with if the `{\foottext@F#1}` is followed by an additional control sequence, `\modifyfootnote@` which is initially `\relax`:

```
\let\modifyfootnote@=\relax
```

and which the user can change with `\modifyfootnote`,

```
\def\modifyfootnote#1{\def\modifyfootnote@{#1}}
```

We've now essentially established the required definition, except that there's always the horrid possibility of a footnote only one token long, which has been typed without braces, so we actually need a `\futurelet\next` to worry about this:

```
\def\vfootnote@#1{\insert\footins
  \bgroup
  \floatingpenalty=20000
  \interlinepenalty=\interfootnotelinepenalty
  \leftskip=0pt \rightskip=0pt
  \spaceskip=0pt \xspaceskip=0pt
  \rm \splittopskip=\ht\strutbox \splitmaxdepth=\dp\strutbox
  \locallabel@\noindent@{\foottext@F#1}\modifyfootnote@
  \footstrut\futurelet\next\fo@t}
```

Here `\fo@t` calls `\f@@t` when a group follows, but `\f@t` otherwise:

```
\def\f@t{\ifcat\bgroup\noexpand\next\expandafter\f@@t\else
  \expandafter\f@t\fi}
```

(this is a slight redefinition from plain, using the K-method). Since `\f@t` is called when we have a single token following instead of a group, we just insert that token, followed by `\@foot`, to be defined presently.

```
\def\f@t#1{#1\@foot}
```

`\f@@t` is also slightly changed from plain TeX:

```
\def\f@@t{\bgroup\aftergroup\@foot
  \afterassignment\FNSSP@\let\next@=}
```

Here the `\afterassignment\FNSSP@` adds `\FNSSP@` right after the `{` that the `\let\next@=` swallows, thereby discarding any space that might come after the `{`, and also taking care of the possibility that an invisible construction follows the `{`.¹

¹Since `\afterassignment` works on single tokens, this is the situation where we *must* have a single control sequence that is defined to mean `\FNSS@\pretendspace@`.

Finally, `\@foot` will be changed in two ways. First, we change `\strut` to

```
\lower\dp\strutbox\vbox to\dp\strutbox{}
```

and we add an `\unskip` before this, in case a mistaken space appears before the closing `}`. In addition, as we will see in sections 5 and 6,

```
\foottext calls \vfootnote@
```


while

```
\footnote sets \fn@true
and calls \footmark \vfootnote@
```

Note that although a `\footnote` is essentially like a `\footmark`, `\foottext` combination, nevertheless, `\foottext` is never called indirectly, but is used only when it actually appears in the input file, presumably matching a previous `\footmark` that appears directly in the input file.

Now `\foottext`, which leaves `\iffn@` false, begins with `\prevanish@`, since it is supposed to be invisible. Consequently, if `\iffn@` is false, we want to end with a `\postvanish@`, but if `\iffn@` is true, we simply want to reset `\iffn@` false:

```
\def\@foot{\unskip
\lower\dp\strutbox\vbox to\dp\strutbox{}}\egroup
\iffn@\expandafter\fn@false\else
\expandafter\postvanish@\fi}
```

 A style file might replace the `\rm` in the definition of `\vfootnote@` by `\eightpoint`, for example, so that footnotes would be set in 8 point type with, say, `\baselineskip=10pt`. But a user who tries to change things on the fly,

```
\footnote{\baselineskip10pt . . . }
```

will be sorely disappointed! As noted on page 244, this construction produces

```
\insert\footins\bgroup . . . [f\baselineskip10pt . . . ]\egroup
```


and . . . won't be typeset in paragraphs until the `\egroup` is encountered, by which time the value of `\baselineskip` will have been restored to its old value! (This same difficulty occurs in plain TeX).

On the other hand,

```
\modifyfootnote{\baselineskip10pt}
\footnote{...}
```

will have the desired effect, so a user can easily arrange for `\footnote`'s to be treated specially within certain constructions.

25.3. Fancy footnote numbering. Footnotes differ from all other automatically numbered \LaTeX -TeX constructions because of the possibility of “fancy footnote numbering”, whereby footnote numbers begin anew on each page. This section explains the basic strategy to be used in that case.

We will introduce the flag

```
\newif\ifplainfn@
\plainfn@true
\def\fancyfootnotes{\plainfn@false}
```

whose default true value indicates that we are using “plain” rather than “fancy” footnote numbering.

When we are using fancy footnote numbering, `\footmark@C` will continue to be incremented by 1 at each footnote. But we will have a separate counter

```
\newcount\fancyfootmarkcount@
\fancyfootmarkcount@=0
```

in which we store the latest fancy footnote number. We also need a counter in which to store the number of the most recent page on which a footnote appeared:

```
\newcount\lastfnpage@
```

Initially this should not have the value of any page that has appeared. So we will initialize it to -10000, presuming that no one will ever start something at that page number:

```
\lastfnpage@=-10000
```

Remember (Chapter 12) that when `\fancyfootnotes` is in force, \LaTeX will write lines of the form

```
F(number)1
F(number)2
F(number)3
. . .
```

to the `.lax` file, where $\langle\text{number}\rangle_1$ is the page number on which the first footnote occurs, $\langle\text{number}\rangle_2$ the number on which the second footnote occurs, etc. And the next time the file is \TeX 'ed, `\document` will use these lines to define `\fnpages@` to be

```
\\(number)1\\(number)2\\(number)3 . . .
```

Now suppose that we have come to the k^{th} footnote. By looking through `\fnpages@`, we can determine the number K of the page on which it occurs (which might not be the same as the current value of `\pageno`). If K is greater than `\lastfnpage@`, the number of the page on which the previous footnote occurs, then the current footnote is the first on the page. In this case, we will set `\fancyfootmarkcount@` to 1, and use this value for the number of the current footnote. Moreover, we will also change `\lastfnpage@` to K . But if K equals `\lastfnpage@`, so that the previous footnote occurs on this same page, we will just increase `\fancyfootmarkcount@` by 1, and use that value for the number of the current footnote (in this case, we don't have to bother changing `\lastfnpage@`).

If `\footmark@C` has the value k , the following code efficiently sets `\Next@` to the corresponding page number K , or to -10000 if there are fewer than k numbers on the list (we use `\Next@` because of the global `\xdef` [see page 22]):

```
(A)  {\let\\=\or
      \xdef\Next@{\ifcase\number\footmark@C\fnpages@
      \else -10000 \fi}
      }
```

Roughly speaking, the \xdef makes \Next@ mean

```
\ifcase k\or<number>_1\or<number>_2\or<number>_3...\else -10000 \fi
```

which immediately picks out either $\langle \text{number} \rangle_k$ or -10000 . In reality, however, things are trickier, and this code works only by a fluke: As T_EX expands $\langle \text{number} \rangle_{\text{footmark}C}$, it will have to expand out $\langle \text{fnpages} \rangle$ (since this might stand for another digit of the final number that it is looking for); consequently, all the \’s in $\langle \text{fnpages} \rangle$ will be revealed, so that T_EX will know how many \’s the \ifcase actually has. If we had been “careful” and used

```
\ifcase\number\footmarkC\relax\fnpages\else -10000 \fi
```

then we would always get the case -10000 (being sloppy pays off), so a more intricate scheme would be needed. This observation will also be important when we read in data from a .dat file for tables (Volume 2).

25.4. \footmark. In section 1, we introduced the flag \iffn@, which is true when we are processing a \footnote.

When \footmark is not called indirectly by \footnote, but is called directly (so that the flag \iffn@ will be false), we will have to store

```
{V_1}{V_2}{V_3}{V_4}
```

where V_1, \dots, V_4 have their usual significance (Chapter 11 et seq.), because this information will have to be used by some subsequent \foottext (a \label might appear in the \foottext part, and V_2 will be the marker that will appear at the beginning of the \foottext).

This information will be stored in \justfootmarklist@, initially defined to be empty,

```
\let\justfootmarklist@=\empty
```

In this case it will be more convenient for \justfootmarklist@ to be a list of the type introduced in *The T_EXbook*, page 378, so we will be using \rightappend@ (c.f. section 3.7) to append to it.

The definition of `\footmark` begins with some of the same constructions used by `\footnote` in plain to save the space factor (compare page 147), and add the italic correction to the previous letter:

```
\let\@sf=\empty
\ifhmode\edef\@sf{\spacefactor\the\spacefactor}\/\fi
```

Notice that `\/` has already been redefined by *AMS-TEX* to include an `\unskip` before the italic correction; so a space before a `\footmark` (and hence, eventually, before a `\footnote`) is ignored.

We are going to be using a modified compressed format,

```
\def\footmark{\let\@sf=\empty
\ifhmode\edef\@sf{\spacefactor\the\spacefactor}\/\fi
\def\next@{\ifx"\expandafter\nextii@\else
\expandafter\footmark@\fi}
\def\nextii@"#1"{...}
\futurelet\next\next@}
```

where we will define `\footmark@` separately (instead of using `\nextiii@`, which we would temporarily define within `\footmark`). We use this procedure simply because `\footmark@` will turn out to be quite involved, and we don't want `\footmark` to have to make such a complicated definition for `\nextiii@` each time it is used.

The first attempt for defining `\nextii@` might be

```
\def\nextii@"#1"{%
{\let\style=\footmark@S \let\numstyle=\footmark@N
\footmark@F#1
\noexpands@
\let\style=\foottext@S
\Qlabel@{#1}
}
\iffn@\else
{\noexpands@\xdef\Next@{\TheLabel@}{\TheLabel@@}
{\TheLabel@@@}{\TheLabel@@@}}
```

```

\expandafter\rightappend@\Next@\to\justfootmarklist@
\fi
\@sf\relax}

```

Here we use \Qlabel@ to create \Thelabel@, ..., \Thelabel@@@. If our \footmark came from a \footnote, which calls the pair \footmark, \vfootnote@ (see page 246) then any \label's within the \footnote will actually end in the \vfootnote@ part of the construction, and these values will then be used by the \vfootnote@. However, before using \Qlabel@, we first change the meaning of \style from \footmark@S to \foottext@S, so that \Ref within a \footnote{...} will refer to the style of the numbering that is used within the note itself, rather than to the style used for the footmark.

If our \footmark was used directly, so that the flag \iffn@ is false, then the \iffn@else... \fi clause stores the appropriate information. In this case, it is essential that \Thelabel@@ use the value of \foottext@S rather than \footmark@S, since \Thelabel@@ is what we will print at the beginning of the corresponding \foottext.

And then, finally, we restore the space factor with

```
\@sf\relax
```

The \relax is necessary, because \@sf expands out to

```
\spacefactor=(number)
```

and a digit could follow.

Unfortunately, there is problem here, of a type already addressed by $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$ for the `amspt.sty` file: If \footmark is invoked within \text (as in the example on page 52 of the $\mathcal{L}\mathcal{A}\mathcal{T}\mathcal{E}\mathcal{X}$ Manual), we don't want to add things to \justfootmarklist@ 4 times, even though \text sets things 4 times. So $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$ has a flag \iffirstchoice@, which is usually true, but which is set to false within the second, third, and fourth choices of the \mathchoice involved in the definition of \text. We will therefore use

```

\def\nextii@ "#1" {%
\iffirstchoice@
{\let\style=\footmark@S \let\numstyle=\footmark@N
\footmark@F#1

```

```

\noexpands@
\let\style=\foottext@S
\Qlabel@{#1}
}
\iffn@else
{\noexpands@
\xdef\Next@{{\TheLabel@}{\TheLabel@@}
{\TheLabel@@@}{\TheLabel@@@@}}
}
\expandafter\rightappend@\Next@\to\justfootmarklist@
\fi
\fi
\@sf\relax}

```

The definition of `\footmark@`, the routine when `\footmark` is not followed by a quoted number "...", will be quite a bit more complicated.

First, we want to advance `\footmark@C` by 1,

```
\global\advance\footmark@C by 1
```

Next we want to define `\adjustedfootmark@`, which will simply be the value of `\footmark@C` for non-fancy footnotes, but which will be the properly adjusted value when `\fancyfootnotes` is in force. For the latter case, we use the strategy explained in section 3. Note that if the test (A) on page 248 makes `\Next@` have the value -10000, then we presumably have some new footnotes, not recorded on the last run; in this case we just use `\footmark@C` for determining the value of `\adjustedfootmark@` (another run will be necessary to get everything right):

```

\ifplainfn@
\xdef\adjustedfootmark@{\number\footmark@C}
\else
{\let\|= \or
\xdef\Next@{\ifcase\number\footmark@C\fnpages@\else-10000 \fi}}
\ifnum\Next@=-10000
\xdef\adjustedfootmark@{\number\footmark@C}

```

```

\else
\ifnum\Next@=\lastfnpage@
\global\advance\fancyfootmarkcount@ by 1
\else
\global\fancyfootmarkcount@=1
\global\lastfnpage@=\Next@
\fi
\xdef\adjustedfootmark@{\number\fancyfootmarkcount@}
\fi
\fi

```

Then we define `\TheLabel@`, ..., `\TheLabel@@@`, in case it will be needed by a succeeding `\vfootnote@`. Since there is no pre- or post- material, `\TheLabel@@@` will just be the same as `\TheLabel@`:

```

{\noexpands@
\xdef\TheLabel@@{\adjustedfootmark@}
\xdef\TheLabel@\footmark@N
\xdef\TheLabel@@@{\TheLabel@}
\xdef\TheLabel@@\foottext@S
}

```

Notice that in the last step we used `\foottext@S` instead of `\footmark@S`, for the same reasons as with the definition of `\nextii@`.

As before, we then add information to `\justfootmarklist@` unless `\iffn@` is true:

```

\iffn@\else
{\noexpands@
\xdef\Next@{\TheLabel@}\TheLabel@@}
{\TheLabel@@}\TheLabel@@@}
\expandafter\rightappend@\Next@\to\justfootmarklist@
\fi

```

Finally, if we have fancy footnote numbering, we want to write an appropriate line

```

F(current page number)

```

to the .lax file. To do this we will use

```
\ifplainfn@ \else
  \edef\next@{\write\laxwrite@{F\noexpand\the\pageno}}
  \next@
\fi
```

In this `\edef`, the `\laxwrite@` is not expanded because it was created with `\newwrite` and thus with `\chardef` (compare page 55), and plain TeX's `\pageno` is also not expanded, since it was created with a `\countdef` (compare page 179). Consequently, the `\edef` makes `\next@` mean

```
\write\laxwrite@{F\the\pageno}
```

When this (delayed) write is executed, it will consequently print the proper page number.

And, finally, `\footmark@` will end with `\@sf\relax`, just like `\nextii@`.

But again, we have to perform most of this definition only when `\iffirstchoice@` is true, since we don't want to increase the counter 4 times, etc.

Thus, the whole definition of `\footmark` reads:

```
\def\footmark{\let\@sf=\empty
\ifhmode\edef\@sf{\spacefactor\the\spacefactor}\fi
\def\next@{\ifx"\next\expandafter\nextii@\else
\expandafter\footmark@\fi}
\def\nextii@###1{%
\iffirstchoice
{\let\style=\footmark@S \let\numstyle=\footmark@N
\footmark@F##1%
\noexpands@
\let\style=\foottext@S
\Qlabel@{##1}%
}%
```



```

\iffn@else
  {\noexpands@
  \xdef\Next@{\TheLabel@}{\TheLabel@@}%
  {\TheLabel@@@}{\TheLabel@@@}}%
  }%
  \expandafter\rightappend@\Next@\to\justfootmarklist@
\fi
\fi
\@sf\relax}%
\futurelet\next\next@}

```

```

\def\footmark@{%
  \iffirstchoice@
  \global\advance\footmark@C by 1
  \ifplainfn@
  \xdef\adjustedfootmark@{\number\footmark@C}%
  \else
  {\let\=\or
  \xdef\Next@{\ifcase\number\footmark@C\fnpages@
  \else-10000 \fi}}%
  \ifnum\Next@=-10000
  \xdef\adjustedfootmark@{\number\footmark@C}%
  \else
  \ifnum\Next@=\lastfnpage@
  \global\advance\fancyfootmarkcount by 1
  \else
  \global\fancyfootmarkcount=1
  \global\lastfnpage@=\Next@
  \fi
  \xdef\adjustedfootmark@{\number\fancyfootmarkcount}%
  \fi
  \fi
  {\noexpands@
  \xdef\TheLabel@@@{\adjustedfootmark@}%
  \xdef\TheLabel@\footmark@N

```

```

\edef\TheLabel@@@{\TheLabel@}%
\edef\TheLabel@@\foottext@S
}%
\iffn@ \else
  {\noexpands@\xdef\Next@{\TheLabel@}\TheLabel@@}%
  {\TheLabel@@@}\TheLabel@@@}}}%
\expandafter\rightappend@\Next@\to\justfootmarklist@
\fi
\ifplainfn@
\else
  \edef\next@{\write\laxwrite@{F\noexpand\the\pageno}}\next@
\fi
\fi
\footmark@S{\footmark@N{\adjustedfootmark@}}%
\@sf\relax}

```

25.5. `\foottext`. The `\foottext` construction has to get its information from `\justfootmarklist@`, since (compare page 246) `\foottext`'s should only occur after sufficiently many `\footmark`'s have been used. If `\justfootmarklist@` is

$$\backslash\{(stuff)_1\}\backslash\{(stuff)_2\}\backslash\dots$$

and we define

```
\def\next@\@#1#2\next@{\def\next@{#1}\gdef\justfootmarklist@{#2}}
```

then

```
\expandafter\next@\justfootmarklist@\next@
```

defines `\next@` to be `\(stuff)1`, while `\justfootmarklist@` is redefined as `\(stuff)2`....

The definition of `\foottext` will begin with `\prevanish@`. Then we will first test whether `\justfootmarklist@` is empty, and if so, we will issue an error message

There is no `\footmark` for this `\foottext`.

Assuming this hasn't happened, we will use the process just described to make \next@ be the first thing on \justfootmarklist@, and to redefine \justfootmarklist@ to be the remainder.

And then we will use

```
\expandafter\foottext@\next@
```

so that \foottext@ can use the four values of \next@:

```
\def\foottext{\prevanish@
\ifx\justfootmarklist@\empty
\Err@{There is no \noexpand\footmark for this
\string\foottext}\fi
\def\next@\##1##2\next@{\def\next@{##1}%
\gdef\justfootmarklist@{##2}}%
\expandafter\next@\justfootmarklist@\next@
\expandafter\foottext@\next@}
```

Once again, for the \noexpand in the error message, see section 3.4.

\foottext@ simply uses the next four values for defining \TheLabel@, ..., \TheLabel@@@, and then calls \vfootnote@ with the value of \thelabel@@ (the \locallabel@ part of \vfootnote@ sets this to the current value of \TheLabel@@, namely the second value of \next@):

```
\def\foottext@#1#2#3#4{\noexpands@
\xdef\TheLabel@{#1}\xdef\TheLabel@@{#2}%
\xdef\TheLabel@@@{#3}\xdef\TheLabel@@@{#4}}%
\vfootnote@{\thelabel@@}
```

The \vfootnote@ will put in a \postvanish@ at the end, to match the \prevanish@ at the beginning, since \iffn@ will not be true when \foottext is processed. The only thing left to do now, is to

```
\rightadd@\foottext\to\vanishlist@
```

25.6. `\footnote`. Finally, the definition of `\footnote` holds no special surprises. It is practically the same as the combination `\footmark`, `\foottext`, except that we use `\vfootnote@` explicitly instead of `\foottext`, so that the flag `\iffn@`, which we now set to be true, will function properly:

```

\def\footnote{\fn@true
\let\@sf=\empty
\ifhmode\edef\@sf{\spacefactor\the\spacefactor}\fi
\def\next@{\ifx\next\expandafter\nextii@
\else\expandafter\nextiii@\fi}%
\def\nextii@"##1"{\footmark"##1"%
\vfootnote@\let\style=\foottext@S
\let\numstyle=\footmark@N##1}%
\def\nextiii@\footmark
\vfootnote@\foottext@S{\footmark@N{\adjustedfootmark@}}}%
\futurelet\next\next@}

```

Part IV

*Miscellaneous
Constructions*

Chapter 26. Literal mode

The basic mechanism for literal mode is really not all that complicated. However, the final \LaTeX definitions are complicated indeed, because of

- special features that we want to incorporate;
- subtle possibilities for bugs;
- the desire for generality.

In order to temporarily eliminate the third complicating factor, we will first consider what is needed to define the particular version of literal mode in which `*` is the delimiter and `"` is used to serve as the escape character in literal mode.

26.1. In-line literal mode. The basic idea of in-line literal mode, with `*` as the delimiter and `"` as the escape character, is to

```
\def\lit*{\begingroup\litcodes@\litdefs@\tt\lit@}
\def\lit@#1*{#1\endgroup}
```

where `\litdefs@` is the abbreviation

```
\def\litdefs@{\let\0=\empty\def\1{\char42 }\def\ { \char32 }\def\"{\char34 }}
```

and `\litcodes@` changes all the category codes,

```
\def\litcodes@{\catcode'\=12
\catcode'\{=12 \catcode'\}=12
\catcode'\$=12 \catcode'\&=12
\catcode'\#=12
\catcode'\^=12 \catcode'\_ =12
\catcode'\@=12 \catcode'\~=12
\catcode'\%=12 \catcode'\ "=0 }
```

so that these new category codes are in force before `\lit@` reads in its argument.

The first line of this definition says that `\` will be an ordinary character once `\litcodes@` appears. The next two lines make ordinary characters out of the

characters with category codes 1, 2, 3 and 4. Category 5 (end of line) isn't changed for in-line-literal mode, so the next line just handles the character # with category code 6, and the next line handles category codes 7 and 8. We don't bother with category 9 (ignored character) and category 10 (space) will be attended to in a moment. Categories 11 and 12 (letter and other character) need no special treatment. The next line handles category 13 (active); the final line handles category 14 (comment character), leaving category 15 (invalid) alone, and then finally makes " the escape character.

Then `\lit*...*` will print out `'...'`, with each of the tokens `\`, `{`, `}`, `$`, `&`, `#`, `^`, `_`, `@`, `~`, `%` being printed in the `\tt` font. Moreover, within the `\lit*...*` region, `"0` will stand for `\empty`, `"1` will stand for character 42 on the `\tt` font, which is the `*` symbol, `"'` will stand for character 32, which is the `␣` symbol, and `""` will stand for character 34 on the `\tt` font, which is the `"` symbol.

However, we also want spaces to act specially. plain T_EX's `\obeyspaces` makes spaces active, and defines each active space to give one space in the output. But we will want a different definition for literal mode. If we first use

```
{\obeyspaces
\gdef\defspace@{\def {\hskip.5emminus.15em}}}
```

so that `\defspace@` makes the active space give spacing suitable for literal mode, then we can simply add

```
\obeyspaces\defspace@
```

to the end of the definition of `\litcodes@`.

Actually, we will use

```
{\obeyspaces
\gdef\defspace@{\def {\allowbreak\hskip.5emminus.15em}}}
```

Then a line break can occur between spaces. For example, `\lit*A B*`, with four spaces between the letters, could split after the second, giving 'A B'; two spaces remain at the end of the first line, although all subsequent spaces disappear before the second letter. Note that `\allowbreak` is not only shorter than `\penalty0`, but much preferable in this case, since we cannot afford to leave a space after the 0, so would have to add `\relax` to be on the safe side.

I made this choice so that the first line could at least indicate that more than one space was involved, but it looked too weird to have spaces at the beginning of the next line. If that more literal sort of literal mode is preferred, it is only necessary to define the active space to mean

```
\allowbreak\hbox to.5em{\hskip0emminus.15em}
```

In practice, one would presumably be printing `□`'s if the number of spaces needs to be emphasized.

The `\allowbreak` at the beginning of the definition of a space in literal mode does introduce one slight complication: we should

```
\def\lit*{\leavevmode\begingroup . . .}
```

—otherwise a `\lit*` at the beginning of a paragraph will contribute an `\allowbreak` in *vertical* mode, possibly overruling a `\nobreak` that occurs before it.

Moreover, the definition of `\lit@` should be changed to

```
\def\lit@#1*{#1\endgroup\null}
```

so that a `\lit*...*` construction ending with a space won't have this space deleted by any succeeding `\unskip`.

Making spaces active means that we don't have to say `\frenchspacing`, since now there aren't any spaces after punctuation. However, as *The TeXbook* points out (page 381), there's another slight problem, because the `\tt` font has the ligatures `'` and `!`, which print as `¿` and `¡` respectively (sigh). So we must

```
{\catcode'\ '=\active\gdef'\{\relax\lq}}
```

and then use

```
\catcode'\ '=\active\obeyspaces\defspace@
```

at the end of the definition of `\litcodes@`.

Finally, we want to set the `\hyphenchar` of the `\tentt` font (selected by `\tt`) to `-1`, so that hyphenation won't be allowed:

```
\def\lit*{\leavevmode\begingroup\litcodes@\litdefs@
\tt\hyphenchar\tentt=-1 \lit@}
```

od If the possibility of hyphenation is preferred, this last clause can simply be omitted. Or one could add

```
\def\-\{\discretionary{\char45 }{-}{}}
```

to the `\litdefs@`, so that `"-` could be used for a discretionary hyphen.

od In the *L_AS-T_EX Manual*, `\tt` means `\tentt` for the usual 10 point text, but `\ninett` in 9 point type (used for these “small print” sections), and `\eighttt` in 8 point (used for the footnotes). The style file for the manual simply sets the `\hyphenchar` of all three to `-1` at the beginning, but a more complicated scheme could be used. For example, we could have `\tenpoint` define `\pointsize@` to be `t`, and `\ninepoint` define `\pointsize@` to be `n`, and `\eightpoint` define `\pointsize@` to be `e`. And then we could replace the clause

```
\hyphenchar\tentt=-1
```

with

```
\if\pointsize@ t\hyphenchar\tentt=-1 \else
\if\pointsize@ n\hyphenchar\ninett=-1 \else
\hyphenchar\eighttt=-1 \fi\fi
```

(As a matter of fact, the style file for the Manual does make use of `\pointsize@`, for defining the *L_AS-T_EX* logo, since the *A*, *M*, and *S* need to be selected from different fonts for different sizes.)

26.2. *Displayed literal mode.* For “displayed” literal mode, we are going to use `\obeylines` to make `^^M` active, but we will need a new definition of the active `^^M`. It will be convenient to use

```
{\obeylines\gdef\letM@{\let^^M=\CtrlM@}}
```

where `\CtrlM@` will be defined in a moment, so that `\letM@` will make `^^M` have the meaning of `\CtrlM@` once `\obeylines` has appeared.

We can begin defining `\Lit*` by:

```
\def\Lit*{\bigskip\beginingroup
\litcodes@\obeylines\letM@\tt\Lit@}
```

We don't need to disable hyphenation, since we are going to be setting each line as a separate `\hbox` (and therefore the shrink in the definition of the active space will also be irrelevant). For reasons to be discussed in section 3, `\litdefs@` hasn't been included yet.

`\Lit@` will begin setting an `\hbox`, so we first declare

```
\newbox\litbox@
```

and then

```
\def\Lit@{\setbox\litbox@=\hbox\bgroup\litdefs@}
```

Thus, the `\Lit@` will start setting `\box\litbox@` to be whatever is on the current line, with the literal codes in effect, as well as the temporary definitions from `\litdefs@`. When we get to the end of the line, we will encounter a `^^M`, which will have the meaning of `\CtrlM@`, which we can define by

```
\def\CtrlM@{\egroup
\box\litbox@
\Lit@}
```

so that the `^^M` will simply cause the `\hbox` containing the line to be added to the vertical list, and then start setting another `\box\litbox@`.

In order to have the closing `*` end the literal display, the simplest method is to make the `*` *active* during the display and have it defined by

```
\def*{\egroup\endgroup\bigskip}
```

The `\egroup` ends the `\box\litbox@` that started being set at the beginning of the line containing the `*`, but instead of adding this (presumably empty)

box to the vertical list, we simply supply the `\endgroup` that matches the `\begingroup` contributed by `\Lit*`.

In order to do this, we need to do something like

```
{\catcode'\*=\active
 \gdef\defstar@{\def*{\egroup\endgroup\bigskip}}
}
```

so that `\defstar@` will define `*` properly once it is active, and then

```
\def\Lit*{\bigskip\begingroup
 \litcodes@
 \catcode'\*=\active\defstar@
 \obeylines\letM@\tt\Lit@}
```

The only slight problem with our definition is that the very first line of a

```
\Lit*
 . . .
 . . .
 *
```

contributes an empty box. To take care of this, and for later purposes (sections 4 and 8), we will declare a counter

```
\newcount\litlines@
```

which `\Lit*` will set to 0, and then we will have `\CtrlM@` (page 265) simply increase `\litlines@` by 1 if it is 0, but add `\box\litbox@` if `\litlines@` is not 0.

26.3. Notes for the wary. Although we still have many features to add, there are already some important point to be made about our definitions.

First of all, `\Lit*` ends up processing only one line at a time. Consequently, there is no limit on the number of lines before the ending `*`.

Second, we were careful to arrange that `\litdefs@` will be in force only within each individual `\box\litbox@`.

And this is extremely important: After any particular `\box\litbox@` has been placed on the main vertical list, T_EX may decide to exercise the `\output` routine, possibly long before the `\endgroup` supplied by the final `*`. If we had put `\litdefs@` right into the definition of `\Lit*`, then `\litdefs@` would be in effect when the `\output` routine is invoked, which could lead to havoc. For example, suppose that in the book style we have typed

```
\chapter The \LamSTeX\ Co"op\endchapter
```

so that even-numbered pages will use

```
The \LamSTeX\ Co"op
```

in the running heads. Then if this running head happened to be typeset while the literal mode definitions are in force, we would end up with

```
The LAS-TEX-Co"op
```

since the ‘’ symbol is `\char32` in the Computer Modern fonts, while `\char34` is ’’. Similarly, the user should be able to define `\0` and `\1`, and have them used in running heads, without worrying about them changing definitions if the output routine is invoked in the middle of literal mode.

26.4. Prohibiting page breaks. Normally, we want to prohibit breaks between lines of displayed literal material, so we also want to add a penalty after `\box\litbox@`. We will declare a new counter

```
\newcount\interlitpenalty@
\interlitpenalty@=10000
```

and we will have `\CtrlM@` (page 265) add `\penalty\interlitpenalty@` before the current `\box\litbox@` except for the very first box (other constructions, see section 8, can insert other penalties, or give different values to `\interlitpenalty@`). The following definition takes care of this, together with the fact that we also don't want the empty box from the first

line (page 266):

```

\def\CtrlM@{\egroup
\ifcase\litlines@
\advance\litlines@ by 1
\or
\box\litbox@ \advance\litlines@ by 1
\else
\penalty\interlitpenalty@\box\litbox@
\fi
\Lit@

```

Notice that with this arrangement, `\penalty\litpenalty@` ends up being added after each `\box\litbox@`, *except* the last box, so that a page break can occur at the `\bigskip` supplied by the closing `*` (page 265); an explicit `\penalty` could be added before the `\bigskip` if we wanted to discourage or encourage a page break at that point.

26.5. Indentation. We usually want displayed literal constructions to be indented by a certain amount. So we declare a new dimension

```
\newdimen\litindent
```

with the default value

```
\litindent=20pt
```

and we add `\hskip\litindent` at the beginning of each box:

```
\def\Lit@{\setbox\litbox@=\hbox\bgroup\litdefs@\hskip\litindent}
```

26.6. TAB's. Proper interpretation of the TAB character (`^^I`) is another feature that we want to add. We will use

```
{\catcode'\^^I=\active\gdef\letTAB@{\let^^I=\TAB@}}
```

so that `\letTAB@` will make an active `^^I` mean `\TAB@`, which we will define in a moment, and then we will add

```
\catcode'\^^I=\active\letTAB@
```

to the definition of `\Lit*`, before the `\obeylines`.

For `\TAB@`, we want, first of all, to allow for variances in the spacing for TAB's. So we will create a new counter, with the default value 8,

```
\newcount\littab@
\littab@=8
```

as well as the construction `\littab`, to change the value,

```
\def\littab#1{\littab@=#1\relax}
```

Every time that we encounter a `^~I` while we `\setbox\litbox@`, we want to supply an `\egroup`, so that `\box\litbox@` will be complete, add extra space at the end of `\box\litbox@` so that it has the requisite width (namely the next multiple of `\littab@` times the width of `\hbox{\tt0}`), and then include all this at the beginning of another

```
\setbox\litbox@=\hbox\bgroup\litdefs@
```

The TeXbook, page 391, gives a simple way to find the required extra space: If `\dimen@` is the width of our box, and `\dimen@ii` is `\littab@` times the width of `\hbox{\tt0}`, then to see how many times `\dimen@` fits within `\dimen@ii`, we can simply

```
\divide\dimen@ii by \dimen@
\multiply\dimen@ii by \dimen@
```

Here `\dimen@` will be converted to its equivalent in scaled points (i.e., 1pt will become 65536). At the end, `\dimen@ii` will be the largest multiple of `\dimen@` that is \leq `\dimen@ii`, so we will just have to increase `\dimen@` by `\littab@` times the width of `\hbox{\tt0}` to move to the next tab.¹ We have to remember, however, that all our boxes begin with an extra `\litindent` amount of space:

```
\def\tab@{\egroup
\dimen@=\wd\litbox@
\advance\dimen@ by -\litindent
```

¹Although TeX indicates dimensions with decimal points, internally all dimensions are expressed in terms of scaled points. Consequently, these calculations will not have any rounding errors.

```

\setbox0=\hbox{\tt0}%
\dimen@ii=\littab@\wd0
\divide\dimen@ by \dimen@ii
\multiply\dimen@ by \dimen@ii
\advance\dimen@ by \littab@\wd0
\advance\dimen@ by \litindent
\setbox\litbox@=\hbox\bgroup\litdefs@\hbox to\dimen@
{\unhbox\litbox@\hfil}}

```

26.7. Widow control. In order to provide control over widow lines before literal displays, we will make literal displays behave very much like displayed formulas. We first use

```

\def\Lit*{\ifhmode
  $$\abovedisplayskip=\bigskipamount
  \abovedisplaysshortskip=\bigskipamount
  \belowdisplayskip=0pt \belowdisplayshortskip=0pt
  \postdisplaypenalty=10000
  $$\vskip-\baselineskip\else\bigskip\fi
  \begingroup . . .

```

to produce an empty displayed formula right above the displayed literal mode material. The settings for `\abovedisplayskip` and for the ‘short’ version ensure that a `\bigskip` will precede this formula. The settings for `\belowdisplay` and for the ‘short’ version ensure that there is no extra space between this empty displayed formula and the displayed literal mode material. The `\vskip-\baselineskip` deletes the extra space that this empty formula contributes. Finally, the empty display and the display literal mode material are kept on the same page by the `\postdisplaypenalty`.

Consequently, TeX’s page breaking decisions before `\Lit*` material, will be the same as before displayed formulas.

If we are already in vertical mode, we just add the `\bigskip` (we don’t want to add an empty formula in that case, since `$$...$$` in vertical mode actually produces an extra blank line [*The TeXbook*, page 316]).

bd As mentioned on page 85 of the *L^AS-T_EX Manual*, changing the values of `\abovedisplayskip`, etc., in this definition will change the amount of space

before a literal display. One could even add a clause

```
\predisplaypenalty=...
```


changing `\predisplaypenalty` from its default value of 10000, to allow, or encourage, page breaks before a literal display (if this done, then a `\penalty` of that amount should be added before the `\bigskip` in the `\else` clause above). However, the last line on page 85 is just wrong: changing `\displaywidowpenalty` within the formula won't affect anything.


As with displayed formulas (compare section 16.1), special care is required in the definition of `*` made by `\defstar@` (page 265), in case an invisible construction follows. Basically we need to augment the definition to

```
\egroup\endgroup\bigskip
\vskip-\parskip
\noindent@@\futurelet\next\pretendspace@
```

to start a `\noindent`'ed paragraph, and take care of the fact that an invisible construction may follow (compare page 146). But now we have an extra complication: we must skip over the space following the `*` before adding the `\noindent@@`, so that our `\noindent`'ed paragraph doesn't begin with an extra space. So we really need

```
\egroup\endgroup\bigskip
\vskip-\parskip
\def\next@{\noindent@@\futurelet\next\pretendspace@}
\FNSS@\next@
```

 As in the case of a displayed formula (page 101), users must be warned against adding “invisible” constructions at the end of a literal display that ends a paragraph.

 The treatment of the end of display literal mode has been changed from version 1 of *L_AT_EX*, where a displayed formula was added at the end also (with undesirable side effects).

26.8. Page breaks. Within a literal display we want `\displaybreak` to force a page break when it appears on a line by itself; similarly `\allowdisplaybreak` should allow a page break, and `\allowdisplaybreaks` should allow a page break at that line and all succeeding lines.

Within the definition of `\Lit*` we will add

```
\def\displaybreak{\egroup\break\litlines@=0 \Lit@}
```

The first `\egroup` ends the `\box\litbox@` that was being set at the beginning of the line containing the

```
"displaybreak
```

The `^^M` at the end of this line is still there, and will end an empty `\box\litbox@` started by the `\Lit@`. Since we have set `\litlines@=0`, however, this box will not appear on the vertical list, nor will a penalty appear before the next `\box\litbox@`.

Thus,

```
<a line of the literal display>
"displaybreak
<the next line of the literal display>
```

will produce

```
<a line of the literal display>
\penalty -10000
appropriate \baselineskip glue for next box
<the next line of the literal display>
```

The appropriate `\baselineskip` glue will disappear after the page break at the `\penalty-10000`, so the next page will start with the second line (preceded by suitable `\topskip` glue).

Similarly, we will define

```
\def\allowdisplaybreak{\egroup\allowbreak\litlines@=0 \Lit@}
```

Then

```

<a line of the literal display>
"allowdisplaybreak
<the next line of the literal display>

```

will produce

```

<a line of the literal display>
\penalty 0
appropriate \baselineskip glue for next box
<the next line of the literal display>

```

Finally, we will define

```

\def\allowdisplaybreaks{\egroup\allowbreak
\interlitpenalty=0 \litlines@=0 \Lit@}

```

In this case,

```

<a line of the literal display>
"allowdisplaybreaks
<the next line of the literal display>

```


produces

```

<a line of the literal display>
\penalty 0
appropriate \baselineskip glue for next box
<the next line of the literal display>
\penalty 0
. . .

```

with a `\penalty0` after all succeeding lines of the literal display, except the very last (see page 268).

 This manual contains numerous page references to material in a literal display. But a `\pagelabel` right before or right after the literal display couldn't be counted upon to give the proper page number, especially since many displays were allowed to split across pages. So the style file contains the definition

```
\def\8#1\8{\pagelabel{#1}}
```

Then on page 219 I could type

```
\rightadd@#2\to\overlonglist@
\edef\next@{\global\let\cname\exstring@#2@C\endcsname
=\expandafter\noexpand\cname HL@C#1\endcsname}\next@ "8NameHLcounter"8
```

so that `\ref{NameHLcounter}` could be used on page 229 to refer to the page containing this line.

26.9. `\Litbox`. For the construction

```
\Litbox#1=*
. . .
*
```

we can first use

```
{\catcode'\*=\active
\gdef\defboxstar@{\def*{\egroup\egroup\endgroup}}
}
```

so that `\defboxstar@` defines `*` once `*` is active, in a different manner than `\defstar@` did (adding an extra `\egroup`, and omitting the remaining material). Then we want a definition like

```
\def\Litbox#1*{\begingroup\defboxstar@
\litcodes@\tt\catcode'\^I=\active\letTAB@
\obeylines\letM@\global\setbox#1=\vbox\bgroup
\litindent=0pt \litlines@=0 \Lit@}
```

The `\litindent=0pt` eliminates extra space at the beginning of each `\box\litbox@`. All these boxes are simply stacked inside the `\vbox`, and

then the closing `*` supplies an `\egroup` to end the (empty) `\box\litbox@` already begun, another `\egroup` to end the `\vbox`, and a final `\endgroup` to match the `\begingroup`.

We might as well also add in the definition of `\allowdisplaybreak` and `\allowdisplaybreaks`, in case the created box is later `\unvbox`'ed.

But there is a slight problem with our whole scheme, because we don't want the assignment `\setbox#1` to be global when `#1` is even (section 1.3). To get around this, we will declare a new box

```
\newbox\Litbox@
```

and `\global\setbox\Litbox@`, and then after the `\endgroup` supplied by the concluding `*` we will `\setbox#1\box\Litbox@` for even `#1`, but add a `\global` for odd `#1`:

```
\def\Litbox#1={\begingroup
\ifodd#1\relax\aftergroup\global\fi
\aftergroup\setbox\aftergroup#1%
\aftergroup\box\aftergroup\Litbox@
\defboxstar@
\litcodes@\tt\catcode'\^^I=\active\letTAB@
\obeylines\letM@\global\setbox\Litbox@=\vbox\bgroup
\litindent=Opt \litlines@=0 \Lit@}
```

The various `\aftergroup` tokens—`\global` (if `#1` is odd), `\setbox`, `#1`, `\box`, and `\Litbox@`—are saved up and performed after the final `\egroup` provided by the final `*`; in other words, `\setbox#1\box\Litbox@` is performed after `\box\Litbox@` has been (`\global`'ly) set.

26.10. The general definition. Now that we have all the pieces, let's try to put them together, and allow for greater generality.

We begin with things that don't depend on the particular choice of the literal delimiter or backslash.

```

\newdimen\litindent
\litindent=20pt
\newbox\litbox@
\newbox\Litbox@
\newcount\interlitpenalty@
\interlitpenalty@=10000
\newcount\litlines@

{\obeyspaces\gdef\defspace@{%
  \def {\allowbreak\hskip.5emminus.15em\relax}}

{\obeylines\gdef\letM@{\let^^M=\CtrlM@}}

{\catcode'\ '=\active\gdef'\relax\lq}}

\def\CtrlM@{\egroup
  \ifcase\litlines@
    \advance\litlines@ by 1
  \or
    \box\litbox@ \advance\litlines@ by 1
  \else
    \penalty\interlitpenalty@ \box\litbox@
  \fi
  \Lit@}

\def\Lit@{\setbox\litbox@=\hbox\bgroup\litdefs@
  \hskip\litindent}

\newcount\littab@
\littab@=8
\def\littab#1{\littab@=#1\relax}

{\catcode'\^^I=\active\gdef\letTAB@{\let^^I=\TAB@}}

```

```

\def\TAB@{\egroup
\dimen@=\wd\litbox@
\advance\dimen@ by -\litindent
\setbox0=\hbox{\tt0}%
\dimen@ii=\littab@\wd0
\divide\dimen@ by \dimen@ii
\multiply\dimen@ by \dimen@ii
\advance\dimen@ by \littab@\wd0
\advance\dimen@ by \litindent
\setbox\litbox@=\hbox\bgroup\litdefs@
\hbox to\dimen@{\unhbox\litbox@\hfil}}

```

We want to allow the possibility that no literal backslash has been chosen. For this purpose, we will first

```
\let\litbs@=\relax
```

and then have `\litdelimitern#1` redefine `\litbs@`, so that we can insert `\litbs@` within our definitions whether or not a literal backslash has been chosen.

`\litdelimitern#1` must define `\litbs@` so that it first sets the category code of `#1` to 0. For this we can use

```
\edef\litbs@{\catcode'\string#1=0 ...
```

In this `\edef`, the only thing that gets expanded is the `\string#1`. Since this gives us a character `C` of type 12,

```
\catcode'C=0
```

without the usual `\` after the `'` is quite legitimate; that's fortunate, since

```
\catcode'\#1=0
```

would be totally misinterpreted as

```
\catcode 351=0      (35 is the category code of #)
```

When we say `\litdelimiter` we also want `\litbs@` to save away the definition

```
\def"{\char'\string}
```

in `\litbs@@`, and similarly for any other argument that may be used with `\litdelimiter`. Here we will need our old `\expandafter\noexpand` trick (pages 126 and 161):

```
\let\litbs@=\relax
\let\litbs@@=\relax

\def\litbackslash#1{%
  \edef\litbs@{%
    \catcode'\string#1=0
    \def\noexpand\litbs@@{\def\expandafter\noexpand
      \csname\string#1\endcsname{\char'\string#1}}}%
  }%
```

`\litcodes` can now be defined by

```
\def\litcodes@{\catcode'\=12
  \catcode'\{=12 \catcode'\}=12
  \catcode'\$=12 \catcode'\&=12
  \catcode'\#=12
  \catcode'\^=12 \catcode'\_ =12
  \catcode'\@=12 \catcode'\~=12 \catcode'\ "=12
  \catcode'\;=12 \catcode'\:=12
  \catcode'\!=12 \catcode'\?=12
  \catcode'\%=12
  \litbs@\catcode'\ '= \active\obeyspaces\defspace@}
```

Notice that we need the `\catcode'\ "=12` since `"` is normally active in \LaTeX . The `\litbs@` is placed after all the other `\catcode`'s, since it may effect a further category code change (e.g., the category code of `"` may then be changed to 0).

The extra `\catcode's` for `;` and `:` and `!` and `?` are inserted just in case any of these have been made active for French styles (compare 3.10).

Now comes a tricky part. `\litdelimiter#1` will define `\Lit#1`, and part of this definition will be to make `#1` active and then properly define it. So we want a construction, `\activate@#1`, which will define `#1` properly when it is active. But putting

```
\def#1{...}
```

in our definition won't work, since the `#1` will be read in before `#1` actually is active! So we will resort to the `\lowercase` trick. We first use

```
\lccode'\~='#1
```

to make the `\lccode` of the *active* character `~` be `#1` (only an ordinary character is allowed for the literal delimiter, so the `'#1` is fine). This means that when `~` appears within a `\lowercase` it will be turned into an *active* `#1`.

So then we can do something like

```
\lowercase{%
  \gdef\defdelimiter@{\def~{...}}
}
```

so that `\defdelimiter@` will define our delimiter `#1` properly once it is active. We will need a similar, but different, definition for the redefinition of `#1` for `\Litbox`, so we will let `\activate@` have two arguments—the first, which will always be either 0 or 1, determining which value the global scratch token `\Next@` (see page 22) will be given:

```
\def\activate@#1#2{\lccode'\~='#2%
\lowercase{%
  \if0#1%
    \gdef\Next@{\def~{\egroup\endgroup
  \bigskip\vskip-\parskip
  \def\next@{\noindent@\futurelet\next\pretendspace}%
  \FNSS@\next@}}%
```

```

\else
\gdef\Next@{\def~{\egroup\egroup\endgroup}}%
\fi
}%
}}

```

Another thing `\litdelimiter#1` will do is to set `\litdelim@` to be the character code for #1:

```
\edef\litdelim@{\char'#1}
```

since this will be used by `\litdefs@`:

```

\def\litdefs@{\let\0=\empty\def\1{\litdelim@}}%
\def\ {\char32 }\litbs@@}

```

Remember that `\litbs@@` is either `\relax` or the proper definition of `\` if `\litdelimiter` has been used, etc.

Finally, `\litdelimiter#1` then defines `\lit#1`, `\Lit#1` and the construction `\Litbox##1=#1`. The definitions are just those in the previous sections, except that `\Lit#1` will use

```
\catcode'#1=\active \activate@0#1\Next@
```

to get the literal delimiter #1 active, and give it the proper definition, while `\Litbox##1=#1` will use

```
\catcode'#1=\active \activate@1#1\Next@
```

to get the proper definition for that case:

```

\def\litdelimiter#1{%
\edef\litdelim@{\char'#1}%
\def\lit#1{\leavevmode\begingroup\litcodes@\litdefs@
\tt\hyphenchar\tentt=-1 \lit@}%
\def\lit@##1#1{##1\endgroup\null}%

```

```

\def\Lit#1{\ifhmode$$abovedisplayskip=\bigskipamount
\abovedisplaysshortskip=\bigskipamount
\belowdisplayskip=Opt \belowdisplayshortskip=Opt
\postdisplaypenalty=1000
$$\vskip-\baselineskip\else\bigskip\fi
\beginingroup\litlines@=0
\catcode'#1=\active \activate@0#1\Next@
\def\displaybreak{\egroup\break\litlines@=0 \Lit@}%
\def\allowdisplaybreak{\egroup\allowbreak\litlines@=0
\Lit@}%
\def\allowdisplaybreaks{\egroup\allowbreak
\interlitpenalty@=0 \litlines@=0 \Lit@}
\litcodes@\tt\catcode'\^^I=\active\letTAB@
\obeylines\letM@\Lit@}%
\def\Litbox##1=#1{\beginingroup
\ifodd##1\relax\aftergroup\global\fi
\aftergroup\setbox\aftergroup##1\aftergroup\box
\aftergroup\Litbox@
\def\allowdisplaybreak{\egroup\allowbreak
\litlines@=0 \Lit@}%
\def\allowdisplaybreaks{\egroup\allowbreak
\interlitpenalty@=0 \litlines@=0 \Lit@}%
\catcode'#1=\active \activate@1#1\Next@
\litcodes@\tt\catcode'\^^I=\active\letTAB@
\obeylines\letM@\global\setbox\Litbox@
=\vbox\bgroup\litindent=Opt \litlines@=0 \Lit@}%
}

```

bd 26.11. *Nicer syntax*. Long before I wrote literal mode for \LaTeX , I had my own literal mode, based upon making * permanently active, with

* ... *

giving in-line literal mode, and

```
**
. . .
. . .
**
```

giving displayed literal mode. Thus, * worked quite analogously to \$, except that it gave literal mode instead of math mode.

Based on the constructions of the previous section, we might implement this approach as follows:

```
\catcode'\*=\active
\def*\{ \futurelet\next\star@}
\def\star@{\ifx\next*\expandafter\star@@\else
\expandafter\star@@@\fi}
\def\star@@@{(previous definition for \lit*)}
\def\star@@*{(previous definition for \Lit*)}
```

except that \star@@ should then (temporarily) redefine * as

```
\def**{\egroup\endgroup ... }
```

Actually, there is a subtle bug here: The \futurelet\next causes T_EX to read the token after the *, and since this has been done *before* the \litcodes@, this next token will have the wrong category code permanently imprinted on it. Consequently, we have to start with

```
\catcode'\*=\active
\def*\{\begingroup\litcodes@\futurelet\next\star@}
```

to get things right.

Also, the syntax for \Litbox should be changed correspondingly, to

```
\Litboxn=**
. . .
**
```

I found this arrangement so much more pleasant than the \lit and \Lit syntax (even after I had abbreviated \lit to \l and \Lit to \L) than I have used it throughout this manual.

Making `*` active does necessitate some small changes in other parts of `LATEX`. First of all, I changed the plain definition of `\ast` to

```
\def\ast{\string*}
```

so that `\ast` just gives the category 12 `*`, and can thus be used both in text and in math mode (giving the symbol `*` in text, but `*` in math).

Then I had to examine all the places where `*` already appears in `lamstex.tex`.

- (1) Its appearance in the definition of `\fnsymbol` requires no change, since the `*` appearing there is a type 12 `*`.
- (2) However, the definition of `\starparts@` needs to be restated, after `*` has been made active, because it uses a `*` as part of the syntax for the subsidiary control sequence `\next@`.
- (3) Similarly, the definition of `\starparts@@` uses `*` as part of its syntax, so it needs to be restated.
- (4) And the same was true of `\iabbrev`. Moreover, in this case, the `*` in the replacement text needs to be replaced with `\noexpand*`.

There are also a few definitions for mathematics where `*` appears.

- (1) The first is in the definition of `\keybin@` (section 3.10). Since `\ast` would now be used instead of `*` in a math formula, in the definition of `\keybin@` the

```
\ifx\next*
```

clause needs to be replaced with

```
\ifx\next\ast
```

- (2) It also appears in `\boldkey`. Although `\boldkey` is meant to be used with symbols on the keyboard, rather than control sequences, it would probably be reasonable to change the clause

```
\ifx#1\mathcharii@203 \else
```

to

```
\ifx#1\ast\mathcharii@203 \else
```

and agree that `\boldkey` can also be used with `\ast`.

I would have loved to use this nicer syntax no matter what delimiter the user chooses, but it seemed utterly hopeless to arrange this. Nevertheless, to encourage people, here in its entirety is the additional code that makes * and " the literal delimiter and escape character with this nicer syntax. With almost no modifications, it should function for other pairs, although one must then look carefully through all the \LaTeX macros to see what modifications may be needed because of the fact that another character has been made active. It is doubly easier than the code of the previous section, not only because we are dealing with a specific choice of literal delimiter and backslash, but also because the literal delimiter is active to begin with. Nevertheless, the subtle bug mentioned on page 282 introduces the possibility for an even yet more subtle bug, so there is an additional detail that is discussed at length starting on page 286.

```

\catcode'\*=\active

\def\ast{\string*} % can be used both in text and in math

\def\iabbrev*#1#2{\ifindexing@\toks@{#2}%
\immediate\write\ndx@
f\noexpand\abbrev\noexpand*\noexpand#1{\the\toks@}}\fi}

\def*{\begingroup\litcodes@\futurelet\next\star@}

\def\litcodes@{\catcode'\=12
\catcode'\{=12 \catcode'\}=12
\catcode'\$=12 \catcode'\&=12
\catcode'\#=12
\catcode'\^=12 \catcode'\_ =12
\catcode'\@=12 \catcode'\~=12
\catcode'\%=12
\catcode'\'=\active \catcode'\ "=0 \obeyspaces\defspace@}

\def\star@{\ifx\next*\expandafter\star@\else\expandafter\star@@\fi}

%%% Code for *...*

\def\star@@{\leavevmode\litdefs@\tt\hyphenchar\tentt=-1 \star@@@}%

\def\star@@@#1*{#1\endgroup}

```

```

\def\litdefs@{\let\0=\empty\def\1{\char42 }\def\"{\char34 }\def\ {\char32 }}

%%% Code for ** ... **

\def\newstar@{\def**{\egroup\endgroup
\bigskip\vskip-\parskip
\def\next@{\noindent@@\futurelet\next\pretendspace@}%
\FNSS@\next@}}


\def\star@@*{\endgroup\ifhmode$$\abovedisplayskip=\bigskipamount
\abovedisplaysshortskip=\bigskipamount
\belowdisplayskip=0pt \belowdisplaysshortskip=0pt
\postdisplaypenalty=10000
$$\vskip-\baselineskip\else\bigskip\fi
\begingroup\litcodes@
\litlines@=0
\newstar@
\def\displaybreak{\egroup\break\litlines@=0 \Lit@}%
\def\allowdisplaybreak{\egroup\allowbreak\litlines@=0 \Lit@}%
\def\allowdisplaybreaks{\egroup\allowbreak
\interlitpenalty@=0 \litlines@=0 \Lit@}%
\tt\catcode'\^^I=\active\letTAB@\obeylines\letM@\Lit@}

\def\boxstar@{\def**{\egroup\egroup\endgroup}}

\def\Litbox#1=**{\begingroup
\ifodd#1\relax\aftergroup\global\fi
\aftergroup\setbox
\aftergroup#1\aftergroup\box\aftergroup\Litbox@
\boxstar@
\def\allowdisplaybreak{\egroup\allowbreak\litlines@=0 \Lit@}%
\def\allowdisplaybreaks{\egroup\allowbreak
\interlitpenalty@=0 \litlines@=0 \Lit@}%
\tt\litcodes@\catcode'\^^I=\active\letTAB@\obeylines\letM@%
\global\setbox\Litbox@=\vbox\bgroup\litindent=Opt \litlines@=0 \Lit@}

\catcode'\@=\active

```

 The boxed additions to the definition of `\star@@` are needed because of a strange T_EX “feature”.
Suppose we type

```
\hangafter-2 \hangindent=20pt
Here are several ...
lines of text.
\vskip1in
Here are several ...
lines of text.

Here are several ...
lines of text.
```


Then the first paragraph, implicitly ended by the `\vskip1in`, will have hanging indentation of 20 points for the first two lines (like these small print notes), while the next two paragraphs will be treated normally. But now suppose we type

```
\hangafter-2 \hangindent=20pt
Here are several ...
lines of text.
{\vskip1in}
Here are several ...
lines of text.

Here are several ...
lines of text.
```

Then the second paragraph will also have hanging indentation for the first two lines!! Reason: The `\vskip1in`, implicitly ending the first paragraph, causes T_EX to restore `\hangafter` and `\hangindent` to their default values, as with any other locally defined T_EX parameters.¹ So, after the `}` that follows the `\vskip1in`, the values of `\hangafter` and `\hangindent` are still *-2* and *20pt*, respectively!!

¹ *The T_EXbook*, page 103, states that “T_EX automatically restores [the values of `\hangindent` and `\hangafter`] at the end of every paragraph, and (by local definitions) whenever it enters internal vertical mode. For example, hanging indentation that might be present outside of a `\vbox` construction won’t occur inside that `vbox`, unless you ask for it inside.” I would take that to mean that the restoration of values is done globally at other times (i.e., when ending a paragraph), but apparently that is not the case.

 If our definition of `\star@@` didn't have the `\endgroup` before the

```
\ifhmode$$ . . . $$
\vskip-\baselineskip\else\bigskip\fi
```


then something like

```
\hangafter-2 \hangindent=20pt
Here are several ...
lines of text.
**
. . .
**
```

would entail a `\vskip` within a group, and thus give rise to this problem: text following the closing `**`, which is really a new paragraph, would continue to have the same hanging indentation. Once we are safely past that `\vskip`, we reinsert the

```
\begingroup\litcodes@
```

that we initially had.

 As if that weren't confusing enough, it has to be admitted that this extra code really isn't needed, after all, because `\newstar@` defines `**` in terms of `\noindent@@`, so that the closing `**` essentially contributes a

```
\endgroup
\bigskip\vskip-\parskip
\noindent@@
```

and thus essentially

```
\endgroup
\bigskip\vskip-\parskip
\par
\noindent@
```

Now the `\par` causes the values of `\hangafter` and `\hangindent` to be restored to their default values, outside the group ended by the `\endgroup` (but `\noindent@` alone doesn't end a paragraph—it merely starts an unindented paragraph in vertical mode, and has no effect in horizontal mode).

But there's obviously no point tempting fate by relying on `\noindent@@` always being used instead of `\noindent@`.

Chapter 27. Literal mode in heading levels

As mentioned in Chapter 25, our definitions allow literal mode to work within `\footnote's`. It might seem that it should be just as easy to allow literal mode constructions within heading levels, but here the situation is much more complex.

Remember that something like

```
\hl1{Extra \lit*}* errors}
```

must not only typeset ‘Extra } errors’, it must also send

```
Extra \lit*}* errors
```

off to the `.toc` file, and it's not very clear how we are going to get these tokens, with unbalanced braces, stored inside a control sequence!¹ Moreover, a `\footnote`, no matter how formatted, usually involves an `\insert\footins{...}`, so a style file designer who wants to change the appearance of a `\footnote` probably won't to have to worry about the trickery involved in allowing category changes. But it seems much less reasonable to commit heading levels to something like

```
\global\setbox1=\vbox{...}
```

since heading levels for other style files might have to be handled quite differently.

Of course, in the great majority of cases, even when literal mode is used, it won't be needed in header levels. And even when literal mode material does occur in header levels, usually only small snatches are needed, which can be

¹ If we are willing to process the argument a token at a time, appropriately changing category codes each time we hit a `\lit` token, then it can be done (although we might have problems if the user has substituted some other control sequence for `\lit`). But it wouldn't be pleasant: since we can't put individual, unbalanced, braces into a token list, each time we hit a (non-literal-mode) `{` we would have to record this fact, and then add the `{` back in when the matching `}` is discovered, i.e., we would practically have to rewrite `TeX's` scanner in `TeX` macros.

handled quite easily with special definitions. For example, for this manual, where control sequences appear quite often in heading levels, I defined

```
\def\CS#1{{\tt\char'134 #1}}
```

so that the next section could be typed as

```
\section{Literal mode in \CS{HL} and \CS{h1}}
```

With a few definitions for the backslash `\` and the curly braces `{` and `}`, virtually any literal mode material can be included in heading levels.

For this reason, `lamstex.tex` does not directly address the problem of literal mode in heading levels. However, there is a subsidiary file, `lith1.tex`, which adds new definitions that allow literal mode to be incorporated, albeit somewhat indirectly, within heading levels.

27.1. *Literal mode in \HL and \h1.* If the file `lith1.tex` is `\input`, before a `\litdelimiter` and `\litbackslash` are declared, then `\lit*...*` will generally act as before, but two special extensions will be introduced:

- (1) On the one hand, we can type

```
\lit n*...*
```

for $n = 0, \dots, 9$. This will not typeset `...` in literal mode, but simply store the corresponding literal mode tokens in a special storage space, one for each of $0, \dots, 9$.

- (2) On the other hand, the combination

```
\lit n
```

(where the next symbol is *not* a `*`) will simply give a copy of whatever is stored in storage space n .

So, for example, the current section could be typed as

```
\lit0*\HL*
\lit1*\h1*
\section{Literal mode in \lit0 and \lit1}
```

`lith1.tex` creates new boxes

```
\expandafter\newbox\csname lit@0\endcsname
\expandafter\newbox\csname lit@1\endcsname
. . .
```

for the storage locations, and makes `\lit` a control sequence with an argument,

```
\lit#1
```

If `*` is going to be the literal delimiter, then when the argument `#1` is `*` we use `\lit@@@`, which is essentially the old `\lit*`, but if `#1` is not `*` (and thus presumably one of `0, \dots, 9`), we set

```
\count@=#1
```

and then use

```
\futurelet\next\lit@@
```

The `\futurelet` is needed to see whether the next character is a `*` or not. If it isn't (so that we have something like `'\lit0 and'`), then we simply use

```
\unhcopy\csname lit@\number\count@\endcsname\null
```

(the `\null` is added for the same reason that is was added to the original definition of `\lit@` [page 263]). But if we now have a `*`, so that we are in the case

```
\litn*...*
```

we use the routine `\lit@@@`.

We might define `\lit@@@` by

```
\def\lit@@@*{\prevanish@\begingroup
\litcodes@\litdefs@\lit@@@}
```

```

\def\lit@@@#1*{\toks@=#1}
\global\expandafter\setbox
\csname lit@\number\count@\endcsname
=\hbox{\tt\the\toks@}
\endgroup\postvanish@}

```

with \prevanish@ and \postvanish@ added to make constructions like \lit n * . . . * invisible, just in case they get used in a paragraph.

With such definitions,

```

\lit0*\HL*
\lit1*\hl*1
\section{Literal mode in \lit0 and \lit1}

```

will indeed produce the current section title. On the other hand, if we are making a .toc file, then this section will simply appear as

```

\section{27.1}
{Literal mode in \lit0 and \lit1}

```

so we will also want to have

```

\lit0*\HL*
\lit1*\hl*

```

written to the .toc file first.

This might seem fairly straightforward (after all the trickery to which we've become accustomed),

```

\def\lit@@@#1*{\toks@=#1}
\iftoc@
\edef\next@{\write\toc@{\noexpand\noexpand
\noexpand\lit\number\count@*\the\toks@*}\next@}
\fi
. . .

```

but, alas, it can fail in a subtle way.

Suppose that we wanted a heading like

1. Comparing _ and \space

and therefore typed

```
\lit0*\ " *
\lit1*\space*
\HL1 Comparing \lit0 and \lit1\endHL
```

Then the first line would cause the .toc file to contain

```
\lit0 *\ \ *
```

Reason: When we have dutifully established \litcodes@ and \litdefs@ before exercising \lit@@@@, the token list \toks@ contains two tokens: the first token is a type 12 \, and the second token is 'control-space', i.e., the control sequence whose name is " ' when " is the escape character, and ' \ ' when \ is the escape character, etc. When T_EX goes to print that control sequence in the .toc file, it will simply print it as ' \ '. To put it another way, the tokens ' \ \ ' that get written really consist of a type 12 \ followed by a type 0 \ followed by a space, but once the tokens are written, the category codes become irrelevant.

To get around this problem, we use the following byzantine strategy. After using \litcodes@ to change the category codes, we add

```
\catcode'\ "=12
```

so that " is just an ordinary character. Then \toks@ will be just the ones that we want to write to the .toc file. The problem, of course, is that they are no longer the tokens that we want to put in the \hbox: we really have to read in the * . . . * material once again, this time *without* making the special change for ". Although we cannot get T_EX to back up and read the argument again, nevertheless we can reread the argument, by first *writing* the token list \toks@ to a temporary file, and then reading it in again with the proper codes!

```
\newwrite\tempwrite@
\newread\tempread@
```

```

\def\lit@@@*{\prevanish@\begingroup\litcodes@
\catcode'\ "=12 \lit@@@}

\def\lit@@@@#1*{%
\toks@=#1}%
\iftoc
\edef\next@{\write\toc@{\noexpand\noexpand
\noexpand\lit\number\count@*\the\toks@}\next@
\fi
\immediate\openout\tempwrite@=\jobname.tmp
\immediate\write\tempwrite@{\the\toks@}
\immediate\closeout\tempwrite@
\catcode'\ "=0 \litdefs@
\immediate\openin\tempread@=\jobname.tmp
\read\tempread@ to \next@
\immediate\closein\tempread@
\global\expandafter
\setbox\csname lit@\number\count@\endcsname
=\hbox{\tt\next@}
\endgroup
\postvanish@}

```

Unfortunately, that doesn't quite work either, because there is no way that `\next@` can reflect the exact number of spaces that occurred at the end of the `*...*` sequence that we wrote to `\tempwrite`, since spaces at the end of a line are always stripped off by TeX as it reads. So yet another fillip has to be added: We will always add a `*` at the end of the sequence (this `*` can't occur within the sequence, although "1 can appear to indicate this character), and then we will strip off the `*` and everything following (presumably just the space inserted by the `^^M` at the end of the line) from `\next@`:

```

\def\lit@@@@#1*{%
\toks@=#1}%
\iftoc
\edef\next@{\write\toc@{\noexpand\noexpand
\noexpand\lit\number\count@*\the\toks@}\next@
\fi

```

```

\immediate\openout\tempwrite@=\jobname.tmp
\immediate\write\tempwrite@{\the\toks@*}%
\immediate\closeout\tempwrite@
\catcode'\ "=0 \litdefs@
\immediate\openin\tempread@=\jobname.tmp
\read\tempread@ to \next@
\immediate\closein\tempread@
\def\nextii@##1*##2\nextii@{\def\next@{##1}}%
\expandafter\nextii@\next@\nextii@
\global\expandafter
\setbox\csname lit@\number\count@\endcsname
=\hbox{\tt\next@}
\endgroup
\postvanish@}

```

27.2. The general definitions. The previous section indicated definitions to be used when * is the literal delimiter, and " is the literal backslash. Now we will give the code in general.

The file `lithl.tex` begins, like `lamstex.tex` itself, with

```
\catcode'\@=11
```

As illustrated here, we will always use double horizontal lines for code that is in subsidiary files, rather than in `lamstex.tex` itself.

Since we are going to use `\new...` constructions we then declare (compare page 38)

```
\let\alloc@=\alloc@@
```

First we declare new boxes,

```

\expandafter\newbox\csname lit@0\endcsname
. . .
\expandafter\newbox\csname lit@9\endcsname

```

and the input and output streams for the `.tmp` file,

```
\newwrite\tempwrite@
\newread\tempread@
```

Since we are going to be changing the category code of the literal backslash, if it has been chosen, `\litbackslash` will have to store extra information that allows us to do this. We declare a counter, initially with value `-1`,

```
\newcount\litbackslashno@
\litbackslashno@=-1
```

and we change the definition of `\litbackslash#1` so that it sets the value of `\litbackslashno@` to the character code of `#1`:

```
\def\litbackslash#1{%
  \edef\next@{\litbackslashno@='\string#1}\next@
  \edef\litbs@{%
    \catcode'\string#1=0
  }
  \def\noexpand\litbs@{\def\expandafter\noexpand
    \csname\string#1\endcsname{\char'\string#1}}}
```

Then we add the necessary definitions of `\lit@` et al. to the definition of `\litdelimiter`, with the category code of character `\litbackslashno@` changed back to 0 if `\litbackslashno@` isn't `-1` (if it is, we haven't declared an escape character for literal mode, so we don't have to worry about things like `\"`):

```
\def\litdelimiter#1{%
  \edef\litdelim@{\char'#1}%
  \def\lit##1{\ifx##1#1\let\next@=\lit@\else
    \count@=##1\relax\def\next@{\futurelet\next@\lit@@}\fi
  \next@}%
  \def\lit@{\leavevmode\begingroup\litcodes@\litdefs@
    \tt\hyphenchar\tentt=-1 \lit@@@}%
```

```

\def\lit@@@#1#1{##1\endgroup\null}%
\def\lit@@{\ifx\next#1\let\next@=\lit@@@}\else
\def\next@{\unhcopy\csname lit@\number\count@\endcsname
\null}\fi
\next@}%
\def\lit@@@#1{\prevanish@
\begingroup\litcodes@\ifnum\litbackslashno=-1 \else
\catcode\litbackslashno=12 \fi\lit@@@}%
\def\lit@@@#1#1{\toks@={##1}%
\iftoc@
\edef\next@{\write\toc@{\noexpand\noexpand
\noexpand\lit\number\count@#1\the\toks@#1}}\next@
\fi
\ifnum\litbackslashno=-1 \def\next@{\the\toks@}\else
\immediate\openout\tempwrite@=\jobname.tmp
\immediate\write\tempwrite@{\the\toks@#1}%
\immediate\closeout\tempwrite@
\catcode\litbackslashno=0 \litdefs@
\immediate\openin\tempread@=\jobname.tmp
\read\tempread@ to\next@
\immediate\closein\tempread@
\def\nextiii@###1#1###2\nextiii@{\def\next@{###1}}%
\expandafter\nextiii@\next@\nextiii@
\fi
\global\expandafter
\setbox\csname lit@\number\count@\endcsname
=\hbox{\tt\next@}%
\endgroup\postvanish@}%
\def\Lit#1{${\ifhmode$\abovedisplayskip=\bigskipamount
\abovedisplayshortskip=\bigskipamount
\belowdisplayskip=0pt \belowdisplayshortskip=0pt
\postdisplaypenalty=1000
$$\vskip-\baselineskip\else\bigskip\fi
\begingroup\litlines@=0
\catcode'#1=\active \activate@0#1\Next@
\def\displaybreak{\egroup\break\litlines@=0
\Lit@}%

```

```

\def\allowdisplaybreak{\egroup\allowbreak
\litlines@=0 \Lit@}%
\def\allowdisplaybreaks{\egroup\allowbreak
\interlitpenalty@=0 \litlines@=0 \Lit@}
\litcodes@\tt\catcode'\^^I=\active\letTAB@
\obeylines\letM@\Lit@}%
\def\Litbox##1=#1{\begingroup
\ifodd##1\relax\aftergroup\global\fi
\aftergroup\setbox
\aftergroup##1\aftergroup\box\aftergroup\Litbox@
\def\allowdisplaybreak{\egroup\allowbreak
\litlines@=0 \Lit@}%
\def\allowdisplaybreaks{\egroup\allowbreak
\interlitpenalty@=0 \litlines@=0 \Lit@}%
\catcode'#1=\active \activate@1#1\Next@
\litcodes@\tt\catcode'\^^I=\active\letTAB@
\obeylines\letM@\global\setbox\Litbox@=\vbox%
\bgrouplitindent=Opt \litlines@=0 \Lit@}%
}

```

Finally, we reassign `\alloc@` its original definition from plain T_EX, and make `@` active again:

```

\def\alloc@#1#2#3#4#5{\global\advance\count1#1by\@ne
\ch@ck#1#4#2\allocationnumber=\count1#1
\global#3#5=\allocationnumber
\wlog{\string#5=\string#2\the\allocationnumber}}
\catcode'\@=\active

```

bd If the alternate syntax of section 26.11 is used, something different would be needed. For example, we might create `\SL` and `\UL` (store literal mode material and use literal mode material), so that `\SLn*...*` stores the material in location n , while `\ULn` uses it. With a little work, we could even arrange for `*n*...*` to work like `\SLn*...*`, so that only `\UL` would be needed. (For the case of literal mode material that happened to begin with a digit, like `0...*`, we would then have to use `*"00*...*` to print it.)

Chapter 28. Title, author, etc., in the default style

There is, happily, not much interesting about `\title`, `\author`, `\affil`, and `\date` in the default style, except that, for the sake of economy, we carefully arrange to avoid introducing any flags to tell us which of these constructions have been used.

We allow the possibility of an empty title, so we declare

```
\newbox\titlebox@
\setbox\titlebox@=\vbox{}
```

and we

```
\righadd@\title\to\overlonglist@
```

since we want `\overlong\title` to work. Then `\title...\endtitle` will just set `\box\titlebox@`. This box won't actually be printed until the `\maketitle` appears; the main purpose of this sort of arrangement is to allow `\title`, `\author`, `\affil`, and `\date` to occur in any order before the `\maketitle`, so that the user doesn't need to know in which order these various elements have to be specified.

The `\title...\endtitle` instructions for setting `\box\titlebox@` are similar to those used for setting the `\vbox` in '`\HL@1`' (page 210), except that we might as well follow other $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{T}\mathcal{E}\mathcal{X}$ constructions and allow `\title` and `\endtitle` to function separately, instead of having `\endtitle` be part of the syntax for `\title`:¹

```
\def\title{\begingroup\let@
\global\setbox\titlebox@=\vbox\bgroup\tabskip\hss@
\halign to\hsize\bgroup
\bf\hfil\ignorespaces##\unskip\hil\cr}
\def\endtitle{\crcr\egroup\egroup\endgroup
\overlong@false}
```

¹There was really no point doing that for `\HL...\endHL`, since something like `\chapter...\endchapter` would have to be defined in terms of `\HL...\endHL` anyway.

Although `\maketitle` is going to print `\box\titlebox@` even if it is empty (the idea is to leave some space for a hand-written title, or perhaps a title with some weird special symbols, etc.), no author information is going to be printed unless `\author... \endauthor` explicitly appears.

The definition of `\author` is exactly analogous to the definition of `\title`,

```

\newbox\authorbox@
\rightadd@\author\to\overlonglist@
\def\author{\begingroup\Let@
  \global\setbox\authorbox@=\vbox\bgroup\tabskip\hss@
  \halign to\hsize\bgroup
  \rm\hfil\ignorespaces##\unskip\hfil\cr}
\def\endauthor{\crr\egroup\egroup\endgroup
  \overlong@false}

```

except that initially `\box\authorbox@` will be void; thus, `\maketitle` will be able to use the test `\ifvoid\authorbox@` to tell whether `\author` has been used. `\rm` was added just in case some other font has already been selected, for some weird reason.

And `\affil` is exactly analogous:

```

\newbox\affilbox@
\def\affil{\begingroup\Let@
  \global\setbox\affilbox@=\vbox\bgroup\tabskip\hss@
  \halign to\hsize\bgroup
  \rm\hfil\ignorespaces##\unskip\hfil\cr}%
\def\endaffil{\crr\egroup\egroup\endgroup
  \overlong@false}

```

`\date` is a little different, since we don't create a box. Instead, we define `\date@`, initially set equal to `\relax`:

```

\let\date@=\relax
\def\date#1{\gdef\date@{\ignorespaces#1\unskip}}

```

We add the appropriate `\ignorespaces` and `\unskip` at this stage, since that is much easier than trying to insert them into an already defined `\date@` later on.

The definition of `\today` is taken right from *The T_EXbook* (page 406):

```
\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}
```

And then, finally, `\maketitle` simply puts everything together. We use

```
\hrule \height0pt \vskip-\topskip
```

to get to the very top of the page, and then

```
\vskip24pt plus12pt minus12pt
\unvbox\titlebox@
```

to put (stretchable) space before the (possibly empty) title. Then we add some more space and the author(s), but only if there are some,

```
\ifvoid\authorbox@\else
\vskip12pt plus6pt minus3pt\unvbox\authorbox@ \fi
```

The affiliation and date are handled similarly, except that the date is put inside a `\centerline` (with `\rm` explicitly stated). And then, finally, some extra space is added before the first material of the document proper:

```
\def\maketitle{\hrule \height0pt \vskip-\topskip
\vskip24pt plus12pt minus12pt
\unvbox\titlebox@
\ifvoid\authorbox@ \else
\vskip12pt plus6pt minus3pt \unvbox\authorbox@ \fi
\ifvoid\affilbox@ \else
\vskip10pt plus5pt minus2pt \unvbox\affilbox@ \fi
\ifx\date@\relax\else
\vskip6pt plus2pt minus1pt \centerline{\rm\date@} \fi
\vskip18pt plus12pt minus6pt}
```

Chapter 29. The bibliography

\LaTeX 's bibliography constructions, an extension of those originally used in `amspt.sty`, are really quite adequate for most bibliography requirements. They show that various “fields” of information, allowed to appear in any order, can be put together properly by \TeX itself, without resorting to an external program like `BIBTEX`. Moreover, as explained on page 98 of the \LaTeX Manual, the bibliographic entries can be labelled, and thus cited within the text using `\ref`, so that the proper number for a bibliographic entry is printed automatically (after enough passes).

Of course, `BIBTEX` also allows bibliographic entries to be selected from a data base and sorted in any of numerous desired ways. Moreover, the `BIBTEX` approach has the advantage that the final result (the `.bbl` file) consists of standard (well, almost standard) \TeX code, and is thus easily edited and modified. By contrast, it may be quite difficult to coerce \LaTeX 's bibliography macros into performing as desired, and many special sorts of \TeX trickery had to be built into the macros for this purpose. (With either approach, careful proofreading of the bibliography—rarely attempted by the authors, alas—is advisable, to check that special situations have been handled correctly.)

Since many people have already made extensive data bases for `BIBTEX`, which they presumably don't want to go to waste, \LaTeX now provides an interface with `BIBTEX`, as explained in the next chapter. (Of course, it would be nice if there were a `LIBTEX` program, working like `BIBTEX`, but producing a `.bbl` file with \LaTeX code instead of \LaTeX code.)

Others may prefer using \LaTeX 's bibliography macros, however, especially since they provide features missing from `BIBTEX`. I have added all features that the AMS has added to `amspt.sty` (though often with modified syntax), not to mention a few more of my own. All in all, this was a rather harrowing experience—I now understand the perils of creeping featureism. I ended up deferring the bibliography macros to the very end, rightly dreading all the details that would be involved. On the other hand, when I finally came to writing this chapter, it turned out that the description of the macros, and the strategy behind them, went rather smoothly.

Most important of all, from the point of view of the user or style file designer, once \LaTeX 's hidden macros have taken care of all the messy details, the final process of printing the information from all the fields, in the proper

order, and with proper punctuation and spacing, is fairly straightforward, and thus easily modified if some other sort of arrangement is needed.

29.1. `\cite`. The `amspt.sty` has a `\cite` construction, which simply prints its argument within brackets. When we are interfacing with `BIBTEX`, `\cite` will work essentially the same as in `LATEX`, but we will first give a default definition. We will still allow `\cite` to producing something like [Knuth1984, page 123], but the syntax for indicating the additional information ‘page 123’ will be changed, to

```
\cite(page~123){Knuth1984}
```

This is close to `LATEX`’s

```
\cite[page~123]{...}
```

syntax for optional arguments; however the use of the brackets has been avoided because they represent letters on Scandinavian keyboards. Of course, any `)`’s in the additional text must be hidden in braces,

```
\cite({section 1(a)}){...}      [... , section 1(a)]
```

There’s nothing very surprising about the definition of `\cite`, which uses standard techniques:

```
\def\cite{%
  \def\nextii@##1##2-{\rm[]-##2}, {##1\/-}{\rm[]}}%
  \def\nextiii@##1-{\rm[]-##1\/-}{\rm[]}}%
  \def\next@{\ifx\next\expandafter\nextii@\else
    \expandafter\nextiii@\fi}%
  \futurelet\next\next@}

```

bd Note that we always specify `\rm [and]`. We put `##1` and `##2` inside braces, in case a font change instruction is added. (When we are interfacing with `BIBTEX`, a font change instruction will only be allowed in the optional argument #1 (see page 349).

29.2. Features of L_AM_S-T_EX's bibliography macros. The original `amspt.sty` command `\Refs` is called

```
\makebib
```

in L_AM_S-T_EX, and this `\makebib` now requires a matching

```
\endmakebib
```

at the end of the entries. With the

```
\makebib
. . .
\endmakebib
```

region, the general `amspt.sty` syntax

```
\ref ... \endref
```

has been changed to

```
\bib ... \endbib
```

since `\ref` already has another use in L_AM_S-T_EX.

Within `\bib... \endbib` we can use the fields

```
\no \key
\by \bysame
\paper \jour \vol \issue \yr \toappear
\pg \pp
\book \inbook \publ \publaddr
\paperinfo \bookinfo \finalinfo
```

which correspond to those from the original `amspt.sty`, with `\pg` and `\pp` replacing `\page` and `\pages`, since `\page` has another use in L_AM_S-T_EX.

Some changes have been made in conformity with changes by the AMS:

- `\key` now automatically adds brackets, and sets its field in `\bf`. Thus, `\key C1` gives `[C1]`, which was previously typed as `\key \bf C1`.

- `\inbook` normally prints only the book title, not preceded by ‘in ’ [which many people don’t like very much], although, as discussed below, this can be changed.
- `\issue` now prints ‘no. ’ before its field, though I think that’s very bad (`\issue` was originally designed for something like ‘Special commemorative issue’).
- The AMS has also changed `\finalinfo`, so that it is preceded by a comma, rather than a period after all the previous information. I think that’s even worse, and have kept the old arrangement (see page 312 for further discussion of this particular point.)

As in the AMS’s new `amspt.sty`, there is no longer a `\manyby` to indicate the start of a sequence of `\bysame`’s. The first reference is simply typed as `\by`, with `\bysame` used for the subsequent ones. In addition, `\bysame` now prints a horizontal rule of fixed width, rather than one that varies with the width of the first instance. (This makes the macros considerably easier to write, but it’s what journals always use anyway, so there’s no point apologizing for the shortcut.)

`\moreref` has been changed to `\morebib`. As before, something like

```
\bib \no 2 \by L. Auslander
\paper On the Euler characteristic of
compact locally affine spaces
\jour Comment. Math. Helv. \vol 35 \yr 1961 \pp25--27
\morebib
\paper \rm II
\jour Bull. Amer. Math. Soc. \vol67 \yr1961 \pp 405--406
\endbib
```

will produce

```
┌ 2. L. Auslander, On the Euler characteristic of compact locally affine spaces, Com- ┐
  ment. Math. Helv. 35 (1961), 25–27; II, Bull. Amer. Math. Soc. 67 (1961),
└ 405–406. ┘
```

If part II of this paper had appeared in the same journal, but in a different

volume,

```
\bib \no 2 \by L. Auslander
\paper On the Euler characteristic of
compact locally affine spaces
\jour Comment. Math. Helv. \vol 35 \yr 1961 \pp25--27
\morebib
\paper \rm II
\vol36 \yr1961 \pp 13--15
\endbib
```

the output would look like

```
┌ 2. L. Auslander, On the Euler characteristic of compact locally affine spaces, Com-
└ ment. Math. Helv. 35 (1961), 25–27; II 36 (1961), 13–15. ─
```

`\morebib` remembers that there was a `\jour` before, so it prints the `\vol`, `\yr`, and `\pp` fields for the second title even though no `\jour` is given for that title.

By the way, this example illustrates one of those innumerable circumstances where a completely automated system won't give the optimal results: journal volume numbers in the default style happen to be printed without commas following the preceding field, which looks just fine for the 35 following 'Comment. Math. Helv.', but not so fine after the shortened title 'II'; in this case we would probably want to change the input to

```
\paper \rm II,
```

(see also page 311).

This "remembering" feature of `\morebib` is generally quite convenient, but

it doesn't give the desired results in certain other situations. For example,

```

\bib
\no 3 \by Stefan Banach
\paper Sur la d\ecomposition des ensembles de points
  en parties respectivement congruents
\jour Fund. Math. \vol 6 \yr 1925 \pp 244--277
\morebib
\book \OE uvres
\publ \ 'Editions Scientifiques de Pologne
\publaddr Warsaw \yr 1967
\endbib

```

will produce

- ```

┌ 3. Stefan Banach, Sur la décomposition des ensembles de points en parties respectivement congruents, Fund. Math. 6 (1925), 244–277; (1967), Œuvres, Éditions Scientifiques de Pologne, Warsaw.
└

```

Here the 1967 got printed first, as if it were the `\yr` for a `\jour`, because `\morebib` remembered that a `\jour` appeared before.

So there is now `\anotherbib`, which clears out such information. If we use `\anotherbib` instead of `\morebib` we will get the desired result:

- ```

┌ 3. Stefan Banach, Sur la décomposition des ensembles de points en parties respectivement congruents, Fund. Math. 6 (1925), 244–277; Œuvres, Éditions Scientifiques de Pologne, Warsaw, 1967.
└

```

By the way, the AMS does not seem to have retained this “remembering” feature for `\morebib`. At any rate, the *AMS-TEX Version 2.0 User's Guide*

gives an example like

```

\bib \no 7
\by P. D. Lax and C. D. Levermore
\paper The small dispersion limit for the KdV equation.~\rm I
\jour Comm. Pure Appl. Math. \vol 36 \yr1983
\pp 253--290
\morebib\paper \rm II
\jour Comm. Pure Appl. Math.
\vol 36 \yr 1983 \pp 571--594
\morebib\paper \rm III
\jour Comm. Pure Appl. Math.
\vol 36 \yr 1983 \pp 809--829
\endbib

```

to produce

```

┌ 7. P. D. Lax and C. D. Levermore, The small dispersion limit for the KdV equa-
  tion. I, Comm. Pure Appl. Math. 36 (1983), 253–290; II, Comm. Pure
  Appl. Math. 36 (1983), 571–594; III, Comm. Pure Appl. Math. 36 (1983),
└ 809–829.

```

which is an obvious example of overkill. The listing

```

┌ 7. P. D. Lax and C. D. Levermore, The small dispersion limit for the KdV equa-
  tion. I, Comm. Pure Appl. Math. 36 (1983), 253–290; II, 571–594; III,
└ 809–829.

```

would have been preferable by far.

There are now four new fields, as added by the AMS:

- `\ed` and `\eds` are for one editor, or several editors, respectively, of a book; the first adds “ed.” after the editor’s name, while the second adds “eds”; all the information is enclosed in parentheses.
- `\lang`, for the original language of a translation, prints its information, enclosed in parentheses, at the very end, after the final punctuation for the `\bib` entry. I have followed the AMS macros in this regard—`\lang` basically becomes `\finalinfo`—although I don’t think that’s the optimal solution.

- `\transl` is for translation information, preceding the `\jour` or `\book` to which it pertains; thus, an entry with `\transl` may well have more than one `\jour` or `\book`—in essence, `\transl` functions something like `\anotherbib`.

As examples,

```
\bib\no9 \by S. Kripke
\paper Semantical analysis of intuitionistic logic \rm I
\inbook Formal Systems and Recursive Functions
\eds J. Corssely and M. A. E. Dummett
\publ North-Holland \yr1965 \pp92--130
\endbib
```

produces

```
┌ 9. S. Kripke, Semantical analysis of intuitionistic logic I, Formal Systems and
└ Recursive Functions (J. Corssely and M. A. E. Dummett, eds.), North-
  Holland, 1965, pp. 92–130. ─
```

and

```
\bib\no6 \by O. A. Ladyzhenskaya
\book Mathematical problems in the dynamics of a viscous
incompressible fluid \bookinfo 2nd rev. aug. ed.
\publ ‘‘Nauka’’ \publaddr Moscow \yr 1970
\lang Russian
\transl English transl. of 1st ed.
\book The mathematical theory of viscous
incompressible flow
\publ Gordon and Breach \publaddr New York
\yr 1963; rev. 1969
\endbib
```

produces

6. O. A. Ladyzhenskaya, *Mathematical problems in the dynamics of a viscous incompressible fluid*, 2nd rev. aug. ed., "Nauka", Moscow, 1970 (Russian); English transl. of 1st ed., *The mathematical theory of viscous incompressible flow*, Gordon and Breach, New York, 1963; rev. 1969.

As you can see from these examples, the default style, in conformity with the AMS's changes, now prints both paper titles and book titles in italics, except for book titles produced by `\inbook`. However, there are new AMS constructions

```
\bookinquotes
\paperinquotes
```

which have also been added in L^AT_EX. Typing

```
\bookinquotes
```

after `\makebib` will cause all the book entries to be placed in quotes (and in roman type), and similarly for `\paperinquotes`.

`\paperinquotes` and `\bookinquotes` can be used together, but I personally feel that one, and only one, of these commands should always be used, to distinguish between papers and books. The necessity for this is well illustrated by one of the AMS's examples in the User's Guide:

4. V. I. Arnol'd, A. N. Varchenko, and S. M. Guseĭn-Zade, *Singularities of differentiable maps. I*, "Nauka", Moscow, 1982. (Russian)

Until I looked at the input,

```
\bib \no 4 \by V. I. Arnol'$d, A. N. Varchenko,
and S. M. Guse\u\i n-Zade
\book Singularities of differentiable maps.~\rm I
\publ 'Nauka' \publaddr Moscow \yr 1982
\lang Russian
\endbib
```

I didn't know this was a book! (Normally a book would have something like 'Volume 1' in its title.)

The Kripke example on page 308 appears on page 263 of the second edition of *The Joy of T_EX*, but with

```
\inbook in Formal Systems and Recursive Functions
```

to explicitly add ‘in’ before the book title. Unfortunately, that won’t work very well if `\bookinquotes` has been specified! Instead, I have added

```
\ininbook
```

to add ‘in’ before all book titles specified by `\inbook`; if `\bookinquotes` has also been specified, the quotes will go only around the book title itself.

The AMS has extended the use of (the old) `\nofrills` within the bibliography: `\nofrills` after a field suppresses the punctuation that would normally occur. The above mentioned example was actually given as

```
\bib
\no9 \by S. Kripke
\paper\nofrills Semantical analysis of
intuitionistic logic \rm I;
\inbook in Formal Systems and Recursive Functions
\eds J. Corssely and M. A. E. Dummett
\publ North-Holland \yr1965 \pp92--130
\endbib
```

to produce

```
┌ 9. S. Kripke, Semantical analysis of intuitionistic logic I; in Formal Systems and Recursive Functions (J. Corssely and M. A. E. Dummett, eds.), North-Holland, 1965, pp. 92–130. └
```

`\nofrills` has been replaced by `\nopunct` and `\nospace` in $\text{L}_{\text{A}}\text{M}\text{S}-\text{T}_{\text{E}}\text{X}$ (and its positioning has been changed), and this usage now extends to bibliography items also: In $\text{L}_{\text{A}}\text{M}\text{S}-\text{T}_{\text{E}}\text{X}$ the above example could be typed as

```
\ininbook
```



```

\bib
\no9 \by S. Kripke
\nopunct\paper Semantical analysis of
intuitionistic logic \rm I;
\inbook Formal Systems and Recursive Functions
\eds J. Corssely and M. A. E. Dummett
\publ North-Holland \yr1965 \pp92--130
\endbib

```

Note that on page 305 we might use

```
\nopunct \paper \rm II,
```

if we weren't sure about the treatment of punctuation for the next field.

The AMS also allows `\nofrills` to occur before a field name, in order to suppress punctuation after the *previous* field. For L_AM_S-T_EX, where `\nopunct` and `\nospace` always precede the fields, I have added

```

\noprepunct
\noprespace

```

For example, to print

```

┌ 7. P. D. Lax and C. D. Levermore, The small dispersion limit for the KdV equa-
  tion. I, Comm. Pure Appl. Math. 36 (1983), 253–290 (overview); II, 571–
└ 594; III, 809–829.

```

we can use

```

\bib \no 7
\by P. D. Lax and C. D. Levermore
\paper The small dispersion limit for the KdV equation.~\rm I
\jour Comm. Pure Appl. Math. \vol 36 \yr1983
\pp 253--290
\noprepunct\finalinfo (overview)
\morebib\paper \rm II
\pp 571--594

```

```
\morebib\paper \rm III
\pp 809--829
\endbib
```

thereby suppressing the punctuation on the field immediately preceding the `\finalinfo` (we might not be sure just which field this is).

As another example, note that in \LaTeX the input

```
\bib
\key C \by H. Cartan
\paper Operations dans les construction acycliques
\inbook Seminaire H. Cartan 1954--55
\bookinfo Expos'e 6 \publ ENS \publaddr Paris
\finalinfo Reprinted by W. A. Benjamin, New York (1967)
\endbib
```

produces

```
[C] H. Cartan, Operations dans les construction acycliques, Seminaire H. Cartan
1954--55, Exposé 6, ENS, Paris. Reprinted by W. A. Benjamin, New York
(1967).
```

with the information from the `\finalinfo` field following a period after all the other fields (see page 304). To print this in the AMS's manner,

```
[C] H. Cartan, Operations dans les construction acycliques, Seminaire H. Cartan
1954--55, Exposé 6, ENS, Paris, reprinted by W. A. Benjamin, New York
(1967).
```

we can type

```
. . . .
\publ ENS \publaddr Paris
\noprespace\noprepunct\finalinfo , reprinted by
W. A. Benjamin, New York (1967).
\endbib
```

Notice also that, once again in conformity with the AMS's changes, punctuation is supplied automatically after `\finalinfo`, unless it is preceded by `\nopunct`.

Finally, it turned out that one more such modifier was needed. When book or paper titles are printed in quotes, we sometimes want to suppress the quotation marks. For example,

```
┌ 2. L. Auslander, "On the Euler characteristic of compact locally affine
└   spaces," Comment. Math. Helv. 35 (1961), 25-27; "II," Bull. Amer. Math.
    Soc. 67 (1961), 405-406. ─┘
```

looks much better if we suppress the quotations around the II,

```
┌ 2. L. Auslander, "On the Euler characteristic of compact locally affine
└   spaces," Comment. Math. Helv. 35 (1961), 25-27; II, Bull. Amer. Math.
    Soc. 67 (1961), 405-406. ─┘
```

To obtain this, we would type

```
\morebib
\noquotes\paper \rm II
. . .
```

29.3. Storing the fields. One of the basic problems in the bibliography macros is that we want to be able to type things like

```
\by ... \paper ... \jour ...
```

instead of

```
\by {...} \paper {...} \jour {...}
```

but we also don't want to force the various elements to be typed in a particular order, or force all of them to appear.

The solution to this problem is to have boxes, say `\bybox@`, `\paperbox@`, `\jourbox@`, ..., in which to store the information from these various fields, and to make definitions like

```
\def\by{\unskip\egroup \setbox\bybox@=\hbox\bgroup}
\def\paper{\unskip\egroup \setbox\paperbox@=\hbox\bgroup}
```

The idea is that the `\egroup` matches the `\bgroup` from the previous field, and then we start storing the current field. (The `\unskip` before the `\egroup` simply removes any extraneous space at the end of the previous field; it is convenient to get rid of this space at the very outset, instead of worrying about it later.) The definition of `\bib` will have to start with an extra `\bgroup`, which will be closed by the `\egroup` of the first field that follows, while `\endbib` will supply a final `\egroup`, and then take the material in all these boxes, and suitably arrange them.

This whole idea works because `\hbox` can be ended by `\egroup`, rather than an explicit `}`. However, several details intrude:

(1) If the user adds one of the line breaking commands, `\nolinebreak`, `\allowlinebreak`, `\linebreak`, or `\newline` at the end of a field, we want the corresponding `\penalty` to be inserted *after* the punctuation that will follow the field when it is printed, not before.

Recall (section 3.5) that the \LaTeX definitions of `\nolinebreak`, `\allowlinebreak`, and `\linebreak` have an extra element `\kerns@` at the end, while `\newlines` has `\nkerns@`. Originally these are both `\relax`, but now we will change them, so that they involve very small kerns like

```
\kern-1sp \kern1sp
```

The presence of such `\kern`'s, presumably not ever explicitly inserted by the user, will allow us to recognize that such commands were typed, and to deal with them suitably (details are presented in section 10).

(2) Some people like to prepare a "template" with all possible fields,

```
\bib
\no
\key
\paper
. . .
```

and then fill in only the necessary fields. So we have to deal with the possibility that certain fields are empty.

(3) Finally, there is the phenomenon reported by Michael Downes in TUGBOAT, Volume 11, No. 4. Normally when \TeX is setting text it inserts a discretionary break after hyphens, en- and em-dashes, and explicitly typed

discretionary hyphens `\-`. But these are omitted when setting an `\hbox` (in restricted horizontal mode). If the `\hbox` is later `\unhbox`'ed, and made part of a paragraph, TeX will attempt hyphenation, as usual, if it can't set the paragraph without hyphenation. But when this second attempt is made, hyphenations are inserted only by the hyphenation algorithm—possible break points after hyphens, dashes and discretionary hyphens are not added at this stage. Since the possible break points after hyphens, dashes and discretionary hyphens weren't added in the original `\hbox`, they have thus been lost forever. So, for example, a compound word like “Nebraska–Lincoln” will not be able to break properly at the end of a line.

The solution to this problem will be to set everything as a `\noindent`'ed paragraph within a `\vbox` with `\hsize=\maxdimen`, and then take the `\vbox` apart. So we will be using definitions like

```
(A) \def\paper{\unskip\egroup
     \setbox\paperbox@=\vbox\bgroup \hsize=\maxdimen }
```

In this situation, line breaking commands at the end of a field have to be handled a bit differently. We will add things like

```
\null\kern-1sp\kern1sp
```

so that the `\null` will keep the `\kern`'s from disappearing at the end of the line.

29.4. Starting the bibliography macros. The definition of `\makebib` begins

```
\def\makebib@W{Bibliography}
\def\makebib{\begingroup
  \rm
  \bigbreak
  \centerline{\smc\makebib@W}
  \nobreak\medskip
  \sfcode'\.=1000 \everypar={}\parindent=0pt
```

We use `\makebib@W` instead of specifying ‘Bibliography’ explicitly, so that `\newword\makebib` can be used to change this heading (compare section 23.6 and Chapter 24). `\rm` is added, just in case another font has been selected.

The space factor code of the period is changed to 1000 because almost all periods in a bibliography will come from abbreviations (except for the periods at the ends of each entry, and those occur at the ends of paragraphs). We add `\everypar={}` just in case `\everypar` was non-empty before, since each entry begins as a nonindented paragraph, and we might as well set `\parindent` to 0pt, though that shouldn't matter.

Once `\makebib` has appeared, we might be using not only `\nopunct` and `\nospace`, but also `\noprepunct` and `\noprespace` and `\noquotes` before various elements of a `\bib... \endbib` entry. Since all of these can precede almost any construction, we want to save ourselves all the agony of section 20.1, and simply have each of these set a flag. `\nopunct` and `\nospace`, which already have definitions, will thus have to be changed by `\makebib`, and, to be on the safe side, we will reset the flags to be false,

```
\def\makebib{\bigbreak
\centerline{\smc Bibliography}
\nobreak\bigskip
\sfcode'\.=1000 \everypar={}\parindent=0pt
\def\nopunct{\nopunct@true}
\def\nospace{\nospace@true}
\nopunct@false\nospace@false
```

As mentioned in section 3, `\lkerns@`, and `\nkerns@` have to be changed for the bibliography macros; these new definitions also have to be added to `\makebib`:

```
\def\makebib{\bigbreak
\centerline{\smc Bibliography}%
\nobreak\bigskip
\sfcode'\.=1000 \everypar={}\parindent=0pt
\def\nopunct{\nopunct@true}%
\def\nospace{\noprepunct@true}%
\nopunct@false\nospace@false
\def\lkerns@{\null\kern-1sp\kern1sp}%
\def\nkerns@{\null\kern-2sp\kern2sp}%
}
```

The `\endmakebib` simply supplies the `\endgroup` that matches the `\begingroup` with which `\makebib` begins:

```
\let\endmakebib=\endgroup
```

Next we add the flags and definitions

```
\newif\ifnoprepunct@
\newif\ifnoprespace@
\newif\ifnoquotes@
\def\noprepunct{\noprepunct@true}
\def\noprespace{\noprespace@true}
\def\noquotes{\noquotes@true}
```

And then we declare all the boxes needed to hold various constructions:

```
\newbox\nobox@
\newbox\keybox@
\newbox\bybox@
\newbox\paperbox@
\newbox\paperinfobox@
\newbox\jourbox@
\newbox\volbox@
\newbox\issuebox@
\newbox\yrbox@
\newbox\pgbox@
\newbox\ppbox@
\newbox\bookbox@
\newbox\inbookbox@
\newbox\bookinfobox@
\newbox\publbox@
\newbox\publaddrbox@
\newbox\edbox@
\newbox\edsbox@
\newbox\langbox@
\newbox\translbox@
\newbox\finalinfobox@
```

29.5. `\bibinfo@`. Each of various constructions within a `\bib... \endbib` entry, like `\paper`, `\jour`, ..., may be preceded by `\nopunct`, ..., `\noquotes`, and we need an easy way to keep track of this information. We will store all the necessary information for a `\bib... \endbib` entry in a control sequence `\bibinfo@` (initially empty at the beginning of each `\bib`). Each field like `\paper`, `\jour`, ..., will first update `\bibinfo@`, based on the current values of the flags `\ifnopunct@`, ..., `\ifnoquotes@`—these values will simply depend on whether `\nopunct`, ..., `\noquotes` occurred before this field (but after the preceding field).

Remember that `\paperbox@`, `\jourbox@`, ..., are simply TEX integers,¹ which we can produce with `\the\paperbox@`, `\the\jourbox@`, If `\paperbox@` happens to have the value 28 (it did the last time I checked), and `\jourbox@` happens to have the value 30, and `\paper` and `\jour` are preceded by any of `\nopunct`, ..., `\noquotes`, then we want `\bibinfo@` to be

$$28, x_1 x_2 x_3 x_4 x_5 30, y_1 y_2 y_3 y_4 y_5$$

[or $30, y_1 y_2 y_3 y_4 y_5 28, x_1 x_2 x_3 x_4 x_5$ if `\jour` appears before `\paper`], where

$$\begin{aligned} x_1 \text{ is } & \begin{cases} 1 & \text{if } \backslash\text{nopunct} \text{ precedes } \backslash\text{paper} \\ 0 & \text{otherwise} \end{cases} \\ x_2 \text{ is } & \begin{cases} 1 & \text{if } \backslash\text{nospace} \text{ precedes } \backslash\text{paper} \\ 0 & \text{otherwise} \end{cases} \\ x_3 \text{ is } & \begin{cases} 1 & \text{if } \backslash\text{noprepunct} \text{ precedes } \backslash\text{paper} \\ 0 & \text{otherwise} \end{cases} \\ x_4 \text{ is } & \begin{cases} 1 & \text{if } \backslash\text{noprespace} \text{ precedes } \backslash\text{paper} \\ 0 & \text{otherwise} \end{cases} \\ x_5 \text{ is } & \begin{cases} 1 & \text{if } \backslash\text{noquotes} \text{ precedes } \backslash\text{paper} \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

and similarly for y_1, \dots, y_5 .

To conserve memory, and allow subsequent macros to work quickly, we want `\bibinfo@` to contain only necessary information; boxes not preceded by any of `\nopunct`, ..., `\noquotes` simply shouldn't show up.

¹ More precisely (*The TeXbook*, page 121), the `\newbox` routine will `\chardef\paperbox@`, etc., and TeX allows `\chardef`'d quantities to be used as integers.

The routine \setbibinfo@#1, used when #1 is \paperbox@, ..., suitably expands \bibinfo@ if necessary, based on the current values of the flags \ifnopunct@, ..., \ifnoquotes@:

```
\def\setbibinfo@#1{\edef\next@{\ifnopunct@1\else0\fi
\ifnospace@1\else0\fi\ifnoprepunct@1\else0\fi
\ifnoprespace@1\else0\fi\ifnoquotes@1\else0\fi}%
\def\nextii@{00000}%
\ifx\next@\nextii@
\else
\xdef\bibinfo@{\bibinfo@\the#1,\next@}%
\fi}
```

Here we first set \next@ to the proper $x_1 \dots x_5$. Then, if this sequence is 00000 (because none of \nopunct, ..., \noquotes appeared) we do nothing; otherwise, we add

```
\the#1,x1x2x3x4x5
```

to \bibinfo@. An \xdef was needed to define \bibinfo@ globally because we will be using \setbibinfo@ within a group, to pass information on beyond that group (section 12).

There is a corresponding \getbibinfo@#1, which will \let\next@=x₁, ..., \let\nextv@=x₅ when '\the#1,' appears in \bibinfo@, or simply \let them =0 otherwise:

```
\def\getbibinfo@#1{%
\ifx\bibinfo@\empty
\let\next@=0\let\nextii@=0\let\nextiii@=0%
\let\nextiv@=0\let\nextv@=0%
```

```

\else
\edef\next@{\def
\noexpand\next@####1\the
#1,####2####3####4####5####6####7\noexpand\next@
{\let\noexpand\next@=####2\let\noexpand\nextii=####3%
\let\noexpand\nextiii=####4\let\noexpand\nextiv=####5%
\let\noexpand\nextv=####6}%
\noexpand\next@\bibinfo@\the#1,00000\noexpand\next@}%
\next@
\fi}

```

By now it should be no problem to unravel this (compare page 82 and Chapter 17). For the sake of speed, we have made a special clause for the (usual) case where `\bibinfo@` is empty. We use the rather rare assignments `\let\next@=0` or `\let\next@=1`, etc., so that later we can use simple `\if` tests,

```
\if\next@1 . . .
```

to test for the value.

Note, finally, that these definitions work because `\paperbox@`, `...`, all represent 2-digit numbers. There would be ambiguity if, for example, one had the value 9 and another had the value 19; fortunately, that can't happen, since `\newbox` only creates numbers greater than 9.

29.6. Additional flags. Next we introduce the flags for determining the treatment of paper and book titles:

```

\newif\ifbookinquotes@
\def\bookinquotes{\bookinquotes@true}
\newif\ifpaperinquotes@
\def\paperinquotes{\paperinquotes@true}
\newif\ifininbook@
\def\ininbook{\ininbook@true}

```

We will also need a flag

```
\newif\ifopenquotes@
```

which we will set true after printing each \paper or \book title that has begun with ‘‘. The next field will then supply the closing ’’ at the right time, and reset the flag to false, using the routine

```
\def\closequotes@{\ifopenquotes@''\openquotes@false\fi}
```

In addition, we need special flags to deal with the possibilities of \morebib, \anotherbib, and \transl. The definitions of these constructions aren't given until section 14, but some discussion is necessary now. These three constructions act almost like an \endbib \bib pair, first printing the information already collected (but without ending the paragraph), and then collecting new information. For efficiency, these constructions and \endbib all call upon a common construction, \endbib@, which does all the work of printing the accumulated information, except that certain flags will have to be set differently for the various constructions.

First we have the flags

```
\newif\ifbeginbib@
\newif\ifendbib@
```

The flag \ifbeginbib@ will, for example, determine whether or not we should print \no and \key information; it will be true when we are setting the first part of a \bib... \endbib entry, but false if we are printing a \morebib part. The flag \ifendbib@ will, for example, determine whether we should print the final period at the end of the entry; it will be false if we are printing the first part of an entry that has a \morebib part to follow, though it will be true when we then set the \morebib part.

We also need the flags

```
\newif\ifprevjour@
\newif\ifprevbook@
```

to pass information to \morebib and \anotherbib about a \jour or \book in the main part.

29.7. \bib. For defining \bib we first introduce a new dimension

```
\newdimen\bibindent@
```

with the default value

```
\bibindent@=20pt
```

in terms of which the hanging indentation for `\bib` items will be specified. This makes it easier to add commands to change this indentation. (The default style doesn't have any such commands, but other styles do.) Then we define

```
\def\bib{\global\let\bibinfo@=\empty
\global\let\translinfo@=\relax
\beginbib@true
\begingroup
\noindent@hangindent\bibindent@ \hangafter1
\bib@}
```

The `\global\let\bibinfo@=\empty` clears out `\bibinfo@` from the previous `\bib.. \endbib`. `\translinfo@`, which will play a role later, also has to be cleared out. Then we set `\ifbeginbib@` to be true, begin a group, start a `\noindent@ed` paragraph with hanging indentation `\bibindent@` after the first line, and call `\bib@`.

`\bib@` has to start by setting `\nobox@`, `\keybox@`, ... , to void boxes. We introduce the abbreviation

```
\def\v@id#1{\setbox#1=\box\voidb@x}
```

and then

```
\def\bib@{\v@id\nobox@ \v@id\keybox@ \v@id\bybox@
\v@id\paperbox@ \v@id\paperinfobox@
\v@id\jourbox@ \v@id\volbox@ \v@id\issuebox@ \v@id\yrbox@
\v@id\pgbox@ \v@id\ppbox@
\v@id\bookbox@ \v@id\inbookbox@ \v@id\bookinfobox@
\v@id\publbox@ \v@id\publaddrbox@
\v@id\edbox@ \v@id\edsbox@
\v@id\langbox@ \v@id\translbox@ \v@id\finalinfobox@
\bgroup}
```

As explained in section 3, the `\bgroup` will immediately be closed by an `\egroup` at the beginning of the `\no` or `\key` or ... that occurs next.

29.8. The basic construction. The basic definition (A) on page 315 is going to be changed in several ways. First of all, when #1 is `\paperbox@`, `\jourbox@`, etc., we want to add

```
\unskip\setbibinfo{#1}\egroup
\setbox#1=\vbox\bgroup . . .
```

so that `\bibinfo@` will have the proper information regarding any `\nopunct`, `\dots`, `\noquotes` that precede `\paper`, `\jour`, `\dots`. Since the information in `\bibinfo@` is recorded using `\xdef`'s, this can be done before the `\egroup`; note also that since each `\nopunct`, `\dots`, `\noquotes` will occur within a field, and thus eventually within some `\bgroup` `\dots` `\egroup`, we don't have to worry about their effects unexpectedly promulgating to another field.

Inside the `\vbox` we will want to have

```
\hsize=\maxdimen
```

as well as

```
\leftskip=0pt \rightskip=0pt
```

(even if, for some strange reason, the bibliography is being set with other values, we want to have `\leftskip` and `\rightskip` be zero inside the preliminary `\vbox`), and also

```
\hbadness=10000 \hfuzz=\maxdimen
```

(so that `Underfull` and `Overfull` boxes will be reported only later, not during the `\vbox` step), all followed by

```
\noindent
```

to preclude any extra indentation in our test `\vbox`.

But if we have an empty field (page 314), in which case our `\vbox` will have width `Opt`,¹ we want to reset our box to `\box\voidb@x`. We can do this with

```
\unskip\setbibinfo@#1\egroup
\def\aftergroup@{\ifdim\wd#1=0pt \setbox#1=\box\voidb@x\fi}
\setbox#1=\vbox\bgroup \aftergroup\aftergroup@ . . .
```

Here the `\aftergroup@` is performed immediately after the `\egroup` from the next field (or from `\endbib`) that eventually matches the `\bgroup`.

In addition, in a few cases we will need to add something extra right after the first `\egroup` (namely, settings for the flags `\ifprevjour` and `\ifprevbook`). So for this general case we define


```
\def\Setnonemptybox@#1#2{%
  \unskip\setbibinfo@#1\egroup#2%
  \def\aftergroup@{\ifdim\wd#1=0pt
    \setbox#1=\box\voidb@x\fi}%
  \setbox#1=\vbox\bgroup \aftergroup\aftergroup@
  \hsize=\maxdimen \leftskip=0pt \rightskip=0pt
  \hbadness=10000 \hfuzz=\maxdimen
  \noindent}

```

and then for the more common case we define

```
\def\setnonemptybox@#1{\Setnonemptybox@#1\relax}

```

 Notice that we used `\noindent@` in the definition of `\bib`, but `\noindent` in the definition of `\Setnonemptybox@`. That is because there is a difference between the situation

(A) `\bib \pagelabel{...} \no . . . \endbib`

where the invisible `\pagelabel` occurs before any field, and something like

(B) `\bib . . . \paper \pagelabel{...} . . . \endbib`

¹ The `\hsize=\maxdimen` within our `\vbox` simply describes the width of paragraphs of text within the `\vbox`, not necessarily the width of the `\vbox` itself.

where the `\pagelabel` occurs within a field.

In case (A), the `\bib` contributes a `\bgroup` (via the `\bib@`), and the `\no` supplies the ending `\egroup`. Within this group, the `\pagelabel{. . .}` has already deleted the final space, but even if it didn't, the `\unskip` before the `\egroup` contributed by `\no` would get rid of it. Consequently, nothing except a `\write` is contributed by this group. But if we tried to be too careful, and replaced the `\bgroup` in `\bib@` with

```
\bgroup\futurelet\next\pretendspace@
```

then, since the invisible `\pagelabel` follows, the results would be wrong:

- (1) First `\pretendspace` would contribute `\hskip-1pt\hskip1pt`.
- (2) Then the `\prevanish@` in `\pagelabel` would set `\saveskip@` to `1pt` and remove the `\hskip1pt`.
- (3) Then the `\postvanish@` in `\pagelabel` would add back the `1pt` and then delete the next space, if any.
- (4) Consequently, the `\unskip\egroup` in `\no` would remove the final `1pt`, leaving an extra `\hskip-1pt` at the beginning.

In case (B), however, the `\noindent` (which eventually calls the combination `\futurelet\next\pretendspace@`) makes everything work out correctly if an invisible element occurs first (followed by something else). Of course, this won't work if a `\pagelabel` appears within an *empty* field, but that doesn't seem worth worrying about!

29.9. \no, \key, . . . Now we are ready to give definitions of `\no`, `\key`, . . . , many of which are quite similar. First we have

```
\def\no{\setnonemptybox@nobox@}
\def\key{\setnonemptybox@keybox@\bf}
\def\by{\setnonemptybox@bybox@}
```

Notice that `\key` specifies `\bf` for the font.

For `\bysame` we simply specify the desired rule, rather than waiting for user input:

```
\def\bysame{\setnonemptybox@bybox@
\leaders\hrule\hskip3em\null}
```

The `\null` is essential, to prevent the `\unskip` from the next field from deleting this `\leaders` glue (compare the `LATEX` Manual, page 225).

The definition of `\paper` is a little more complicated, because we have to begin with ‘ ‘ when `\ifpaperinquotes@` is true, except not when `\noquotes` precedes `\paper`. To find out this latter information, we use

```
\getbibinfo@\paperbox@
```

which will `\let\nextv@=1` if `\noquotes` preceded `\paper`, but will `\let\nextv@=0` otherwise:

```
\def\paper{\setnonemptybox@\paperbox@
\ifpaperinquotes@
\getbibinfo@\paperbox@
\if\nextv@1\else‘ ‘\fi
\else
\it
\fi}
```

Notice that we’ve specified `\it` when `\ifpaperinquotes@` is false (even if `\noquotes` preceded `\paper`).

The next definition reverts to the simple case,

```
\def\paperinfo{\setnonemptybox@\paperinfobox@}
```

but the definition of `\jour` uses `\Setnonemptybox@`, since we need to set `\ifprevjour@` true (for possible use by `\morebib` later on):

```
\def\jour{\Setnonemptybox@\jourbox@\prevjour@true}
```

The next definitions are simple (`\vol` merely adds `\bf` for the font),

```
\def\vol{\setnonemptybox@\volbox@\bf}
\def\issue{\setnonemptybox@\issuebox@}
\def\yr{\setnonemptybox@\yrbox@}
```

But `\toappear` is handled quite specially:

```
\def\toappear{\noprepunct\finalinfo(to appear)}
```

Then come another two simple definitions:

```
\def\pg{\setnonemptybox@pgbox@}
\def\pp{\setnonemptybox@ppbox@}
```

The definition of \book has both the complexities of \paper (involving quotation marks) and \jour (setting a flag),

```
\def\book{\Setnonemptybox@bookbox@prevbook@true
\ifbookinquotes@\getbibinfo@bookbox@
\if\nextv@1\else'\fi
\else
\it
\fi}
```

and the definition of \inbook has yet another clause, because of the flag \ifinbook@:

```
\def\inbook{\Setnonemptybox@inbookbox@prevbook@true
\ifininbook@ in \fi
\ifbookinquotes@\getbibinfo@inbookbox@
\if\nextv@1\else'\fi
\fi}
```

(In this case we don't change fonts even if there are no quotation marks.)

Then comes another bunch of simple definitions,

```
\def\bookinfo{\setnonemptybox@bookinfobox@}
\def\publ{\setnonemptybox@publbox@}
\def\publaddr{\setnonemptybox@publaddrbox@}
\def\ed{\setnonemptybox@edbox@}
\def\eds{\setnonemptybox@edsbox@}
\def\lang{\setnonemptybox@langbox@}
\def\finalinfo{\setnonemptybox@finalinfobox@}
```

This leaves only `\transl`, `\morebib`, and `\anotherbib`, all of which are treated somewhat similarly—it will be easiest to understand their definitions later (section 14) after examining how `\endbib` puts all the information together.

29.10. Manipulating the `\vbox`'es. At the end of a `\bib... \endbib` entry, we will have stored all our information in various `\vbox`'es, and we now have to get information out of each such box, #1, with a construction `\getbox@#1`. Most of the time we could simply use

```
\def\getbox@#1{\setbox0=\vbox{\unvbox#1
\setbox0=\lastbox
\global\setbox1=\hbox{\unhbox0 \unskip\unskip\unpenalty}}
\unhbox1 }
```

Here the inner `\box0` (`\lastbox`) is the final line of this one line paragraph (having `\hsize=\maxdimen`), at the end of which we have (see *The T_EXbook*, page 100),

```
\penalty 10000 (\parfillskip glue) (\rightskip glue)
```

which we delete with the `\unskip\unskip\unpenalty`.

But since line breaking commands may have occurred (either in the middle of the field, or at the end), our `\vbox` may have several `\hbox`'es, separated by glue and possibly penalties (`\clubpenalty`, etc.), and we may have to use a `\loop` to discover all these boxes. We add `\vskip-10000 pt` at the beginning as a marker, the assumption that such a `\vskip` won't be inserted otherwise.


```
\def\getbox@#1{\setbox0=\vbox{\vskip-10000pt
\unvbox#1%
\setbox0=\lastbox
\global\setbox1=\hbox{\unhbox0 \unskip\unskip\unpenalty}
\ifdim\lastskip=-10000pt
\else
\loop
\ifdim\lastskip=-10000 pt
```

```

\else
\unskip\unpenalty\setbox0=\lastbox
\global\setbox1=\hbox{\unhbox0 \unhbox1}%
\repeat
\fi}%
\unhbox1 }

```

Notice that we use the test `\ifdim\lastskip=-10000pt` before entering the `\loop`, even though it is also used within the `\loop`; this allows us to bypass the `\loop` completely if it is unnecessary.

 The combination `\setbox0=\lastbox` occurs rather often, so right before the definition of `\getbox@` we have

```
\def\setboxz1@{\setbox\z@\lastbox}
```

and then we use `\setboxz1@` throughout. However, for the sake of readability, we will always simply print `\setbox0=\lastbox`.

29.11. Line breaking commands. Next we have to deal with the problem of a line breaking command appearing at the end of one of our fields, since the appropriate penalties must be inserted *after* the punctuation that is normally added after the field. These line breaking commands produce

```
\null\kern-n sp \kern n sp
```

at the end of the horizontal list produced by `\getbox@`. More specifically,

(1) `\nolinebreak` produces

```
\penalty10000 ⟨possible glue⟩ \null\kern-1sp\kern1sp
```

The `⟨possible glue⟩` comes about because `\nolinebreak` remembers the glue before it, and inserts it after the `\penalty10000`, so that this glue does *not* disappear. Of course, it is unlikely that any one would want to have such a non-disappearing space at the end of a field, but we might as well allow for the possibility.

(2) Similarly, `\allowlinebreak` produces

```
\penalty0 <possible glue> \null\kern-1sp\kern1sp
```

(3) `\linebreak` produces

```
\penalty-10000 <Opt glue> \null\kern-1sp\kern1sp
```

Here the `\null\kern-1sp\kern1sp` comes from the last `\hbox` in the `\vbox`, while the `\penalty-10000 <Opt glue>` comes from the previous `\hbox`, the `<Opt glue>` being the `\rightskip` glue, and the `\penalty-10000` the penalty from `\linebreak`.

(4) `\newline` produces

```
\null \hfil \penalty-10000 <Opt> \null\kern-2sp\kern2sp
```

Now the `\null\hfil\penalty-10000 <Opt glue>` comes from the next to last `\hbox`, the `<Opt glue>` again being the `\rightskip` glue, and the `\null\hfil\penalty-10000` coming from the `\newline`.

Since such small `\kern`'s were presumably not inserted by the user, their presence indicates that one of the line breaking commands occurred; the value of the `\kern`, together with the `\penalty` before the `\null`, allows us to distinguish the four cases.

We introduce the routine `\adjustpunct@#1`, where `#1` is normally some punctuation symbol, which will find such information, delete all the `\kern`'s, `\penalty`'s, etc., add the punctuation `#1`, and then put back the proper `\penalty`. We begin with

```
\def\adjustpunct@#1{\count@=\lastkern
```

so that `\count@` is the number of scaled points in `\lastkern`. This will normally either be 0, because there is no `\kern`, or if a `\kern` was inserted by the user, it will be something reasonably large or large negative, presumably `>2` or `<-2`. In such cases, we just want to add the punctuation `#1`, except that this must be followed by `\closequotes@`, in case `'` needs to be added after

the punctuation, because the previous field began with ‘‘:

```
\def\adjustpunct@#1{\count@=\lastkern
\ifnum\count@=0 #1\closequotes@\else
\ifnum\count@>2 #1\closequotes@\else
\ifnum\count@<-2 #1\closequotes@\else
. . .
```

In the remaining, more interesting case, we use

```
\unkern\unkern\setbox0=\lastbox
```

to get rid of the

```
\null \kern-n sp\kern n sp
```

Then we save the previous glue in `\skip@` and remove it:

```
\skip@=\lastskip \unskip
```

Now we’ve reached the point where the original `\penalty` is revealed; we store it in `\count@@` and remove it:

```
\count@@=\lastpenalty \unpenalty
```

Moreover, in the case of `\newline` (`\count@` is 2), we also get rid of the `\null\hfil`:

```
\ifnum\count@=2 \unskip \setbox0=\lastbox \fi
```

Now we can add the punctuation `#1`, followed by `\closequotes@`; if `\skip@` is non-zero, because of the (possible glue) in the case of `\nolinebreak` or `\allowlinebreak`, we add this first:

```
\ifdim\skip@=0pt \else \hskip\skip@ \fi
#1\closequotes@
```

In the case of `\newline` we then add back a `\null\hfill`, and in all cases we add back the original `\penalty`, now stored in `\count@`:

```

\def\adjustpunct@#1{\count@=\lastkern
  \ifnum\count@=0 #1\closequotes@\else
  \ifnum\count@>2 #1\closequotes@\else
  \ifnum\count@<-2 #1\closequotes@\else
  \unkern\unkern\setbox0=\lastbox
  \skip@=\lastskip \unskip
  \count@=\lastpenalty\unpenalty
  \ifnum\count@=2 \unskip\setbox0=\lastbox\fi
  \ifdim\skip@=0pt \else \hskip\skip@ \fi
  #1\closequotes@
  \ifnum\count@=2 \null\hfill\fi
  \penalty\count@
\fi\fi\fi}

```

29.12. Adding punctuation before a field. The routine `\adjustpunct@` figures prominently in the routine

```
\prepunct@#1#2
```

which adds #1, some sort of punctuation, before the field corresponding to #2, which is `\paperbox@`, . . .

Before we consider this routine, we need to reconsider the information in `\bibinfo@`, which we use with the routine `\getbibinfo@` to funnel information from `\bibinfo@` into `\next@`, . . . , `\nextv@`. Note that the information derived from the value of `\next@` and `\nextii@`, i.e., whether or not `\nopunct` or `\nospace` preceded the field, is not used when printing the field; this information has to be used by the *next* field, to decide whether to print the finishing punctuation and space, before printing the new material from that next field. Consequently, the information from the value of `\next@` and `\nextii@` has to be passed along to the next field. We will pass this information along by setting the values of the flags `\ifnopunct@` and `\ifnospace@` once again, thus providing them with a dual role.

Now our definition of `\prepunct@` begins

```

\def\prepunct@#1#2{\getbibinfo@#2%
% first part
\ifnopunct@
\else
\if\nextiii@0\adjustpunct@#1\fi
\fi
\closequotes@
\ifnospace@
\else
\if\nextiv@0\space\else\fi
\fi
% second part
\nopunct@false \nospace@false
\if\next@1\nopunct@true\fi
\if\nextii@1\nospace@true\fi

```

Before examining the first part of this code, consider the second part: this sets `\ifnopunct@` to be true if `\nopunct@` preceded the field and `\ifnospace@` to be true if `\nospace@` preceded the field.

The first part of the code should now make more sense, when we remember that those same assignments will have been made by the field which precedes this field: The test `\ifnopunct@` is true if the previous field is not supposed to have punctuation at the end, and `\nextiii@` will be 1 (rather than 0) if the current field was followed by `\noprepunct`. In either of these cases, we don't supply any punctuation; otherwise, we use `\adjustpunct@#1` to supply the punctuation #1. Even when no punctuation is supplied, we need `\closequotes@`, in case the previous field set `\ifopenquotes@` true (if `\adjustpunct@` was used, the `\closequotes` within it has already set `\ifopenquotes@` to be false [page 321], so we won't be adding closing quotes twice). Space following the punctuation is handled similarly, using the flags `\ifnospace@` and the value of `\nextiv@`.

```

\def\prepunct@#1#2{\getbibinfo@#2%
\ifnopunct@

```

```

\else
  \if\nextiii@0\adjustpunct@#1\fi
\fi
\closequotes@
\ifnospace@
\else
  \if\nextiv@0\space\else\fi
\fi
\nopunct@false\nospace@false
\if\next@1\nopunct@true\fi
\if\nextii@1\nospace@true\fi}

```

`\prepunct@#1#2` is usually followed immediately by `\getbox@#2`, so we introduce an abbreviation for that combination:

```
\def\ppunbox@#1{\prepunct@{#1}#2\getbox@#2}
```

29.13. `\endbib@`. Now we're ready to actually typeset all the information acquired by a `\bib... \endbib` entry. First we

```
\let\semicolon@=;
```

so that we can use `\semicolon@` in place of `;` everywhere. This is done solely for the benefit of French styles (compare section 3.10), which can then redefine `\semicolon@` (presumably as something like `⸗`).

The final command `\endbib` is defined in terms of `\endbib@`, which does most of the work. We begin with

```

\def\endbib@{%
  \ifbeginbib@
    \ifvoid\nobox@
      \ifvoid\keybox@\else
        \hbox to\bibindent@{[\getbox@\keybox@]\hss}\fi
      \else
        \hbox to\bibindent@{\hss\getbox@\nobox@. }
      \fi
  . . .

```


Thus, when \ifbeginbib@ is true (which will usually be the case, unless we are printing remaining information from \morebib, \anotherbib, or \transl), if \box\nobox@ isn't void we start our \noindent'ed paragraph with

```
\hbox to\bibindent@{\hss\getbox@\nobox@. }
```

a box of width \bibindent@, which is also the amount of hanging indentation we will be using, containing the information in \box\nobox@. This information (from \getbox@\nobox@) is followed by a period and a space, and preceded by \hss (rather than \hfil, just in case the number is too long). Because the \ifvoid test always tells us whether or not a particular field has been set, we don't need any flags for this. Note also that \getbox@#1 uses \unvbox#1 (page 328), so that once we print the information for a field, the box containing that information has become void again.

If \box\nobox@ is void, we instead use the information in \box\keybox@, if that isn't void. In this case we surround the information with brackets, and have it start at the left margin, with the additional space at the right.

In the other case, where \ifbeginbib@ is false, we use

```
\else
  \nopunct@true
  \ifvoid\bybox@\else\getbox@\bybox@\fi
\fi
```

We first set \ifnopunct@ to be true because we don't want the first field of our \morebib or \anotherbib or \transl portion to add any punctuation to the end of the previous field, which will already have been supplied with a closing ; (see page 338ff.). Normally \box\bybox@ will be void for this subsequent portion, but \by could conceivably be used after \anotherbib, so we add that in.

After this preamble, we start to add the other various elements in the required order. Although \transl isn't defined until section 14, it does end up storing its information in \box\translbox@, just like other fields; if this box is nonvoid, then we must be setting this second portion of the entry, and this information should come first, so we now add

```
\ifvoid\translbox@\else\ppunbox@,\translbox@\fi
```

to print the information.

Next comes

```

\ifvoid\paperbox@
\else
\ppunbox@,\paperbox@
\ifpaperinquotes@
\if\nextv@1\else\openquotes@true\fi
\fi
\fi

```

Thus, if `\box\paperbox@` isn't void we add the information in it, with appropriate punctuation and spacing before it. The `\ppunbox@` routine has already used `\getbibinfo@\paperbox@` to get information from `\bibinfo@`. When the flag `\ifpaperinquotes@` is true (so that we are enclosing paper titles in quotation marks), then the paper title normally has begun with ‘ ‘ and we want to set `\ifopenquotes@` to be true, so that the next routine will know to add the ’ ’; however, we don't do this when the `\bibinfo@` has caused `\nextv@` to be 1, since this indicates that a `\noquotes` preceded the `\paper`, in which case the opening ‘ ‘ will not have been included in the paper title.

After the paper title, we add any `\paperinfo`:

```

\ifvoid\paperinfobox@\else\ppunbox@,\paperinfobox@\fi

```

Then comes the information in `\jourbox@`, `\volbox@`, `\issuebox@`, Information from `\volbox@`, ... , should be added only if there was

- (a) information in `\jourbox@`;
- (b) or in the case of `\morebib`, information in a previous `\jourbox@`.

In the latter case, `\ifprevjour@` will have been set true, so we use

```

\test@false
\ifvoid\jourbox@\else\test@true\ppunbox@,\jourbox@\fi
\ifprevjour@\test@true\fi
\iftest@
<material for \vol, \issue, \yr, \pp, \pg>
\fi

```

For \volbox@ we use

```
\ifvoid\volbox@\else\ppunbox@\relax\volbox@\fi
```

since there is no punctuation before the information in \volbox@. Similarly,

```
\ifvoid\issuebox@
\else\prepunct@\relax\issuebox@ no.~\getbox@\issuebox@\fi
\ifvoid\yrbox@\else\prepunct@\relax\yrbox@
(\getbox@\yrbox@)\fi
\ifvoid\ppbox@\else\ppunbox@,\ppbox@\fi
\ifvoid\pgbox@\else\prepunct@,\pgbox@
p.~\getbox@\pgbox@\fi
```

puts “no. ” before the \issue, which has no punctuation before it, encloses the \yr in parentheses, with no punctuation before it, and adds “p. ” before \pg.

Next comes information for a book. As with \jour, succeeding information will be printed only if there is a \book or \inbook or when \ifprevbook@ is true. The flag \ifbookinquotes@ also requires steps similar to those used for \paper:

```
\test@false
\ifvoid\bookbox@\else
\test@true\ppunbox@,\bookbox@
\ifbookinquotes@\if\nextv@1\else\openquotes@true\fi\fi
\fi
\ifvoid\inbookbox@\else
\test@true\ppunbox@,\inbookbox@
\ifbookinquotes@\if\nextv@1\else\openquotes@true\fi\fi
\fi
\ifprevbook@\test@true\fi
\iftest@
(subsidiary fields for \book)
\fi
```

The subsidiary fields include \edbox@, \edsbox@, \bookinfo@, \publbox@, \publaddrbox@, and once again \yrbox@, \ppbox@ and

`\pgbox@`, since the latter three can be with `\paper` as well as with `\book`. There's nothing here that we haven't already considered:

```

\ifvoid\edbox@\else\prepunct@\relax\edbox@
  (\getbox@\edbox@, ed.)\fi
\ifvoid\edsbox@\else\prepunct@\relax\edsbox@
  (\getbox@\edsbox@, eds.)\fi
\ifvoid\bookinfo@@\else\ppunbox@,\bookinfo@\fi
\ifvoid\publbox@\else\ppunbox@,\publbox@\fi
\ifvoid\publaddrbox@\else\ppunbox@,\publaddrbox@\fi
\ifvoid\yrbox@\else\ppunbox@,\yrbox@\fi
\ifvoid\ppbox@\else\prepunct@,\ppbox@
  pp.~\getbox@\ppbox@\fi
\ifvoid\pgbox@\else\prepunct@,\pgbox@
  p.~\getbox@\pgbox@\fi

```

Then come the finishing touches. At this point the flag `\ifendbib@` will be important. It will have been set true if we are finishing off the whole `\bib... \endbib` entry, but it will be false if we are simply printing the first portion of an entry, with a further portion from `\morebib` or `\anotherbib` or `\transl` to follow. The possibility of a `\finalinfo` field will also play a role. We begin with the case where there was no `\finalinfo` field:

```

\ifvoid\finalinfo@
  \ifendbib@
    \ifnopunct@\else.\closequotes@\fi
  \else
    \ifvoid\langbox@\else\space(\getbox@\langbox@)\fi
    /\semicolon@\closequotes@

```

...

In other words, if we are truly at the end, we print a period, unless a `\nopunct` preceded the previous field, which will be indicated by `\ifnopunct@` being true; the `\closequotes@` is added just in case the previous field happened to be a `\jour` or `\book` in quotes.

Otherwise (`\morebib` or `\anotherbib` or `\transl` information follows), we add any `\lang` field information in parentheses, and then a semicolon (with

the italic correction `\/` before it, in case the previous field was set in a slanted font), again calling on `\closequotes@` to supply any necessary closing quotes. Notice that no space has been added after the semicolon. The `\prepunct@` or `\ppunbox` for the next field will add in that space, even though the extra punctuation will not be added (because we set `\ifnopunct@` to be true when `\ifbeginbib@` is false [page 335]).

For the other case, where there is a `\finalinfo` field, we use

```
\else
\ifendbib@
\ppunbox@{.\spacefactor3000\relax}\finalinfobox@
\ifnopunct@\else.\fi
\else
\ppunbox@,\finalinfobox@\//\semicolon@\fi
\fi
```

So, if we are at the very end, we add a period before the `\finalinfo` field (unless a `\nopunct` or `\nopreunct` tells us not to), and then a final period (unless a `\nopunct` preceded the `\finalinfo`). After the period before the `\finalinfo` material we change the `\spacefactor` to 3000, since that period is not an abbreviation. If we're not at the very end, so that the `\finalinfo` applies only to the first portion of the entry, then we want a comma before this information, and a semicolon (with `\/` before it).

Finally, we add

```
\ifvoid\langbox@\else\space(\getbox@\langbox@)\fi
```

at the very end, to take care of a `\lang` field that has to go after everything else.

Our whole definition thus reads:

```
\def\endbib@{%
\ifbeginbib@
\ifvoid\nobox@
\ifvoid\keybox@\else
\hbox to\bibindent@{[\getbox@\keybox@]\hss}\fi
```

```

\else
  \hbox to\bibindent@{\hss\getbox@\nobox. }\fi
\ifvoid\bybox@\else\getbox@\bybox@\fi
\else
  \nopunct@true
  \ifvoid\bybox@\else\ppunbox@\relax\bybox@\fi
\fi
\ifvoid\translbox@\else\ppunbox@,\translbox@\fi
\ifvoid\paperbox@\else\ppunbox@,\paperbox@\ifpaperinquotes@
  \if\nextv@1\else\openquotes@true\fi\fi\fi
\ifvoid\paperinfobox@\else\ppunbox@,\paperinfobox@\fi
\test@false
\ifvoid\jourbox@\else\test@true\ppunbox@,\jourbox@\fi
\ifprevjour@\test@true\fi
\iftest@
  \ifvoid\volbox@\else\ppunbox@\relax\volbox@\fi
  \ifvoid\issuebox@
    \else\prepunct@\relax\issuebox@ no.~\getbox@\issuebox@\fi
  \ifvoid\yrbox@\else\prepunct@\relax\yrbox@
    (\getbox@\yrbox@)\fi
  \ifvoid\ppbox@\else\ppunbox@,\ppbox@\fi
  \ifvoid\pgbox@\else\prepunct@,\pgbox@ p.~\getbox@\pgbox@\fi
\fi
\test@false
\ifvoid\bookbox@\else\test@true\ppunbox@,\bookbox@
  \ifbookinquotes@\if\nextv@1\else\openquotes@true\fi\fi
\fi
\ifvoid\inbookbox@\else\test@true\ppunbox@,\inbookbox@
  \ifbookinquotes@\if\nextv@1\else\openquotes@true\fi\fi
\fi
\ifprevbook@\test@true\fi
\iftest@
  \ifvoid\edbox@\else\prepunct@\relax\edbox@
    (\getbox@\edbox@, ed.)\fi
  \ifvoid\edsbox@\else\prepunct@\relax\edsbox@
    (\getbox@\edsbox@, eds.)\fi
  \ifvoid\bookinfobox@\else\ppunbox@,\bookinfobox@\fi

```

```

\ifvoid\publbox@\else\ppunbox@,\publbox@\fi
\ifvoid\publaddrbox@\else\ppunbox@,\publaddrbox@\fi
\ifvoid\yrbox@\else\ppunbox@,\yrbox@\fi
\ifvoid\ppbox@\else\prepunct@,\ppbox@ pp.~\getbox@\ppbox@\fi
\ifvoid\pgbox@\else\prepunct@,\pgbox@ p.~\getbox@\pgbox@\fi
\fi
\ifvoid\finalinfobox@
\ifendbib@
\ifnopunct@\else.\closequotes@\fi
\else
\ifvoid\langbox@\else\space(\getbox@\langbox@)\fi
\semicolon@\closequotes@
\fi
\else
\ifendbib@
\ppunbox@{.\spacefactor3000\relax}\finalinfobox@
\ifnopunct@\else.\fi
\else
\ppunbox@,\finalinfobox@\semicolon@\space\fi
\fi
\ifvoid\langbox@\else\space(\getbox@\langbox@)\fi
}

```

29.14. \endbib, \morebib, \anotherbib, and \transl. The construction \endbib begins with

```
\unskip\egroup
```

just like all other fields, and then it calls \endbib@ with the flag \ifendbib@ set true (\ifbeginbib@ has already been set true by \bib), finishing up with a \par and then an \endgroup to match the \begingroup supplied by the original \bib:

```

\def\endbib{\unskip\egroup
\endbib@true\endbib@\par\endgroup}

```

`\morebib` uses an `\endbib@` to set the material so far, just like `\endbib`, except that we set `\ifendbib@` to be false:

```
\def\morebib{\unskip\egroup
\endbib@false\endbib@}
```

And then we want to call `\bib@` again, to deal with the following material, except that we also need to reset `\bibinfo@` to be empty and `\ifbeginbib@` to be false:

```
\def\morebib{\unskip\egroup
\endbib@false\endbib@
\global\let\bibinfo@=\empty
\beginbib@false
\bib@}
```

And `\anotherbib` is almost the same, except that we also set `\ifprevjour@` and `\ifprevbook@` to be false before the `\bib@`, so that no previous material will be “remembered”:

```
\def\anotherbib{\unskip\egroup
\endbib@false\endbib@
\global\let\bibinfo@=\empty
\beginbib@false
\prevjour@false\prevbook@false
\bib@}
```

Finally, `\transl` is something like a cross between `\morebib` and a more standard field. We want to begin with something like

```
\unskip\egroup
\endbib@false\endbib@}
```

in order to print the previous information.

Then we want to call \bib@ again, after first setting \ifbeginbib@ to be false, as with \morebib. Now we start with

```
\beginbib@false
\bib@
\egroup
\def\aftergroup@{...}
\egroup\setbox\translbox@=\vbox\bgroup
\aftergroup\aftergroup@ \hsize=\maxdimen . . .
\noindent@}
```

so that we will be putting the following field into \box\translbox@ in essentially the same way that \paper, \jour, etc., put their fields into boxes, using \setnonemptybox@. The one thing we are missing from \setnonemptybox@ is the procedure for storing information about any \nopunct, ..., \noquotes that precedes the \transl. This information should be stored in \bibinfo@, except that any other information in \bibinfo@ should be emptied out, ready to be refilled, if necessary, with current information for the fields following \transl.

To do this, we first store the information in \translinfo@, before the \egroup,

```
\def\transl{\unskip
\xdef\translinfo@{\the\translbox@,\ifnopunct@1\else0\fi
\ifnospace@1\else0\fi\ifnoprepunct@1\else0\fi
\ifnoprespace@1\else0\fi0}
```

(we simply use 0 as the last value, without bothering about the value of \ifnoquotes@, since the value is never used in setting the translation information anyway), and then we set \bibinfo@ to be \translinfo@ before calling \bib@:

```
\def\transl{\unskip
\xdef\translinfo@{\the\translbox@,\ifnopunct@1\else0\fi
\ifnospace@1\else0\fi\ifnoprepunct@1\else0\fi
\ifnoprespace@1\else0\fi0}%
```

```
\egroup
\endbib@false
\endbib@
\global\let\bibinfo@=\translinfo@
\beginbib@false
\bib@
\egroup
\def\aftergroup@{\ifdim\wd\translbox@=0
  \setbox\translbox@=\box\voidb@x\fi}%
\setbox\translbox@=\vbox\bgroup
\aftergroup\aftergroup@
\hsize=\maxdimen \leftskip=Opt \rightskip=Opt
\hbadness=10000 \hfuzz=\maxdimen
\noindent@}
```

And that jolly well finishes off the bibliography macros!

Chapter 30. Interfacing with BIBTEX

It turns out that we can easily prepare an `.aux` file on which BIBTEX can operate, so that a `LATEX` file can use the resulting `.bbl` file. The control sequence

```
\UseBibTeX
```

will set things up for this. When `\UseBibTeX` is specified (preferably close to the beginning of the document), a

```
\bibliographystyle{...}
```

line should also appear somewhere.

Furthermore, when `\UseBibTeX` is specified, `\cite` will essentially function just as in `LATEX`, rather than as in the previous chapter, except that we will continue to use `(...)` for an optional argument rather than `[...]`. As in `LATEX`, a citation of the form

```
\cite{(key)1,(key)2,...}
```

had better not have spaces after the commas, because this multiple argument is going to be passed directly to the `.aux` file, on which BIBTEX operates, and BIBTEX will insist that no space appears. (Of course, spaces will be made to appear after the commas in the output.)

Finally, when `\UseBibTeX` is specified, instead of a

```
\makebib  
.  
.  
.  
\endmakebib
```

region, a line

```
\bibliography{(source list)1,(source list)2,...}
```

should appear, at the appropriate place. In this case, no specific references should be given, either in L^AT_EX or in L^AM_S-T_EX format. The references will be supplied by the .bbl file that will eventually be made, and which the \bibliography command will read in, and any additions or changes would be made directly in that file.

30.1. \UseBibTeX. We first declare new streams for writing the .aux file, and reading the .bbl file:

```
\newwrite\auxwrite@
\newread\bbl@
```

Now \UseBibTeX must first open up an .aux file,

```
\immediate\openout\auxwrite@=\jobname.aux
```

and then redefine \cite. We will simply

```
\let\cite=\BTcite@
```

where \BTcite@ is defined separately afterwards. That allows a style file to redefine aspects of \BTcite@ that deal with the material actually printed.

\UseBibTeX will also define \nocite,

```
\def\nocite#1{\immediate\write\auxwrite@{\string\citation{#1}}
```

and \bibliographystyle,

```
\def\bibliographystyle#1{\immediate\write\auxwrite@
{\string\bibstyle{#1}}}
```

Finally, it must define \bibliography, and this is somewhat more complex. To begin with, we want to

```
\def\bibliography@W{Bibliography}
```

so that \newword\bibliography can be used. Then we start with

```
\def\bibliography#1{\immediate\write\auxwrite@
  {\string\bibdata{#1}}
  . . .
```

Then we have to see whether a .bbl file exists. If it doesn't, we simply give a message to that effect, but if the .bbl file does exist, we read in a special macro file bibtex.tex before we \input the .bbl file, since the bibtex.tex macros will make this a file that we can process,

```
\def\bibliography#1{\immediate\write\auxwrite@
  {\string\bibdata{#1}}%
  \immediate\openin\bbl@=\jobname.bbl
  \ifeof\bbl@
    \W@{No .bbl file}%
  \else
    \immediate\closein\bbl@
    \begingroup
    \input bibtex
    \input\jobname.bbl
    \endgroup
  \fi}
```

Putting this all together, we have

```
\def\UseBibTeX{\immediate\openout\auxwrite@=\jobname.aux
  \let\cite=\BTcite@
  \def\nocite##1{\immediate\write\auxwrite@
    {\noexpand\citation{##1}}}%
  \def\bibliographystyle##1{\immediate\write\auxwrite@
    {\string\bibstyle{##1}}}%
  \def\bibliography@W{Bibliography}%
  \def\bibliography##1{%
    \immediate\write\auxwrite@{\string\bibdata{##1}}%
    \immediate\openin\bbl@=\jobname.bbl
    \ifeof\bbl@
      \W@{No .bbl file}%
```

```

\else
\immediate\closein\bb1@
\begingroup
\input bibtex
\input\jobname.bbl
\endgroup
\fi}%
}

```

We use `\string's` rather than `\noexpand's` (section 3.4) so that no extra spaces will appear in the `.aux` file, since BIBTEX doesn't seem to like that!

The definition of `\BTcite@` (compare the definition of `\cite` in the previous chapter) leaves the parsing for commas to `\BTcite@@... \BTcite@@`:

```

\def\BTcite@{%
\def\nextii@(#1)##2{%
{\rm[]\BTcite@@##2,\BTcite@@{\rm,}##1\}/{\rm[]}%
\immediate\write\auxwrite@{\string\citation{##2}}}%
\def\nextiii@##1{{\rm[]\BTcite@@##1,\BTcite@@\}/{\rm[]}%
\immediate\write\auxwrite@{\string\citation{##1}}}%
\def\next@{\ifx\next(\expandafter\nextii@
\else\expandafter\nextiii@\fi}%
\futurelet\next\next@}%

```

Notice that `\nextii@`, used in the case of an optional argument (...), explicitly inserts a comma and space (in the `\rm` font—see the small print note on page 349). `\BTcite@@... \BTcite@@` will be defined so that the comma and space is *not* added after the final part of the argument.

The definition of `\BTcite@@` uses standard parsing methods. For each individual `{key}`, combination in the argument `#1`, we will call `\BTcite@@@{key}` followed by `{\rm,}`, *except for the last such {key}*, where the comma will simply be dropped:

```

\def\BTcite@@#1,{\BTcite@@@{#1}\futurelet\next\BTcite@@@}
\def\BTcite@@@{\ifx\next\BTcite@@
\expandafter\eat@\else{\rm, }\expandafter\BTcite@@\fi}

```

Here `\BTcite@@@` will basically be equivalent to `\ref` (compare the definition on page 92):


```
\catcode'\~=11
\def\BTcite@@@#1{\nolabel@\cite{#1}\relax
\def\nextii@##1~##2\nextii@{##1}%
\csL@{#1}\expandafter\nextii@\Next@\nextii@\fi}
\catcode'\~=active
```

The `\nolabel@\cite{#1}\relax` means that we will initially get an error message or warning message

No `\label` found for `(key)`.

which is reasonable, since a `\cite` is essentially like a `\ref`; the label will be created when we have a `\bibitem` from the `.bbl` file. As in L^AT_EX, three passes will normally be required:

- (1) First each `\cite` creates a `\citation` line in the `.aux` file.
- (2) Then BIB_TE_X creates a `.bbl` file, with a `\bibitem` for each key cited. When we T_EX the file again, the `\bibitem`'s in the `.bbl` file will be treated like `\label`'s, and write appropriate information (forward references) to the `.lax` file.
- (3) When we T_EX the file yet again, these references will finally be inserted.

 Because BIB_TE_X makes `\cite{...}` work like `\ref{...}`, it is then not permissible to say things like

`\cite{\bf ...}`

Once can type

By `{\bf\cite{(key)1,(key)2}` we have ...

to print something like

By [3, 4] we have ...

(`\rm` has been explicitly specified for the brackets and comma definitions), but a change to the style file should be used if all citations are supposed to be in `\bf`. It is even possible to type something like

By `{\bf\cite(\it page~123){key}}` we have ...

to print something like

By [3, page 123] we have ...

(there are no restrictions on the material occurring in the optional argument). In the definition of `\BTcite@` we typed `{##1\}` in the definition of `\nextii@` to allow for the fact that font change instructions might occur in this argument.

bo If we have a multi-key `\cite`,

`\cite{key}_1, {key}_2, ...}`

then `{key}_1` will give a warning message like

```
Warning: No \label found for {key}_1.
1.35 ... \cite{key}_1}
```

while `{key}_2` will give a warning message like

```
Warning: No \label found for {key}_2.
1.35 ... \cite{key}_2}
```

Although it would be nicer if we got a single message involving

```
1.35 ... \cite{key}_1, {key}_2, ...}
```

this doesn't really seem worth worrying about.

The `.bbl` file will have `\bibitem`'s, but the definition of `\bibitem` is found only in the file `bibtex.tex`. However, we will state

```
\let\newblock=\relax
```

even though `\newblock` occurs only in the `.bbl` file that is read in after the `bibtex.tex` file; the reason for this is that different styles might

want to change the definition of `\newblock`. (Many L^AT_EX style files define `\newblock` to be `\hskip.11em plus.33em minus.07em`, but the default L^AM_S-T_EX style leaves no extra space between various parts of a bibliographical entry.)

Similarly, we define a default routine

```
\def\beginthebibliography@#1{\rm
  \setbox0\hbox{#1\ } \bibindent@=\wd0
  \bigbreak
  \centerline{\smc\bibliography@W}%
  \nobreak\medskip
  \sfcode'\.=1000 \everypar{\}\parindent=0pt}
```

which will be used by the

```
\begin{thebibliography}
```

that occurs in the `.bbl` file; style files can redefine this, to produce different sorts of formatting for the information in the `.bbl` file.

30.2. The bibtex.tex file. The file `bibtex.tex` is read in right before the `.bbl` file; its purpose is to make sense out of the various L^AT_EX macros that are going to appear.

The file begins with

```
%\input cd.tox
%\input islands.tox
%\input bib.tox
%\input alignat.tox
%\input lists.tox
%\input cardord.tox
%\input anynum.tox
%\input dblacc.tox
%\input literal.tox
```

(As in section 27.2, we are using double horizontal lines for code that is in the subsidiary file `bibtex.tex`, rather than in `lamstex.tex` itself). Although

these lines are commented out, it might be wise to uncomment them, if it turns out that adding the extra material from `bibtex.tex` unduly strains the implementation of TeX that is being used.

This is followed by

```
\catcode'\@=11
\let\alloc@=\alloc@@
```

since we will be using a `\newcount` and `\newdimen` (compare page 38).

Next we have to deal with the fact that the `.bbl` file usually begins with things like

```
\newcommand{\noopsort}[1]{-}
\newcommand{\printfirst}[2]{#1}
```

and perhaps there will be `\newcommand`'s for control sequences without arguments also. So we have to interpret `\newcommand` correctly.

For a situation like

```
(A) \newcommand{\foo}{bar}
```

where a control sequence `\foo` without arguments is being defined, we can simply let `\newcommand#1` mean `\define#1`. But the general case is more complicated.

First we define `\args@` so that `\args@1` makes `\toks@` be the token list `##1`, and `\args@2` makes `\toks@` be the token list `##1##2`, etc. (we want the double `##`'s because `\toks@` will then be used within an `\edef`). The definition looks pretty strange,

```
\def\args@#1{\count@=1 \toks@={}%
\loop
\edef\next@{\toks@={\the\toks@#####\number\count@}}%
\next@
\ifnum\count@<#1 \advance\count@ by 1
\repeat}
```

because we have the *octuplet* `#####`, even though we want to end up with only a pair `##`. The reason we need this octuplet is that the `\loop... \repeat`

construction (section 3.9) defines `\iterate` in terms of ‘...’; when this definition is made, the octuplet `#####` then coalesces into a quadruplet `####` (unfortunately, even the redefinition of `\loop... \repeat` doesn’t get around *this* problem), and then the `\edef` turns this quadruplet into a pair `##`.

Now that `\args@` is defined, a situation like

```
\newcommand{\foo}[2]{...}
```

can be taken care of with

```
(B) \def\newcommand#1[#2]{\args@#2
     \edef\next@{\noexpand\define\noexpand#1\the\toks@}
     \next@}
```

For example,

```
\newcommand{\switchargs}[2]{#2#1}
```

first uses `\args@2` to make `\toks@` be the token list `##1##2`. Then the `\edef` makes `\next@` be

```
\define\switch#1#2
```

so we end up with

```
\define\switch#1#2{#2#1}
```

For the general case we must use

```
\def\newcommand#1{\def\nextiv@{#1}%
\def\nextii@[##1]{\args@##1%
\edef\next@{\noexpand\define\expandafter\noexpand\nextiv@
\the\toks@}%
\next@}%
\def\nextiii@{\edef\next@{\noexpand\define
\expandafter\noexpand\nextiv@}\next@}%
\def\next@{\ifx\next[\expandafter\nextii@\else\expandafter
\nextiii@\fi}%
\futurelet\next\next@}
```

Thus, we first save #1 in `\nextiv@`, and then use a `\futurelet` to see if the next token is `[`. If it is, we use `\nextii@`, whose definition is based on (B), except that we need

```
\expandafter\noexpand\nextiv@
```

to get back the `\noexpand#1` (compare pages 126 and 161). If the next token isn't `[`, then we use `\nextiii@`, based on (A), again using

```
\expandafter\noexpand\nextiv@
```

to get back the `\noexpand#1`.

We also need to replicate the definitions of several special commands from L^AT_EX:

```
\def\em{\ifdim\fontdimen1 \the\font>0pt \rm\else\it\fi}
\def\mbox{\leavevmode\hbox}
```

(The actual definition of `\mbox` in L^AT_EX is

```
\def\mbox#1{\leavevmode\hbox{#1}}
```

but the above definition works just as well [and even allows category changes, if that matters].)

To deal with the various items of the bibliography, we will need a new counter

```
\newcount\bibitemcount@
```

for automatically numbered items, and a dimension

```
\newdimen\bibindent@
```

for the hanging indentation of items.

A `\bibitem`, which is written by `BIBTEX`, might have an “optional argument” [...], so we will need maneuvers similar to that for `\newcommand`. The case

```
\bibitem[...] {key}
```

will call

```
\bibitem@{...} {key}
```

while the case

```
\bibitem {key}
```

without the optional argument will call

```
\bibitem@{\number\bibitemcount} {key}
```

where `\bibitem@#1#2` takes care of making the `{key}` #2 act like a `\label` with the value #1, and then printing #1 at the beginning of the bibliographical data that follows:

```
\def\bibitem{%
  \def\nextii@{##1}##2{\bibitem@{##1}{##2}}%
  \def\nextiii@##1{\global\advance\bibitemcount@ by 1
  \bibitem@{\number\bibitemcount@}{##1}}%
  \def\next@{\ifx\next[\expandafter\nextii@\else
  \expandafter\nextiii@\fi}%
  \futurelet\next\next@}%
```

In order for `\bibitem@#1#2` to make #2 act like a `\label` with the value #1, we simply have to

```
{\def\thelabel@{#1}
  \let\thelabel@@=\empty \let\thelabel@@@=\empty
  \let\thelabel@@@@=\empty
  \label{#2}}
```

—remember that `\label{...}` will always make sense when `\thelabel@` is defined, and will record the value of this label in terms of `\thelabel@`, ..., `\thelabel@@@` (we only care about `\thelabel@`, since it is used for `\ref`, the only thing used by `\cite`).

And then we start the bibliographic entry with

```
\par\hangafter 1 \hangindent=\bibindent@
\noindent@ \hbox to \bibindent@{#1\hfil}\ignorespaces
```

So the definition is

```
\def\bibitem@#1#2{%
  {\def\thelabel@{#1}%
   \let\thelabel@@=\empty \let\thelabel@@@=\empty
   \let\thelabel@@@=\empty
   \label{#2}}%
 \par
 \hangafter1 \hangindent=\bibindent@
 \noindent@ \hbox to \bibindent@{#1\hfil}\ignorespaces}
```

Finally, as soon as we hit the

```
\begin{thebibliography}{...}
```

we want everything to be properly set up. Hopefully, this is the only `\begin` that will occur in the `.bbl` file. So we give an error message in any other case:

```
\def\begin#1#2{%
  \def\next@{#1}\def\nextii@{thebibliography}%
  \ifx\next@\nextii@
  \beginthebibliography@{#2}%
  \else
  \Err@{I can't deal with \string\begin{#1}}%
  \fi}
```

Recall that `\beginthebibliography@` (page 351) prints the heading, and sets everything else up properly.

At the very end of the file we will encounter

```
\end{thebibliography}
```

Again, hopefully this is the only `\end` that will occur in the `.bbl` file, and we define

```
\def\end#1{%
  \def\next@{#1}\def\nextii@{thebibliography}%
  \ifx\next@\nextii@
  \else
  \Err@{I can deal with \string\end{#1}}%
  \fi}
```

(The fact that we are redefining `\end` isn't important, since the definitions of `bibtex.tex`, together with the `.bbl` file are all read in within a `\begingroup ... \endgroup` region (page 348.)

Finally, we reassign `\alloc@` its original definition from plain `TEX`, and make `@` active:

```
\def\alloc@#1#2#3#4#5{\global\advance\count1#1by\@ne
  \ch@ck#1#4#2\allocationnumber=\count1#1
  \global#3#5=\allocationnumber
  \wlog{\string#5=\string#2\the\allocationnumber}}
\catcode'\@=\active
```

We *don't* bother with counters like `\list@C1`, or any thing else created with a `\new...` construction, because it would create confusion if we had to restore them in the `.tex` files—each `\newcount` construction will use up another possibility for a counter, even if we happen to be using the same name as before. `\newif` works differently, so flags are set equal to `\relax`, and we also

```
\let\list@@C=\relax
. . .
```

Intermingled with these redefinitions we also have

```
\let\keeplisting=\undefined
\let\list=\undefined
\let\runitem=\undefined
. . .
```

And at the very end we make `@` active again,

```
\catcode'\@=\active
```

On the other hand, the file `lists.tex` essentially consists of all the definitions for `\lists`, except for things like

```
\expandafter\newcount\csname list@C1\endcsname
```

that aren't removed by `lists.tox`.

Chapter 31. \purge'ing and \unpurge'ing

There's nothing mysterious about \purge and \unpurge—they simply read in a .toc or .tex file:

```
\def\unpurge#1{\input #1\relax}
\def\purge#1{\input #1.toc\relax}
```

Each .toc file clears up the memory space used for certain control sequence definitions, by redefining these control sequences. Control sequences meant for the user (those without @ in their names) are set equal to \undefined, which L^AT_EX always keeps undefined, so that their use will give an error message. On the other hand, control sequences used internally by L^AT_EX (those with @ in their names) are simply set equal to \relax. For example, the file

```
lists.toc
```

which eliminates everything for making lists, starts (again, as in section 27.2, we use double horizontal lines for code in subsidiary files)

```
\catcode'\@=11
\let\listformatbi@=\relax
. . .
\let\listformtmi@=\relax
. . .
\let\listformatti@=\relax
. . .
\let\listformate@=\relax
```

thereby removing from memory the initial definitions of section 18.1.

Then, for the definitions in section 18.2, we

```
\expandafter\let\csname list@P1\endcsname=\relax
. . .
\expandafter\let\csname list@Q1\endcsname=\relax
. . .
```

Part V

*Islands
and the
Output Routine*



Chapter 32. Packaging figures, tables, . . . , with captions

The previous Parts have covered almost all the standard \LaTeX constructions that involve `\label`'s (as well as miscellaneous constructions that don't). But `\caption`'s, in a `\Figure`, `\Table`, or the more general `\island` construction, have been postponed to this Part, because they interact so intimately with the `\output` routine.

32.1. Preliminaries. First we want to disable certain commands from plain \TeX , since they would conflict with \LaTeX 's approach to these insertions:

```
\let\topinsert=\undefined
\let\midinsert=\undefined
\let\pageinsert=\undefined
```

Next, we deal with the rather simple matter of producing crop marks around an `\hybw` or `\Hbyw`. We first need a flag for telling whether `\Figureproofing` or `\noFigureproofing` is in force:

```
\newif\iffigproofing@
\def\Figureproofing{\figproofing@true}
\def\noFigureproofing{\figproofing@false}
```

The special `\Hbyw` construction is reduced to `\hbyw`, but also sets a flag, to indicate that this `\hbyw` will need further processing later on:

```
\newif\ifHby@
\def\Hbyw#1{\global\Hby@true\hbyw\vsz{#1}}
```

The `\global` is needed because the later processing will occur after the current group is completed.

¹In version 1 of \LaTeX , `\table` was used to indicate a class of `\island`'s in the main file, and to indicate a specific table construction in the "tables" file produced with \LaTeX - \TeX ; this implicit inconsistency became explicit in $\mu\TeX$, where the tables are specified within the main file. Consequently, the `\table` class of `\island`'s has been renamed `\Table`. For consistency, `\figure` has likewise been renamed `\Figure`, and then `\figureproofing` was renamed `\Figureproofing` (although it does not act on `\Figure`'s per se, but only on `\hbyw`'s).

An `\hbyw#1#2` is basically going to be

```
\vbox to#1{\hbox to#2{\vfil}}
```

(TeX allows `\hbox to#2{}` without an `\hfil` within the braces, but the `\vfil` is necessary, since there is already something in the `\vbox`.)

We will actually use

```
\hbox{(left crop marks)
\vbox to#1{\hbox to#2{\vfil}
(right crop marks)}
```

when crop marks are required, which we will produce by first adding the ones on the left side, and then those on the right.

To produce the crops marks, we first store a horizontal rule 5 points wide (and default width .4pt) in `\box0`, and declare its height to be 0pt, so that it won't interfere with anything else:

```
\setbox0=\vbox{\hrule \width5pt}\ht0=0pt
```

Now

```
\vbox to#1{\hrule \height5pt \width.4pt\vfil
\hrule \height5pt \width.4pt}
```

gives the vertical parts of the left crops,

|

|

with the baseline at the bottom of this box. To add in the horizontal parts,

┌

└

we move back the .4pt width of these marks, and then `\rlap` a copy of `\box0`, containing the horizontal rule, to give the bottom horizontal rule, and also `\rlap` a copy that has been raised by the height #1, to give the top one.

```
\vbox to#1{\hrule \height5pt \width.4pt\vfil\hrule
           \height5pt \width.4pt}
\kern-.4pt\rlap{\copy0}\raise#1\hbox{\rlap{\copy0}}
```

A similar construction produces the crop marks on the right, using `\llap` instead of `\rlap`:

```
\vbox to#1{\hrule \height5pt \width.4pt\vfil\hrule
           \height5pt \width.4pt}
\kern-.4pt\llap{\copy0}\raise#1\hbox{\llap{\box0}}
```

Of course, we only want to add these crop marks when `\iffigproofing@` is true. But we also don't want to add the crop marks if `\ifHby@` happens to be true—in that case, the final height is to be determined later, based on the height of the caption, and we will deal with the crop mark problem later.

```
\def\hbyw#1#2{%
\hbox{%
\ifHby@
\else
\iffigproofing@
\setbox0=\vbox{\hrule \width5pt}\ht0=0pt
\vbox to#1{\hrule \height5pt \width.4pt\vfil\hrule
           \height5pt \width.4pt}%
\kern-.4pt\rlap{\copy0}\raise#1\hbox{\rlap{\copy0}}%
\fi
\fi
\vbox to#1{\hbox to#2{\vfil}}%
\ifHby@
```

```

\else
\iffigproofing@
\ vbox to#1{\hrule \height5pt \width.4pt\vfil\hrule
\height5pt \width.4pt}%
\kern-.4pt\llap{\copy0}\raise#1\hbox{\llap{\box0}}%
\fi
\fi}}

```

32.2. Starting an `\island`. Next comes the general mechanism whereby

```
\island <space or other material> \caption{...} \endisland
```

packages the `<space or other material>` together with the caption into a box.

At first sight, `\island`'s and `\claim`'s seem to be quite similar, since they can be associated into different "classes", and since there is a `\newisland` construction much like `\newclaim`. But they function somewhat differently, since `\island`'s are not themselves numbered—it is the `\caption`'s on the `\island`'s that must be numbered (and one uses `\caption"..."` to get a quoted number, etc.); consequently, the `\claim` definitions cannot be mimicked too directly.

We begin by declaring the counter and control sequences associated to general `\island`'s:

```

\newcount\island@C
\let\island@P=\empty
\let\island@Q=\empty
\def\island@S#1{#1\null.}
\let\island@N=\arabic
\def\island@F{\rm}

```

We add the `\null` in case #1 ends with an upper-case letter (compare page 208).

Just as `\claim`'s have a `\claimclass@`, our `\island`'s will have an `\islandclass@`, which will be `'...'` when we use

```
\island \c{...}
```


(or when we use something created by \newisland with this \c{...}), and empty if we omit the \c{...}.

Similarly, a construction created with \newisland, say \map, will define \islandtype@ to be \map, analogous to \claimtype@, while an ordinary \island will define \islandtype@ to be \island.

Analogous to \claim@@P, etc., we define

```
\def\island@@@P{\csname\exxx@islandtype@ @P\endcsname}
\def\island@@@Q{\csname\exxx@islandtype@ @Q\endcsname}
\def\island@@@S{\csname\exxx@islandtype@ @S\endcsname}
\def\island@@@N{\csname\exxx@islandtype@ @N\endcsname}
\def\island@@@F{\csname\exxx@islandtype@ @F\endcsname}
```

while for \island@@@C we

```
\def\island@@@C{\csname island@C\islandclass@\endcsname}
```

We want to give an error message if an \island is not used within some sort of \...place, so we declare a flag

```
\newif\ifplace@
```

which such constructions can set true, and which each \island will eventually set false. Moreover, we want a flag

```
\newif\ifisland@
```

which each \island will set true, so that constructions like \Aplace{...} can give an error message if '...' is not some sort of \island.

If \island is used when \ifplace@ is false, we will give an error message. Otherwise, we will initialize \islandclass@ to be empty and \islandtype@ to be \island, and then use a \futurelet to see if a \c comes next:

```

\def\island{%
  \ifplace@
    \def\next@{\let\islandclass@=\empty
      \def\islandtype@{\island}%
      \futurelet\next\island@}%
    \else
      \long\def\next@##1\endisland{\Err@{\noexpand\island must
        be used after some type of \string\...place}}}%
    \fi
  \next@}

```

Notice that when we give the error message, we also swallow the whole `\island... \endisland` construction, to eliminate confusion. The `\long` is required because the `\caption` before the `\endisland` is allowed to contain a blank line. For the `\noexpand`, see section 3.4.

If `\island` is followed by `\c` we will use `\island@c`. Otherwise we will need yet another `\futurelet`, to see if the next token is a `{`, since we want to give an error message if a suitable group `{...}` for the “caption prefix” doesn’t come next:

```

\def\island@{\ifx\next@c\let\next@=\island@c\else
  \def\next@{\futurelet\next\island@@}\fi\next@}

```

Here `\island@@` will call `\island@@@` if a group `{...}` does follow, but give an error message otherwise:

```

\def\island@@{%
  \ifcat\bgroup\noexpand\next
    \let\next@=\island@@@
  \else
    \def\next@{\Err@{\noexpand\island must be followed by
      a {prefix} for \string\caption's}}%
    \fi
  \next@}

```

(See section 3.4 for the use of `\noexpand` in the error message.)

Finally, if we've made it this far without an error message, i.e., if we've been scanning

```
\island {...}
```

then we will save '...' in \captionprefix@, and start to store everything in a box:

```
\global\setbox\islandbox@=\vbox\bgroup
```

Although the \global really isn't necessary here, some of our routines will require \global, so we will use it in all cases.

We therefore first want to declare a new box \islandbox@. Moreover, we also want a new counter \captioncount@, which is quite different from \island@C: it records the number of \caption's that occur within the current \island (remember that an \island can have more than one \caption):

```
\newbox\islandbox@
\newcount\captioncount@
\def\island@@@#1{\def\captionprefix@{#1}%
\captioncount@=0
\global\setbox\islandbox@=\vbox\bgroup}
```

The \egroup matching this \bgroup might be supplied by the \endisland; in most cases, however, it will be supplied by a \caption, which will then go on to store the \caption argument, and then combine this properly with \box\islandbox@.

Leaving these details aside for the moment, let us now consider \island@c, which we obtain when we have

```
\island \c{...}
```

As we might expect from our experience with \newclaim, a construction created by \newisland is going to lead us directly to \island\c{...}. Since such constructions must also be used only within a suitable \...place construction, this means that we *must repeat our \ifplace@ test*.

When `\ifplace@` is false, we again want to give an error message, but instead of the general message

```
\island must be used after some type of \...place
```

we want a more specific message mentioning our particular construction, say `\map`, which has been recorded in `\islandtype@`. Moreover, as before, we want to swallow the entire

```
\map ... \endmap
```

text. This requires much the same tomfoolery that we have engaged in before (compare page 215). If we

```
\edef\next@{\long\def\noexpand\next@###1\expandafter
\noexpand\csname end\exxx@\islandtype@\endcsname{\noexpand\Err@
{\noexpand\noexpand\expandafter\noexpand\islandtype@ must be
used after some type of \noexpand\string\noexpand\...place}}}
```

then this `\edef` makes `\next@` mean

```
\long\def\next@###1\endmap{\Err@{\noexpand\map must be
used in some type of \string\...place}}
```

Consequently, `\next@` executes this definition. So calling `\next@` yet once again produces the error message, swallowing the `\map... \endmap` text in the process.

If `\island\c{...}` is used when `\ifplace@` is true, we will define `\islandclass@` to be `'...'`. Then, similarly to `\claim`, we want to use

```
\csname island@C...\endcsname
```

as the counter. So we will create this counter, using `\newcount@` (see section 3.6 and page 121), and set it to 0 if it does not already exist. Before proceeding further however, we now need yet another `\futurelet` to check that this

```
\island \c{...}
```

is followed by yet another group {...} to specify a caption prefix; in fact, we will need \FNSS@, to skip over any space after the \c{...}:

```

\def\island@c\c#1{%
  \ifplace@
  \def\next@\def\islandclass@{#1}%
  \expandafter\ifx\csname island@C#1\endcsname\relax
  \expandafter\newcount@\csname island@C#1\endcsname
  \global\csname island@C#1\endcsname=0 \fi
  \FNSS@\island@c@}%
\else
\def\next@{%
  \edef\next@\def\noexpand\next@#####1\expandafter
  \noexpand\csname end\expandafter\exxx@
  \islandtype@\endcsname
  {\noexpand\Err@{\noexpand\noexpand\expandafter\noexpand
  \islandtype@ must be used in some type of
  \noexpand\string\noexpand\...place}}}%
  \next@\next@}%
\fi
\next@}

```

Then \island@c@ works analogously to the way \island@@ worked before:

```

\def\island@c@{%
  \ifcat\bgroup\noexpand\next
  \let\next@=\island@c@@
  \else
  \def\next@{\Err@{\noexpand\island\string\c
  {\expandafter\string\islandclass@} must
  be followed by a {prefix} for \string\caption's}}%
  \fi
\next@}

\def\island@c@@#1{\def\captionprefix@{#1}%
  \captioncount@=0
  \global\setbox\islandbox@=\vbox\bgroup}


```

32.3. Starting a `\caption`. As we are setting `\box\islandbox@` we will probably encounter a `\caption`, which we have to consider next, before worrying about the `\endisland`.

First of all, we

```
\rightadd@\caption\to\nofrillslist@
```

since the default style adds a period at the end of the `\caption` automatically, and we want to allow `\nopunct` to override this.

 This is a change from version 1: Even though captions are often whole sentences, where it might seem normal for the user to add the punctuation, it is preferable to require that this punctuation not appear in the input file, because it could create problems in the `.tic` file, where the caption might be followed by something like dot leaders.

Then we will need a box,

```
\newbox\captionbox@
```

to store `\caption`'s, and since we allow more than one `\caption` in an `\island`, we will need another box,

```
\newbox\Captionbox@
```

to store the accumulated `\caption`'s.

A `\caption` will first be encountered within an `\island`, where we have already begun

```
\global\setbox\islandbox@=\vbox\bgroup
```

and it will normally first supply the matching `\egroup` that will finish `\box\islandbox@`, before scanning its argument. But we don't want to add an `\egroup` if our `\caption` is simply a subsequent `\caption` within the

same `\island`. So we want something like

```
\def\caption{%
  \ifnum\captioncount@=0
    \let\next@=\egroup
  \else
    \let\next@=\relax
  \fi
  \next@
  \advance\captioncount@ by 1
  . . .
```

But there is an extra detail to worry about: If the `\caption` was preceded by `\nopunct`, then we have `\nopunct@true` after the `\caption`, but this will then be hidden by the `\egroup`. So we actually use

```
\def\caption{%
  \ifnum\captioncount@=0
    \ifnopunct@
      \def\next@{\egroup\nopunct@true}%
    \else
      \let\next@=\egroup
    \fi
  \else
    \let\next@=\relax
  \fi
  \next@
  \advance\captioncount@ by 1
  \futurelet\next\caption@}

```

Note that the `\captioncount@` is always set or advanced outside of the various boxes that we build, so we don't need to use `\global` with it. Note also that we didn't worry about `\nospace`. That's because this control sequence, although allowed before `\caption`, won't have any effect (in the default style, where nothing follows the final punctuation of a caption; if for some reason a style did have a space after this punctuation, a slightly more complicated routine would be needed).

The `\futurelet` at the end of the definition is needed to see whether `\caption` is followed by a quoted number "...", and `\caption@` simply calls `\caption@q` when a quoted number follows, and `\caption@@` in general:

```
\def\caption@{\ifx\next"\expandafter\caption@q\else
\expandafter\caption@@\fi}
```

Here `\caption@q` and `\caption@@` each first define `\TheLabel@`, ..., `\TheLabel@###`, and then call

```
\finishcaption@
```

where `\finishcaption@` will be defined in the next section. All this is quite analogous to the situation for `\claim`. Since `\caption`'s will be written to the .tic file, we also need some of the subsidiary information that was added for `\HL` and `\hl`: `\caption@q` will set `\ifquoted@` to be true and define `\Qlabel@###`, while `\caption@@` will set it to be false, so that we can use the `\QorTheLabel@###` construction that was used for `\HL` and `\hl`:

```
\def\caption@q"#1"{\quoted@true
{\noexpands@
\let\pre=\island###P \let\post=\island###Q
\let\style=\island###S \let\numstyle=\island###N
\Qlabel@{#1}\let\style=\relax\xdef\Qpref@{#1}}%
\finishcaption@}

\def\caption@@{\quoted@false
\global\advance\island###C by 1
{\noexpands@
\xdef\TheLabel###{\number\island###C}}%
\xdef\TheLabel@\island###N
\xdef\TheLabel###@{\island###P\TheLabel@\island###Q}}%
\xdef\TheLabel@@\island###S
\xdef\Thepref@{\TheLabel###}}%
\finishcaption@}
```

32.4. *Formatting a \caption*. Analogous to \claimformat@, we want to have \captionformat@, to describe, in the very simplest terms, how a caption should be formatted

```
\long\def\captionformat@#1#2#3{%
#1 {\island@@@F#2} #3\punct@.}
```

Here #1 will be the “caption prefix”, #2 will be the properly formatted caption number, and #3 will be the argument of \caption. We use \long because we allow #3 to contain a blank line (the default style doesn’t treat new paragraphs in a \caption in any particularly interesting way, but other styles might.) The ‘\punct@.’ allows a \nopunct before the \caption to eliminate the period (for a caption that ends with a ! for example); a \null is not added here (compare pages 166 and 208) since nothing follows the period. A \nospace before a \caption will have no effect, in the default style, but is allowed.

Unfortunately, the actual formatting of a caption may be quite a complex affair. In the default style, for example, if the caption can be set on a single line whose width is less than the width of \box\islandbox@, we center that line under \box\islandbox@; otherwise, we set the caption as a paragraph whose width is the same as that of \box\islandbox@. In addition, when a single \island has multiple \caption’s, we have the usual problem of stacking these paragraphs on top of each other, so we will want to insert struts.

Many, many, other arrangements might be used; captions might be printed on the side of the figure, etc. It is impossible to anticipate all the things that different style files might do, but necessary modifications for other styles should be clear from a careful study of the proceedings used throughout this chapter.

We begin by adding struts to the definition of \captionformat@:

```
\long\def\captionformat@#1#2#3{\rm\strut
#1 {\island@@@F#2} #3\punct@.\strut}
```

In other styles, \rm might be replaced by something like \tenpoint, so that the caption will be in 10 point type no matter what the current font may be; the \strut should follow, so that it will be the right size. Of course, some styles might even need different \captionformat@’s, for different sorts of \island’s.

Then we want to have

```
\widerthanisland@#1#2#3
```

to test whether `\captionformat@#1#2#3` produces an `\hbox` that is wider than `\box\islandbox@`, setting `\iftest@` to be true if it is wider, and setting it to be false otherwise. Basically, we want to

```
\setbox0=\hbox{\captionformat@{#1}{#2}{#3}}
```

and measure the width of `\box0`.

This fails, however, if the caption contains a displayed formula, like

```
$$\alpha=\dots$$
```

Indeed, in restricted horizontal mode `$$` simply stands for an empty formula (*The T_EXbook*, page 287), so the `\alpha` is outside of math mode, and will give a (mysterious) error message. (It also fails if the caption contains a `\linebreak`, although presumably this will only be put to avoid an `Overfull \hbox` when the caption is longer than the island.)

So instead (compare page 315) we first

```
\setbox0=\vbox{\hsize=\maxdimen
\noindent@@\captionformat@{#1}{#2}{#3}%
\par
\setbox0=\lastbox}
```

where the inner `\setbox0=\lastbox` simply removes¹ the last box in (the main) `\box0`. (See Chapter 8 for the `\noindent@@`.) If our caption argument `#3` consists of simple text, everything will have been set on one line, which will be `\lastbox`, so our final `\box0` will be empty, and therefore have width `0pt` (compare the footnote on page 324). In this case, we will simply set the same material in `\box\captionbox@`, and set `\iftest@` to be true if the width of this box is greater than the width of `\box\islandbox@` and false otherwise.

¹ Recall that `\lastbox` removes the last box (something like `\global\setbox1=\lastbox` keeps the last box in `\box1` if we need it).

But if #3 contains a displayed formula, or more than one paragraph of text, or anything else that creates more than one line of text (including perhaps, so much text that it won't even fit on a line of length \maxdimen), then there is still a line left after we eliminate \lastbox, so the final \box0 does not have width 0pt. In this case, we also want \iftest@ to be true, since we again want to reformat #3 within a \vbox having the width of \box\islandbox@.

After we have set a box for testing purposes, we must be careful to \unlabel@ (section 16.5), to ignore any \label's for future setting of #3, for otherwise we will be using a \label more than once when we do the final setting. We will also use \noset (section 16.4), just in case \Reset or \Offset appear.

We define our test by:

```
\long\def\widerthanisland@#1#2#3{%
  \test@true
  \setbox0=\vbox{\hsize=\maxdimen
    \noindent@@\captionformat@{#1}{#2}{#3}%
    \par\setbox0=\lastbox}%
  \ifdim\wd0=0pt
    \setbox\captionbox@=\hbox{\noset@\unlabel@
      \captionformat@{#1}{#2}{#3}}%
    \ifdim\wd\captionbox@ > \wd\islandbox@\else\test@false\fi
  \fi}
```

Now we can define \captionformat@@, which does the actual formatting by:

```
\long\def\captionformat@@#1#2#3{%
  \widerthanisland@{#1}{#2}{#3}%
  \iftest@
    \global\setbox\captionbox@=\vbox{\hsize=\wd\islandbox@
      \vskip-\parskip
      \noindent@@\noset@\unlabel@
      \captionformat@{#1}{#2}{#3}\par}%
  \fi}
```

```

\else
\global\setbox\captionbox@=
\hbox to\wd\islandbox@{\hfil\box\captionbox@\hfil}%
\fi}

```

The `\vskip-\parskip` is used to eliminate any extra `\parskip` glue before our `\noindent@`'ed paragraph.

Finally, `\finishcaption@#1`, where `#1` will be the argument of `\caption`, first stores `#1` in `\entry@`,

```
\def\entry@{#1}
```

for writing to the `.tic` file later (section 5), and then takes care of details like ignoring initial and final spaces,

```

{\locallabel@
\captionformat@{\expandafter\ignorespaces\captionprefix@\unskip}%
{\ifx\thelabel@@\empty\unskip\else\thelabel@@\fi}%
{\ignorespaces#1\unskip}}

```

If there is only one `\caption` so far, our final `\box\Captionbox@` will just be a `\vbox` containing `\box\captionbox@`:

```
\global\setbox\Captionbox@=\vbox{\box\captionbox@}%
```

(We do this so that `\box\Captionbox@` is always a `\vbox`; that way, we can always `\unvbox\Captionbox@` when necessary.)

Otherwise, we want to create a new `\box\Captionbox@` by combining the old `\box\Captionbox@` with `\box\captionbox@` with a `\smallskip` between them:

```

\global\setbox\Captionbox@=\vbox{\unvbox\Captionbox@
\smallskip\box\captionbox@}%

```

In addition to the `\smallskip`, `\strut`'s will give the proper line spacing.

Note that the \unvbox'ed \Captionbox@ and \box\captionbox@ will not have any additional space added between them.¹

At this point we also want to add appropriate \write\tic@s for writing to the .tic file (page 187). Although it would seem reasonable to include these \write@s within \box\Captionbox@,¹ we will instead attach them to \box\islandbox@, which will always appear on the same page as the corresponding captions. (At some preliminary stage of the macros, \box\Captionbox@ was subjected to further processing, which would have been complicated by the presence of \write@s. That is no longer the case, so that the \write@s could be included in \box\Captionbox@, but there's nothing wrong with the other approach, and I didn't want to rewrite this part of the routines, with the attendant risk of introducing some new bugs.)

We will define \ticwrite@, the appropriate collection of \write\tic@s, in section 5, and we will use

```
\ifnum\captioncount@=1
  \global\setbox\islandbox@=\vbox{\ticwrite@\vbox{\box\islandbox@}}%
  \global\setbox\Captionbox@=\vbox{\box\captionbox@}
\else
  \global\setbox\islandbox@=\vbox{\unvbox\islandbox@
    \setbox0=\lastbox
    \ticwrite@\box0 }
  \global\setbox\Captionbox@=\vbox{\unvbox\Captionbox@
    \smallskip\box\captionbox@}
\fi
```

Note that when \captioncount@ is 1, our new \box\islandbox@ is a \vbox containing \ticwrite@ at top, and then the old \box\islandbox@ enclosed in another layer of \vbox'ing. The reason for this is that \box\islandbox@ might contain more than one thing (in particular, it might contain the \write for a \pagelabel), and we don't want to uncover any of these layers when

¹*The TeXbook*, page 282: "The vertical list inside that box is appended to the current vertical list, without changing it in any way." *The TeXbook* goes on to say that "The value of \prevdepth is not affected." That is, the value of \prevdepth after the \unvbox is the value it had before the \unvbox, not the depth of the last box produced by the \unvbox. This will be important later (page 445).

¹Note that, unlike \insert@s, which have an effect only if they "migrate" out to the main vertical list, \write@s will take place when enclosed in boxes, no matter to what depth.

we `\unvbox\islandbox@` in the next case, where `\captioncount@` is >1 . In that latter case, the

```
\unvbox\islandbox@
\setbox0=\lastbox@
```

temporarily removes the old `\box\islandbox@` from the list, inserts the `\ticwrite@` (after any `\write's` that were inserted from previous steps), and then puts back the old `\box\islandbox@`. This procedure keeps the `\write's` for multiple `\caption's` in the correct order, and also keeps them at the top of the `\vbox`, so that if necessary we can always use `\lastbox` to examine the non-`\write` part.

And after all that, we simply reset `\ifnopunct@` and `\ifnospace@` to be false (we need to do this for `\ifnospace@`, even though `\nospace` has no effect on a `\caption`, because `\nospace` before a `\caption` will not give an error message, and will not reset `\ifnospace@` to be false):

```
\long\def\finishcaption@#1{\def\entry@{#1}%
{\locallabel@
\captionformat@
{\expandafter\ignorespaces\captionprefix@\unskip}%
{\ifx\thelabel@\empty\unskip\else\thelabel@\fi}%
{\ignorespaces#1\unskip}}%
\ifnum\captioncount@=1
\global\setbox\islandbox@
=\vbox{\ticwrite@\vbox{\box\islandbox@}}%
\global\setbox\Captionbox@=\vbox{\box\captionbox@}%
\else
\global\setbox\islandbox@
=\vbox{\unvbox\islandbox@
\setbox0=\lastbox
\ticwrite@\box0}%
\global\setbox\Captionbox@
=\vbox{\unvbox\Captionbox@
\smallskip
```

```

\box\captionbox@}%
\fi
\nopunct@false\nospace@false}

```

32.5. \ticwrite@. Analogous to \Sixtoc@ (section 23.7), we define the routine \Sixtic@ by

```

\def\Sixtic@{\ifx\macdef@\empty\else
\def\next@##1##2\next@\def\macdef@{##1##2}}%
\expandafter\next@\macdef@\next
\edef\next@
{\noexpand\six@\tic@\macdef@
\space\space\space\space\space\space
\space\space\space\space\space\space\noexpand\six@}%
\next@\let\macdef@=\relax\fi}

```

When \tocfile has been specified, and the input file has something like

```

\island \c{<class>}{<caption prefix>} . . .
\caption{<the caption>}
\endisland

```

we will want to write

```

\island \c{<class>}{<caption prefix>}{<formatted caption number>}
<the caption>
\Page ...

```

to the .tic file (with line breaks in <the caption> after every sixth space).

If the file has something like

```

\Figure . . . \caption{<the caption>} \endFigure

```

using \Figure or \Table, or anything else created by \newisland, then we want to write something like

```

\Figure {<formatted caption number>}
<the caption>
\Page ...

```

(Multiple `\caption's` for a single `\island` will simply give rise to multiple copies of such data, with the corresponding (formatted caption number's.) We also want to have `\nopunct` appear before the `\island` or `\Figure` or whatever, when it appears before the corresponding `\caption`. Finally, as with `\HL` and `\hl`, quoted numbers should appear with quotes also (and `\style` should be allowed to appear within the quotes).

For the first line, we use the following code, resorting to the standard `\expandafter\noexpand` trick (compare pages 126 and 161):

```
\def\next@{\island}
\edef\next@{\write\tic@{\ifnopunct@
\noexpand\noexpand\noexpand\nopunct\fi
\ifx\islandtype@\next@\noexpand\noexpand\noexpand\island
\noexpand\string\noexpand\c{\islandclass@}
{\captionprefix@}{\QorTheLabel@@@}}}}
\else
\noexpand\noexpand\expandafter\noexpand
\islandtype@{\QorTheLabel@@@}\fi}
\next@
```

We use `\string\c` rather than `\noexpand\c`, because a space after `\c` usually looks extraneous in the `.tic` file. As with `\HLtoc@` and `\hltoc@`, this should all be done within a group with `\noexpands@` and `\let\style=\relax`.

Next we use

```
\expandafter\unmacro@\meaning\entry@\unmacro@
\Sixtic@
```

to produce `\write's` for the `\caption`, and finally we add the `\write` for the page number:

```
\def\ticwrite@{%
\iftoc@
{\noexpands@\let\style=\relax
\def\next@{\island}%
\edef\next@{\write\tic@{%
\ifnopunct@\noexpand\noexpand\noexpand\nopunct\fi
```



```

\ifx\islandtype@\next@
\noexpand\noexpand\noexpand\island
\noexpand\string\noexpand\c{\islandclass}%
{\captionprefix}{\QorThelabel@@@}%
\else
\noexpand\noexpand\expandafter\noexpand
\islandtype@{\QorThelabel@@@}\fi}%
\next@}%
\expandafter\unmacro@\meaning\entry@\unmacro@
\Sixtic@
\write\tic@{\noexpand\Page{\number\pageno}{\page@N}%
{\page@P}{\page@Q}^^J}%
\fi}

```

For a style where `\captionformat@` involved `\addspace@`, so that `\nospace` before a `\caption` would have an effect, we would probably want to add

```
\ifnospace@\noexpand\noexpand\noexpand\nospace\fi
```

in the `\edef`.

32.6. \Htrim@. A special procedure is needed to deal with an `\Hbyw`. This has been temporarily set to a box of height `\vsize`, so that `\box\islandbox@` will also have this height. The procedure

```
\Htrim@#1
```

is used with `#1` being the amount of space that we want to leave between the tall box and the caption, and is used only when `\ifHby@` is true:

```

\def\Htrim@#1{%
\ifHby@
. . .
\fi}

```

It is supposed to change `\box\islandbox@` to a box of height

$$\vsize - \#1 - \text{height of the caption box,}$$

so that this new box, together with the caption box and #1 space between them, will take up the entire page. In addition, if `\ifFigureproofing` is true, then we need to put crop marks around this new `\box\islandbox@`.

So we start out by setting

```
\dimen@=\vsize
```

If there was a caption we

```
\advance\dimen@ by -\ht\Captionbox@
\advance\dimen@ by -#1
```

Now we basically want to

```
\setbox\islandbox@=\vbox
{\hbyw\dimen@{\wd\islandbox@}}
```

In order to allow crop marks around this box, however, we need to have `\ifHby@` be false (page 365). But we'll need to know the value of `\ifHby@` later on (page 390 ff.). So we will set `\global\Hby@false` before we do this, and restore `\global\Hby@true` afterwards (remember that this whole routine is being done only when `\ifHby@` is true; we use `\global` assignments here since they were necessary previously).

Before we actually

```
\global\setbox\islandbox@=\vbox
{\hbyw\dimen@{\wd\islandbox@}}
```

we must first extract any `\ticwrite's` that may have appeared in the old `\box\islandbox@`, and make sure that they appear in the new one. To do this, we can first store `\wd\islandbox@`,

```
\dimen@ii=\wd\islandbox@
```

and then

```
\global\setbox\islandbox@=\vbox
{\unvbox\islandbox@
\setbox0=\lastbox
\hbyw\dimen@\dimen@ii}
```

Recall (page 380) that the

```
\unvbox\islandbox@\setbox0=\lastbox
```

leaves the \write's at the top of \box\islandbox@, while deleting the other part. Unfortunately, that's not quite adequate, because we will then lose any \pagelabel information that may have been in \box\islandbox@. So instead we use

```
\global\setbox\islandbox@=\vbox
{\unvbox\islandbox@
 \setbox0=\lastbox
 \vbox to0pt{\vss\box0}\nointerlineskip
 \hbyw\dimen@\dimen@ii}
```

Although no extra space is inserted before the \vbox to0pt (see the footnote on page 379), we need \nointerlineskip to prevent extra space between that box and the next \hbyw.

The complete code is:

```
\def\Htrim@#1{%
 \ifHby@
 \dimen@=\vsize
 \ifnum\captioncount@=0
 \else
 \advance\dimen@ by -\ht\Captionbox@
 \advance\dimen@ by -#1%
 \fi
 \global\Hby@false
 \dimen@ii=\wd\islandbox@
 \global\setbox\islandbox@=\vbox
 {\unvbox\islandbox@ \setbox0=\lastbox
 \vbox to0pt{\vss\box0}\nointerlineskip
 \hbyw\dimen@\dimen@ii}%
 \global\Hby@true
 \fi}
```

32.7. Other accoutrements for `\endisland`. The `\islandpairedata` and `\islandtripledata` constructions (section 10) essentially operate by temporarily creating `\islands` and storing `\box\islandbox@` in other places. During such processes we want to abort most of the additional operations of `\endisland`, so we need a flag

```
\newif\ifdata@
```

to tell us when we are performing such temporary maneuvers.

We also need a way of testing for the class of an `\island`, since this will determine how the caption is attached to the `\island`; in the default style, for example, an `\island` with class `T` (i.e., a `\Table`), has the caption above, instead of below. The definition of `\iclasstest@` is analogous to `\classtest@` for `\claim` (section 22.11):

```
\def\iclasstest@#1{\def\next@{#1}\ifx\next@\islandclass@
\test@true\else\test@false\fi}
```

We don't bother with a corresponding `\itypetest@`, because the class of an `\island` should be sufficient information: Unlike the situation for `\claim`, where, for example, `\claim\c{T}{Theorem}` and `\claim\c{T}{Lemma}` might be used so that Theorem's and Lemma's are numbered together, it's unlikely that any one would want Tables and Figures, or Maps and Plates to share a common numbering. Of course, a style file that did want to allow such things could introduce a suitable `\itypetest@`.

32.8. `\endisland`. Finally, we are ready for `\endisland`. If there was no `\caption`, then it must provide the `\egroup` that matches the `\bgroup` in `\island`,

```
\ifnum\captioncount@=0 \expandafter\egroup\fi
```

Next come the instructions for putting the island and caption together, which will *not* be done when we are simply collecting data for constructions

like `\islandpairedata`, etc.,

```
\ifdata@
\else
. . .
. . .
\fi
```

In the case of a table, the default style leaves a `\bigskip` between the caption and the island, so we want to start with something like

```
\iclasstest@{T}
\iftest@
\Htrim@\bigskipamount
```

Remember that `\Htrim@` operates only when we have an `\Hbyw`. In this case, it makes `\box\islandbox@` just high enough so that its height plus the height of `\box\Captionbox@` plus `\bigskipamount` is `\vsize`.

Actually, instead of `\bigskip`, we really want less space, because the last line of the caption above the table will have a depth of 3.5pt (the depth of a strut). So we really want something like

```
\skip@ = -3.5pt \advance\skip@ by \bigskipamount
\Htrim@\skip@
```

That way, we know exactly how much space we are getting: enough to make the baseline of the first line of the caption 12pt plus `\bigskipamount` below the `\island`.

We will actually use something like

```
{\rm\global\skip1 = -\dp\strutbox}\global\advance\skip1 by
\bigskipamount
\Htrim@{\skip1}
```

to remind other style files to insert something like `\tenpoint` for the `\rm`, so that `\skip1` will have the proper value. We need `\skip1` because of the

\global assignment. Since it turns out that \skip1 will be used quite frequently, we will

```
\skipdef\skipi@=1
```

so that we can use \skipi@ to abbreviate \skip1.

Then we want

```
\global\setbox\islandbox@=\vbox
{\ifnum\captioncount@=0
  \else
    \box\Captionbox@
    \nointerlineskip
    \vskip\skipi@
  \fi
\box\islandbox@}
```

In addition to the \vskip\skipi@ that we provide, there would normally also be \lineskip glue between the caption and \box\islandbox@; we use \nointerlineskip to suppress this, so that there is just one glue between the two boxes.

The new \box\islandbox@ will eventually be taken apart by the \output routine (knowing that there is just one glue between the two boxes makes this easier), and the pieces will be put back on the page in the proper way. In the default style, \box\Captionbox@ and (the original) \box\islandbox@ will each be centered; at that time, the \vskip\skipi@ glue between them will be able to stretch or shrink with the other glue on the page.

In the default style, all \island's other than \Table's have their captions below, with a \medskip between them. In this case we actually want to *increase* the \medskip by 3.5pt, since the first line of the caption has a strut of height 8.5pt:

```
\else
{\rm\global\skipi@ = \dp\strutbox}
\global\advance\skipi@ by \medskipamount
\Htrim@\skipi@
```

```

\global\setbox\islandbox@=\vbox
{\box\islandbox@
 \ifnum\captioncount@=0
 \else
 \nointerlineskip
 \vskip\skipi@
 \box\Captionbox@
 \fi}
\fi

```

The only thing left to do is to give an error message in case the final `\box\islandbox@` ends up larger than `\vsize`, and reset the height of `\box\islandbox@` to `\vsize`. The message should be something like

```

\island is larger than page
\Figure is larger than page
etc.,

```

when there are no captions, and

```

\island with \caption is larger than page
\Figure with \caption is larger than page
etc.,

```

otherwise.

After all this (occurring within an `\ifdata@\else...\fi` region), we can reset `\Hby@false` and set `\island@true`, to give proper information to `\Aplace`, etc. The complete code is:

```

\skipdef\skipi@=1
\def\endisland{%
 \ifnum\captioncount@=0 \expandafter\egroup\fi
 \ifdata@
 \else
 \iclasstest@{T}%
 \iftest@
 {\rm \global\skipi@ = -\dp\strutbox}%
 \global\advance\skipi@\bigskipamount

```

```

\Htrim@\skip@
\global\setbox\islandbox@=\vbox
{\ifnum\captioncount@=0
  \else
  \box\Captionbox@
  \nointerlineskip
  \vskip\skipi@
  \fi
  \box\islandbox@}%
\else
{\rm \global\skipi@ = \dp\strutbox}%
\global\advance\skipi@ by \medskipamount
\Htrim@\skipi@
\global\setbox\islandbox@=\vbox
{\box\islandbox@
  \ifnum\captioncount@=0
  \else
  \nointerlineskip
  \vskip\skipi@
  \box\Captionbox@
  \fi}%
\fi
\ifHby@
\else
\dimen@=\ht\islandbox@ \advance\dimen@ by \dp\islandbox@
\ifdim\dimen@ > \vsize
\def\next@{\island}%
\Err@{%
  \ifx\Islandtype@\next@\noexpand\island\else
  \expandafter\noexpand\islandtype@\fi
  \ifnum\captioncount@=0 \else
  with \noexpand\caption\fi
  is larger than page}%
\ht\islandbox@=\vsize
\fi
\fi
\fi

```

```
\global\Hby@false \island@true}
```

32.9. \newisland. The `\newisland` construction is similar to `\newclaim`, but, as with `\NameHL` and `\Namehl`, we write the information to the `.tic` file (and, as with `\newclaim`, we first make sure that #1 hasn't already been defined). As before, we use `\newcount@` (section 3.6 and page 121):

```
\def\newisland#1\c#2#3{\define#1{}}%
\iftoc@\immediate\write\tic@{\noexpand\newisland
  \noexpand#1\string\c{#2}{#3}^^J}\fi
\expandafter\def\csname\exstring@#1@S\endcsname{\island@S}%
\expandafter\def\csname\exstring@#1@N\endcsname{\island@N}%
\expandafter\def\csname\exstring@#1@P\endcsname{\island@P}%
\expandafter\def\csname\exstring@#1@Q\endcsname{\island@Q}%
\expandafter\def\csname\exstring@#1@F\endcsname{\island@F}%
\expandafter\def\csname end\exstring@#1\endcsname
  {\endisland}%
\expandafter
\ifx\csname island@C#2\endcsname\relax
  \expandafter\newcount@\csname island@C#2\endcsname
  \global\csname island@C#2\endcsname=0
\fi
\edef\next@{\noexpand\expandafter\noexpand\let\noexpand
  \csname\exstring@#1@C\noexpand\endcsname=
  \csname island@C#2\endcsname}%
\next@
\def#1{\def\islandtype@{#1}\island@c{#2}{#3}}
```

\LaTeX defines `\Figure` and `\Table` directly in terms of `\newisland`:

```
\newisland\Figure\c{F}{Figure}
\newisland\Table\c{T}{Table}
```

32.10. Manipulating \island's. The constructions `\islandpairedata` and `\islandtripledata`, the basic constructions behind `\Figurepair`, etc.,

store the `\island`'s and `\caption`'s from their arguments in appropriate boxes, which we first declare:

```

\newbox\islandboxi
\newbox\islandboxii
\newbox\islandboxiii
\newbox\captionboxi
\newbox\captionboxii
\newbox\captionboxiii

```

(These boxes, as well as `\islandpairdata`, etc., have no @'s in them, so that they will be accessible to the user.)

`\islandpairdata{...}{...}` is meant to be used when each ... is some sort of `\island` like

```
\Figure _ _ _ \endFigure
```

Basically, we just want to execute the first

```
\Figure _ _ _ \endFigure
```

without any `\...place`, and store `\box\islandbox@` in `\box\islandboxi` and `\box\Captionbox@` in `\box\captionboxi`, and similarly for the second

```
\Figure _ _ _ \endFigure
```

Since `\island`'s normally work only when `\ifplace@` is true, we will simply temporarily declare this within a group. We will also set `\ifdata@` to be true to disable most of the `\endisland` routine.

```

\long\def\islandpairdata#1#2{%
  {\place@true \data@true
  #1%
  \global\setbox\islandboxi=\box\islandbox@
  \global\setbox\captionboxi=\box\Captionbox@
  #2%
  \global\setbox\islandboxii=\box\islandbox@
  \global\setbox\captionboxii=\box\Captionbox@
  }}

```

The definition of `\islandpairbox` has been changed from version 1 (which appears on page 73 of the *L^AT_EX* Manual). First of all, we have to allow for pairs of `\island`'s neither of which has a `\caption`. Moreover, the `\medskip` should be modified as in `\endisland`. The biggest change, however, is that instead of simply creating a `\vbox`, we instead `\setbox\islandbox@` to this `\vbox`:

```

\long\def\islandpairbox#1#2{%
  \islandpairdata{#1}{#2}%
  \dimen@=\ht\captionboxi
  \ifdim\ht\captionboxii > \dimen@
    \dimen@=\ht\captionboxii\fi
    \ifdim\dimen@ > Opt
      \ifdim\ht\captionboxi < \dimen@
        \global\setbox\captionboxi
          =\vbox to\dimen@{\unvbox\captionboxi\vfil}\fi
      \ifdim\ht\captionboxii < \dimen@
        \global\setbox\captionboxii
          =\vbox to\dimen@{\unvbox\captionboxii\vfil}\fi
    \fi
    \global\setbox\islandbox@=\vbox{\hbox to\hsize
      {\hfil\box\islandboxi\hfil\box\islandboxii\hfil}}%
    \ifdim\dimen@ > Opt
      \nointerlineskip
      {\rm \global\skipi@=\dp\strutbox}
      \global\advance\skipi@ by \medskipamount
      \vskip\skipi@
      \hbox to\hsize
        {\hfil\box\captionboxi\hfil\box\captionboxii\hfil}
    \fi}%
}

```

Then `\Figurepair` is defined by

```

\def\Figurepair#1\and#2\endFigurepair{\island@true
  \islandpairbox{\Figure#1\endFigure}{\Figure#2\endFigure}}

```

Thus, `\Figurepair` produces the same effect as if we had made an `\island` where the `(appropriate space)` is the combination of the `(appropriate space)`'s for the two islands, and where the caption is the combination of the captions for the two islands.

In version 1 the definition was somewhat different: `\Figurepair` treated the entire combination as a single new island without a caption. The advantage of the new approach is that when our `\box\islandbox@` is taken apart by the `\output` routine (page 388), there will be `\vskip\skipi@` glue between the combined `(appropriate space)`'s and the combined captions, which will again be able to stretch or shrink with the other glue on the page. In particular, if a `\Figure` and a `\Figurepair` appear on the same page, the space before the caption will have stretched or shrunk by the same amount for each.

The definitions of `\islandparboxa`, to put the captions above the combined spaces, and `\Tablepair`, which uses this, won't be given, since they are quite analogous. Similarly, `\islandtripledata`, etc., follow quite similarly.

Chapter 33. An overview Placing the packaged figures, tables, ...

We come now to what is perhaps the most interesting part of \LaTeX 's general document processing features—its handling of inserted Figures and other `\island`'s.

Chapter 8 of the \LaTeX Manual describes the default style's fairly rigorous set of rules for figure placement. Of course, innumerable different rules might be specified by a style designer, so we certainly cannot claim to have anticipated all problems.

In fact, as the Manual concedes (pages 68 and 70), although the current solution will probably be quite adequate in practice, it isn't ideal. The aim of \LaTeX 's automatic placement mechanism is not to provide perfection, but to get as much mileage as possible out of \TeX 's own `\insert` mechanism. Although the details of the \LaTeX constructions take up the next few chapters, they are fairly simple in concept; a \TeX pert intending to change the `\output` routine should not rush in, but at least this is not a region where \TeX angels fear to tread (a thorough understanding of pages 122–125 of *The \TeX book* is in order, however).

33.1. `\place`. Before we get to the interesting part, we will consider the “low level” construction `\place`, used in a construction like

```
\place{\island ... \endisland}
```

to simply insert `\box\islandbox@` at the current point in the file.

The `\place` construction begins by declaring `\place@true`, so that the `\island` construction won't give an error message, and `\island@false`, so `\ifisland@` will be true only if the argument of `\place@` actually involves an `\island` of some sort:

```
\def\place#1{\place@true \island@false  
#1 . . .
```

The argument `#1`, presumably `\island... \endisland`, will simply create `\box\islandbox@` and set `\island@true`. So if `\ifisland@` is still false, we

will give an error message:

```
\Err@{Whoa ... there's no \string\Figure, \string\Table,
etc., here}
```

Otherwise, we will just use

```
\box\islandbox@
```

to typeset the box at that point.

Then we will reset `\ifplace@` to be false:

```
\def\place#1{\place@true\island@false
#1%
\ifisland@
\box\islandbox@
\else
\Err@{Whoa ... there's no \string\Figure, \string\Table,
etc., here}%
\fi
\place@false}
```

33.2. Automatic placement. Having gotten that out of the way, we are ready, willing, and able to tackle the automatic placement routines.

Here is the general principal. Whenever something is `\...place'd` it will simply be added to a certain insertion class; for this purpose we might as well preempt plain TeX's `\topins` insertion class, since we have already disallowed its use for its original purposes.

Remember that for every insertion class, like `\topins`, there is

- An associated box, `\box\topins`, which is where the material from that insertion class appears when a page has been completed and the `\output` routine is called.

At that time, the height of `\box255` may well be different than `\vsize`, since the `\output` routine is going to combine `\box255` with the material in `\box\topins` (and possibly other material), to make the box that TeX will `\shipout`.

In plain `TeX`, material from the `\footins` and `\topins` insertion classes are handled rather simply: the `\pagecontents` macro simply `\unvbox`'es each of these boxes, keeping everything in `\box\topins` at the top of the page, and everything in `\box\footins` at the bottom. But there is no need for such a restricted approach: it's quite possible for the `\output` routine to extract various pieces from `\box\topins`, putting some pieces at the top and other pieces at the bottom. Thus, some figures in the `\topins` insertion class can just as well be placed at the bottom of the page (hopefully, before the footnotes) as at the top of the page.

- An associated counter `\count\topins`, which tells `TeX` what proportion of the height plus depth of `\box\topins` to subtract from `\vsize` in computing the proper height for `\box255`. In plain `TeX`, `\count\topins=1000`, indicating that `\box255` should leave room for the full height plus depth of `\box\topins`; this will remain the same in `LATEX`.
- An associated dimension register, `\dimen\topins`, which indicates the maximum size of `\box\topins` that will be allowed per page.
- An associated glue register, `\skip\topins`, which indicates extra space to leave blank in `\box255` if there happens to be anything in `\box\topins`.

As we will soon see, `\dimen\topins` and `\skip\topins` are the keys to making `\topins` work for us.

Notice that `TeX`'s insertion mechanism already does most of the work we want: boxes are added and withdrawn from any insertion class in a "first in first out" manner; and `TeX` does all the calculations to make sure that `\box255` is made sufficiently short to accommodate anything in the insertion class.

Of course, we usually don't want `TeX` to allow more than two things in `\topins` to appear per page. But this can be controlled by `\dimen\topins` (which can be set to any desired value at any time); if `\dimen\topins` is the sum of the height plus depths of the first two boxes in `\topins`, then `TeX` will allow only these two boxes to be added to `\box\topins`.

There is also the question of the glue to be left below a figure at the top of a page, which we will call `\belowtopfigskip`, or above a figure at the bottom of the page, which we will call `\abovebotfigskip`. For the moment, let's pretend that these two values are the same. Then before

`\box\islandbox@` is inserted into `\topins`, we will simply declare the height of `\box\islandbox@` to be greater than it already is by the `<dimen>` part of `\abovebotfigskip`, and we will similarly increase `\dimen\topins` enough to leave space for the `<dimen>` part of this glue. Thus, when a box in `\box\topins` is returned to its proper height, there will be room to add `\abovebotfigskip` glue.

The only question is, what should we do when the two amounts of glue that we want to leave are different, as, for example, in the default style, which has

```
\belowtopfigskip = 15pt plus 5pt minus 5pt
\abovebotfigskip = 18pt plus 6pt minus 6pt
```

Fortunately, `\skip\topins` allows us to deal with this:

- (1) We will be increasing `\dimen\topins` by an additional 18pt for each `\Aplace`.
- (2) But we will also set

```
\skip\topins = <dimen> part of \abovebotfigskip - \belowtopfigskip
              = -3pt
```

Then:

- If no members of the insertion class appear on the page, all these quantities are irrelevant.
- If just one member of the insertion class appears (at top), then `TEX` leaves -3pt extra space for it; i.e., `\box255` will be 3pt *longer* than it should be to allow the insertion plus 18pt space to fit on a page. Consequently, when we add only 15pt instead of the 18pt, everything will work out just right.
- If one member of the insertion class appears at the top and one (or more) at the bottom, everything still works, because `TEX` only leaves the -3pt extra space once. This properly adjusts the top member of the insertion class, while the bottom member, with 18pt glue above it, also works out right.

The analysis seems equally valid (and easier) if `\belowtopfigskip` is greater than `\abovebotfigskip`; however, as we'll see at the end of section 6 and in section 36.5, it really introduces quite different problems.

33.3. Setting things up. First we declare the glues below figures at the top of the page and above figures at the bottom of the page, and give them their default values; at the same time, we introduce a `(dimen)` register for the minimum amount of text to be allowed on a page that is otherwise filled with figures:

```
\newskip\belowtopfigskip
\belowtopfigskip=15pt plus 5pt minus 5pt
\newskip\abovebotfigskip
\abovebotfigskip=18pt plus 6pt minus 6pt
\newdimen\minpagesize
\minpagesize=5pc
```

To get `\skip\topins` to be the `(dimen)` part of `\belowtopfigskip` – `\abovebotfigskip`, we use the following code (compare page 136):

```
\dimen@=\belowtopfigskip
\advance\dimen@ by -\abovebotfigskip
\skip\topins=\dimen@
```

(If a style file changes `\belowtopfigskip` and `\abovebotfigskip`, these lines should be repeated, to reset `\skip\topins` correctly.)

Initially, `\dimen\topins` will be `0pt`; it will be increased only as we `\Aplace` things.

```
\dimen\topins=0pt
```

In order to set `\dimen\topins` correctly, we are going to have to keep track of various things. In particular, we will use a counter

```
\newcount\topinscount@
```

to keep track of how many boxes still remain in the `\topins` insert.

33.4. How \Aplace works. An extra layer of complications envelops \LaTeX 's automatic placing mechanism because of the existence of so many different varieties of `\dotsplace's`. In order to get a better idea of the main features, in this section we will consider a simpler style in which there is only

`\Aplace`, and we will also assume that `\Aplace` is being used only in vertical mode. Code surrounded by dashed lines rather than solid lines will be changed when we get to the final definitions later on.

(1) Each

```
\Aplace{\island ... \endisland}
```

will essentially

```
\insert\topins\box\islandbox@
```

where `\box\islandbox@` is the completed box—containing both the `\island` and its caption(s), which the `\island...endisland` creates—except that we will first increase the height of `\box\islandbox@` by the `<dimen>` part of `\abovebotfigskip`, so that T_EX will be trying to leaving room for this box plus this amount of space. T_EX doesn't allow us to say

```
\advance\ht\islandbox@ by \abovebotfigskip
```

so we have to take a roundabout route and say

```
(A) \dimen@=\ht\islandbox@
     \advance\dimen@ by \abovebotfigskip
     \ht\islandbox@=\dimen@
```

(2) Then we will

```
\advance\dimen@ by \dp\islandbox@
```

so that `\dimen@` is the height plus depth of `\box\islandbox@`, plus the `<dimen>` part of `\abovebotfigskip`. The value of `\dimen@` is information that we are going to store, whether or not we plan to allow the `\Aplace`'d `\island` to appear in `\box\topins`. Instead of keeping a list of these dimensions, it is simpler to store empty `\hbox`'s of the corresponding width inside a `\vbox`, which we call `\topinsdims@`; the routine for doing this will be called `\storedim@`:

```
\newbox\topinsdims@
```

```
\def\storedim@{\global\setbox\topinsdims@
=\vbox{\hbox to\dimen@{}%
\unvbox\topinsdims@}}
```

Notice that we put the new information on the *top* of the `\vbox`. That is because it is easiest to get information by taking things off of the *bottom* of the `\vbox`, with `\lastbox` (see the footnote on page 376). Note also that no glue will be added between the `\hbox`'s (see the footnote on page 379).

(3) Each time we encounter an `\Aplace`, we must decide whether or not the corresponding

```
\box\islandbox@ ,
```

which we have placed in the `\topins` insertion class, should be allowed to appear in `\box\topins` at the next `\output` routine.

The default style uses the rule that there should be at most two `\island`'s per page. Recall that `\topinscount@` is supposed to be the number of insertions currently in the `\topins` insertion class. So if `\topinscount@` is greater than or equal to 2, there are already at least two insertions waiting to go on the current page. In this case, we don't want to allow the current insertion to appear in `\box\topins`. But if `\topinscount@` is less than 2, so that there is at most one insertion waiting to go on the current page, then we do want to allow the current insertion to appear in `\box\topins` (of course, it will then be up to `TEX` to decide whether there is actually room on the page for this insertion).

We initially have

```
\dimen\topins=0pt
```

This means that initially no inserts will be placed in `\box\topins` during the `\output` routine. If we decide to allow the current insertion to appear in `\box\topins`, we need to communicate that decision to `TEX`, as indicated on page 397, by increasing `\dimen\topins` by `\dimen@`,

```
\ifnum\topinscount@ > 1 \else
(increase \dimen\topins appropriately) \fi
```

Again, we will need a roundabout route for increasing `\dimen\topins`,

```
\advance\dimen@ by\dimen\topins
\global\dimen\topins=\dimen@
```

We define a routine for doing this, `\advancedimtopins`, which contains one more element,

```
\def\advancedimtopins@{%
  \ifnum\pageno=1
  \else
  \advance\dimen@ by \dimen\topins
  \global\dimen\topins=\dimen@
  \fi}
```

In other words, aside from any other considerations influencing our decision as to whether or not an `\Aplace'd \island` should be allowed to appear in `\box\topins`, it will never be allowed on page 1.

Simple modifications of `\advancedimtopins@` allow numerous other style decisions. For example, in the book style, each `\chapter` starts a new page and then sets `\iffirstchapterpage@` to be true; then in the definition of `\advancedimtopins@`, the clause

```
\ifnumpageno=1
\else
```

is replaced by

```
\iffirstchapterpage@
\else
```

This prevents anything from appearing in `\box\topins` when the first page of any `\chapter` is presented to the `\output` routine.

(4) Once we have properly increased `\dimen\topins`, we can `\insert` our `\box\islandbox` into `\topins` (it is important to do the `\insert` *after* taking care of `\dimen\topins`, because TeX decides whether or not to move an `\insert` onto the current page at the time the `\insert` is made).

(5) Finally, the counter `\topinscount@` is supposed to be the number of boxes currently in the `\topins` insertion class, so each time an `\Aplace` occurs we need to

```
\global\advance\topinscount@ by 1
```

(it will be the duty of the `\output` routine to suitably *decrease* `\topinscount@` by the number of `\island`'s that finally get placed on that page).

For the actual definition, `\Aplace`, like `\place`, will begin by setting `\place@true` and `\island@false`, and end by resetting `\place@false`. After calling the argument #1 to set a box, and set `\ifisland@` to be true, we give an error message if `\ifisland@` is still false,

```
\def\Aplace#1{\place@true \island@false
#1%
\ifisland@
. . .
. . .
\else
\Err@{Whoa ... there's no \string\Figure, \string\Table,
etc., here}
\fi
\place@false}
```

(1) Assuming we don't give an error message, the first thing will be

```
\dimen@=\ht\islandbox@
\advance\dimen@ by \abovebotfigskip
\ht\islandbox@=\dimen@
```

(2) This will be followed by

```
\advance\dimen@ by \dp\islandbox@
\storedim@
```

(3) Next comes

```
\ifnum\topinscount@ > 1 \else \advancedimtopins@ \fi
```

(4) Then we are ready to

```
\insert\topins{\penalty0 \splittopskip=0pt \floatingpenalty=0
\box\islandbox@}
```

The `\penalty0` allows this insertion to be treated as a split insertion, and thus float to the next page if it does not fit on this page (*The T_EXbook*, pages 123–124). Other penalties, like plain T_EX's `\penalty200`, might be used by style designers to encourage T_EX to keep the insertion on the same page. The `\splittopskip=0pt` insures that only 0pt glue precedes this split insertion; this is probably unnecessary, since the inserted box is quite unlikely to be smaller than `\topskip`, but there's no point taking chances, especially since at one point (page 407) it will be crucial to know that this glue is indeed 0pt. The `\floatingpenalty=0` means that there will be no extra page breaking penalties if later inserts are allowed to split also (i.e., to appear on later pages); this is also not really necessary, since `\floatingpenalty` has the default value 0.

(5) Finally, we

```
\global\advance\topinscount@ by 1
```

So our definition is

```
-----
\def\Aplace#1{\place@true \island@false
#1%
\ifisland@
\dimen@=\ht\islandbox@
\advance\dimen@ by \abovebotfigskip
\ht\islandbox@=\dimen@
\advance\dimen@ by \dp\islandbox@
\storedim@
\ifnum\topinscount@ > 1 \else \advancedimtopins@ \fi
\insert\topins{\penalty0 \splittopskip=0pt
\floatingpenalty=0
\box\islandbox@}%
\global\advance\topinscount@ by 1
```

```

\else
  \Err@{Whoa ... there's no \string\Figure, \string\Table,
    etc., here}%
\fi
\place@false}
-----

```

33.5. How the `\output` routine works. Now let's consider what the `\output` routine must do. At the time that it is invoked, `\box\topins` will contain zero, one, or two boxes, with all other members of the insertion class still held over. The boxes in `\box\topins` occur in the order that they were `\insert`'ed, i.e., the first `\insert` produces the top box, the second `\insert` produces the box below it. Unfortunately, that's the exact reverse of the order we want, because the only way that we can extract boxes from `\box\topins` is by means of a construction like

```

\setbox0=\vbox{%
  \unvbox\topins
  \globally\setbox1=\lastbox}

```

which makes `\box1` be the bottom box. In the case where `\box\topins` contains at most two boxes, this isn't really all that burdensome, but in the general situation, where something like `\AAplace` may have been used, the reverse order becomes a true impediment. So we will define a routine

```

\fliptopins@

```

that recreates `\box\topins` with the sub-boxes in reverse order. At the same time, the number of boxes will be stored in a counter, `\flipcount@`, whose value will be used later on.

(1) To begin, we set `\flipcount@` to 0. Then we `\unvbox\topins` within another box, into which `\vskip1pt` is inserted at the beginning, as a marker (compare page 328). And within this box, in which `\box\topins` has already

been destroyed, we initialize the new `\box\topins`:

```
\def\fliptopins@{\global\flipcount@=0
\setbox0=\vbox{%
\vskip1pt
\unvbox\topins
\global\setbox\topins=\vbox{}}
```

If `\box0` looks like

```
\vskip1pt
\penalty0
⟨box1⟩
\penalty0
⟨box2⟩
\penalty0
⟨box3⟩
. . .
```

then, as we start to take off boxes and penalties from the bottom of this list, `\lastskip` will be 1pt when we have reached the top; at any other point `\lastskip` will be 0pt, since there is no other glue on the list. So the procedure

```
\loop
\ifdim\lastskip=0pt
\global\advance\flipcount@ by 1
\setbox0=\lastbox
\global\setbox\topins=\vbox{\unvbox\topins\box0}
\unpenalty
\repeat}
```

should set `\flipcount@` to the number of boxes, and also reconstruct `\box\topins`, in the reverse order.

But `\box0` can have a different structure also: when a member of `\topins`

has been deferred to a later page (i.e., split), `\box\topins` looks like

```
\vskip0pt
<box1>
\penalty0
<box2>
\penalty0
<box3>
. . .
```

where the `\vskip0pt` is the `\splittopskip` glue, which we have made certain will be `0pt` (page 404). So we have to use the more complicated

```
\loop
\test@false
\ifdim\lastskip=0pt
\unskip
\ifdim\lastskip=0pt
\test@true
\fi
\fi
\iftest@
\setbox0=\lastbox
\global\setbox\topins=\vbox{\unvbox\topins\box0}
\unpenalty
\repeat}
```

We want to avoid all this rigamorole if `\box\topins` is void to begin with, so our final definition is

```
\newcount\flipcount@
\def\fliptopins@{\global\flipcount@=0
\ifvoid\topins\else
\setbox0=\vbox{%
\vskip1pt
\unvbox\topins
\global\setbox\topins=\vbox{}}%
```

```

\loop
  \test@false
  \ifdim\lastskip=0pt
    \unskip
    \ifdim\lastskip=0pt
      \test@true
    \fi
  \fi
  \iftest@
    \global\advance\flipcount@ by 1
    \setbox0=\lastbox
    \global\setbox\topins=\vbox{\unvbox\topins\box0}%
    \unpenalty
  \repeat}%
\fi}

```

(2) After the `\output` routine has done

```
\fliptopins@
```

to get `\box\topins` properly set up, it essentially does

```
\shipout\vbox{\makeheadline\pagebody\makefootline}
```

just like the plain `TEX` `\output` routine. As in plain `TEX`, `\pagebody` is not much more than

```
\vbox to\vsizel{\boxmaxdepth=\maxdepth \pagecontents}
```

Actually, the `LATEX` `\pagebody` routine is a little more complicated, because marginal notes for index entries are attached at this point, but for now let's ignore that complication.

`\pagecontents` is the routine where interesting things happen. In plain `TEX`, `\pagecontents` basically stacks up

```

things in \box\topins
contents of \box255
things in \box\footins, preceded by footnote rule

```

But we need a more complicated routine, because if `\box\topins` contains two boxes, we need to put the first box (i.e., the *bottom* box, since we've already applied `\fliptopins@`) at the top of the page, with suitable space below it, and the other box at the bottom.

We start with

```
\def\pagecontents{%
  \ifnum\flipcount@ > 0
    \setbox\topins=\vbox{\unvbox\topins\global\setbox1=\lastbox}
```

to get the bottom box of `\box\topins` into `\box1`.

Then we want to take this `\box1` apart, to get the caption and island boxes that constitute it, together with the glue between them,

```
\setbox0=\vbox{\unvbox1
  \global\setbox1=\lastbox
  \global\skipi@=\lastskip
  \unskip
  \global\setbox3=\lastbox}
```

Here we use the fact that there is just one glue between the two boxes (page 388).

Then we use

```
\centerline{\box3}
\nointerlineskip
\vskip\skipi@
\centerline{\box1}
\vskip\belowtopfigskip
```

to put everything back, with the island box and the caption box both centered, and `\belowtopfigskip` following; the `\nointerlineskip` is used to make certain that no extra glue creeps in.

The original `\box0`, appearing in `\box\topins`, had its height artificially increased by `\abovebotfigskip`. But the individual pieces of this box still

have their correct heights, so the

```
\centerline{\box3}
\nointerlineskip
\vskip\skipi@
\centerline{\box1}
```


causes a net decrease of `\abovebotfigskip` in height, to which we have then added a net increase of `\belowtopfigskip`. As explained on pages 398 and 400, T_EX has already left exactly enough space to allow this change from `\abovebotfigskip` to `\belowtopfigskip`, so it looks like everything should work out just right. Actually, however, we will change the


```
\vskip\belowtopfigskip
```

to

```
\ifdim\ht255 < \minpagesize \else
\vskip\belowtopfigskip \fi
```

for reasons discussed in section 7.

 If our `\island` had no caption, then `\box1` would actually be the `\hbyw`, or other material, `\skipi@` would be `0pt`, and `\box3` would be void. This makes no difference in the current situation, where all the sub-boxes are treated the same, but other styles might have to make explicit checks to see whether `\box0` contained a caption or not.

 It is even conceivable that the style would need to know what class of `\island` was contained in `\box0`. This information would probably best be recorded in a list like the list `\AAlist@` discussed in sections 34.3, 34.4, and 34.8.

Once we've taken care of figures at the top of the page we use

```
\ifdim\ht255 < \minpagesize \vfil \else \unvbox255 \fi
```

to print the text in `\box255`, unless there is too little to go on the page; then we finish the definition off with a routine `\bottomfigs@`, which adds the figure

at the bottom if \box\topins still has a box left in it, after which we add the footnotes, just as in plain T_EX:

```
-----
\def\pagecontents{%
  \ifnum\flipcount@ > 0
    \setbox\topins=\vbox{\unvbox\topins
      \global\setbox1=\lastbox}%
    \setbox0=\vbox{\unvbox1
      \global\setbox1=\lastbox
      \global\skipi@=\lastskip
      \unskip
      \global\setbox3=\lastbox}%
    \centerline{\box3}\nointerlineskip
    \vskip\skipi@
    \centerline{\box1}%
    \ifdim\ht255 < \minpagesize \else
      \vskip\belowtopfigskip \fi
  \fi
  \ifdim\ht255 < \minpagesize \vfil \else \unvbox255 \fi
  \bottomfigs@
  \ifvoid\footins\else
    \vskip\footins\footnoterule\unvbox\footins\fi}

\def\bottomfigs@{%
  \ifnum\flipcount@ > 1
    \nointerlineskip
    \vskip\abovebotfigskip
    \setbox0=\vbox{\unvbox\topins
      \setbox0=\lastbox
      \unvbox0
      \global\setbox1=\lastbox
      \global\skipi@=\lastskip
      \unskip
      \global\setbox3=\lastbox}%
  \fi
}
```

```

\centerline{\box3}\nointerlineskip
\vskip\skipi@
\centerline{\box1}%
\fi}

```

Note that the `\nointerlineskip` at the very beginning of the definition of `\bottomfigs@` is quite essential: without it, extra glue would be added before the `\centerline{\box3}`. Even extra `\lineskip` glue might not fit at this stage, but, in fact, the glue might be bigger, because `\box3` could be empty (for example, an empty caption above a table), in which case `\baselineskip` glue would be used!

bd Since we went to the trouble of taking apart `\box\islandbox@`, in order to allow the glue between the island and the caption to stretch or shrink on the page, we might also want to take apart `\box\Captionbox@`, so that the glue between multiple captions could also participate.

bd When we have an `\island` at top, followed by material from `\box255`,

```

. . .
\centerline{\box1}
\vskip\belowtopfigskip
\unvbox255
. . .

```

the first line of `\box255` will have glue above it determined by the value of `\topskip`; for example, in the default style there will usually be enough glue to make the height of the first line be 10pt. So the space between the bottom of `\box1` and the baseline of this first line will be `\belowtopfigskip + 10pt`. If we were being fussy, we might increase `\belowtopfigskip` by 2pt to compensate for this. However, even that wouldn't work precisely if `\box1` has some depth.

Probably none of this matters much when `\belowtopfigskip` is a fairly large value, but a more precise routine could be made: for `\island`'s like `\Figure`, with their caption below, in the `\endisland` routine (page 389) we could add

```
\kern-\prevdepth
```

after the `\box\Captionbox@` in the final `\vbox`

(3) After the `\shipout`, we will use

```
\advancepageno
```

to advance the page number.

(4) Now that `\pagecontents` has properly arranged the boxes in `\topins` on the page, the `\output` routine must set things up properly for the next page. First of all

```
\global\advance\topinscount@ by -\flipcount@
```

insures that `\topinscount@` will still represent the number of elements in the `\topins` insertion class. Taking care of `\dimen\topins` requires more work.

At the time that the `\output` routine was called, `\dimen\topins` was the height plus depth, plus extra `\abovebotfigskip` space, for the first two boxes in the `\topins` insertion class. After the `\output` routine is done, we have to reset `\dimen\topins` to the same quantity for the first two boxes that remain in this class.

Remember that `\box\topinsdims@` is a `\vbox` of empty `\hbox`'s whose widths are the appropriate dimensions for the various boxes in the `\topins` insertion class. Since these boxes were added at the *top* (page 401), the bottom box has the dimension for the first box in `\topins`, and so forth. In order to keep our information current, we will remove boxes from the bottom, the number of boxes we remove being `\flipcount@`, and reset `\dimen\topins` to the sum of the widths of the next two bottom boxes: this will ensure that on the next page, at most two of the next boxes in `\topins` will actually be allowed to appear in `\box\topins`.

To do this, we first

```
\global\setbox\topinsdims@=
\vbox{%
\unvbox\topinsdims@
\count@=0
\loop
\ifnum\count@ < \flipcount@
\setbox0=\lastbox
\advance\count@ by 1
\repeat
```

In other words, the new `\box\topinsdims@` will start out with all the boxes originally in this box (from the `\unvbox\topinsdims@`); and then we will use `\setbox0=\lastbox` the appropriate number of times to get rid of `\flipcount@` boxes at the bottom.

Before finishing off this box, we need to set `\dimen\topins` to the sum of the widths of the next two bottom boxes. The only way we can get at these widths is by using `\lastbox`; since we don't want to throw away these next two boxes, however, we will store them in `\box1`, and then `\unvbox1` before completing the box. For purposes of generalization later on, we will write this as a `\loop` also:

```

\dimen@=0pt
\count@=0
\setbox2=\vbox{ }
\loop
  \ifnum\count@ < 2
    \setbox0=\lastbox
    \advance\dimen@ by \wd0
    \setbox2=\vbox{\box0 \unvbox2}
    \advance\count@ by 1
  \repeat
\unvbox2
\global\dimen\topins@=\dimen@

```

The whole routine will be called `\resetdimtopins@`:

```

-----
\def\resetdimtopins@{%
\global\advance\topinscount@ by -\flipcount@
\global\setbox\topinsdims@=
\vbox{%
\unvbox\topinsdims@
\count@=0
\loop
\ifnum\count@ < \flipcount@ \setbox0=\lastbox
\advance\count@ by 1
\repeat

```



```

\dimen@=0pt
\count@=0
\setbox2=\vbox{}%
\loop
\ifnum\count@ < 2
\setbox0=\lastbox
\advance\dimen@ by \wd0
\setbox2=\vbox{\box0 \unvbox2}%
\advance\count@ by 1
\repeat
\unvbox2
\global\dimen\topins=\dimen@}}

```

We aren't going to formally define an `\output` routine for our specialized style, saving for Chapter 36 numerous additional details (some discussed in the next section of this chapter). But here is a summary of the main points:

- (1) The `\output` routine first uses `\fliptopins@` to get `\box\topins` into the right order, computing `\flipcount@` in the process.
- (2) Then it ships out a `\vbox` whose main part is `\pagecontents`, which we have already considered in detail.
- (3) At this point, when any delayed `\write's` have been done, we can use

```
\advancepageno
```

to advance the page number.

- (4) Then we use

```
\resetdimtopins@
```

to properly adjust `\topinscount@` and `\dimen\topins` for the next page.

- (5) Unlike the plain routine, where `\box255` is always used up, our routine will not use `\box255` if its height is less than `\minpagesize`. So at this point we use

```
\ifvoid 255 \else \unvbox255 \penalty\outputpenalty
```

This puts the material in `\box255` back on the top of the list, together with whatever `\penalty` precedes it, since this was recorded in `\outputpenalty` (*The T_EXbook*, pages 125 and 254).

(6) Then we end, as in plain T_EX, with

```
\ifnum\outputpenalty > -20000 \else \dosupereject \fi
```

33.6. When insertions float. Although the theory behind our definition of `\Aplace` may seem fine, it has one annoying flaw that becomes particularly significant when we are dealing with straight text, unrelieved by displayed formulas, or other material that introduces shrink or stretch on the page.

To take a particular example, let us suppose that we have set

```
\parskip=0pt
\topskip=10pt
\baselineskip=12pt
\vsizе=478pt (= 39 × 12pt + 10pt)
```

so that exactly 40 lines of text will fit on a page, without any stretch or shrink allowed between paragraphs. (Styles with such specifications usually arrange for the heights of any Chapter headings, etc., to be an integer multiple of `\baselineskip`. When illustrations are to be included, `\belowtopfigskip` and `\abovebotfigskip` should probably have stretch and shrink at least half of `\baselineskip`, unless care is taken to insure that the heights of all `\island`'s, when added to these `\vskip`'s, are also an integer number times `\baselineskip`. However, for purposes of illustration, we will simply continue to use our default values of `\belowtopfigskip` and `\abovebotfigskip`.)

Now suppose that near the bottom of page 2 we `\Aplace` something that will not fit on that page, so that the `\insert` will split. If `'\showbox\topins'` is added to the `\output` routine, then before page [1] is shipped out, the `.log` file will contain the following information about `\box\topins` (which happens to be `\box253`):

```
> \box253=void
```

But before page [2] is shipped out, we will have

```
> \box253=
  \vbox(0.0+0.0)x0.0
```

this `\vbox` being the part that is split off from the

```
\insert\topins{\penalty0 ... }
```

This might seem like an insignificant difference, but it's not: Since `TEX` has added something (albeit something trivial) to `\box\topins`, it has also allowed for `\skip\topins=-3pt` glue. If `'\showbox255'` is added to the `\output` routine, then the log file will have

```
> \box255=
  \vbox(481.0+...)x...
  .\glue(\topskip)
  . . .
```

In other words, `\box255` is now 3pt *too high*. In our particular example, it will probably be an Underfull box. `TEX` doesn't report Underfull boxes when `\box255` is packaged for use by the `\output` routine (*The TEXbook*, page 400), and when we `\unvbox255`, everything will probably still turn out all right.

But if `\skip\topins` were as large as 12pt, then we would definitely have an extra line on our page. Consequently, the `\output` routine will have to take care to trim `\box255` down to the proper height, and anything left over will have to be put back on the main vertical list at the end of the `\output` routine. (Such a final step in the `\output` routine will be needed for other reasons also; for example, when there are figures at both the top and bottom of the page, and the space between is less than `\minpagesize`, so that no text is allowed to appear on the page, everything in `\box255` has to be put back).

Notice that if `\belowtopfigskip` were *greater* than `\abovebotfigskip`, then our problem would be much more severe: In this case, `\box255` would be *too short*, and there would be no way to rectify the situation. This is discussed further in section 36.5.

33.7. What happens to an \Hbyw? When we have an \island involving an \Hbyw, the height of \box0 will have been increased so that it is 18pt larger than \vsize. T_EX has chosen \box255 so that

$$\begin{aligned} \vsize &= \text{height of } \box255 + \text{height of } \box0 + \vskip\topins \\ &= \text{height of } \box255 + \vsize + 18\text{pt} - 3\text{pt} \\ &= \text{height of } \box255 + \vsize + 15\text{pt} \end{aligned}$$

So \box255 will be an empty box of height -15pt (see below for further elucidation), which would make things come out just right when it is added back in with the other material. But we won't be adding the material from \box255 in this situation, since its height is less than \minpagesize;¹ consequently, if we were to add \vskip\belowtopfigskip in this case, we would get an Overfull box that is 15pt too high. That is why we had to use

```
\ifdim\ht255 < \minpagesize \else
\vskip\belowtopfigskip \fi
```

In our final definition, page 457, we will actually use more refined emendations.

This apparently anomalous situation, where \box255 has negative height, more or less follows from the rules on page 123 of *The T_EXbook*. The first time that an \Hbyw insertion is considered, it will be split, according to Step 4, at the \penalty0, leaving a box of height \vsize + 15pt. When the next page is made, this box will definitely be added, at Step 1, which will decrease the \pagegoal g by this height, so that it is now -15pt.

Note that, as a consequence, an \Hbyw will always appear at least one page after it is \Aplace'ed, even if there is nothing else on the page at the time.

¹ When we \unvbox255 at the end of the \output routine—see (5) on page 415, and page 451—to put unused material from \box255 back at the top of the current page, the contents of such an empty box will simply disappear, since T_EX discards such glue at the top of the current page.

Chapter 34. \Aplace, \AAplace and \Bplace

We are now ready to consider the real definition of \Aplace, together with \AAplace and \Bplace.

Unlike \Cplace, \Mplace and \MXplace, which can only be used between paragraphs, or in a special \Par... \endPar region, \Aplace, \AAplace, and \Bplace can be used directly within paragraphs. Nevertheless, it is conceivable that the latter will also be used in a \Par... \endPar region (probably because some \Cplace, \Mplace, or \MXplace is also being used in that region). So we first have to take care of some preliminaries related to \Par... \endPar regions.

The \LaTeX Manual states, on page 67, that blank lines within a \Par... \endPar region will give an error message, and this was indeed true in version 1 of \LaTeX . But there is really no need for this, and the implementation was quite imperfect, so that feature has been dropped in version 2.

34.1. Figures, etc., within \Par... \endPar First we need a flag to tell whether we are in such a paragraph, a counter to tell how many times a \...place occurred within this paragraph, and a box in which to store the text of such a paragraph:

```
\newif\ifPar@
\newcount\Parcount@
\newbox\Parbox@
```

The \island's that are \...place'ed within such regions will be stored in other boxes,

```
\expandafter\newbox\csname Parfigbox1\endcsname
\expandafter\newbox\csname Parfigbox2\endcsname
\expandafter\newbox\csname Parfigbox3\endcsname
\expandafter\newbox\csname Parfigbox4\endcsname
\expandafter\newbox\csname Parfigbox5\endcsname
```

Creating these boxes with \csname... \endcsname is easier and more convenient than trying to make a '\Parfigbox' macro with an argument.

It turns out that we are also going to need `<dimen>` registers to store the depths of various paragraphs:

```
\expandafter\newdimen\csname Parprev1\endcsname
\expandafter\newdimen\csname Parprev2\endcsname
\expandafter\newdimen\csname Parprev3\endcsname
\expandafter\newdimen\csname Parprev4\endcsname
\expandafter\newdimen\csname Parprev5\endcsname
\expandafter\newdimen\csname Parprev6\endcsname
```

We need six `<dimen>` registers because `'\Parprev1'` will store the depth of the paragraph before the `\Par... \endPar` region, `'\Parprev2'` will store the depth of the text before the first `\island`, etc. As before (compare pages 14, 29, 75, and 128), we use quotes around `\Parprev1` to indicate that it is a single control word.

In addition to all this, within each `\Par... \endPar` region we will be keeping a list,

```
\Parlist@
```

which will simply be a list of letters.

The definition of `\Par` begins by ending the previous paragraph, storing the depth of the last line of that paragraph in `'\Parprev1'`, setting `\ifPar@` to be true, and initializing `\Parcount@` to be 1 and `\Parlist@` to be empty:

```
\par
\csname Parprev1\endcsname=\prevdepth
\Par@true
\global\Parcount@=1
\global\let\Parlist@=\empty
```

Then we begin to set `\box\Parbox@`,

```
\setbox\Parbox@=\vbox\bgroup
```

We want this `\vbox` to begin with a `\break` for latter purposes (section 35.4). Our whole definition is:

```

\def\Par{\par
\csname Parprev1\endcsname=\prevdepth
\global\Parcount@=1
\Par@true\global\let\Parlist@=\empty
\global\setbox\Parbox@=\vbox\bgroup\break}

```

The definition of `\endPar` will not be given until section 35.4.

34.2. `\place@`. The constructions `\Aplace`, `\AAplace`, and `\Bplace` have much in common, and they are going to be defined in terms of a control sequence `\place@#1#2` with a rather strange combinations of arguments:

```

\Aplace   will be defined in terms of \place@ a\Aplace@
\AAplace  will be defined in terms of \place@ A\AAplace@
\Bplace   will be defined in terms of \place@ b\Bplace@
. . .

```

Here the first argument is simply a convenient single character marker, whose role will eventually become clear; the second argument, `\Aplace@`, `\AAplace@`, or `\Bplace@`, will then do the main work.

As an example of how this is going to work out, let's consider the definition of `\Aplace`, which will actually come later:

```

\def\Aplace#1{\prevanish@
\place@true \island@false
#1%
\place@ a\Aplace@
\postvanish@}

```

where we begin with `\prevanish@` and end with `\postvanish@` to make the `\Aplace` invisible.

The definition of `\place@` will be of the form

```

\def\place@#1#2{%
\ifisland@

```

```

. . .
\else
\Err@{Whoa ... there's no \string\Figure,
\string\Table, etc., here}
\fi
\place@false}

```

to produce an error message if the argument of `\Aplace` didn't involve some sort of `\island`.

If we are in vertical mode, we will just call argument #2, but in horizontal mode we call `\vadjust{#2}`. For example, we will call `\Aplace@` when we are using `\Aplace` in vertical mode and

```
\vadjust{\Aplace@}
```

in horizontal mode.

So far we haven't even mentioned argument #1. And, quite concomitantly, we haven't mentioned what happens if we happen to be in a `\Par... \endPar`. In this case, we want to globally set the box '`\Parfigbox1`' to be `\box\islandbox@` if this is the first `\...place@`, or globally set the box '`\Parfigbox2`' to `\box\islandbox@` if it is the second, etc. We will keep track of which `\...place@` we are at with the counter `\Parcount@`, which was set to 1 by the `\Par`, and which will be increased with each `\...place@`. So the box we want to be setting is

```
\csname Parfigbox\number\Parcount@\endsname
```

and to globally set this box to `\box\islandbox@` we need to say

```

\expandafter\expandafter\expandafter
\global\expandafter\setbox
\csname Parfigbox\number\Parcount@\endcsname=\box\islandbox@

```

followed by

```
\global\advance\Parcount@ by 1
```


to keep the \Parcount@ counter accurate.

And, finally, here is where argument #1 comes in. When it comes time to properly tear apart \box\Parbox@, we will need to know what sort of \dotsplace's produced the various \Parfigbox's. We keep this information in the list \Parlist@, which, for example, will be 'abaAb' when our \Par...\endPar region contains the sequence

```

\Aplace
\Bplace
\Aplace
\AApalce
\Bplace

```

Each time we \place@, we will simply add the marker, argument #1, at the right of \Parlist@,

```
\xdef\Parlist@{\Parlist@#1}
```

The only other detail is that an error message should be produced if more than five \dotsplace's appear in the \Par...\endPar region. Here is the complete code:

```

\def\place@#1#2{%
  \ifisland@
  \ifhmode
  \ifPar@
  \ifnum\Parcount@ > 5
    \Err@{Only 5 \string\place's allowed per
      \string\Par...\noexpand\endPar paragraph}%
  \else
    \expandafter\expandafter\expandafter
    \global\expandafter\setbox
    \csname Parfigbox\number\Parcount@\endcsname
    =\box\islandbox@
    \global\advance\Parcount@ by 1
  \xdef\Parlist@{\Parlist@#1}%
  \fi
}

```

```

\else
\adjust{#2}%
\fi
\else
#2%
\fi
\else
\Err@{Whoa ... there's no \string\Figure,
\string\Table, etc., here}%
\fi
\place@false}

```

34.3. `\Aplace` and `\AAplace`. As we have already indicated, `\Aplace` is defined directly in terms of `\place@`,

```

\long\def\Aplace#1{\prevanish@ \place@true \island@false
#1%
\place@ a\Aplace@
\postvanish@}

```

so that it calls `\Aplace@`. The `\long` is needed since `#1` can contain a `\caption` with blank lines.

The definition of `\Aplace@` will then involve most of the preliminary definition of `\Aplace` on page 404:

```

\dimen@=\ht\islandbox@
\advance\dimen@ by \abovebotfigskip
\ht\islandbox@=\dimen@
\advance\dimen@ by \dp\islandbox@
\storedim@
\ifnum\topinscount@ > 1 \else \advancedimtopins@ \fi
\insert\topins{\penalty0 \splittopskip=0pt
\floatingpenalty=0
\box\islandbox@}
\global\advance\topinscount@ by 1

```

But `\Aplace@` is going to be more complicated, for several reasons. First of all, we are going to define `\AAplace` indirectly in terms of `\Aplace@` by using a flag `\ifAA@`:

```
\long\def\AAplace#1{\prevanish@ \place@true \island@false
#1%
\place@ A\AAplace@
\postvanish@}

\newif\ifAA@

\def\AAplace@{\AA@true\Aplace@\AA@false}
```

Even if we weren't going to use this approach, we would still need a way of knowing whether a box in the `\topins` insertion class was produced with `\Aplace` or `\Bplace` on the one hand, or by `\AAplace` on the other hand. We will keep this information in a list `\AAlist@`, initially defined to be empty, with 0 representing the `\Aplace` or `\Bplace` case, and 1 representing the `\AAplace` case. When `\ifAA@` is true, so that we are considering an `\AAplace`, we will add 1 to the end of this list, otherwise we will add 0. Moreover, in the `\AAplace` case, we always want to call the `\advancedimtopins@` routine, no matter what value `\topinscount@` has, since `\AAplace`'s are supposed to allow any number of `\island`'s per page.

Finally, we want the definition of `\Aplace@` to begin with an `\allowbreak`, because this may help `TEX` check whether the insertion fits.

Reason: When a `\vadjust{\Aplace@}` occurs within a paragraph, the main vertical list will have something like

```
\hbox{line  $L_1$  of text}
\Aplace@
\baselineskip glue
\hbox{line  $L_2$  of text}
```

Suppose that the line L_1 won't fit on the page, so that `TEX` will break the page before this line, which will be at, or near, the top of the next page. `TEX` computes the page total t at line L_1 after a penalty or glue that follows this line (*The T_EXbook*, page 133, lines 17–18). Consequently, if the `\Aplace@` contains no penalty or glue, `TEX` will read the `\Aplace@` before it knows that line L_1 must go on the next page; thus, it will conclude that the insertion doesn't fit (near the bottom of the current page), instead of

concluding that it does fit (near the top of the next page). The `\allowbreak` causes TeX to compute the page total t before it sees the `\Aplace@`, so that it will realize in time that the line doesn't fit. (The procedure isn't foolproof, however; even if line L_1 does fit, TeX may end up breaking before this line if a break point with smaller cost occurred earlier).

The full definition of `\Aplace@` is:

```

\let\AAlist@=\empty
\def\Aplace@{\allowbreak
  \dimen@=\ht\islandbox@
  \advance\dimen@ by \abovebotfigskip
  \ht\islandbox@=\dimen@
  \advance\dimen@ by \dp\islandbox@
  \storedim@
  \ifAA@
    \xdef\AAlist@{\AAlist@1}%
    \advancedimtopins@
  \else
    \xdef\AAlist@{\AAlist@0}%
    \ifnum\topinscount@>1 \else \advancedimtopins@ \fi
  \fi
  \insert\topins{\penalty0 \splittopskip=0pt
    \floatingpenalty=0
    \box\islandbox@}%
  \global\advance\topinscount@ by 1 }

```

34.4. `\Bplace`. The definition of `\Bplace` is analogous to that for `\Aplace`:

```

\long\def\Bplace#1{\prevanish@ \place@true \island@false
  #1%
  \place@ b\Bplace@
  \postvanish@}

```

with all the work going into the definition of `\Bplace@`.

A \Bplace is supposed to force an \island to the bottom of the page if it would normally go at the top of the current page. In other words

```
\ifnum\topinscount@=0
```

we need to force the \island to the bottom of the page. To do this, we will basically first \Aplace an island consisting of an empty box

```
\vbox to -\belowtopfigskip{}
```

of negative height $-\text{\belowtopfigskip}$, so that when this island is followed by $\text{\vskip\belowtopfigskip}$ we will be back at the top of the page. We actually want to our island to be a box of the form

```
\vbox{\vbox to -\belowtopfigskip{}}
```

so that it can be taken apart, like any other \box\islandbox@ , with our \unvbox 'ing mechanism, so we start with

```
\setbox0=\vbox{\vbox to -\belowtopfigskip{}}
```

The next steps,

```
\dimen@=\ht0
\advance\dimen@ by \abovebotfigskip
```

set \dimen@ to $\text{\abovebotfigskip} - \text{\belowtopfgiskip}$, and are thus equivalent to

```
\dimen@=-\skip\topins (c.f. page 399)
```

after which we want to set

```
\ht0=\dimen@
```

and then

```

\storedim@
\advancedimtopins@
\insert\topins{\box0}
\global\advance\topinscount@ by 1

```

as well as

```

\xdef\AAlist@{\AAlist@0}

```

This should then be followed by the instructions for \Aplace'ing the \box\islandbox@ that we want to force to the bottom:

```

\def\Bplace@{\allowbreak
\ifnum\topinscount@=0
\setbox0=\vbox{\vbox to -\belowtopfigskip{}}%
\dimen@=-\skip\topins
\ht0 =\dimen@
\storedim@
\advancedimtopins@
\insert\topins{\box0}%
\global\advance\topinscount@ by 1
\xdef\AAlist@{\AAlist@0}%
\fi
\dimen@=\ht\islandbox@
\advance\dimen@ by \abovebotfigskip
\ht\islandbox@=\dimen@
\advance\dimen@ by \dp\islandbox@
\storedim@
\xdef\AAlist@{\AAlist@0}%
\ifnum\topinscount@>1 \else \advancedimtopins@ \fi
\insert\topins{\penalty0 \splittopskip=0pt
\floatingpenalty=0
\box\islandbox@}%
\global\advance\topinscount@ by 1 }

```

34.5. Changing \pagecontents. Our definition of \Bplace forces the first change in our preliminary definition of \pagecontents (page 411). Recall that we first

```
\setbox\topins=\vbox{\unvbox\topins
\global\setbox1=\lastbox}
```

so that \box1 now contains the material we want to put on top, and then took \box1 apart with

```
\setbox0=\vbox{\unvbox1
\global\setbox1=\lastbox
\global\skipi@=\lastskip
\unskip
\global\setbox3=\lastbox}
```

In the case where \box1 is

```
\vbox{\vbox to -\belowtopfigskip{}}
```

the new \box1 will be \vbox to -\belowtopfigksip{}, while \skipi@ will be 0pt and \box3 will be empty (compare the small print remark on page 410). The problem now is that when we add the combination

```
\centerline{\box3}
\nointerlineskip
\vskip\skipi@
\centerline{\box1}
```

the last \hbox,

```
\centerline{\box1}=(\hbox to\hsize{...})
```

will not have the negative height -\belowtopfigskip that we want it to have: an \hbox never has negative height or depth (*The T_EXbook*, page 77). So we

must change this to

```
\centerline{\box3}
\nointerlineskip
\vskip\skipi@
\ifdim\ht1 < 0pt \box1 \else \centerline{\box1}\fi
```

34.6. `\breakisland@` and `\printisland@`. The construction by which we just took apart `\box1` will appear several times, so we introduce the abbreviation

```
\def\breakisland@{\global\setbox1=\lastbox
\global\skipi@=\lastskip
\unskip
\global\setbox3=\lastbox}%
```

Similarly, for the combination that puts the `\island` back on the page we introduce the abbreviation

```
\def\printisland@{\centerline{\box3}%
\nobreak
\nointerlineskip
\vskip\skipi@
\ifdim\ht1 < 0pt \box1 \else \centerline{\box1}\fi}
```

Of course, the final `\ifdim... \fi` clause will normally be equivalent to `\centerline{\box1}`, and will be operative only when our macros have created a `\box1` of negative height. Similarly, although the added `\nobreak` will sometimes be needed (page 439), at other times it will be superfluous.

bo For other styles, which treat `\island`'s differently, a completely different `\printisland@` routine might be in order; in such cases `\endisland` may combine `\box\islandbox@` and `\box\Captionbox@` in an entirely different way.

34.7. `\bottomfigs@`. The possibility of `\AAplace`'s, and thus more than two `\island`'s per page, forces us to change the preliminary version of `\bottomfigs@` (page 411) to a loop. We also use the newly defined routines `\breakisland@` and `\printisland@` in the definition.

```

\def\bottomfigs@{%
  \count@=1
  \loop
  \ifnum\count@ < \flipcount@
  \nointerlineskip
  \vskip\abovebotfigskip
  \setbox\topins=\vbox{\unvbox\topins\setbox0=\lastbox
    \unvbox0
    \breakisland@}%
  \printisland@
  \advance\count@ by 1
  \repeat}

```

As we will see later (page 439), there may be occasions during the use of this `\bottomfigs@` routine when the `\box1` in the `\printisland@` routine has negative height.

34.8. `\resetdimtopins@`. We also have to change the definition of the routine `\resetdimtopins@`. Recall that at the time `\resetdimtopins@` is called, `\box\topinsdims@` will be a `\vbox` of the form

```

. . .
\hbox to <dimen3>{}
\hbox to <dimen2>{}
\hbox to <dimen1>{}

```

where `<dimen1>` is the width of the first box that was still in the `\topins` insertion class just before this page was made, `<dimen2>` is the width of the second such box, etc. Of these boxes in the `\topins` insertion class, `\flipcount@` actually appeared in `\box\topins`, and were thus put on the page; the first thing we do is to diminish `\topinscount@` by `\flipcount@`, so that `\topinscount@` is then the number of boxes still in the insertion class.

At this time, the list `\Aalist@` (of 0's and 1's) will have as many members

as `\box\topinsdims@`. As we perform the next step,

```
\global\setbox\topinsdims@=\vbox{%
  \unvbox\topinsdims@
  \count@=0
  \loop
    \ifnum\count@ < \flipcount@ \setbox0=\lastbox
    \advance\count@ by 1
  \repeat
```

which removes the bottom `\flipcount@` boxes from `\box\topinsdims@`, we are also going to want to remove the first `\flipcount@` members from (the left end of) `\AAlist@`. To do this, we use

```
\global\setbox\topinsdims@=\vbox{%
  \unvbox\topinsdims@
  \count@=0
  \def\next@##1##2\next@{\gdef\AAlist@{##2}}
  \loop
    \ifnum\count@ < \flipcount@ \setbox0=\lastbox
    \expandafter\next@\AAlist@\next@
    \advance\count@ by 1
  \repeat
```

The next `\loop` in the definition,

```
\loop
  \ifnum\count@ < 2
  \setbox0=\lastbox
  \advance\dimen@ by \wd0
  \setbox2=\vbox{\box0 \unvbox2}
  \advance\count@ by 1
\repeat
```

requires more modifications. After the second iteration of the `\loop`, we want to continue as long as the boxes we encounter correspond to `\AAplace'd`

material. To determine whether that is the case, we

```
\edef\nextiii@{\AAlist@}
\def\next@##1##2\next@{\def\nextii@{##1}\def\nextiii@{##2}}
```

so that `\expandafter\next@\nextiii@\next` defines `\nextii@` to be the next element of `\AAlist@`, and removes it from the copy `\nextiii@` of `\AAlist@` (we use a copy because we don't want these elements to be removed once the `\vbox` is finished).

Our `\loop` will iterate at least two times and at most `\topinscount@` times. For values of `\count@` in between, we continue the iteration only when `\nextii@` is 1:

```
\loop
  \test@false
  \ifnum\count@ < \topinscount@
    \expandafter\next@\nextiii@\next@
    \ifnum\count@ < 2
      \test@true
    \else
      \if\nextii@ 1\test@true\fi
    \fi
  \iftest@
    \setbox0=\lastbox
    \advance\dimen@ by \wd0
    \setbox2=\vbox{\box0 \unvbox2}%
    \advance\count@ by 1
  \repeat
```

Thus, our whole (and final) definition is:

```
\def\resetdimtopins@{%
  \global\advance\topinscount@ by -\flipcount@
  \global\setbox\topinsdims@=\vbox
  {\unvbox\topinsdims@
   \count@=0
   \def\next@##1##2\next@{\gdef\AAlist@{##2}}}%
```

```

\loop
  \ifnum\count@ < \flipcount@ \setbox0=\lastbox
  \expandafter\next@\AAlist@\next@
  \advance\count@ by 1
  \repeat
\dimen@=0pt
\count@=0
\setbox2=\vbox{}%
\edef\nextiii@{\AAlist@}%
\def\next@##1##2\next@{\def\nextii@{##1}%
  \def\nextiii@{##2}}%
\loop
  \test@false
  \ifnum\count@ < \topinscount@
  \expandafter\next@\nextiii@\next@
  \ifnum\count@ < 2
  \test@true
  \else
  \if\nextii@ 1\test@true\fi
  \fi
  \fi
  \iftest@
  \setbox0=\lastbox
  \advance\dimen@ by \wd0
  \setbox2=\vbox{\box0 \unvbox2}%
  \advance\count@ by 1
  \repeat
\unvbox2
\global\dimen\topins=\dimen@}

```

Chapter 35. \Cplace, \Mplace, and \MXplace

Next we consider the variants \Cplace, \Mplace, and \MXplace, which must be used between paragraphs, or in the special \Par... \endPar construction.

35.1. \Place@. Just as \Aplace, \AAplace, and \Bplace are defined in terms of \place@, the \Cplace, \Mplace, \MXplace constructions are defined in terms of \Place@. This is similar to \place@, except that (1) in horizontal mode if we are not in a \Par... \endPar region, we will give an error message rather than using a \vadjust, and (2) if we are in a \Par... \endPar region we will add a \vadjust{\break}:

```
\def\Place@#1#2{%
  \ifisland@
  \ifhmode
  \ifPar@
  \ifnum\Parcount@ > 5
    \Err@{Only 5 \string\place's allowed per
      \string\Par...\noexpand\endPar paragraph}%
  \else
    \expandafter\expandafter\expandafter
    \global\expandafter\setbox
    \csname Parfigbox\number\Parcount@\endcsname
    =\box\islandbox@
    \global\advance\Parcount@ by 1
    \xdef\Parlist@{\Parlist@#1}%
    \vadjust{\break}%
  \fi
  \else
    \Err@{\noexpand#2allowed only in a
      \string\Par...\noexpand\endPar paragraph}%
  \fi
  \else
    #2%
  \fi
}
```

```

\else
  \Err@{Whoa ... there's no \string\Figure, \string\Table,
    etc., here}%
\fi
\place@false}

```

Note that the `\adjust{\break}` occurs within the `\box\Parbox@` that we are setting; it will be used (in section 4) when we pull the box apart.

35.2. `\Cplace@`. We first declare a new flag and a new `<dimen>` register:

```

\newif\ifC@
\newdimen\Cdim@

```

The definition of `\Cplace` is analogous to all the previous `\...place's`,

```

\long\def\Cplace#1{\prevanish@ \place@true \island@false
#1%
\Place@ c\Cplace@
\postvanish@}

```

leaving all the work to `\Cplace@`.

`\Cplace@` basically just calls `\Aplace@`, except that when `\topinscount@` is 0, so that there are no insertions waiting to go on the page, it sets the flag `\ifC@` to be true, and the `<dimen>` `\Cdim@` to `\pagetotal`; this information will then be used by the `\output` routine.

```

\def\Cplace@{\allowbreak
\ifnum\topinscount@ > 0
\else
  \global\C@true
  \global\Cdim@=\pagetotal
\fi
\Aplace@}

```

Here we start with `\allowbreak`, even though this occurs in the `\Aplace@`, because it might give `TEX` the opportunity to call the `\output` routine, which will reset `\topinscount@`.

35.3. \Mplace@ and \MXplace@. The \Mplace and \MXplace constructions are going to be defined together, much like \Aplace and \AAplace. First we define

```
\long\def\Mplace#1{\prevanish@ \place>true \island>false
#1%
\Place@ m\Mplace@
\postvanish@}
```

so that the work goes into defining \Mplace@.

Similarly, we define

```
\long\def\MXplace#1{\prevanish@ \place>true \island>false
#1%
\Place@ M\MXplace@
\postvanish@}
```

where we define \MXplace@ in terms of \Mplace@ and a flag:

```
\newif\ifMX@
\def\MXplace@{\MX@true\Mplace@\MX@false}
```

The definition of \Mplace@ begins with \allowbreak,

```
\def\Mplace@{\allowbreak . . .
```

just as with other \...place's. Then, however, it is quite different, because we must calculate whether or not \box\islandbox@ will fit on the current page, together with \abovebotfigskip glue above it (there is no need to consider the glue below the box, since it is allowed to disappear at a page break). Actually, \abovebotfigskip is merely supposed to be the minimum amount of glue before the box: if \lastskip is less than \abovebotfigskip, then we will \unskip before adding this glue, and if \lastskip is equal or greater to \abovebotfigskip, we will do nothing. Moreover, this glue will disappear if there is nothing else on the current page.

We begin with

```

\dimen@=\ht\islandbox@
\advance\dimen@ by \dp\islandbox@
\ifdim\pagetotal=0pt
\else
\ifdim \lastskip < \abovebotfigskip
\advance\dimen@ by \abovebotfigskip
\advance\dimen@ by -\lastskip
\fi
\fi

```

so that `\dimen@` is the amount of space required for the box.

Then we

```

\advance\dimen@ by \pagetotal

```

and use the test

```

\ifdim\dimen@ > \pagegoal

```

If this test is true, then there is *not* enough space for the box, so we use

```

\Aplace@

```

to convert the `\Mplace` to an `\Aplace`.

On the other hand, when the test

```

\ifdim\dimen@ > \pagegoal

```

is false, so that `\box\islandbox@` *does* fit, we basically want to treat the

\Mplace like a \place, except for adding the requisite space:

```

\nointerlineskip
\ifdim\lastskip < \abovebotfigskip
  \removelastskip
  \vskip\abovebotfigskip
\fi
\setbox0=\vbox{\unvbox\islandbox@
  \breakisland@}
\printisland@

```

Unlike the situation for \bottomfigs@, which simply adds things to the \vbox created by \pagebody, the \nobreak appearing in the definition of \printisland@ is required here, when we are simply contributing things to the main vertical list.

Now we have to make sure that no later \Aplace will end up at the top of this page, and hence out of order. We handle this in much the same way as \Bplace, by essentially \Aplace'ing a box of height $-\text{\belowtopfigskip}$ if \topinscount@ is 0:

```

\ifnum\topinscount@=0
  \setbox0=\vbox{\vbox to-\belowtopfigskip{}}
  \dimen@ = -\skip\topins
  \ht0 = \dimen@
  \storedim@
  \advancedimtopins@
  \insert\topins{\box0}
  \global\advance\topinscount@ by 1
  \xdef\AAlist@{\AAlist0}
\fi

```

Finally, suppose that \ifMX@ is true. Then we are considering an \MXplace, which means that we want to prohibit an \island from appearing at the bottom of the page. To do this, we essentially want to \Aplace a box of height $-\text{\abovebotfigskip}$ if \topinscount@ is 1. (This box of negative height will be contributed to the page by \bottomfigs@, a circumstance alluded to on page 431.)

So we start with

```
\setbox0=\vbox{\vbox to-\abovebotfigskip{}}
```

In this case, making the height of \box0 bigger by \abovebotfigskip is just the same as making it Opt, and we set \dimen@ to this value before the \storedim@:

```
\ht0=Opt
\dimen@=Opt
\storedim@
\advancedimtopins@
\insert\topins{\box0}
\global\advance\topinscount@ by 1
\xdef\AAlis@{\AAlis@}
```

And after all that, we add

```
\nointerlineskip
\vskip\belowtopfigskip
```

below our \Mplace'd \island:

```
\def\Mplace@{\allowbreak
\dimen@=\ht\islandbox@
\advance\dimen@ by \dp\islandbox@
\ifdim\pagetotal=Opt \else
\ifdim\lastskip < \abovebotfigskip
\advance\dimen@ by \abovebotfigskip
\advance\dimen@ by -\lastskip
\fi
\fi
\advance\dimen@ by \pagetotal
\ifdim\dimen@ > \pagegoal
\Aplace@
```

```

\else
\nointerlineskip
\ifdim\lastskip < \abovebotfigskip
\removelastskip
\vskip\abovebotfigskip
\fi
\setbox0=\vbox{\unvbox\islandbox@
\breakisland@}%
\printisland@
\ifnum\topinscount@=0
\setbox0=\vbox{\vbox to-\belowtopfigskip{}}%
\dimen@=-\skip\topins
\ht0=\dimen@
\storedim@
\advancedimtopins@
\insert\topins{\box0}%
\global\advance\topinscount@ by 1
\edef\AAlist@{\AAlist@0}%
\fi
\ifMX@
\ifnum\topinscount@=1
\setbox0=\vbox{\vbox to-\abovebotfigskip{}}%
\ht0=0pt
\dimen@=0pt
\storedim@
\advancedimtopins@
\insert\topins{\box0}%
\global\advance\topinscount@ by 1
\edef\AAlist@{\AAlist@0}%
\fi
\fi
\nointerlineskip
\vskip\belowtopfigskip
\fi}

```

35.4. \endPar. An \Aplace, \AAplace, or \Bplace within a \Par... \endPar region simply stores the corresponding \box\islandbox@ in some

`\Parfigbox n` and the letter a, A, or b in `\Parlist@` (section 34.2), while a `\Cplace`, `\Mplace`, or `\MXplace` stores the `\box\islandbox@`, and the letter c, m or M, and also adds a `\vadjust{\break}` within the `\box\Parbox@` that we are setting (section 1).

When we get to the `\endPar` that matches the `\Par`, we will first supply the `\egroup` that ends the setting of `\box\Parbox@`,

```
\def\endPar{\egroup . . .
```

The remaining task is to take the material in `\box\Parbox@` and restructure it as if all the `\Cplace`'s, `\Mplace`'s and `\MXplace`'s actually occurred between paragraphs. The idea is to use `\vsplit` to take `\box\Parbox@` apart, splitting it after the lines where `\vadjust{\break}`'s were added at these `\. . .place`'s, and treat the pieces as separate paragraphs. The pieces into which we split `\box\Parbox` will be stored in new boxes that we declare:

```
\expandafter\newbox\csname Parbox1\endcsname
\expandafter\newbox\csname Parbox2\endcsname
\expandafter\newbox\csname Parbox3\endcsname
\expandafter\newbox\csname Parbox4\endcsname
\expandafter\newbox\csname Parbox5\endcsname
```

To split `\box\Parbox@`, we use a `\loop`,

```
\count@=1
\loop
  \ifnum\count@ < \Parcount@
```

This `\loop` will be enclosed in a group where we have set

```
\vfuzz=\maxdimen \vbadness=10000
```

so that `Overfull` and `Underfull` `\vbox`'es will not be reported. In this group we will also set `\splitmaxdepth` to `\maxdimen`, since we already know where our boxes will split, and don't want to impose any extraneous constraints on their depths, and we will set `\splittopskip=\ht\strutbox`, so that in essence a strut will have been added at the beginning of each box.

Before beginning the \loop, we will

```
\setbox0=\vsplit\Parbox@ to\ht\Parbox@
```

Because of the \break at the beginning of \box\Parbox@ (page 420), this simply splits off the \break and the following \parskip glue into \box0 (which will not be used), so that the remaining \box\Parbox@ doesn't begin with extraneous glue.

Now we will continue to

```
\vsplit\Parbox@ to \ht\Parbox@
```

storing the results in '\Parbox1', '\Parbox2', ... The \vadjust{\break}'s that we inserted by \Cplace, \Mplace, or \MXplace will force these new (Underfull) boxes to end after the line in which this \dotsplace's appeared.

At the same time, we want to store the depth of '\Parbox1' in the <dimen> register '\Parprev2', the depth of '\Parbox2' in '\Parprev3', ... (recall, page 420, that '\Parprev1' is used to store the depth of the line before the \Par begins):

```
\loop
\ifnum\count@ < \Parcount@
\expandafter\expandafter\expandafter\global
\expandafter\setbox
\csname Parbox\number\count@\endcsname
=\vsplit\Parbox@ to \ht\Parbox@
\count@@=\count@ \advance\count@@ by 1
\global\csname Parprev\number\count@@\endcsname
=\dp\csname Parbox\number\count@\endcsname
\advance\count@ by 1
\repeat
```

When this \loop is over, \box\Parbox@ will contain the text after the line containing the last \dotsplace (or will be void if there was no such text). Even though we need to \global\setbox'\Parboxn', the \vsplit opera-

tion defines the remainder of the split `\box\Parbox@` globally, so the final `\box\Parbox@` will persist after the end of the group.¹

Having finished with this, we now insert the `\parskip` glue, and get ready for another `\loop` to put things back:

```
\vskip\parskip
\count@=1
```

For this loop we first define a routine

```
\def\nextv@##1##2\nextv@{\def\next@{##1}\gdef\Parlist@{##2}}
```

that extracts the first letter from `\Parlist@` and stores it in `\next@`.¹

Our `\loop` has several stages:

(1) We begin

```
\loop
\ifnum\count@ < \Parcount@
\dimen@=\csname Parprev\number\count@\endcsname
\advance\dimen@ by \ht\strutbox
\ifdim\dimen@ < \baselineskip
\advance\dimen@ by -\baselineskip \vskip -\dimen@
\else
\vskip\lineskip
\fi
```

Thus, we first set `\dimen@` to be the depth of the previous line plus the height of a strut, which will be the height of the first line of the next piece of text (unless this contains some extra tall symbol). If `\dimen@` is less than `\baselineskip`, then we want to insert

$$\text{\baselineskip} - \text{\dimen@}$$

¹This does not seem to be stated explicitly anywhere in *The TeXbook*, though page 259 deals with related matters.

¹We chose `\nextv@`, which appears nowhere else in \LaTeX -TeX except in the definition of `\root`, to make certain that this will not be redefined by any of the routines to follow.

extra glue, or equivalently,

$$-(\dimen@ - \baselineskip)$$

hence the code

```
\advance\dimen@ by -\baselineskip \vskip -\dimen@
```

Otherwise, the two lines are already too far apart for `\baselineskip` glue to be used, so we just add the `\lineskip` glue.

(2) Next we

```
\unvbox\csname Parbox\number\count@\endcsname
```

to put the next part of the text on the vertical list; recall (see the footnote on page 379), that no extra space is added before or after this material, and that its depth does not influence `\prevdepth`, which is why we have stored the various depths in registers, and have added the extra space between these various pieces by hand.

(3) The next step is to deal with the corresponding `\dots`place'd material, stored in

```
\box\csname Parfigbox\number\count@\endcsname
```

We first move this material into `\box\islandbox@`,

```
\global\setbox\islandbox@
=\box\csname Parfigbox\number\count@\endcsname
```

and then use

```
\expandafter\nextv@\Parlist@\nextv@
```

so that `\next@` will have the value

- a if the material was `\dots`place'd with `\Aplace`
- A if the material was `\dots`place'd with `\AAplace`
- b if the material was `\dots`place'd with `\Bplace`
- c if the material was `\dots`place'd with `\Cplace`
- m if the material was `\dots`place'd with `\mplace`
- M if the material was `\dots`place'd with `\Mplace`

Then we simply pretend that we are performing the corresponding `\dotsplace` on `\box\islandbox@` at this point (where we are in vertical mode):

```
\if a\next@Aplace@\else
\if A\next@AAplace@\else
\if b\next@Bplace@\else
\if c\next@Cplace@\else
\if m\next@Mplace@\else
\if M\next@MXplace@\fi\fi\fi\fi\fi\fi
```

After all this is done we can reset `\ifPar@` to be false.

- (4) If the `\Par...\endPar` region happened to end with some sort of `\dotsplace`, our final `\box\Parbox@` will be void, and we want to set

```
\prevdepth=\csname Parprev\number\count@\endcsname
```

so that `\prevdepth` has the value for the last line of the last piece of text that was `\unvbox`'ed.

Otherwise, we first have to add glue before this final piece, as before,

```
\dimen@=\csname Parprev\number\count@\endcsname
\advance\dimen@ by \ht\strutbox
\ifdim\dimen@ < \baselineskip
\advance\dimen@ by -\baselineskip \vskip -\dimen@
\else
\vskip\lineskip
\fi
```

and then store the depth of this remaining `\box\Parbox@` before `\unvbox`'ing it, so that we can finally set `\prevdepth` to that depth:

```
\dimen@=\dp\Parbox@
\unvbox\Parbox@
\prevdepth=\dimen@
```


Our whole definition reads:

```

\def\endPar{\egroup
\count@=1
{\vbadness=10000 \vfuzz=\maxdimen
\splitmaxdepth=\maxdimen \splittopskip=\ht\strutbox
\setbox0=\vsplit\Parbox@ to\ht\Parbox@
\loop
\ifnum\count@ < \Parcount@
\expandafter\expandafter\expandafter\global
\expandafter\setbox
\csname Parbox\number\count@\endcsname
=\vsplit\Parbox@ to\ht\Parbox@
\count@=\count@ \advance\count@@ by 1
\global\csname Parprev\number\count@\endcsname
=\dp\csname Parbox\number\count@\endcsname
\advance\count@ by 1
\repeat}%
\vskip\parskip
\count@=1
\def\nextv@##1##2\nextv@{\def\next@{##1}}%
\gdef\Parlist@{##2}}%
\loop
\ifnum\count@ < \Parcount@
\dimen@=\csname Parprev\number\count@\endcsname
\advance\dimen@ by \ht\strutbox
\ifdim\dimen@ < \baselineskip
\advance\dimen@ by -\baselineskip \vskip -\dimen@
\else
\vskip\lineskip
\fi
\unvbox\csname Parbox\number\count@\endcsname
\global\setbox\islandbox@
=\box\csname Parfigbox\number\count@\endcsname
\expandafter\nextv@\Parlist@\nextv@
\if a\next@\Aplace@\else
\if A\next@\AAplace@\else

```

```
\if b\next@\Bplace@\else
\if c\next@\Cplace@\else
\if m\next@\Mplace@\else
\if M\next@\MXplace@\fi\fi\fi\fi\fi
\advance\count@ by 1
\repeat
\global\Par@false
\ifvoid\Parbox@
\prevdepth=\csname Parprev\number\count@\endcsname
\else
\dimen@=\csname Parprev\number\count@\endcsname
\advance\dimen@ by \ht\strutbox
\ifdim\dimen@ < \baselineskip
\advance\dimen@ by -\baselineskip \vskip -\dimen@
\else
\vskip\lineskip
\fi
\dimen@=\dp\Parbox@
\unvbox\Parbox@
\prevdepth=\dimen@
\fi}
```

Chapter 36. The `\output` routine
Ta-ran-ta-ra! Ta-ran-ta-ra! Ta-ran-ta-ra!

The \LaTeX -TeX default `\output` routine mimics as closely as possible the plain TeX `\output` routine, defined in terms of `\makeheadline`, `\pagebody`, `\makefootline`, etc.

To begin with, we redefine `\folio` to reflect \LaTeX -TeX's approach to page numbers (and numbering in general):

```
\def\folio{\page@F  
  \page@S{\page@P\page@N{\number\page@C}\page@Q}}
```

Since we are not using negative values of `\pageno` to indicate roman numerals, we might as well also simplify the definition of `\advancepageno`:

```
\def\advancepageno{\global\advance\pageno by 1 }
```

36.1. `\plainoutput`. Since plain TeX sets `\output={\plainoutput}`, our new `\output` routine will be specified by modifying `\plainoutput`, which is defined in plain TeX by

```
\def\plainoutput{\shipout\vbox{\makeheadline  
  \pagebody\makefootline}  
  \advancepagno  
  \ifnum\outputpenalty > -20000 \else\dosupereject\fi}
```

We will need a new flag

```
\newif\ifspecialsplit@
```

which should be set true when `\box\topins` is

```
\vbox(0.0+0.0)x0.0
```

because of a split floating insertion (section 33.6). At the very beginning of the definition of `\plainoutput` (even before the `\fliptopins@`), we will use the

code

```
\specialsplit@false
\ifvoid\topins\else\ifdim\ht\topins=0pt
\specialsplit@true
\fi\fi
```

to set `\ifspecialsplit@` to be true only if `\box\topins` has height 0pt, but isn't void (the latter test must be made explicitly because a void `\box\topins` will also satisfy the condition `\ifdim\ht\topins=0pt`), and thus precisely when (section 33.6) our `\box255` is too high by `-\skip\topins` (= 3pt), in which case we will eventually have to prune down `\box255`.

Remember that we are going to be using code like

```
\ifdim\ht255 < \minpagesize \else\unvbox255 \fi
```

so that we are not going to `\unvbox255` at all when the test

```
\ht255 < \minpagesize
```

is true. But when `\ifspecialsplit@` is true, we really want the test

```
\ht255 < \minpagesize - \skip\topins
```

(0) To achieve this, we will simply use

```
\specialsplit@false
\ifvoid\topins\else\ifdim\ht\topins=0pt
\specialsplit@true
\advance\minpagesize by -\skip\topins
\fi\fi
```

(`\minpagesize` will be restored to its usual value at the end of the `\output` routine, since TeX implicitly encloses the `\output` routine within a group.)

After this step, we follow the procedure outlined on page 415:

(1) We use `\fliptopins@`, to prepare `\box\topins` for use by `\pagecontents`, which will take care of properly positioning any 'island's that have shown up in `\box\topins`.

(2) Instead of plain T_EX's direct `\shipout`, we will declare a new box

```
\newbox\outbox@
```

and also

```
\let\shipout@=\shipout
```

and then

```
\setbox\outbox@=\vbox{\makeheadline\pagebody\makefootline}
{\noexpands@\let\style=\relax
\shipout@\box\outbox@}
```

The `\noexpands@` and `\let\style=\relax` are necessary before the `\shipout@` because various `\write's` in `\pagebody` will contain page numbering control sequences, whose expansion we want to prohibit (compare pages 58 and 87), and they might also contain `\style` (pages 204 and 382).

bd The only reason for this approach is that it makes the `\output` routine easier to modify in certain ways. For example, if we want a style file that produces "crop marks" around the completed pages (as was done for the L_AT_EX Manual, and for this manual), instead of redefining `\plainoutput`, we just have to define a suitable routine

```
\def\crop#1{\vbox{ ... #1 ... }}
```

which puts crop marks around a box #1, and then

```
\def\shipout@{\shipout\crop{\box\outbox@}}
```

(3)–(6) Then come

```
\advancepageno
\resetdimtopins@
\ifvoid 255 \else \unvbox255 \penalty\outputpenalty
\ifnum\outputpenalty > -20000 \else \dosupereject \fi
```

The whole definition is:

```

\newif\ifspecialsplit@
\newbox\outbox@
\let\shipout@=\shipout

\def\plainoutput{\specialsplit@false
\ifvoid\topins\else
\ifdim\ht\topins=0pt
\specialsplit@true
\advance\minpagesize by -\skip\topins
\fi\fi
\fliptopins@
\setbox\outbox@=\vbox{\makeheadline
\pagebody\makefootline}%
{\noexpands@ \let\style=\relax
\shipout@\box\outbox@}%
\advancepageno
\resetdimtopins@
\ifvoid 255 \else \unvbox 255 \penalty\outputpenalty \fi
\ifnum\outputpenalty > -20000 \else \dosupereject \fi}

```

Notice that the `\noexpands@... \shipout@` occurs after `\box\outbox@` is set, so any font control sequences in `\headline` or `\footline` will not be `\relax` at that time.¹ We also put the `\noexpands@... \shipout@` inside a group. In this particular definition, that extra group is irrelevant, but it would be important if the `\output` routine were modified to add additional material at the top of the next page, before the `\unvbox255` material. For example, in the index, continuation lines are added containing `{\it continued\}` (see page 514), and we don't want `\it` to be `\relax` when these lines are added.

36.2. `\pagebody`. The default \LaTeX -TEX style keeps `\makeheadline` and `\makefootline` the same as in plain TEX, and `\pagebody` is the same except that any index entries placed in the `\margin@` insertion class, for proofing, are placed at the side:

¹This is a somewhat unnecessary precaution, since `\headline` and `\footline` normally contain things like `\tenrm` or `\tenit`, rather than `\rm` or `\it`, which may have different values if a different pointsize command is currently in effect; but `\tenpoint` or `\tenpoint\it` would be equally valid.

```

\def\pagebody{\vbox to\vsize{\boxmaxdepth=\maxdepth
\ifvoid\margin@
\else
\rlap{\kern\hsize\vbox to0pt{\kern4pt\box\margin@\vss}}%
\fi
\pagecontents}

```

The `\kern\hsize` moves the entries over to the right side of the page, the `\rlap` allows them to stick into the right margin, the `\vbox to0pt` allows them to be arbitrarily long, and the `\kern4pt`, chosen empirically, starts the entries slightly below the top line, to look better.

36.3. \pagecontents. The only thing left to define is `\pagecontents`, for which we have given a preliminary definition on page 411, with the modification of section 34.5.

We will need one final flag

```

\newif\ifonlytop@

```

which will be set true when the only thing on the page is a figure on top.

The definition of `\pagecontents` begins

```

\def\pagecontents{\onlytop@false
\ifdim\ht255 < \minpagesize
\ifnum\flipcount@ < 2
\ifvoid\footins
\onlytop@true
\fi\fi\fi

```

Thus, we set `\onlytop@true` only when three conditions all hold: (1) `\ht255` is less than `\minpagesize`, so that we will not `\unvbox255` after the top figure; (2) `\flipcount@` is less than 2, so that there are no figures to go after the one on top; (3) `\box\footins` is void, so that there are no footnotes to go after the figure on top either.

The main change in `\pagecontents` is that special work may be needed if `\flipcount@` has the value 1 and `\ifC@` is true, so that there is just one

`\island` in `\box\topins` and it was `\Cplace'd`; in this case, `\Cdim@` has the value of `\pagetotal` at the time of the `\Cplace`, so that `\Cdim@` is the amount of text preceding the `\Cplace`. We will want to change the placement of this `\island` from top to bottom if `\Cdim@` is greater than half `\ht255` (i.e., if the `\island` appears in the lower half of the page). To test for this we use

```

\test@false
\ifC@
\ifnum\flipcount@=1
\global\multiply\Cdim@ by 2
\ifdim\Cdim@ > \ht255
\test@true
\fi
\fi
\fi
\global\C@false

```

(taking the opportunity to reset `\ifC@` as soon as the test has been made).

(a) When `\iftest@` is true, we basically want to put the `\island` below the rest of the page instead of at the top,

```

\iftest@
\unvbox255
\setbox\topins=\vbox{\unvbox\topins
\global\setbox0=\lastbox}
\setbox0=\vbox{\unvbox0
\breakisland@}
\nointerlineskip
\vskip\abovebotfigskip
\printisland@

```

But this doesn't quite work, since there is really just enough space for the `\vskip\belowtopfigskip`, not for the `\vskip\abovebotfigskip` (which might differ considerably in some styles).

So we will first set

```

\dimen@=\ht 255
\advance\dimen@ by \skip\topins

```


so that \dimen@ is \ht255 plus the difference in the space we are going to add (page 399), and then replace \unvbox 255 with

```
\setbox1=\vsplit 255 to \dimen@
\unvbox1
```

(anything remaining in \box255 will be put back on the main vertical list later). During the \vsplit we want to set \vfuzz=\maxdimen, \vbadness=10000, \splitmaxdeth=\maxdepth, and \splittopskip=\topskip, as in the definition of \endPar (page 442):

```
{\vfuzz=\maxdimen \vbadness=10000
 \splitmaxdepth=\maxdepth \splittopskip=\topskip
 \setbox1=\vsplit 255 to \dimen@
 \unvbox1}
```

The new value of \box255 will persist after the end of the group (see page 444).

bob If \skip\topins is negative, i.e., if \belowtopfigskip is greater than \abovebotfigskip, our \vsplit simply gives the original \box255, and we just have to hope that the glue can stretch enough to make up the difference. Compare section 5.

And then, having put the proper amount of \box255 at the top of the page, we add the \island in \box\topins, with the proper amount of space above it:

```
\setbox\topins=\vbox{\unvbox\topins
 \global\setbox0=\lastbox}
\setbox0=\vbox{\unvbox0
 \breakisland@}
\nointerlineskip
\vskip\abovebotfigskip
\printisland@
```

(b) When \iftest@ is false (the usual case), we simply put the first \island

in `\box\topins`, if any, at the top,

```
\ifnum\flipcount@ > 0
\setbox\topins=\vbox{\unvbox\topins
\global\setbox0=\lastbox}%
\setbox0=\vbox{\unvbox0
\breakisland@}%
\printisland@
```

Normally, this should be followed by `\belowtopfigskip`. But we don't want to do that if nothing else goes on the page, because the figure may be too large to allow `\belowtopfigskip` glue below it. Instead we use

```
\ifonlytop@
\kern-\prevdepth \vfill
\else
\vskip\belowtopfigskip
\fi
```

The `\kern-\prevdepth` is inserted for the following reason.¹ We might have an island whose height is less than or equal to `\vsize`, but whose height plus depth exceeds `\vsize`. Since `\pagecontents` will become

```
\vbox to\vsize{(the top \island) \vfill}
```

this will give an `Overfull \vbox` (without the `\vfill` we wouldn't have this problem, because the depth of the `\island` would just become the depth of the `\vbox to\vsize`). The `\kern-\prevdepth` eliminates this extra depth, so that `\vfill` can safely be added.

Our preliminary definition of `\pagecontents` then had

```
\ifdim\ht255 < \minpagesize \vfill \else \unvbox255 \fi
```

¹ Compare the `\ifraggedbottom\kern-\dimen@\vfil\fi` in the plain TeX definition of `\pagecontents` (*The TeXbook*, page 364).

but now we will have to prune down \box255 when \ifspecialsplit@ is true. This is similar to the routine used for a \Cplace:

```
\ifspecialsplit@
  {\vfuzz=\maxdimen \vbadness=10000
  \splitmaxdepth=\maxdepth \splittopskip=\topskip
  \dimen@ii=\ht255 \advance\dimen@ii by \skip\topins
  \setbox1=\vsplit255 to\dimen@ii
  \unvbox1}%
\else
  \unvbox255
\fi
```

We might as well also replace the

```
\ifdim\ht255 < \minpagesize
  \vfill
```

with

```
\ifdim\ht255 < \minpagesize
  \ifonlytop@ \else \vfill \fi
```

so that a page containing a single large figure will have \vfill below it rather than two \vfill's (this just might make a difference if some one were trying to reposition things).

Finally, having taken care of the text, we just have to add \bottomfigs@, and the footnote material, if any:

```
\def\pagecontents{\onlytop@false
  \ifdim\ht255 <\minpagesize
    \ifnum\flipcount@ < 2
      \ifvoid\footins
        \onlytop@true
      \fi\fi\fi
  \test@false
```

```

\ifC@
\ifnum\flipcount@=1
\global\multiply\Cdim@ by 2
\ifdim\Cdim@ > \ht255
\test@true
\fi
\fi
\fi
\global\C@false
\iftest@
\dimen@=\ht255
\advance\dimen@ by \skip\topins
{\vfuze=\maxdimen \vbadness=10000
\splitmaxdepth=\maxdepth \splittopskip=\topskip
\setbox0=\vsplit255 to\dimen@
\unvbox0}%
\global\setbox\topins=\vbox{\unvbox\topins
\global\setbox1=\lastbox}%
\setbox0=\vbox{\unvbox1
\breakisland@}%
\nointerlineskip
\vskip\abovebotfigskip
\printisland@
\else
\ifnum\flipcount@ > 0
\global\setbox\topins=\vbox{\unvbox\topins
\global\setbox1=\lastbox}%
\setbox0=\vbox{\unvbox1
\breakisland@}%
\printisland@
\ifonlytop@ \kern-\prevdepth \vfil
\else \vskip\belowtopfigskip \fi
\fi
\fi
\ifdim\ht255 < \minpagesize
\ifonlytop@ \else \vfill \fi
\else

```

```

\ifspecialsplit@
  {\vfuzz=\maxdimen \vbadness=10000
  \splitmaxdepth=\maxdepth \splittopskip=\topskip
  \dimen@ii=\ht255 \advance\dimen@ii by \skip\topins
  \setbox0=\vsplit255 to\dimen@ii
  \unvbox0}%
\else
  \unvbox255
\fi
\fi
\bottomfigs@
\ifvoid\footins\else
  \vskip\skip\footins\footnoterule\unvbox\footins\fi}

```

god When `\vskip\topins` is small we might elect simply to forget about the flag `\ifspecialsplit@`, and always `\unvbox255` when its height is greater than or equal to `\minpagesize`. Only very very rarely would we encounter difficulties, and these could always be fixed by hand—after all, nothing is perfect (compare section 13).

36.4. And we are done! This finishes off everything concerned with the first part of the \LaTeX -TEX Manual, except for front and back matter, which are dealt with in Part VI. Commutative diagrams are dealt with in Volume II, but all that material could just as well be considered as separate files that we could `\input`. And all the material for tables, except for a few things like `\paste` and `\measuretable` are actually in a separate file (except in $\mu\text{T}^n\text{TEX}$); these are also dealt with in Volume II. So let us now skip to the very end of the `lamstex.tex` file.

First, we introduce the more formal `\enddocument` as a synonym for `\bye`:

```
\let\enddocument=\bye
```

Then, since we have no `\new...` constructions to use, we restore `\alloc@` to its original plain meaning (see page 38), so that any further uses of the `\new...` constructions will write to the `.log` file:

```

\def\alloc@#1#2#3#4#5{\global\advance\count1#1by\@ne
\ch@ck#1#4#2\allocationnumber=\count1#1
\global#3#5=\allocationnumber
\wlog{\string#5=\string#2\the\allocationnumber}}

```

And finally, we make `@` active:

```

\catcode'\@=\active

```

36.5. When `\box255` is too small. At the end of section 33.6, we mentioned the fact that `\box255` will be too small, rather than too big, when `\belowtopfigskip` is greater than `\abovebotfigskip`, so the splitting maneuvers used in defining `\pagecontents` won't do us any good.

Fortunately, that situation is unlikely to occur, since good style design calls for more space to be left above constructions than below them. (This holds for headers also; they look better when the space above exceeds the space below, something that more `TEX` style file designers should bear in mind.)

I don't see any easy way to deal with the problem of a style that chooses a value of `\belowtopfigskip` greater than `\abovebotfigskip`. The best I can suggest in that case is the following.

First of all, `\Aplace` will also have to be disallowed within paragraphs (except for special `\Par... \endPar` regions, which reduce back to use between paragraphs). Then, when we `\Aplace` an `\island` we would make a special check if `\topinscount@` is 0 (so that this is the first `\island` to be considered for the page). In this case, we would use the same calculations as for `\Mplace`, to see if the `\island` will fit. If it does fit, we would simply insert it (essentially converting the `\Aplace` to an `\Mplace`). If it doesn't fit, so that it will have to float, we would set

```

\skip\topins=0pt

```

so that `\box255` for this page will have just the right size. Finally, we would add

```

\global\skip\topins=(original value of \skip\topins)

```

at the end of the `\output` routine.

36.6. The endgame. Although the `\dosupereject`'s in the `\output` routine ensure that all `\Aplace`'d material will eventually make its way onto the page, some of this material may have to be printed after all the other text, with the final pages consisting entirely of `\island`'s. These `\island`'s will still be allocated only two to a page, so it will probably be necessary to change some of the final `\Aplace`'s to `\Mplace`'s or `\AAplace`'s in order to improve the appearance of these final pages.

There doesn't seem to be any easy way to have \LaTeX do this automatically. We might try to change `\dimen\topins` to `\vsize` when we have run out of text (i.e., when it is no longer true that `\outputpenalty > -20000`), but this change would be made too late—after \TeX has already decided not to include some members of the `\topins` insertion class in `\box\topins`, since that decision is made at the time of the `\insert`.

One possibility is to have a box, say `\box\remainingplaces@`, in which we store copies of all the `\dotsplace`'d `\islands`, eliminating copies as `\pagecontents` uses up boxes in `\box\topins`. Then, when we run out of text, we would change `\pagecontents` so that it ignores `\box\topins`, and instead takes things from `\box\remainingplaces@`, putting as many on the page as will fit. At this stage we would change the definition of `\dosupereject` from

```
\ifnum\insertpenalties > 0
  \line{} \kern-\topskip \nobreak \vfill \supereject \fi
```

to

```
\ifvoid\remainingplaces@ \else
  \line{} \kern-\topskip \nobreak \vfill \supereject \fi
```

36.7. The endgame once again. In addition, there is one minor defect that is encountered even in plain \TeX . The plain \TeX file

```
\vsize=22pt
\hsize=2in
\raggedright
```

Here is some stuff, taking up more than a line.

```
\pageinsert \hbox{A}\vfil\hbox{B}\endinsert
```

```
\bye
```

will produce *three* pages of output: The text will all occur on page 1; the

```
\vbox to\vsizel\hbox{A}\vfil\hbox{B}}
```

will take up page 2; and page 3 will be *blank* (except for the page number).

If we add

```
\def\plainoutput{%
\showbox255 \showbox\topins \showthe\outputpenalty
\shipout\vbox{\makeheadline\pagebody\makefootline}%
\advancepageno
\ifnum\outputpenalty>-20000 \else\dosupereject\fi}
```

at the beginning of this file (compare page 416), the `.log` file will show that before page [1] is shipped out `\box255` contains the text, `\box\topins` is

```
> \box253=
\vbox(0.0+0.0)x0.0
```

(because of the split insert) and the `\outputpenalty` is 10000 (set by `TEX` because the chosen breakpoint for the page was not a penalty item).

Before page [2] is shipped out, `\box255` will be an empty box, while `\box\topins` will now be the `\vbox to\vsizel\hbox{A}\vfil\hbox{B}` (and the value of `\outputpenalty` will still be 10000).

Finally, before page [3] is shipped out, we will have

```
> \box255=
\vbox(22.0+0.0)x144.54, glue set 12.0fill
(A) .\glue(\topskip) 10.0
     .\hbox(0.0+0.0)x144.54
     .\glue 0.0 plus 1.0fill
```

and `\outputpenalty` will be `-1073741824` (= `-'10000000000`). As explained in *The TEXbook*, page 264, `TEX` inserted the equivalent of

```
\line{ } \vfill \penalty -'1000000000
```

into the main vertical list when it saw the `\end` from the `\bye` (apparently because this `\end` was seen before the text had been output, though I don't understand the details).

The \LaTeX -TeX file

```
\vsize=22pt
\hsize=2in
\raggedright
```

```
\minpagesize=22pt
\Figureproofing
```

Here is some stuff, taking up more than a line.

```
\Aplace{\Figure \Hbyw{1in} \endFigure}
```

```
\bye
```

will also produce a blank third page,² although the .log file will tell a somewhat different story if we add the appropriate `\show's` to the definition of `\plainoutput` in `lamstex.tex`.

Before page [1] is shipped out, `\outputpenalty` will be `-20000` (presumably from a `\dosupereject`).

Before page [2] is shipped out, `\box255` will be

```
\vbox(-15.0+0.0)x0.0
```

(see page 418), and `\outputpenalty` will again be `-20000`.

Before page [3] is shipped out, `\outputpenalty` will again be `-20000`, and we will have

```
> \box255=
\vbox(22.0+0.0)x144.54, glue set 22.0fill
.\glue(\topskip) 10.0
(B) .\hbox(0.0+0.0)x144.54
.\kern -10.0
.\penalty 10000
.\glue 0.0 plus 1.0fill
```

This `\vbox`, somewhat different from (A), has come from the material inserted by `\dosupereject`.

²If we left `\minpagesize` at 5pc we'd get infinitely many pages!

[With some choices of T_EX parameters, the plain T_EX file may cause a page break to be taken before the `\pageinsert` is seen, so that we might end up with (B) instead of (A).]

Such empty pages present a formidable problem if they occur at the end of a book chapter, rather than at the end of a complete document, since they can cause the next chapter to begin with the wrong page number. For such situations we can add a check for empty pages. For example, at the beginning of the L_AT_EX definition of `\plainoutput` we can test if `\box\footins` is void and `\box\topins` is either void or has height 0pt, and if so we can then add

```
\setbox0=\vbox{\unvcopy255 \unskip}
\ifdim\ht0=0pt
\global\advance\pageno by -1
\fi
```





so that the blank page (B) would simply have the same page number as the previous page.


Since we might encounter the plain T_EX situation (A) also, the more complex test

```
\setbox0=\vbox{\unvcopy255
\unskip \unpenalty \unkern
\global\setbox1=\lastbox
\unskip}
\ifdim\ht0=0pt \ifdim\ht1=0pt
\global\advance\pageno by -1
\fi\fi
```

is actually required.


Of course, this test is memory-intensive, since we essentially have to keep two copies of `\box255` in memory at once. The book style (Chapter 41) allows the user to type `\FlushedFigs` when this test is appropriate, but to avoid the test with `\NoFlushedFigs`.

-  36.8. *The endgame once again.*
-  36.9. *The endgame once again.*
-  36.10. *The endgame once again.*
-  36.11. *The endgame once again.*

 36.12. *The endgame once again.*

. . .

Finally, let's hope that our `\output` routine doesn't allow this to happen!

 36.13. *A final warning.* With all the subtleties involved in the `LATEX` `\output` routine, it should perhaps be mentioned that sometimes even the plain `TEX` `\output` routine will produce incorrect results. On page 74, the page might have been broken before the final

where

V_1 = value of `\ref{label}`
 V_2 = value of `\Ref{label}`
 V_3 = value of `\nref{label}`
 V_4 = value of `\pref{label}`

with a badness of only 295. But `TEX` included this material on the page, and then reported an

Overfull `\vbox` (0.22726pt too high) while `\output` is active !!

I must have printed thousands, if not tens of thousands, of pages of `TEX` output before I ever encountered this problem, but it can occur, so you might want to know what happened.

The footnote on the page has a height of 57.5pt and a depth of 2.5pt, and, as in plain `TEX`, `\skip\footins` was 12pt plus 4pt minus 4pt. The thick footnote rule contributed no extra space [in plain `TEX`, `\footnoterule` is

```
\kern-3pt
\hrule width2truein \kern2.6pt
```

for a total height of 0pt, and in this manual it was

```
\kern-3pt
\hrule height1pt width 2truein
\kern2pt
```

again having a total height of 0pt].

At the time the footnote was considered, in the middle of the page, TeX carefully checked that the height plus depth of the footnote, plus the height plus depth of the page so far, plus `\skip\footins`, was less than the page goal, which was the `\vsize` of 480pt. Since this was true, so that the insert fit, the goal g for the page was reduced by 72pt [the height plus depth of the footnote plus the `\dimen` part of `\skip\footins`], to 408pt. (The stretch and shrink part of `\skip\topins` was added to the page total t).

After that, TeX considered the page total t each time it encountered a possible break point; this page total reflects only the *height* of the material so far, not its depth. At the point where the page was finally broken, TeX calculated that

$$t=444.72726 \text{ plus } 35.0 \text{ minus } 37.0 \quad g=408.0$$

with a badness of 97. The 'minus 37.0' meant that the page total could be shrunk down to 407.72726, which seems to allow just enough room for the footnote in `\pagebody`:

$$407.72726\text{pt} + 12\text{pt} + 60\text{pt} = 479.72726\text{pt} \\ < 480\text{pt}$$

But there are two flaws in this calculation:

- (1) In the first place, when we make `\pagebody`, which is a `\vbox to\vsize`, the depth 2.5pt of the footnote at the bottom of the box is irrelevant, since it does not contribute to the height of the box. Thus, we seem to have an extra 2.5pt of breathing room.
- (2) Unfortunately, the depth of `\box255`, which no longer appears at the bottom of the `\vbox to\vsize`, is *not* irrelevant, and must be added in.

It happens that the depth of the display at the bottom of page 74 is 3pt (there is a strut of depth 3pt inserted into each line of $\mathcal{A}\mathcal{M}\mathcal{S}$ -TeX's `\align`, which was used for the display). Consequently, the height of `\pagebody` is

$$407.72726\text{pt} + 3\text{pt} + 12\text{pt} + 57.5\text{pt} = 480.22726\text{pt}$$

leading to the `Overfull \vbox (0.22726pt too high)`.

(You can easily simulate this phenomenon with the plain TeX file

```

\size=480pt
\topskip=0pt
\insert\footins{\hbox{\vrule height 57.5pt
  depth 2.5pt width 0pt}}          %% the ‘‘footnote’’
\hbox{}
\nointerlineskip
\vskip0pt minus34pt                %% the shrink
\hbox{\vrule
  height 444.72726pt depth 3pt width 0pt}  %% the ‘‘text’’
\bye

```

which gives a single page with an `Overfull \vbox`, instead of two pages with a seriously `Underfull \vbox` on the first page. However, the two-page output will occur once the depth of the “text” exceeds 4pt, the value of `\maxdepth`.)

Moral: In theory, the `\output` routine really ought to examine the depth of `\box255` before blithely `\unvbox`ing it, because it might prove necessary to prune down `\box255`, just as we did when `\ifspecialsplit@` was true (page 457). In practice, this refinement is probably just a terrible waste of time. It might become important when there is almost no stretch or shrink on a page (though one would presume that `\vskip\footins` would need to have some stretch and shrink in such cases).

Part VI

***Front and
Back Matter***

Chapter 37. Front Matter
(Table of Contents, List of Figures, Tables, etc.)

Now that we've covered all the formatting of the paper proper, we only have to worry about the formatting of the .toc and .tic files for the front matter, and the .ndx file for the index.

As we've seen in section 15.2, the `\predocstyle` and `\postdocstyle` commands simply `\input` files with the extension .stf and .stb, respectively. In the default style, we say

```
\predocstyle{lamstex}
```

to `\input lamstex.stf`, and

```
\postdocstyle{lamstex}
```

to `\input lamstex.stb`. In this chapter we consider the file `lamstex.stf`, while the next chapter considers `lamstex.stb`.

As in section 27.2, we will be using double horizontal lines for code from these subsidiary files.

37.1. lamstex.stf preliminaries. The file `lamstex.stf`, although it is only a particular example of a front matter style file, nevertheless illustrates all the main points we have to consider.

As usual, we begin with

```
\catcode'\@=11
```

Since we are going to be introducing some `\newdimen`'s, we first

```
\let\alloc@=\alloc@@
```

(compare page 38).

In the default style, we will be setting entries like

Chapter 1. Introduction 3

using dot leaders. This, of course, is merely a style decision, and some modern style designers eschew dot leaders, on the grounds that blank space will work just as well. I suspect, however, that many such supposedly aesthetic decisions were really influenced by the difficulty of handling dot leaders correctly. In any case, the default style uses dot leaders, giving us the opportunity to address the question of long entries, where we will need to use something like

**Chapter 1. This chapter has such a long title that it will be
necessary to split it into two parts 3**

For the moment, we simply define our leaders,

```
\def\dotleaders{\leaders\hbox to10pt{\rm\hfil.\hfil}\hfil}
```

We also want to define the formatting for page numbers. The .toc and .tic files contain lines like

```
\Page {3}{\arabic }{-}
```

to handle the page number, the numbering style, and possible pre- and post-page material. The routine `\Page@#1#2#3#4` tells how these will be combined when typeset:

```
\def\Page@#1#2#3#4{\kern10pt\hbox{\rm#3#2{#1}#4}}
```

Thus, for example,

```
\Page {3}{\roman }{A-}
```

will be printed as ‘A-iii’, with at least 10 points of space between the leaders and the number.

37.2. Setting an entry. We begin with a test

```
\widerthanhsiz@#1#2#3#4
```

to determine whether the material for an entry is longer than `\hsiz`. In the example given above, #1 will be the ‘Chapter 1.’ part of the heading, #2 will

be the title ‘**This chapter ... into two parts**’, #3 will be the `\dotleaders`, and #4 will be the suitable ‘`\Page@...`’.

Displayed formulas and other such nonsense aren’t allowed in heading levels, but captions, at any rate, could have more than one paragraph of text; moreover, in the `.toc` file we might well want to use `\newline` to force a short line when our entry is so long that only one or two of the dot leaders appear. So we will use a test similar to the `\widerthanisland@` test of section 32.4:

```
\long\def\widerthanhsiz@#1#2#3#4{%
  \test@true
  \setbox0=\vbox{\hsiz=\maxdimen
    \rm\noindent@#1#2#3#4\par\setbox0=\lastbox}%
  \ifdim\wd0=0pt
    \setbox0=\hbox{\rm#1#2#3#4}%
    \ifdim\wd0 > \hsiz\else\test@false\fi
  \fi}
```

We use `\noindent@` instead of the more complicated `\noindent@@` here because we will be setting `\everypar{}` while the table of contents is being made. (Actually, even the `\noindent@` is unnecessary, since we will also be setting `\parindent=0pt`.) Instead of `\rm`, other style files might need to insert something like `\tenpoint` (just in case the user has, for example, added `\eightpoint` material at the beginning of the `.toc` file).

Then we define the routine `\setentry@#1#2#3#4`, which sets the entry with parts #1, ..., #4 properly; when the entry will have to be set on more than one line, we call the routine `\longentry@`:

```
\long\def\setentry@#1#2#3#4{%
  \widerthanhsiz@{#1}{#2}{#3}{#4}%
  \iftest@
    \longentry@{#1}{#2}{#3}{#4}%
  \else
    \hbox to\hsiz{\rm\strut#1#2#3#4\strut}%
  \fi}
```

Notice that we added the `\strut`’s after the `\rm`; other styles might have to specify something like `\tenpoint` instead of the `\rm`.

Because our lines have `\strut`'s, which already make the baselines 12pt apart, it is essential that we have

```
\lineskiplimit=0pt
```

while we are setting the table of contents, so that additional `\lineskip` glue isn't inserted between lines (compare page 242). However, instead of making this assignment now, we will make it later, within the definition of `\maketoc`.

For entries that are longer than one line, the default style uses hanging indentation, so that lines after the first are indented by 10 points more than the width of argument #1. We will declare a `(dimen)` register `\thehang@` for the necessary amount of hanging indentation, which will be determined by

```
\setbox0=\hbox{#1}
\thehang@=\wd0 \advance\thehang@ by 10pt
```

All lines before the last will be made shorter than `\hsize`, so that they don't overlap the width of the page number. The default style leaves an extra 20 points leeway, which we arrange with

```
\setbox0=\hbox{#4}
\advance\hsize by -\wd0 \advance\hsize by -20pt
\hangafter 1 \hangindent=\thehang@
```

Now the idea is to set a `\vbox` that looks like

```
Chapter 1. This chapter has such a long title that it will be necessary
to split it into two parts
```

and then use `\lastbox` to extract the last line, to which we will finally add the dot leaders and the page number:

```
\long\def\longentry@#1#2#3#4{%
\setbox0=\hbox{#1} \thehang@=\wd0 \advance\thehang@ by 10pt
\setbox0=\hbox{#4}
\setbox0=\vbox{\advance\hsize by -\wd0
\advance\hsize by -20pt
\hangafter 1 \hangindent=\thehang@
\rm\noindent@\strut\hbox{#1}#2\vphantom{#3#4}\strut}
. . .
```

We put #1 inside an `\hbox` so that any glue within it will have only its natural width, without any stretch or shrink; this is necessary in order to insure that our hanging indentation will truly be the width of #1 plus 10 points. The `\vphantom{#3#4}` is added so that the height and depth of the last line of `\box0` won't change when we add the dot leaders (#3) and the page number (#4). As mentioned before, other styles might have to specify something like `\tenpoint` rather than `\rm`; the `\strut` should come after that, *and also after the `\noindent@`* (so that it doesn't start a paragraph prematurely).

As with footnotes (section 25.2), we will actually want to replace the first `\strut` with

```
\vbox to\ht\strutbox{}
```

and the last `\strut` with

```
\lower\dp\strutbox\vbox to\dp\strutbox{}
```

The latter combination will appear more than once, so we adopt an abbreviation:

```
\def\endstrut@{\lower\dp\strutbox\vbox to\dp\strutbox{}}
```

Although we will have `\lineskiplimit=0pt` when `\longentry@` is being used, within our `\vbox` we want to declare

```
\normalbaselines
```

so that the standard value of `\lineskiplimit` is used there. (In the default style `\normallineskiplimit` happens to be `0pt`, so that this is actually superfluous, but in other styles that is not necessarily the case.)

Now we start to make a new box, in which we `\unvbox0`:

```
\setbox0=\vbox{\unvbox0
\setbox0=\lastbox
. . .
```

This makes the inner `\box0` the last line of the paragraph, a “short” line that ends with `\parfillskip` glue, and then yet another glue, namely `\rightskip`. So we need

```
\unhbox0 \unskip\unskip
```

to get rid of this extra glue, before we add #3 and #4. In addition, the hanging indentation does not insert glue or a kern at the beginning of this box—instead, it merely shifts it to the right in the vertical list. So we use

```
\setbox0=\vbox{\unvbox0
\setbox0=\lastbox
\hbox to\hsize{\kern\thehang@
\unhbox0 \unskip\unskip#3#4\endstrut@}
}
```

to create the desired final line. And then we simply `\unvbox0`:

```
\newdimen\thehang@
\long\def\longentry@#1#2#3#4{\setbox0=\hbox{#1}%
\thehang@=\wd0 \advance\thehang@ by 10pt
\setbox0=\hbox{#4}%
\setbox0=\vbox{\advance\hsize by -\wd0
\advance\hsize by -20pt
\normalbaselines
\hangafter1 \hangindent=\thehang@
\vskip-\parskip
\rm\noindent@\vbox to\ht\strutbox{}}%
\hbox{#1}#2\vphantom{#3#4}\endstrut@}%
\setbox0=\vbox{\unvbox0
\setbox0=\lastbox
\hbox to\hsize{\kern\thehang@
\unhbox0 \unskip\unskip #3#4\endstrut@}%
}%
\unvbox0 }
```

Note that in the last `\setbox0=\vbox` there is no extra glue between the `\unvbox0` and the `\hbox to\hsize` (compare the footnote on page 379); that is why it is important to have the `\vphantom{#3#4}`, to insure that the `\hbox to\hsize` has the same height as the `\lastbox` that it is replacing.

37.3. Further preliminaries for the table of contents. The `\maketoc` command is going to `\input` the appropriate `.toc` file, which will have entries like

```
\HL {1}{\word}{\formatted heading number}
. . .
\Page ...
```

and/or entries like

```
\chapter {\word}{\formatted heading number}
. . .
\Page ...
```

(in which case an extra line like

```
\NameHL 1\chapter
```

should appear at the beginning), and similarly for `\h1`.

We will therefore be redefining `\HL` and `\h1` so that such lines print proper lines for the table of contents. But a few preliminaries are needed to deal with the possibility that something like "B", or even "`\style B`" occurs instead of a `(formatted heading number)`. First, we want a flag

```
\newif\ifemptynumber@
```

which will be set true precisely in the case where we have "" (in the paper proper, this happens when `\thelabel@@` is `\empty`). When it comes time to properly print the argument `'...'` of `\HL` that represents the heading number, we will use

```
\Style@... \Style@
```

so that `\Style@` can use a `\futurelet` to see if ‘...’ begins with a “ (we use `\Style@` because `\style@` is already used, as part of the definition of `\style`). We begin with

```
\def\Style@{\emptynumber@false\futurelet\next\Style@@}
\def\Style@@{\ifx\next"\expandafter\Style@@@
\else\expandafter\Style@@@\fi}
```

For `\Style@@@`, when no “ occurs, we might expect to use

```
\def\Style@@@#1\Style@{#1}
```

Instead, however, we will use

```
\def\Style@@@#1\Style@{\style{#1}}
```

so that each heading level can temporarily define ‘`\style`’ to produce whatever sort of formatting for the number that we want. On the other hand, `\Style@@@` is defined by

```
\def\Style@@@"#1"\Style@{%
\def\next@{#1}\ifx\next@\empty
\emptynumber@true\else#1\fi}
```

Thus, in the special case that our original argument is “”, we simply set `\emptynumber@true`; otherwise we print the argument #1. Notice that if this argument happens to be something like “`\style B`”, the ‘B’ will be formatted according to the manner in which this heading level has temporarily defined `\style` to operate.

For setting `\hl1` entries, we will also declare a new dimension `\digits`,

```
\newdimen\digits
\setbox0\hbox{0.00}
\digits=\wd0
```

so that `\digits` is the width of a double-digit number like '1.12' (the user could reset `\digits` if necessary to accommodate even more digits).

We also want to define

```
\def\maketoc@W{CONTENTS}
```

so that `\newword\maketoc` can be used to choose a different word.

37.4. Starting the `\maketoc` command. The `\maketoc` command first uses `\checkmainfile@` (page 207) to issue an error message if no `\mainfile` command has appeared in the front matter file, since it will have to `\input \mainfile@.toc`, where `\mainfile@` has been defined by `\mainfile`. Then it starts a new page, in case something like a Preface precedes it (the default front matter style file `lamstex.stf` doesn't have special constructions for Prefaces, etc., but the book style, for example, does).¹ Then we produce a centered bold heading 'CONTENTS', followed by some space.

Now we want to `\input` the appropriate `.toc` file, which is when everything interesting will happen. Certain subsidiary definitions will be needed, so we will start with `\begingroup` and add an `\endgroup` after the `\input`, to keep these all local.

Any `\label` or `\pagelabel` in a heading level or a caption will have been written to the `.toc` or `.tic` file (there's no practical way to eliminate them), but they play no further role, and we simply want to ignore them. The same is true of any `\Reset` or `\Offset`. So after the `\begingroup` we want to add

```
\noset@ \unlabel@
```

(section 16.4 and 16.5). Any `\nopunct` or `\nospace` or `\overlong` in the main file will also show up in the `.tic` file. Some styles might want to make use of these (presumably having them set certain flags), but in the default front matter style `lamstex.stf`, we simply want to

```
\let\nopunct=\relax \let\nospace=\relax
\let\overlong=\relax
```

¹ We don't simply put `\checkmainfile@` at the beginning of our file, since a user might omit the `\mainfile` line when simply checking that something like a Preface looks right.

We also add

```
\everypar={}\parindent=0pt
```

for good measure.

Furthermore, as mentioned previously (page 474), it is essential that we set `\lineskiplimit=0pt`. So our definition of `\maketoc` will start

```
\def\maketoc{\checkmainfile@
\par\vfill\break
\noset@ \unlabel@
\let\nopunct=\relax \let\nospace=\relax
\let\overlong=\relax
\everypar={}\parindent=0pt
\lineskiplimit=0pt
```

37.5. Redefining `\HL` and `\hl`. Next we have to redefine `\HL` and `\hl` so that `\HL` and `\hl` entries will be properly set for the table of contents.

To redefine `\HL`, recall (page 477) that it will occur in a combination like

```
\HL {(heading level)}{(heading word)}{(formatted heading number)}
{The heading}
\Page{...}{...}{...}{...}
```

so that we need a definition like

```
\def\HL#1#2#3#4\Page#5#6#7#8{ . . .
```

(except that `#1` will be `##1`, etc., since this entire definition will be made within the definition of `\maketoc`).

The definition should always begin

```
\def\HL#1#2#3#4\Page#5#6#7#8{\def\HLlevel@{#1}
```

```
. . .
```

so that subsequent constructions can know which heading level is being considered. For the default front matter style we will be using

```
\def\HL#1#2#3#4\Page#5#6#7#8{\def\HLlevel@{#1}
\ifnum\HLlevel@ = 1
. . .
. . .
\else
\Err@{\string\HL#1 not defined in this style}%
\fi}
```

so that an error message will be given if \HL {<number>} occurs for any <number> ≠ 1. (Theoretically this shouldn't happen, since the .toc file is generated by the main file, which only recognizes \HL1; but we might as well add this precaution in case the .toc file has been edited.)

When \HLlevel@ is 1, we want to use \setentry@ with the appropriate data,

```
\setentry@{<arg1>}{<arg2>}{<arg3>}{<arg4>}
```

In terms of the original arguments #1, #2, #3, . . . , #8 of \HL (see the previous page), <arg2> will be

```
\bf\ignorespaces#4\unskip
```

<arg3> will be \dotleaders, and <arg4> will be

```
\Page@{#5}{#6}{#7}{#8}
```

As for <arg1>, it will essentially be

```
\bf #2 #3
```

More precisely, it will be

```
\bf \def\next@{#2}\ifx\next@\empty\else#2\space\fi
\let\style=\HL@S \Style@#3\Style@
\ifemptynumber@\else\space\fi
```

So the `<word> #2` is followed by a space, if it is non-empty. Moreover, when `\Style@... \Style@` uses `\style` to produce the formatting of the number, this `\style` will be `\HL@S1` (i.e., `'\HL@S1'`, since `\Hllevel@` was defined to be 1). And we also add a space after this number, except in the case of an empty number. Naturally, other style files might do things differently. Note, moreover, that although `'\HL@S1'` has been defined in `lamstex.tex`, if `\newstyle\HL1` appears in the main file, it will also have to be added to the `.toc` file to produce the same effect. (In practice, of course, only style file designers will use `\newstyle\HL`, except perhaps for transient effects.)

A multi-level `\HL1` heading will have `\\` in it. Although that should probably be edited out, we can at least give `\\` a provisional definition as `'\unskip\space\ignorespaces'`. The only other thing to worry about is the spacing around the heading:

```
\def\HL#1#2#3#4\Page#5#6#7#8{\def\Hllevel@{#1}%
  \ifnum\Hllevel@ = 1
  \bigbreak
  \begingroup
  \def\\{\unskip\space\ignorespaces}%
  \setentry@
  {\bf\def\next@{##2}\ifx\next@\empty\else##2\space\fi
  \let\style=\HL@S \Style@#3\Style@
  \ifemptynumber@\else\space\fi}%
  {\bf\ignorespaces#4\unskip}%
  \dotleaders
  {\Page@{#5}-{#6}-{#7}-{#8}}
  \endgroup
  \nobreak\smallskip
  \else
  \Err@{\string\HL#1 not defined in this style}%
  \fi}
```

The default style for `\hl1` headers is

```
1.12 A short section . . . . . 79
```

where we indent a `\quad` from the left margin, and leave room for a double-digit number like `'1.12'`, followed by an `\enspace`, before the section title.

We have already introduced \digits (page 478), and the definition of \hl introduces nothing especially new:

```
\def\hl#1#2#3#4\Page#5#6#7#8{\def\hllevel@{#1}%
\ifnum\hllevel@ = 1
\setentry@
{\rm\quad \let\style=\hl@@S
\hbox to\digits{\Style@#3\Style@\hfil}\enspace}%
{\rm\ignorespaces#4\unskip}%
\dotleaders
{\Page@{#5}{#6}{#7}{#8}}%
\else
\Err@{\string\hl#1 not defined in this style}%
\fi}
```

In the default style, not only is ‘\hl@W1’ empty, but \hl1 doesn’t print this ⟨word⟩, even if it has been changed by \newword; so we also don’t bother about printing it in the table of contents.

37.6. \NameHL and \NamehL. Finally, \maketoc has to deal with alternate names like \chapter and \section. If the main file has

```
\NameHL 1 \chapter
\NamehL 1 \section
```

then these lines should also appear in the .toc file (before the use of \chapter or \section, respectively).

First of all, we have to redefine \NameHL so that, for example,

```
\chapter {⟨word⟩}{2}
. . .
\Page...
```

will typeset the same thing as

```
\HL {1}{⟨word⟩}{2}
. . .
\Page...
```

In other words, we want to

```
\def\chapter#1#2#3\Page{\HL{1}{#1}{#2}#3\Page}
```

(we don't need braces around the second #3, since it is delimited by \Page). So in general we want \NameHL#1#2 to mean

```
\def#2##1##2##3\Page{\HL{#1}{##1}{##2}##3\Page}
```

and, similarly, we want \Namehl#1#2 to mean

```
\def#2##1##2##3\Page{\hl{#1}{##1}{##2}##3\Page}}
```

37.7. \maketoc. Putting all this together, our definition of \maketoc is:

```
\def\maketoc{\checkmainfile@
\par\vfill\break
\begingroup
\noset@ \unlabel@
\let\nopunct=\relax \let\nospace= \relax
\let\overlong=\relax
\everypar{\parindent=0pt
\lineskiplimit=0pt
\def\HL##1##2##3##4\Page##5##6##7##8{%
\def\HLlevel@{##1}%
\ifnum\HLlevel@=1
\bigbreak
\begingroup
\def\{\unskip\space\ignorespaces}%
\setentry@{\bf\def\next@{##2}\ifx\next@\empty
\else##2\space\fi
\let\style=\HL@@S \Style@##3\Style@
\ifemptynumber@\else\space\fi}%
{\bf\ignorespaces##4\unskip}\dotleaders
{\Page@{##5}{##6}{##7}{##8}}%
\endgroup
\nobreak\smallskip
```

```

\else
  \Err@{\string\HL##1 not defined in this style}%
\fi}%
\def\hl##1##2##3##4\Page##5##6##7##8{%
\def\hllevel@{##1}%
\ifnum\hllevel@=1
  \setentry@{\rm\quad\let\style=\hl@S
  \hbox to\digits{\Style@##3\Style@\hfil}\enspace}%
  {\rm\ignorespaces##4\unskip}\dotleaders
  {\Page@{##5}{##6}{##7}{##8}}%
\else
  \Err@{\string\hl##1 not defined in this style}%
\fi}%
\def\NameHL##1##2{\def##2####1####2####3\Page
{\HL{##1}{####1}{####2}####3\Page}}%
\def\Namehl##1##2{\def##2####1####2####3\Page
{\hl{##1}{####1}{####2}####3\Page}}%
\centerline{\bf\maketoc@W}%
\vskip30pt plus10pt minus10pt
\input\mainfile@.toc
\endgroup}

```

37.8. Lists of Figures, Tables, etc. The commands

```

\makelistFigures
\makelistTables

```

are defined directly in terms of the more general `\makelist` command,

```

\def\makelistFigures{\makelist\c{F}List of
  Figures\endmakelist}
\def\makelistTables{\makelist\c{T}List of
  Tables\endmakelist}

```

The syntax has been changed from that of the $\mathcal{L}_\mathcal{M}\mathcal{S}$ -TEX Manual, page 92, to conform with the general $\mathcal{A}_\mathcal{M}\mathcal{S}$ -TEX and $\mathcal{L}_\mathcal{M}\mathcal{S}$ -TEX syntax for constructions that allow `\` to be used to indicate line breaks.

`\makelist\c#1#2\endmakelist`, like `\maketoc`, first uses the test `\checkmainfile@`, and then ends the previous page. After this we want to add

```
\noset@ \unlabel@
\let\nopunct=\relax \let\nospace=\relax
\let\overlong=\relax
\everypar={}\parindent=0pt
\lineskiplimit=0pt
```

just as in `\maketoc`. Then `\makelist` will

```
\def\listclass@{#1}
```

so that, when we `\input` the `.tic` file we will know which of the `\island`'s to process.

Next `\makelist` must redefine `\island`. In the case of `\HL#1...`, once we had defined `\HLlevel@` as `#1`, the style control sequence `\HL@@S` was defined, so that we could `\let\style=\HL@@S` in our definition of `\HL`. A construction introduced with `\NameHL1` is basically just a synonym for `\HL1`, so we did not need to consider `\HL@@S`, defined in terms of `\HLtype@`. But the situation is different for `\island`'s, where we have only `\island@@S`, defined in terms of `\islandtype@`, which is `\island` for an explicitly typed `\island`, but `\Figure` for a `\Figure`, etc.

To deal with this situation,

```
\Figure {<Figure number>}
  <caption>
\Page ...
```

in the `.tic` file will translate to

```
\island \at@\Figure \c{F}{Figure}{<Figure number>}
(A)   <caption>
\Page ...
```

—here `\at@` is simply an arbitrarily chosen \LaTeX control sequence that the user can't insert—while an explicitly typed `\island`, like

```
\island \c{M}{Map}{<map number>}
```


will be regarded as

```
\island \at@\island \c{M}{Map}{(map number)}
```

so that eventually everything reduces to a common case.

We begin by redefining `\island` in terms of a `\futurelet`, to see whether `\at@` follows:

```
\def\island{\futurelet\next\island@}
\def\island@{\ifx\next\at@\expandafter\island@@\else
\expandafter\island@@@\fi}
```

`\island@@`, where `\at@` follows, has 9 arguments:

```
\long\def\island@@\at@#1\c#2#3#4#5\Page#6#7#8#9
```

For example, in (A), #1 is `\Figure`, #2 is `F`, #3 is `Figure`, #4 is the `(Figure number)`, and #5 is the `(caption)`.

The definition of `\island@@` is similar to the definition of `\HL` except that:

- (1) we use argument #1 to define `\islandtype@`, rather than `\HLlevel@`;
- (2) we print #3 rather than the `(word)` before the `(Figure number)`;
- (3) we process only those `\island`'s whose class #2 is the `\listclass@` for the list we are printing.

```
\long\def\island@@\at@#1\c#2#3#4#5\Page#6#7#8#9{%
\def\islandtype@{#1}%
\def\next@{#2}%
\ifx\next@\listclass@
\setentry@
{\rm#3\let\style=\island@@@S \Style@#4\Style@
\ifemptynumber@\else\space\fi}%
{\rm\ignorespaces#5\unskip}%
\dotleaders
{\Page@{#6}{#7}{#8}{#9}}%
\fi}
```

The definition of `\island@@@`, when no `\at@` follows, is reduced to the case where `\at@\island` follows:

```
\long\def\island@@@ \c#1#2#3#4\Page#5#6#7#8{%
\island@@\at@\island
\c{#1}{#2}{#3}#4\Page{#5}{#6}{#7}{#8}}
```

When something like

```
\newisland\map \c{M}{Map}
```

appears, an occurrence of

```
\map {<map number>}
<caption>
\Page ...
```

should translate to

```
\island \at@\map \c{M}{Map}{<map number>}
<caption>
\Page ...
```

In other words, we should

```
\long\def\map#1#2\Page#3#4#5#6{\island \at@\map
\c{M}{Map}{#1}#2\Page{#3}{#4}{#5}{#6}}
```

So in general, we want to

```
\def\newisland#1\c#2#3{%
\long\def#1##1##2\Page##3##4##5##6{%
\island\at@#1\c{#2}{#3}{##1}##2%
\Page{##3}{##4}{##5}{##6}}
```

Since `\Figure` and `\Table` are `\island` classes that are assumed to be present in all styles, we should also add

```
\newisland\Figure\c{F}{Figure}
\newisland\Table\c{T}{Table}
```

Putting this all together, we have

```
\def\makelist\c#1#2\endmakelist{\checkmainfile@
\par\vfill\break
\begingroup
\noset@ \unlabel@
\let\nopunct=\relax \let\nospace=\relax
\let\overlong=\relax
\everypar{}\parindent=0pt
\lineskiplimit=0pt
\def\listclass@{#1}%
\def\island{\futurelet\next@\island@}%
\def\island@{\ifx\next\at@\expandafter\island@@\else
\expandafter\island@@@\fi}%
\long\def\island@@\at@##1\c##2##3##4##5\Page##6##7##8##9{%
\def\islandtype@{##1}%
\def\next@{##2}%
\ifx\next@\listclass@
\setentry@{\rm##3
\let\style=\island@@@S \Style@##4\Style@
\ifemptynumber@\else\space\fi}%
{\rm\ignorespaces##5\unskip}\dotleaders
{\Page@{##6}{##7}{##8}{##9}}%
\fi}%
\long\def\island@@@\c##1##2##3##4\Page##5##6##7##8{%
\island@@\at@\island\c{##1}{##2}{##3}##4\Page
{##5}{##6}{##7}{##8}}%
\def\newisland##1\c##2##3{\long\def##1###1####2\Page
####3####4####5####6{%
\island\at@##1\c{##2}{##3}{####1}####2\Page
{####3}{####4}{####5}{####6}}}%

```

```

\newisland\Figure\c{F}{Figure}%
\newisland\Table\c{T}{Table}%
\vbox{\Let@\tabskip\centering@\halign to\hsize
  {\bf\hfil\ignorespaces#\unskip\hfil\cr#2\cr}}%
\vskip30pt plus10pt minus10pt
\input\mainfile@.tic
\endgroup}

```

37.9. Fini. Then, as in section 36.4, we finish with

```

\def\alloc@#1#2#3#4#5{\global\advance\count1#1by\@ne
  \ch@ck#1#4#2\allocationnumber=\count1#1
  \global#3#5=\allocationnumber
  \wlog{\string#5=\string#2\the\allocationnumber}}
\catcode'\@=\active

```

Chapter 38. Back matter
The index

Although the “back matter” may include the bibliography, and other matter, the default style file `lamstex.stb` is used only for the index. As explained in Chapter 13 of the \LaTeX -TeX Manual, the `\makeindex` command is going to read in an `.xdx` “expanded index file”, which is produced by the index program from the `.ndx` file that \LaTeX -TeX writes. The index program is discussed in the next chapter. In this chapter, we are only concerned with typesetting the `.xdx` file. In the default style, we will be using a two-column format, with balanced columns at the end.

Page 114 of The \LaTeX -TeX Manual gives a sample piece from an `.xdx` file that illustrates almost all the features that we have to consider; the only additional feature is `\PageSpan`, mentioned on page 117.

38.1. Preliminaries. As with `lamstex.stf`, we begin with

```
\catcode'\@=11
\let\alloc@=\alloc@
```

Since `*` has a special role of indicating options in index entries, if an index entry actually had a `*` in it that symbol will appear as `\asterisk`. So we will need to state

```
.....
\let\asterisk=*
.....
```

so that `\asterisk` will print out a `*`. We’ve printed dotted lines around this definition, since it will actually only be made within the definition of `\makeindex`—for any material inserted before or after `\makeindex`, the control sequence `\asterisk` will simply be undefined.

As with the bibliography (section 29.4), we will be redefining `\lkerns@` and `\nkerns@` for properly placing punctuation before line-breaking penalties,

```
.....
\def\lkerns@{\null\kern-1sp\kern1sp}
\def\nkerns@{\null\kern-2sp\kern2sp}
.....
```

(we don't really need the `\null`'s that occurred previously, since we're not setting `\vbox`'s, but for simplicity we might as well stick to the old definitions). Once again, we use dotted lines, since the redefinitions will only be made later, by `\makeindex`, so that line breaking commands won't change their meaning until we are actually setting the index. On the other hand, we can give a definition of `\adjustpunct@` (section 29.11) right now; it differs from the previous definition only in the omission of `\closequotes@`:

```

\def\adjustpunct@#1{\count@\lastkern
  \ifnum\count@=0 #1\else
  \ifnum\count@>2 #1\else
  \ifnum\count@<-2 #1\else
    \unkern\unkern\setbox0=\lastbox
    \skip@=\lastskip \unskip
    \count@@=\lastpenalty \unpenalty
    \ifnum\count@ = 2 \unskip \setbox0=\lastbox\fi
    \ifdim\skip@ = 0pt \else \hskip\skip@ \fi
    #1%
    \ifnum\count@=2 \null\hfill\fi
    \penalty\count@@
  \fi\fi\fi}

```

(Since `\adjustpunct@` is stated within `lamstex.tex` itself, we could simply state `\let\closequotes@=\relax`, but restating the definition is preferable in case `\purge{bib}` has been used.)

In all the commands appearing in the `.xdx` file,

```

\Entry 10{acetyl-CoA}{-}{-}{-}{-}
\Page {34}
etc.,

```

we are going to want to ignore the space after the last `}`. In addition, since blank lines might be introduced during editing, or by “error messages”, we will want to skip over any such blank lines—we can't afford to introduce a `\par` until we know that there are no more page numbers left to be printed.¹ So we will

¹In version 1 of \LaTeX -S-TEX, the `index` program was careful not to introduce any blank lines,

use `\FNSS@` to get the first non-space token, and then use a `\futurelet\next` to see if the next token is `\par`, which should also be discarded:

```
\def\ignorepars@{\FNSS@\ignorepars@@}
\def\ignorepars@@{\ifx\next\par
\def\next@\par{\futurelet\next\ignorepars@@}\else
\let\next@=\relax\fi\next@}
```

One of the main tasks of the `lamstex.stb` macros is to provide suitable `\mark's` so that appropriate “continuation” lines may be written at the top of a page when a page break has occurred within an index entry, or subentry, etc. And because of this, column-breaking penalties will require more care.

The `\columnbreak` command was not mentioned in the Manual, but was added for the paper style, and mentioned in the Installation Manual. In version 2 we are also supplying `\nocolumnbreak` and `\newcolumn` (for a short column), although `\columnbreak` is really the only important one. Although these commands essentially insert penalties in vertical mode, we have to make sure that the penalties occur at the proper time with respect to the `\mark's` that are emitted by the `\LETTER` and `\Entry` construction.

The column-breaking macros work quite differently from the line-breaking macros because a penalty cannot be removed once it has been placed on the main vertical list: they will not insert any penalties directly, but will simply set the value of a counter, for later use. We declare the counter now,

```
\newcount\ctype@
```

but the definitions

```
.....
\def\nocolumnbreak{\ctype@=1 }
\def\columnbreak{\ctype@=2 }
\def\newcolumn{\ctype@=3 }
.....
```

inserting a `%` before an otherwise blank line if necessary, and users were warned not to introduce blank lines. However, it seems preferable to have the macros skip the blank lines, since they can make the `.xdx` file look so much nicer.

will only be given within the definition of `\makeindex`. We also need to declare a new counter to store the value of `\ctype@`, for use by the `\output` routine, after other constructions have returned `\ctype@` to 0:

```
\newcount\Ctype@
```

It also turns out that column breaks have to be treated quite differently depending on whether they are inserted in the left column or the right column, and we will need a flag

```
\newif\ifleftcolbreak@
```

Now we define a routine `\cbreak@#1#2` that uses the value of `\ctype@`, before resetting it to 0. The definition begins

```
\def\cbreak@#1#2{\ifcase\Ctype@#1\or\nobreak#2\else
. . .
```

Thus, if `\ctype@` is 0 (the usual situation, when no column-breaking command has recently been given), we simply insert #1, which will normally be penalty and glue. If `\ctype@` is 1 (because a `\nocolumnbreak` command has recently been given), we instead insert `\nobreak` and then #2 (which would normally be #1 without the penalty). If `\ctype@` is 2 or 3 (because a `\columnbreak` or `\newcolumn` command has recently been given), we simply insert a `\break`. Then we reset `\ctype@` to 0. Before doing that last step, however, we store `\ctype@` in `\Ctype@`. Moreover, we set `\ifleftcolbreak@` to be false if `\pagetotal > \pageheight` (i.e., we already have enough on the page for the first column), but true otherwise:

```
\def\cbreak@#1#2{%
\ifcase\Ctype@#1\or
\nobreak#2\else
\global\leftcolbreak@true
\ifdim\pagetotal > \pageheight@
\global\leftcolbreak@false
\fi
```



```

\global\Ctype@=\ctype@
\break
\fi
\ctype@=0 }

```

The value of \Ctype@ will be used by the \output routine, which will then globally reset \Ctype@ to 0.

We have another flag

```

\newif\ifshortlastcolumn@

```

for a final index feature not mentioned in the *L_AT_EX* Manual,

```

.....
\def\shortlastcolumn{\shortlastcolumn@true}
.....

```

if we want the right column on the last page of the index to end short, instead of being spread out to match the height of the left column.

38.2. \LETTER and \Entry. The first \Entry after a \LETTER will be treated somewhat specially, so we will need a flag

```

\newif\ifletter@

```

to tell when a \LETTER line has just been set.

\LETTER will end the previous paragraph (in which the various page numbers for a previous entry are being set) and emit a \mark{} to clear the marks. Then we want

```

\penalty-200
\bigskip

```

to encourage a page break (which really means a column break, since we will be setting in double columns). More precisely, we will use

```

\cbreak@{\penalty-200 \bigskip}\bigskip

```

Thus, when a \nocolumnbreak command has just appeared, we insert only the \bigskip, and when a \columnbreak or \newcolumnbreak command

has just appeared, we just insert a `\break`. Then we set `\ifletter@` to be true, center the letter in bold face in its column, and finally add a `\nobreak\medskip`:

```
.....
\def\LETTER#1{\par
  \mark{}%
  \cbreak@{\penalty-200 \bigskip}\bigskip
  \letter@true
  \centerline{\bf#1}%
  \nobreak\medskip}
.....
```

As usual, the dotted lines indicate that this is a definition that will actually be included within the definition of `\makeindex`.

To deal with the necessary `\mark's` to be provided by `\Entry`, it will be convenient to have five token lists:

```
=====
\newtoks\marktoks@i
\newtoks\marktoks@ii
\newtoks\marktoks@iii
\newtoks\marktoks@iv
\newtoks\marktoks@v
=====
```

Moreover, we will use a flag

```
=====
\newif\ifentry@
=====
```

which `\Entry` will set true, and each `\Page` will set false.

Some aspects of our `\Entry` macro are particular to the default style, namely the particular manner in which entries and subentries are typeset. But the really interesting aspect of `\Entry` is the way that it uses the first two arguments (d_1 , d_2 , in the notation of page 115 of the *L_AT_EX* Manual), to determine which `\mark's` should be emitted; these `\mark's` can be used by any style to define “continuation lines” of any desired sort. To deal with all possibilities, we will need a new counter

```
=====
\newcount\di@
=====
```

which will hold the value of d_2 for the *previous* \Entry.

The definition begins

```
\def\Entry#1#2#3#4#5#6#7{\par\entry@true
\marktoks@i={#3}\marktoks@ii={#4}\marktoks@iii={#5}%
\marktoks@iv={#6}\marktoks@v={#7}%
\ifcase#1%
\or
. . .
```

Thus, since the case $\#1 = d_1 = 0$ is not supposed to occur, we consider first the case $\#1 = d_1 = 1$, which means that we have a first order entry, i.e., $\#4, \dots, \#7$ are empty; in this case $\#2 = d_2$ should be 0, since this new first order entry can't have any number of groups agreeing with the previous entry.

We begin with

```
\ifletter@{\else\mark{}}\fi
\cbreak@\relax\relax
\noindent@
\mark{10{\the\marktoks@i}}%
\hangafter 1 \hangindent=.5em{#3}%
```

Thus:

- (1) We issue a `\mark{}` to clear the marks, unless the `\Entry` comes right after a `\LETTER`, which has already done this.
- (2) If a `\nocolumnbreak` appeared right before the `\Entry` we add a `\nobreak`; and if a `\columnbreak` or `\newcolumn` appeared, we add a `\break`, and also set `\ifleftcolbreak@` to the appropriate value and store `\ctype@` in `\Ctype@` before restoring `\ctype@` to 0.
- (3) Then we start an unindented paragraph, which begins with a `\mark`,

```
\mark{10{\the\marktoks@i}}%
```

containing $\#1\#2$, and then a single group containing $\#3$ (without any expansion). The

```
\hangafter1 \hangindent=.5em{#3}%
```

means that after the entry #3 is printed we are set up for printing the various page numbers, with hanging indentation if there are so many that they won't fit on a line. We enclose #3 in a group, in case it involves a font change command, and are careful not to allow a space after {#3}, since punctuation still has to be added.

When the paragraph with this entry is completed, the `\mark{10{...}}` will migrate out so that it occurs right after the first line (before any `\baselineskip` glue); consequently, there will be no legal breakpoint between the `\mark` and this first line, containing the typeset {#3}. If a page break occurs after this entry, but before another entry, which contributes a new `\mark`, then after the page has been `\output` the value of `\botmark` will be

10{#3}

When a style file has to add continuation lines at the beginning of the next page, the 10 will tell it that the style broke at a main entry, which will be contained in the next group. (Actually, we will be using `\splitbotmark` rather than `\botmark` when we do two-column setting.)

In summary, our definition begins:

```
\def\Entry#1#2#3#4#5#6#7{\par\entry@true
\marktoks@i={#3}\marktoks@ii={#4}\marktoks@iii={#5}%
\marktoks@iv={#6}\marktoks@v={#7}%
\ifcase#1%
\or
\ifletter@\else\mark{ }\fi
\cbreak@\relax\relax
\noindent@
\mark{10\the\marktoks@i}%
\hangafter1 \hangindent=.5em{#3}%
\or
```

Now let's consider the next case, $d_1 = \#1 = 2$, so that we have a first order subentry. In this case $d_2 = \#2$ may be either 0 or 1. When $\#2 = 0$, we have a subentry that does *not* have the same main entry as the previous line, like the

situation

```
\Entry 10{abracadabra}{-}{-}{-}
\Page {5}
\Entry 20{acetyl-CoA}{action of}{-}{-}
\page {32}
```

As mentioned on page 116 of the \LaTeX Manual, in some styles this might be printed as

```
abracadabra, 5
acetyl-CoA, action of, 32
```

In the default style, we will still print

```
abracadabra, 5
acetyl-CoA
action of, 32
```

but we still have to treat the `\mark's` differently for the cases $d_2 = \#2 = 0$ and $d_2 = \#2 = 1$.

For the case $\#2 = 0$ we want

```
\ifletter@ \else \mark{-} \fi
\cbreak@ \relax \relax
\noindent
\mark{20{\the\marktoks@i}{\the\marktoks@ii}}%
\hangafter1 \hangindent=.5em{\#3} \par \nobreak
\noindent \hangafter1 \hangindent=1.5em \quad{\#4}%
```

Thus:

- (1) We clear the previous mark (the one for abracadabra).
- (2) We emit the mark

```
20{\#3}{\#4} [20{acetyl-CoA}{action of}, in the above example]
```

and then set up the proper format for printing things. If the page breaks in the middle of this entry, then `\botmark` will have the value

```
20{\#3}{\#4}
```

and the `\output` routine will be clued in as to what sort of continuation lines to add.

The other case, when `#2 = 1`, corresponds to the more common situation where we have something like

```
\Entry 10{acetyl-CoA}{-}{-}{-}
\Page {32}
\Entry 21{acetyl-CoA}{action of}{-}{-}{-}
\Page {45}
```

Here we will use

```
\mark{10{\the\marktoks@i}}%
\cbreak@relax\relax
\noindent@
\mark{21{\the\marktoks@i}{\the\marktoks@ii}}%
\hangafter 1 \hangindent=1.5em\quad{#4}
```


Even when the `\cbreak@` contributes nothing (the usual situation), there is a legal break point between the first `\mark` and the `\noindent@`'ed paragraph that follows, because of the `\parskip` glue (possibly `0pt`) before this paragraph. If the page breaks at this point, then `\botmark` will be this first

```
\mark{10{acetyl-CoA}}
```

which is precisely right: we have broken in the middle of a first level entry. If the page breaks after the sub-entry begins, but before a new `\Entry`, which contributes a new `\mark`, then `\botmark` will be

```
21{acetyl-CoA}{action of}
```

again giving the `\output` routine all the information it needs to decide what sort of continuation lines to insert.

 Although we are using `\cbreak@relax\relax` in all clauses of our `\Entry` definition, other styles might give more detailed choices. For example, it might

be considered better to break right before an entry than right before a subentry. We might use something like

```
\cbreak@{\ifletter@else\penalty-20 \fi}\relax
```

for the case #1 = 1, thereby slightly encouraging a column break before main entries, except those immediately following a \LETTER.

The case $d_1 = \#1 = 3$ begins similarly: When #2 = 0 we use

```
\ifletter@else\mark{}\fi
\cbreak@\relax\relax
\noindent
\mark{30{\the\marktoks@i}{\the\marktoks@ii}{\the\marktoks@iii}}%
\hangafter1 \hangindent=.5em{\#3}\par\nobreak
\noindent@\hangafter1 \hangindent=1.5em\quad{\#4}\par\nobreak
\noindent@\hangafter1 \hangindent=2.5em\quad{\#5}%
```

and when #2 = 1 we use

```
\mark{10{\the\marktoks@i}}%
\cbreak@\relax\relax
\noindent@
\mark{31{\the\marktoks@i}{\the\marktoks@ii}{\the\marktok@iii}}%
\hangafter1 \hangindent=1.5em\quad{\#4}\par\nobreak
\noindent@\hangafter1 \hangindent=2.5em\quad{\#5}%
```

But the case #2 = 2 is a little different, because the previous entry might have been an \Entry 20 or an \Entry 21, and to provide the style with all possible information, we want to supply the correct result. So we use

```
\mark{2\boxed{\number\dii@}{\the\marktoks@i}{\the\marktoks@ii}}%
\cbreak@\relax\relax
\noindent@
\mark{32{\the\marktoks@i}{\the\marktoks@ii}{\the\marktoks@iii}}%
\hangafter1 \hangindent=2.5em\quad{\#5}%
```

where `\dii@` will have the proper value, because, at the very end of this long `\ifcase` (with sub `\ifcase`'s) we finish off the whole definition of `\Entry` with

```
\dii@=#2\relax
\letter@false\ignorepars@}
```

We won't bother writing out the whole definition, since these parts explain everything interesting that is going on (moreover, the whole definition appears only within the definition of `\makeindex`).

38.3. `\Page`, etc. Using `\adjustpunct@` (section 1), we will define `\Page` for the index setting by

```
.....
\def\Page#1{\ifentry@\adjustpunct@,\enspace\else
\adjustpunct@, \fi\entry@false{#1}\ignorepars@}
.....
```

In other words, the first `\Page` after an `\Entry`, when `\ifentry@` is still true, will add a comma after the entry, and then an `\enspace`, while successive `\Page`'s, for which `\ifentry@` will be false, will simply add a comma after that page number and then an ordinary space (with suitable juggling of line-breaking penalties). We put `#1` in a group in case it involves a font change.

The `\Pagespan`, `\Topage` combination requires a bit more manipulation, because the index program may have produced something like

```
\Pagespan {123}
\Topage {123}
```

which should be printed simply as '123', rather than as '123-123'.

We begin with

```
.....
\def\Pagespan#1{\ifentry@\adjustpunct@,\enspace\else
\adjustpunct@, \fi
\entry@false\def\frompage@{#1}{#1}\ignorepars@}
.....
```

Thus, we supply punctuation just as with `\Page`, and before printing `#1` we also store `#1` in `\frompage@`, for use with `\Topage`, defined by


```

.....
\def\Topage#1{\def\next@{#1}\ifx\next@\frompage@
\else\hbox{--}{#1}\fi\ignorepars@}
.....

```

The `\hbox{--}` gives an en-dash at which a line break can't occur (it is what the \LaTeX construction `@--` produces).

The variant `\PageSpan#1#2`, which has a control sequence as its first argument, puts in punctuation in the same manner as `\Pagespan`, and also stores the page argument, #2, in `\frompage@`, as well as the control sequence, #1, in `\pagecs@`. Then it begins a group in which the definition of `\Topage` will be changed, so that the `\Topage` following `\PageSpan` can act differently than when it follows `\Pagespan`:

```

.....
\def\PageSpan#1#2{\ifentry@\adjustpunct@,\enspace
\else\adjustpunct@,\fi\entry@false
\def\pagecs@{#1}\def\frompage@{#2}%
\bgroup\let\Topage=\Topage@\ignorepars@}
.....

```

Then `\Topage@` will apply `\pagecs@` to the pages spanned (or to a single page if the two numbers are the same), and supply the `\egroup` to match the `\bgroup`:

```

.....
\def\Topage@#1#2{\def\next@{#2}\ifx\next@\frompage@
\pagecs@{\frompage@}\else
\pagecs@{\frompage@\hbox{--}#2}\fi
\egroup\ignorepars@}
.....

```

We use `\pagecs@{...}` in case `\pagecs@` is a control sequence with an argument; the `\bgroup` and `\egroup` provide the necessary extra level of grouping in case `\pagecs@` is a font change instruction. We might as well give the definition of `\Topage@` now, rather than within the definition of `\makeindex`, since `\Topage@` can't be typed by the user anyway.

38.4. \Xref, etc. The description of the `*x` option on page 108 of the \LaTeX Manual, and the output it supposedly produces in the `.xdx` file, as described on page 115 of the Manual, is incorrect.¹ There are actually *two* options, `*x` and `*X`. The `*x` option is to be used when only a cross-reference is

¹As mentioned on pages 8–9 of the \LaTeX Installation Manual.

desired, and not a reference to the page on which the entry is marked, while the `*X` option indicates a reference to the page on which the entry is marked, together with a cross reference.

If an entry is always marked with the `*x` option, then the pages on which it is marked are completely irrelevant, and it will give rise to an `\Entryxref` line in the `.xdx` file (possibly followed by various `\Morexref` lines). Otherwise, it will give rise to something like

```

(A)      \Entry ...
          \Xref ...
          \Morexref ...
          . . .
          \Page ...
          \Page ...
          \Xref ...
          \Morexref ...
          . . .

```

with `\Xref...` material occurring both before and after the `\Page`'s because some styles might want to print the *'see also'* before the page numbers instead of after them.

In any case, as the Manual shows on page 115, each `\Xref`, `\Morexref`, and `\Entryxref` will contain `\See`, which can be defined as needed.

In the default style we will be ignoring the `\Xref` and `\Morexref`'s in (A) that come before the first `\Page`; this means that we want `\Xref` and `\Morexref` to operate only when `\ifentry@` is false.

`\Xref`, which will occur after some `\Page`, will add a semi-colon after the page, and then print *'see also'* before its argument. As with the bibliography macros, we use `\semicolon@` instead of `;` for the benefit of French styles (section 29.13), and we add `\/` before it,

```

.....
\def\Xref#1{\ifentry@\else
\def\See##1{\/\adjustpunct@\semicolon@\space
{\it see~also\/}~##1}{#1}\fi\ignorepars@}
.....

```

while `\Morexref` will simply add a comma after the previous cross-reference page, and then a space:

```
.....
\def\Morexref#1{\ifentry@\else
\def\See##1{\adjustpunct@, ##1}{#1}\fi\ignorepars@}
.....
```

An \Entryxref is basically just an \Entry, after which we add the final argument rather than succeeding \Page's.

```
.....
\def\Entryxref#1#2#3#4#5#6#7#8{%
\Entry{#1}{#2}{#3}{#4}{#5}{#6}{#7}%
\def\See##1{\adjustpunct@, {\it see\/}~##1}%
{#8}\ignorepars@}
.....
```

(The \adjustpunct@ is added in case the user adds a line-breaking command, like

```
\Entryxref 10{adrenalin}{-}{-}{-}{-}{\linebreak\See {epinephrine}}
```

by hand.)

Suppose that we want to print the 'see also' before the page numbers instead of after them, which means that we need to use the \Xref's and \Morexref's that appear before the first \Page, ignoring \Xref's and \Morexref's following a \Page.

For this we would introduce a new flag

```
\newif\ifxref@
```

which \Xref would set true and \Entry would set false, and then we could use definitions like

```
\def\Xref#1{\ifentry@
\def\See##1{ {\it see also\/} ##1}{#1}\fi\ignorepars@}
\def\Morexref#1{\ifentry@
\def\See##1{\adjustpunct@, ##1}{#1}\fi\ignorepars@}
\def\Page#1{\ifxref@\newline\else
\ifentry@\adjustpunct@, \enspace\else
\adjustpunct@, \fi\fi\entry@false{#1}\ignorepars@}
```

Then entries without cross-references would appear as before, while entries with cross-references would appear as

acetyl-CoA *see also* derivative, pyruvate 30, 56, 78

Instead of the `\newline` we might prefer ‘. ’ or perhaps `\Xref` would add ‘(`{\it see also\}`)’ while `\Page` would add ‘)’ instead of `\newline`.

Similarly, any punctuation after the last page number would have to be added by the next `\Entry` (the definition of `\makeindex` would also have to take care of the last `\Entry` specially). But no one ever adds this punctuation, so why worry about it.

38.5. Preliminaries for double columns. We first declare a `(dimen)` register to store `\vsize`,

```
\newdimen\pageheight@
\pageheight@=\vsize
```

since `\vsize` will be changed when we are storing material for double columns, and a dimension `\doublepageheight@`, which is a little more than twice `\pageheight@`,

```
\newdimen\doublepageheight@
\doublepageheight@=2\pageheight@
\advance\doublepageheight@ by 1 pc
```

We also declare a `(dimen)` register to store the original `\hsize`,

```
\newdimen\pagewidth@
\pagewidth@=\hsize
```

The `\makeindex` routine will be setting columns 3 inches wide, so we will be changing `\hsize`. Consequently, we must redefine `\makeheadline` and `\makefootline` from plain TeX, so that `\line (= \hbox to \hsize)` is replaced by `\hbox to \pagewidth@`:

```
\def\makeheadline{\vbox to 0pt{\vskip-22.5pt
\hbox to \pagewidth@{\vbox to 8.5pt{\the\headline}\vss}}%
\nointerlineskip}%
\def\makefootline{\baselineskip=24pt
\hbox to \pagewidth@{\the\footline}}%
```

(The definition of \makeheadline isn't really needed for the default style, since we are going to keep \headline={}, but we might as well take care of the general mechanism.)

As in Appendix E of *The T_EXbook*, page 417, we will be setting “pages” of length \doublepageheight@, so that we can automatically balance the columns on the last page. We need another flag

```
\newif\iffirstindexpage@
```

since the first page of the index is special, having a header ‘INDEX’ that extends across both columns.

38.6. \makeindex. The \makeindex routine begins

```
\def\makeindex{\checkmainfile@\par\vfill\break
\begingroup
<all previous definitions enclosed in dotted lines
(including the definition of \Entry)>
\hspace=3in
```

Of course all #'s in any of those <previous definitions> are changed to ##'s.

Next we set

```
\global\vspace=\doublepageheight@
```

so that we will have accumulated slightly more than we need to fill both columns of the page before the \output routine is called (we use \global here because it will be required later, when we set \vspace within the \output routine). And we set

```
\maxdepth=\maxdimen
```

so that the depth of \box255 will always be the actual depth of its last line; this is needed at one point later (compare *The T_EXbook*, page 262–263).

Then we set

```
\global\firstindexpage@true
\global\advance\vspace by -60pt
```

The 60 points is need to leave room for a 30 point heading above both columns.

Next, we set

```
\everypar={}\parindent=0pt
```

just as a precaution, and then

```
\rightskip=0pt plus3em \spaceskip=.3333em \xspaceskip=.5em
```

for ragged right setting (we allow even more “raggedness” than the definition of `\raggedright` in plain T_EX).

And then we want to change the `\output` routine to one called

```
\doublecolumns@
```

before reading in the appropriate `.xdx` file; after this file has been read we will issue an empty `\mark{}`, change the `\output` routine to one called `\balancecolumns@`, for the final page of the index, and then supply the `\endgroup` that matches the `\begingroup` at the beginning of the definition. For anything that follows the `\makeindex`, we will once again have `\output={\plainoutput}`. However, `\vsize` has been globally changed, so we must reset it to `\pageheight@`.

```
\def\makeindex{\checkmainfile@
\par\vfill\break
\begingroup
<definitions from before>
\hsize=3in
\global\vsize=\doublepageheight@
\maxdepth=\maxdimen
\global\firstindexpage@true
\global\advance\vsize by -60pt
\everypar={}\parindent=0pt
\rightskip=0pt plus3em \spaceskip=.3333em \xspaceskip=.5em
```

```

\output={\doublecolumns@}%
\input\mainfile@.xdx
\mark{}%
\output={\balancecolumns@}%
\vfil\break
\endgroup
\global\vsizе=\pageheight@}

```

38.7. \combinecols@. Both \doublecolumns@ and \balancecolumns@ will store the two columns in \box0 and \box2 and then use the routine \combinecolumns@, which prints these boxes side by side, together with the proper headline and footline.

```

\def\makeindex@W{Index}
\def\combinecolumns@{%
\setbox\outbox@=\vbox{\makeheadline
\vbox to\pageheight@{\boxmaxdepth=\maxdepth
\iffirstindexpage@
\vbox to30pt{\vskip10pt
\hbox to\pagewidth@{\hfil\bf
\uppercase\expandafter{\makeindex@W}\hfil}\vfil}
\nointerlineskip
\fi
\wd0=\hsize \wd2=\hsize
\setbox0=\hbox to\pagewidth@{\box0 \kern.5in\box2}
\dimen@=\dp0 \box0 \kern-\dimen@\vfill}
\makefootline}
{\noexpands@ \let\style=\relax
\shipout@\box\outbox@
}

```

This is like the \shipout@ occurring in our definition of \plainoutput (page 452), except that the \pagebody must be replaced by a \vbox to \pageheight@, containing the two boxes with a separation of .5in. There are also two minor adjustments to be made:

- (1) We must explicitly declare the widths of `\box0` and `\box2` to be `\hsize` (`\hsize` is simply the width of paragraphs within these boxes; an inserted `\box` in vertical mode could be any width—compare the footnote on page 324).
- (2) Our `\vbox to\pageheight@` ends with the

```
\hbox to\pagewidth@{\box0 \kern.5in\box2}
```

followed by `\vfill` (preceded by a `\kern`). That is because on the last page, where we balance columns, this `\hbox` will generally be smaller than `\pageheight@`. Most of the time, of course, the `\vfill` will be irrelevant, except that we have to be sure to insert the `\kern` before the `\vfill`, so that the depth of the `\hbox` will not cause an `Overfull box` (compare page 456).

In addition, on the first page of the index, our two columns, `\box0` and `\box2` will have been made 30 points smaller than `\pageheight@`, in order to accommodate the

```
\vbox to30pt{\vskip10pt
\hbox to\pagewidth@{\hfil\bf
\uppercase\expandafter{\makeindex@W}\hfil}\vfil}%
\nointerlineskip
```

that we insert. We use `\makeindex@W`, so that `\newword` can be used to determine the heading that will be printed.

Notice that we have the `\noexpands@ . . . \shipout@` in a group, so that the `\noexpands@` won't interfere with font change instructions in "continuation" lines contributed later (see page 514) by the `\output` routine. (Compare page 452.)

After all this, we simply declare `\iffirstindexpage@` to be false, set `\vsize` to `\doublepageheight@`, its proper value for all index pages after the first, and `\advancepageno`:

```
\def\makeindex@W{Index}
\def\combinecolumns@{%
\setbox\outbox@=\vbox{\makeheadline
```



```

\ vbox to\pageheight@\boxmaxdepth=\maxdepth
\ iffirstindexpage@
\ vbox to30pt{\vskip10pt
\ hbox to\pagewidth@\{ \hfil\bf
\ uppercase\expandafter{\makeindex@W}\hfil}%
\ vfil}%
\ nointerlineskip
\ fi
\ wd0=\hsize \wd2=\hsize
\ setbox0=\hbox to\pagewidth@\{ \box0 \kern.5in\box2}%
\ dimen@=\dp0 \box0 \kern-\dimen@\vfill}%
\ makefootline}%
{\noexpands@ \let\style=\relax
\ shipout@\box\outbox@
}%
\ global\ vsize=\doublepageheight@
\ global\ firstindexpage@false
\ advancepageno}

```

38.8. \doublecolumns@. For the \doublecolumns@ routine, used for all but the last page, we will need a new <dimen> register

```

\newdimen\prevcoldepth@

```

The routine begins with

```

\dimen@=\pageheight@
\iffirstindexpage@ \advance\dimen@ by -30pt \fi

```

so that \dimen@ is the height of the columns we want for the page.

Now we have to consider the possibility that a \columnbreak or a \newcolumn command caused the \output routine to be invoked. More

precisely, we want to insert the code

```
\ifleftcolbreak@
  \global\leftcolbreak@false
  . . .
\else
  - - -
\fi
```

The flag `\ifleftcolbreak@` is set true (by `\cbreak@`) when a `\columnbreak` or `\newcolumn` command has been issued when we are setting the left column,¹ and it is used only in the current test; since we immediately reset it to be false, this means that `\ifleftcolbreak@` is true precisely when we have forced a break in the left column. We will simply let ‘. . .’ for this case be

```
\vbox to\dimen@{\dimen@=\dp255 \unvbox255
  \ifnum\Ctype@=3 \kern-\dimen@\vfill\fi}
\allowbreak
```

In other words, when our `\output` routine has been forced while we still have only enough material for the first column, we do not do any `\shipout`, but simply put material back on the main vertical list. What we put back is a `\vbox to\dimen@` (remember that `\dimen@` is the desired height of columns for this page), containing all the material in `\box255`, either stretched out, when we had a `\columnbreak` (so that `\Ctype@` is 2), or filled with empty space at the bottom, when we had a `\newcolumn` (so that `\Ctype@` is 3). As before (page 510), we need to add a `\kern` before the `\vfill`.

The next time the `\output` routine is called, `\doublecolumns@` will use the rest of the routine, still to be defined, which splits the full `\box255` into two boxes of height `\dimen@`; the first part will then have to be this `\vbox to\dimen@` that we have just added to the main vertical list.¹

Having gotten that consideration out of the way, let’s consider the other situation, where we haven’t forced a break in a left-hand column. Now we want to `\vsplit \box255` into two boxes of height `\dimen@`. During the process we

¹I.e., when `\pagetotal` is \leq `\pageheight@`.

¹The only problem here is that the `\output` routine could conceivably be `\balancecolumns` the next time it is called—see the discussion on page 521 ff.

will set `\splittopskip=\topskip` and `\splitmaxdepth=\maxdepth` (this assignment will be local, since the `\output` routine, `\doublecolumns`, is implicitly enclosed in a group). Splitting off the first column requires no special work,

```
\splittopskip=\topskip \splitmaxdepth=\maxdepth
\setbox0=\vsplit255 to\dimen@
```

But for the second column, we can't simply

```
\setbox2=\vsplit255 to\dimen@
```

This works when `\Ctype@` is 0 or 1 (no column-breaking commands, except perhaps a `\nocolumnbreak`, were given). But when `\Ctype@` is 2 or 3 (a `\columnbreak` or `\newcolumn` command was given) the `\vsplit`'ing is irrelevant—we should simply let `\box2` be `\box255`. And when `\Ctype@` is 3 (the `\newcolumn` case), we need to add `\vfill` at the bottom of the box. For a later step, we also want to store the depth of the last line of the second column in `\prevcoldepth@`.

Since the cases when `\Ctype@` is 0 and 1 are treated the same, we consolidate the cases by first stating

```
\ifnum\Ctype@=0 \global\Ctype@=1 \fi
```

Then we can use

```
\ifcase\Ctype@ \or % case 1 (originally 0 or 1)
\setbox2=\vsplit255 to\dimen@
\global\prevcoldepth@=\dp2
\or % cases 2 and 3
\global\prevcoldepth@=\dp255 % store the depth first
\setbox2=\vbox to\dimen@{\unvbox255
\ifnum\Ctype@=3 \vfill\kern-\prevcoldepth@\vfill\fi}
\fi
```

Once we've determined the columns, `\box0` and `\box2`, in this way, we use

```
\combinecolumns@
```

to `\shipout` the desired page.

Now we have reached the point where we want the `\output` routine to add “continuation” lines (these lines, inserted by the `\output` routine, will go before anything remaining in `\box255`, since we will add these lines before we `\unvbox255`; and both go before anything else that happens to remain on the main vertical list when the page break was chosen).

The default routine adds continuation lines at the beginning of the left column on a new page to reflect the `\mark's` at the end of the right column on the previous page. This means that we will want to use `\splitbotmark` when `\box2` was obtained as a `\vsplit255 to\dimen@`, but `\botmark` otherwise. The specific continuation lines to be constructed from these marks will be contributed by a routine, to be defined shortly,

```
\continue@ (the \mark's) \continue@
```

It will be convenient to consider `\continue@` to be followed by at least 7 (possibly empty) arguments, and we will use

```
\ifcase\Ctype@ \or % case 1, where \vsplit was used
\expandafter\continue@\splitbotmark{}\relax\relax\relax
\relax\relax\relax\continue@
\else % cases 2 and 3
\expandafter\continue@\botmark{}\relax\relax\relax
\relax\relax\relax\continue@
\fi
```

As we will see in a minute, `\continue@` will also attend to an important detail about spacing.

After these continuation lines, we simply use

```
\ifvoid255 \else \unvbox255 \penalty\outputpenalty\fi
```

to finish the `\else` clause of our definition. And, finally, after the entire `\if... \else... \fi` has been done, we reset `\Ctype@` to be 0:

```
\def\doublecolumns@{%
\dimen@=\pageheight@
\iffirstindexpage@ \advance\dimen@ by -30pt \fi
```

```

\ifleftcolbreak@
\global\leftcolbreak@false
\vbox to\dimen@{\dimen@=\dp255
\unvbox255 \ifnum\Ctype@=3 \kern-\dimen@\vfill\fi}%
\allowbreak
\else
\splittopskip=\topskip \splitmaxdepth=\maxdepth
\setbox0=\vsplit255 to\dimen@
\ifnum\Ctype@=0 \global\Ctype@=1 \fi
\ifcase\Ctype@ \or
\setbox2=\vsplit255 to\dimen@
\global\prevcoldepth@=\dp2
\else
\global\prevcoldepth@=\dp255
\setbox2=\vbox to\dimen@{\unvbox255
\ifnum\Ctype@=3 \kern-\prevcoldepth@\vfill\fi}%
\fi
\combinecolumns@
\ifcase\Ctype@ \or \expandafter
\continue@\splitbotmark{}\relax\relax\relax
\relax\relax\relax\continue@
\else
\expandafter\continue@\botmark{}\relax\relax\relax
\relax\relax\relax\continue@
\fi
\ifvoid255 \else \unvbox255 \penalty\outputpenalty \fi
\fi
\global\Ctype@=0 }

```

The \continue@ routine is followed by at least 7 (possibly empty) arguments,

```
\def\continue@#1#2#3#4#5#6#7#8\continue@
```

Since `\continue@` is used in a combination

```
\continue@(value of \splitbotmark or \botmark){}\relax\relax\relax
\relax\relax\relax\continue@
```

the test

```
\def\next@{#1}\ifx\next@\empty
```

isolates the case where the `\mark` is empty (in all other cases the mark will begin with a digit). When this is not the case, we want

```
\noindent@#3 ({\it continued\/})\par
```

as the first continuation line. The default style also introduces slightly differently formatted continuation lines for any sub-entries, where the number of levels of sub-entries is known from the value of `#1`,

```
\ifnum#1>1 \noindent@\enspace{#4 {\it continued\/}}\par\fi
. . .
\ifnum#1>4 \noindent@\enspace{#7 {\it continued\/}}\par\fi
```

After these continuation lines are added, we have to attend to the detail mentioned on page 263 of *The T_EXbook*, except that now we have to consider two different cases:

(1) Suppose first that all of `\box255` has been used up, and that the first box waiting to be placed on the new current page has height h . This box will be preceded by interline glue g , since the page break that created it broke right before the glue. If that box were the first thing to be placed on the new page, the glue g would be irrelevant, since T_EX discards glue when the current page does not contain any boxes (*The T_EXbook*, page 112); instead of this glue, T_EX would simply insert the glue determined by `\topskip`. But this glue will *not* be discarded if continuation lines have already been added.

We have stored the depth of the line preceding this first box in `\prevcoldepth@`; thus, the glue g satisfies

$$\text{\prevcoldepth@} + g + h = \text{\baselineskip}$$

But now we need new glue g' such that

$$\text{\prevdepth} + g' + h = \text{\baselineskip}$$

So we need to add $\text{\prevcoldepth@} - \text{\prevdepth}$ glue; thus, when continuation lines are added, we need to

$\text{\global\advance\prevcoldepth@ by -\prevdepth \kern\prevcoldepth@}$

(We can change \prevcoldepth@ like this, instead of using the more complicated code

$\text{\dimen@=\prevcoldepth@ \advance\dimen@ by -\prevdepth \kern\dimen@}$

because \prevcoldepth@ won't be used any more.)

(2) When \box255 has not been used up, however, the situation is quite different, since this \box255 is what remains from a \vsplit , so that the glue g is simply enough to make

$$g + h = \text{\topskip}$$

We want glue g' such that

$$g' + h + \text{\prevdepth} = \text{\baselineskip}$$

so we need to add

$\text{\baselineskip} - \text{\prevdepth} - \text{\topskip}$

extra glue.

Actually, in both cases (1) and (2), \baselineskip should really be $\text{\baselineskip} + \text{\parskip}$, though this doesn't make any difference in the final result for (1).

The whole definition is thus

```
\def\continue@#1#2#3#4#5#6#7#8\continue@{%
\def\next@{#1}\ifx\next@\empty
```

```

\else
  \noindent#3 ({\it continued\/})\par
  \ifnum#1>1 \noindent\enspace(#4 {\it continued\/})\par\fi
  \ifnum#1>2 \noindent\enspace(#5 {\it continued\/})\par\fi
  \ifnum#1>3 \noindent\enspace(#6 {\it continued\/})\par\fi
  \ifnum#1>4 \noindent\enspace(#7 {\it continued\/})\par\fi
  \ifvoid 255
    \global\advance\prevcoldepth@ by -\prevdepth \kern\prevcoldepth@
  \else
    \skip@=\baselineskip \advance\skip@ by \parskip
    \advance\skip@ by -\prevdepth
    \advance\skip@ by -\topskip
    \vskip\skip@
  \fi
\fi}

```

bd If we had also wanted continuation lines at the top of right hand columns, `\doublecolumns@` would have to be more complicated:

```

\newif\ifleftcol@
\newdimen\leftcoldepth@

\def\doublecolumns@{set \dimen@ as before}
\ifleftcol@
  \global\leftcolfalse@
  \ifleftcolbreak@
    \global\leftcoldepth@=\dp255
    {corresponding part of old \doublecolumns@ routine}
    {add continuation lines,
     adjusting space using \leftcoldepth@}
  \else
    {\splittopskip=\topskip \splitmaxdepth=\maxdepth
     \global\setbox1=\vsplit255 to\dimen@
     \global\prevcoldepth@=\dp1 }
    \box1 \allowbreak
    {add continuation lines,
     adjusting space using \prevcoldepth@}
  \fi

```



```

\else
  \global\leftcol@true
  {the \else part of old \doublecolumns@ routine}
\fi
\global\Ctype@=0 }

```

Thus, \doublecolumns@ would end up being called twice for each page, even if there was no column break in the left column.

38.9. \balancecolumns@. The \balancecolumns@ routine basically follows that of page 417 of *The T_EXbook*, except that we have to be a little more careful about the final glue (the routine in *The T_EXbook* was used for setting *The T_EXbook*, where the balanced columns are then followed by more material), and we also have to allow for a short last column.

We begin with

```

\setbox0=\vbox{\unvbox255 \unskip}
\dimen@=\ht0

```

using \unskip to completely eliminate \vfill that may have been added by the \bye (this \vfill might not be there if the page break actually occurs before the final line); this prevents the depth of \box255 from being counted in the height of \box0.

Now we basically want to divide \dimen@ by 2, and split \box0 into two pieces of that height. But we make a slight adjustment: When we split \box0 into two pieces, the top line of the second piece, which currently has \baselineskip glue before it, should only get \topskip glue. So we use

```

\advance\dimen@ by \topskip
\advance\dimen@ by -\baselineskip
\divide\dimen@ by 2

```

—now \dimen@ is the ideal height for the two pieces into which we hope to split \box0.

Then we set

```

\splittopskip=\topskip

```

and use a `\loop` for splitting. This `\loop` will operate inside a group with `\vbadness=10000`, so that Underfull `\vbox`'es won't be reported unless the columns are actually bad after the balancing act:

```
{\vbadness=10000
  \loop
  \global\setbox3=\copy 0
  \global\setbox1=\vsplit3 to\dimen@
  \ifdim\ht3 > \dimen@ \global\advance\dimen@ by 1pt
  \repeat}
```

Briefly, we try splitting to the ideal `\dimen@`, but if we can't get two columns of the same height we want the left column to be the bigger; so, if the right column turns out to be bigger, we keep increasing `\dimen@` by 1 point until we get a split where the left column is at least as big as the right column. We always `\split` a copy of `\box0` rather than `\box0` itself, so that `\box0` is always available for the next iteration of the `\loop`.

We used `\global\advance` here because we need to know the final value of `\dimen@`. But to keep assignments of `\dimen@` local, we will modify this code to

```
\global\dimen@i=\dimen@
{\vbadness=10000
  \loop
  \global\setbox3=\copy 0
  \global\setbox1=\vsplit3 to\dimen@i
  \ifdim\ht3 > \dimen@ \global\advance\dimen@i by 1pt
  \repeat}
```

so that `\dimen@i` is the final dimension we are interested in.

Now we have to set each of `\box0` and `\box2`, which will be used by `\combinecolumns@`, to a `\vbox to\dimen@i` (at this point we may get an Underfull `\vbox` message):

```
\setbox0=\vbox to\dimen@i{\unvbox1}
\setbox2=\vbox to\dimen@i{\dimen@=\dp3 \unvbox3
  \ifshortlastcolumn@ \kern-\dimen@ \vfill\fi}
```

The second column, \box2 is supplied with \vfill when we want it to be short; as usual (compare page 456 and 510), a \kern is required before the \vfill to prevent the depth of the box from being counted as part of the height, possibly producing a spurious Overfull box.

The complete code is:

```

\def\balancecolumns@{%
  \setbox0=\vbox{\unvbox255 \unskip}%
  \dimen@=\ht0 \advance\dimen@ by \topskip
  \advance\dimen@ by -\baselineskip
  \divide\dimen@ by 2
  \splittopskip=\topskip
  \global\dimen@i=\dimen@
  {\vbadness=10000
  \loop
    \global\setbox3=\copy 0
    \global\setbox1=\vsplit 3 to\dimen@i
    \ifdim\ht3 > \dimen@i \global\advance\dimen@i by 1pt
  \repeat}%
  \setbox0=\vbox to\dimen@i{\unvbox1}%
  \setbox2=\vbox to\dimen@i{\dimen@=\dp3 \unvbox3
  \ifshortlastcolumn@ \kern-\dimen@ \vfill\fi}%
  \combinecolumns@}


```

That completes all the routines from lamstex.stb, so we end, as in lamstex.tex itself, with

```

\def\alloc@#1#2#3#4#5{\global\advance\count1#1by\@ne
  \ch@ck#1#4#2\allocationnumber\count1#1
  \global#3#5\allocationnumber
  \wlog{\string#5\string#2\the\allocationnumber}}
\catcode'\@=\active

```

 Balancing an index can be problematical because of final letters with only a few entries. For example, an index might end

yak (<i>continued</i>)	Z
habitat of, 124	zap, 13–18
yam, 124	z-axis, 12, 256, 300
yang, 187	zebra, 121
y-axis, 89, 345	zed, 103, 105
yclept, 102	Zen, iii, vi, 122
yellow, 103	zenith, 97,103
yellow fever, 108	zero, 11, 14, 56, 79
Yellow Pages, 223	zero-sum, 118
yeti, 301	zeta (ζ), 133
yin, 186	zilch, <i>see</i> zero
yodel, 224	zone, 204, 208
	Zorn's lemma 112, 145

with the first column `Underfull`. This is the best `\vsplit`, for if the left column were any bigger it would have to include the **Z** and the space after it, and at least one more line (not to mention all the extra space that would normally go *before* the **Z**).

If we added a `\newcolumn` right before the zebra `\Entry` in the `.xdx` file (and also specified `\shortlastcolumn`), the output would look acceptable,

yak (<i>continued</i>)	zebra, 121
habitat of, 124	zed, 103, 105
yam, 124	Zen, iii, vi, 122
yang, 187	zenith, 97,103
y-axis, 89, 345	zero, 11, 14, 56, 79
yclept, 102	zero-sum, 118
yellow, 103	zeta (ζ), 133
yellow fever, 108	zilch, <i>see</i> zero
Yellow Pages, 223	zone, 204, 208
yeti, 301	Zorn's lemma 112, 145
yin, 186	
yodel, 224	

Z

zap, 13–18
z-axis, 12, 256, 300

although we would get `Overfull \vbox` messages (the `\newcolumn` causes the `\output` routine to be invoked while we still have `\output={\doublecolumns}`, so the left column will be made into a `\vbox to\pageheight@`).

If we added the `\newcolumn` before the `\LETTER Z` (and also had specified `\shortlastcolumn`), we would get

yak (<i>continued</i>)	Z
habitat of, 124	zap, 13–18
yam, 124	z-axis, 12, 256, 300
yang, 187	zebra, 121
y-axis, 89, 345	zed, 103, 105
yclept, 102	Zen, iii, vi, 122
yellow, 103	zenith, 97,103
yellow fever, 108	zero, 11, 14, 56, 79
Yellow Pages, 223	zero-sum, 118
yeti, 301	zeta (ζ), 133
yin, 186	zilch, <i>see</i> zero
yodel, 224	zone, 204, 208
	Zorn's lemma 112, 145

(again with `Overfull \vbox` messages).

Of course, the most reasonable thing would probably be to add a `\columnbreak` a little before the `yak` entry, so that the left column on the final page would be a little longer.

Chapter 39. The index program

The index program takes the raw data in an `.ndx` file and creates an `.xdx` file. Specifically, it is invoked as

```
index <filename>
```

to operate on `<filename>.ndx`, producing `<filename>.xdx`. Among its many tasks, the index file essentially sorts entries alphabetically. As indicated below, the `*a{<string>}` option is used to indicate that for purposes of alphabetizing, the entry should be treated as if it were `<string>`.

The index program was originally written in Microsoft C, for MS-DOS machines. Porting to other machines will presumably be done by modifying this code. However, this chapter presents the complete specifications for the index program, in case any one wants to write it from scratch.

It should be pointed out that currently the alphabetizing part of the program is essentially set up for English spelling, with the presumption that “foreign” words (e.g., those with accented letters) will be given an `*a{<string>}` option. There should obviously be versions (or perhaps switches) for other languages that will provide transparent alphabetizing for words in those languages, so that the `*a` option will only be needed for words that are “foreign” to those languages.

It might also be mentioned that the current version of the program specifically assumes that the final line ends with a `<carriage-return>`, which should be true of the `.ndx` files that `TEX` writes. Some systems might want to modify this assumption.

39.1. The `.ndx` file. The `.ndx` file may contain certain lines

```
\define<control sequence> . . .
```

that come from `\idefine`'s in the main file, as well as certain lines

```
\abbrev*<control sequence>{...}
```

that come from `\iabbrev`'s in the main file. Moreover, as explained on

page 113 of the \LaTeX -TeX Manual, a sequence of commands of the form

```
\pageorder⟨control sequence⟩{⟨pre page material⟩}{⟨post page material⟩}{⟨order⟩}
```

may have been inserted at the very beginning of the .ndx file (after the .ndx file was automatically made by T E X); here each ⟨control sequence⟩ is some numbering style command, and the ⟨order⟩'s are 1, 2, 3, . . . , in that sequence.

Aside from this, the .ndx file should consist of pairs of lines

```
"⟨entry⟩⟨options⟩"  
{⟨number⟩}{⟨control sequence⟩}{⟨prefix⟩}{⟨postfix⟩}
```

where ⟨number⟩ is a decimal number, ⟨control sequence⟩ is some numbering style control sequence (followed by a space, because T E X will always add the space when it writes to the .ndx file), and ⟨prefix⟩ and ⟨postfix⟩ are arbitrary, possibly empty, strings. The ⟨control sequence⟩ is normally ' $\backslash\text{arabic}_\perp$ '.

Actually, the index program as currently written does not assume that " is the delimiter appearing at the beginning and end of the ⟨entry⟩ lines. It is perfectly legitimate (compare section 10.8) to have something like

```
<⟨entry⟩⟨options⟩>  
{⟨number⟩}{⟨control sequence⟩}{⟨prefix⟩}{⟨postfix⟩}
```

Any left delimiter other than { or \ may be used; the index program takes the ⟨entry⟩⟨option⟩ combination to be everything between the first symbol and the last non-space symbol on the line.

Each ⟨entry⟩ has one of the forms

```
⟨entry⟩1  
⟨entry⟩1, ⟨entry⟩2  
⟨entry⟩1, ⟨entry⟩2, ⟨entry⟩3  
⟨entry⟩1, ⟨entry⟩2, ⟨entry⟩3, ⟨entry⟩4  
⟨entry⟩1, ⟨entry⟩2, ⟨entry⟩3, ⟨entry⟩4, ⟨entry⟩5
```

The ⟨options⟩ are a possibly empty sequence of the following types of entries, where ⟨n⟩ stands for 2, 3, 4 or 5. Actually, 1 is also allowed, but simply treated

as if the 1 weren't there. There are optional spaces between such entries, including an optional space at the beginning and end of the \langle options \rangle :

<code>*a{string}</code>	(alphabetize as)
<code>*\langlen\ranglea{string}</code>	(same for subentry at level \langle n \rangle)
<code>*-{string}</code>	(print before entry)
<code>*\langlen\rangle-{string}</code>	(same for subentry at level \langle n \rangle)
<code>*+{string}</code>	(print after entry)
<code>*\langlen\rangle+{string}</code>	(same for subentry at level \langle n \rangle)
<code>*e{control sequence}</code>	(apply control sequence to entry)
<code>*\langlen\ranglee{control sequence}</code>	(same for subentry at level \langle n \rangle)
<code>*p{control sequence}</code>	(apply control sequence to page number)
<code>*f</code>	(begin page span)
<code>*F</code>	(begin page span, variant form)
<code>*t</code>	(end page span)
<code>*x{string}</code>	(cross reference as)
<code>*X{string}</code>	(list and cross reference as)

In each

```
\abbrev*{control sequence}{...}
```

line, the '...' string should also be a sequence of these possible options, and error messages should be given if it is not, or if `\abbrev` is otherwise used incorrectly.

The `*x` and `*X` options are meant to be mutually exclusive, so an error message should be given if both occur in the same \langle options \rangle list.

39.2. The index program. Here now are the steps taken by the index program.

(I) Preprocessing:

(1) The index program first uses any `\pageorder` lines at the beginning,

```
\pageorder{control sequence}1{pre1}{post1}1
\pageorder{control sequence}2{pre2}{post2}2
. . .
```


to set up an ordering for arrays of the form

$$\langle \text{number} \rangle \langle \text{control sequence} \rangle \langle \text{prefix} \rangle \langle \text{postfix} \rangle$$

Namely, all arrays

$$\langle \text{number} \rangle_1 \langle \text{control sequence} \rangle_1 \langle \text{pre}_1 \rangle \langle \text{post}_1 \rangle$$

come first, with the ordering then determined by the $\langle \text{number} \rangle$'s, all arrays

$$\langle \text{number} \rangle_2 \langle \text{control sequence} \rangle_2 \langle \text{pre}_2 \rangle \langle \text{post}_2 \rangle$$

come next, All other arrays come after these, and are simply ordered first by the order of the $\langle \text{number} \rangle$, and then purely alphabetically by the remaining fields.

Error messages presumably ought to be given if any `\pageorder` line has the wrong form, if the $\langle \text{order} \rangle$'s don't occur in the proper order, or if two different $\langle \text{order} \rangle$'s are assigned to identical material.

(2) Next we consider all lines

$$\backslash \text{abbrev} * \langle \text{control sequence} \rangle \{ . . . \}$$

(which may appear anywhere in the `.ndx` file).

In each case, any occurrence of $* \langle \text{control sequence} \rangle$ in any entry line is replaced by `' . . . '`; then the `\abbrev` lines are eliminated.

(3) All `\define` lines anywhere in the `.ndx` file are stored in a safe place, before being removed; they will be written at the beginning of the `.xdx` file.

(II) Sorting:

(1) For each

$$\langle \text{entry} \rangle \langle \text{options} \rangle$$

line, if $* a \langle \text{string} \rangle$ appears, replace $\langle \text{entry}_1 \rangle$ by $\langle \text{string} \rangle$, and define $\overline{\langle \text{entry}_1 \rangle}$ to be the old $\langle \text{entry}_1 \rangle$. Otherwise define $\overline{\langle \text{entry}_1 \rangle}$ to be empty.

Similarly, if $* \langle n \rangle a \langle \text{string} \rangle$ appears, replace $\langle \text{entry}_n \rangle$ by $\langle \text{string} \rangle$, and define $\overline{\langle \text{entry}_n \rangle}$ to be the old $\langle \text{entry}_n \rangle$. Otherwise define $\overline{\langle \text{entry}_n \rangle}$ to be empty.

(2) Each pair of lines

```
"(entry)<options>"
{(number)}{(control sequence)}{(prefix)}{(postfix)}
```

now gives rise to a collection \mathcal{F} of fields:

E1 is $\langle \text{entry}_1 \rangle$ followed by one space.

E1+ is empty unless $*-\langle \text{string} \rangle$ appears, in which case it is $\langle \text{string} \rangle$ followed by one space (note that $\langle \text{string} \rangle$ may already have one space at the end).

E1- is empty unless $*+\langle \text{string} \rangle$ appears, in which case it is $\langle \text{string} \rangle$ followed by one space.

E2 is $\langle \text{entry}_5 \rangle$ followed by one space (hence just a space if $\langle \text{entry}_5 \rangle$ is empty).

E2- is empty unless $*2-\langle \text{string} \rangle$ appears, in which case it is $\langle \text{string} \rangle$ followed by one space.

E2+ is empty unless $*2+\langle \text{string} \rangle$ appears, in which case it is $\langle \text{string} \rangle$ followed by one space.

E3, E3+, E3-, ..., E5, E5+, E5- are similar.

P# is $\langle \text{number} \rangle$.

Pn is $\langle \text{control sequence} \rangle$ (usually $\langle \text{\arabic{n}} \rangle$).

P- is the $\langle \text{prefix} \rangle$.

P+ is the $\langle \text{postfix} \rangle$.

$\overline{E}1, \dots, \overline{E}5$ are $\langle \overline{\text{entry}}_1 \rangle, \dots, \langle \overline{\text{entry}}_5 \rangle$.

E1c is empty unless $*e\langle \text{control sequence} \rangle$ appears, in which case it is $\langle \text{control sequence} \rangle$, followed by one space.

E2c is empty unless $*2e\langle \text{control sequence} \rangle$ appears, in which case it is $\langle \text{control sequence} \rangle$, followed by one space.

E3c, E4c, E5c are similar, using $*3e$, $*4e$ and $*5e$.

Pc is empty unless $*p\langle \text{control sequence} \rangle$ appears, in which case it is $\langle \text{control sequence} \rangle$, followed by one space.

PT (page type) is f if $*f$ appeared, F if $*F$ appeared, t if $*t$ appeared, and s (for single page) otherwise.

x is empty unless $*x\langle \text{string} \rangle$ appears, in which case it is $\langle \text{string} \rangle$, followed by one space.

X is empty unless $*X\langle \text{string} \rangle$ appears, in which case it is $\langle \text{string} \rangle$, followed by one space.

(3) We first sort these collections \mathcal{F} alphabetically on E1, then among those with identical E1 on E1-, then on E1+, then on E2, ..., E5+. If we ignore the

- and + fields for the moment, this just alphabetizes the entries and subentries. The spaces that were added after each (entry_n) are meant to ensure that these entries are really sorted alphabetically; for example, it produces the order

```
Time
Time table
Time tables
Times
Times tables
```

which will eventually appear in the index as

```
Time
  table
  tables
Times
  tables
```

The other fields, E1-, E1+, ..., allow entries with additional elements to appear in the proper order. For example, for the entry

```
"Time *-{ ' ' } *+{ ' ' }"
```

E1 is Time and E1- is ' '. This will come after the entry line

```
"Time"
```

where E1 is also Time but E1- is empty, so that we will obtain the order

```
Time
"Time"
```

in the index.

After using these fields for sorting, we next use the fields

```
P# Pn P- P+
```

where the sorting order for these arrays is just that described in step (1) of (I).

There may be certain collections \mathcal{F} that are distinct, but agree for all the fields so far. They can be left in any order, say the order in which they appear in the .ndx file, or further sorted purely alphabetically according to the remaining fields $\bar{E}1, \dots, \bar{E}5, E1c, \dots, E5c$ and Pc.

The PT, x and X fields are important for further processing, but need not be considered in the sorting.

After the sorting is completed, the additional spaces added to various fields are no longer needed, and they are deleted in the next step.

(III) Writing the .xdx file:

(1) First, as indicated in step (3) of part (I), all `\define` lines originally in the .ndx file are written to the .xdx file.

(2) The remaining material in the .xdx file will be in sections, each preceded by a blank line, then a line

```
\LETTER ...
```

and then another blank line. The line

```
\LETTER A
```

will normally be the first of these, and will precede all entries that begin with a or A;

```
\LETTER B
```

will precede all entries that begin with b or B; etc. No such line appears for a letter if there are no entries beginning with this letter. Things like `\LETTER !` can appear if entries begin with ! (though normally one would expect the `*a` option to be used for such entries).

(2) In most cases, each collection \mathcal{F} gives rise to a pair of lines in the .xdx file:

```
\Entry d1d2{...}{...}{...}{...}{...}
\Page {...}
```

except that if several collections have the exact same `\Entry` line, then the corresponding `\Page` lines are listed under it in order, with modifications to be mentioned in a minute, involving the PT, x and X fields.

For the first {...} in the \Entry line, '...' is

$$\{E1-\}\{E1c\{E1\}\}\{E1+\}$$

except that E1 is replaced by $\bar{E}1$ if this is non-empty.

Note that there are braces around E1 (or $\bar{E}1$) so that it will be the argument of the control sequence E1c. However, these braces are omitted if E1c is empty. Moreover, all other unnecessary braces are omitted: the first set of braces is omitted if E1- is empty, the third set is omitted if E1+ is empty, and the middle set is also omitted if both are empty. (Remember that this entire sequence, with extraneous braces removed, is enclosed in the first set of braces in the \Entry line.)

The second {...} in the \Entry line is similar, using fields E2-, E2c, E2 and E2+. And similarly for the other groups.

d_1 is the number of {...} that are non-empty, i.e., not of the form {}.

d_2 is the number of {...} that agree with the most recent \Entry line.

Normally, the \Page lines have the form

$$\backslash\text{Page } \{Pc\{\{P-\}\{Pn\{P\#\}\}\{P+\}\}\}$$

But Pn is omitted when it is '\arabic{...}' (the usual case). Moreover, superfluous braces are removed, as before, when fields are empty.

Duplicate \Page lines for the same entry are eliminated. But if duplicates actually come from lines for which Pc is different, then each \Page line after the first is printed as

```
%% ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
%
% \Page {Pc}{...}
% Duplicate page number formatted differently
%
%% \\\\////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

where Pc is the control sequence for that line. For readability, there should be a blank line before and after this "error message" (the current version of the index program actually prints things slightly differently).

When PT is f, however, `\Page` is replaced by `\Pagespan`. And if PT is F, then `\Page` is replaced by `\PageSpan`; moreover, the line is then written in the form

```
\Pagespan {Pc}{P-}{Pn{P#}}{P+}
```

with the Pc as a separate argument. Extraneous braces are removed, as usual.

When PT is t, `\Page` is replaced by `\Topage`. However, if it was preceded by a line for which PT is F, then once again we write

```
\Topage {Pc}{P-}{Pn{P#}}{P+}
```

with the Pc as a separate argument.

When a `\Page` line for which PT is s is encountered between lines for which PT is f or F and the corresponding line for which PT is t, it is ignored, *unless it has* a different Pc. In this case the new `\Page` line is printed as the error message

```
%% ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
%
% \Page {new Pc{...}}
% Page {new Pc{...}} within span starting at old Pc{...}
%
%% \//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

Other such error messages that should be issued are indicated on pages 118–119 of the *L^AT_EX* Manual.

This description is complete only for entries that do not have an `*x` or `*X` option. For entries that do have these options, the processing is somewhat different. Recall that `*x` indicates a cross-reference without a reference to the page on which the entry is marked, while `*X` indicates a cross-reference together with a reference to the page.

The situation where at least one occurrence of `(entry)` has an `*x` or `*X` option can happen in different ways:

Case 1: Every occurrence of the `<entry>` involves `*x` (and thus not `*X`):

```
<entry> ... *x{<string1>}
<entry> ... *x{<string5>}
<entry> ... *x{<string9>}
```

In this case the `.xdx` file should contain

```
\Entryxref d1d2{...}{...}{...}{...}{...}{\See {<string1>}}
\Morexref {\See {<string5>}}
. . .
```

Case 2: The opposite case, in which at least one of the following occurs:

(1) `<entry>` occurs at least once with the `*X` option, so that the corresponding page number should appear. Other occurrences of `<entry>` may or may not have `*x` options.

(2) `<entry>` occurs at least once with *neither* `*x` nor `*X`, and thus there is at least one page number that should appear because no cross-referencing is involved at all. Other occurrences of `<entry>` may have either `*x` or `*X` options.

In these cases, let `<string1>`, `<string5>`, `...`, be all the `<string>`'s, in alphabetical order, that occur in either `*x` or `*X` options, and let

```
\Page {...}
\Page {...}
. . .
\Page {...}
```

be the lines that would be written considering all instances that do not involve the `*x` option (but possibly the `*X` option). These lines may actually involve `\Pagespan`, `\PageSpan` and `\Topage`.

Then the `.xdx` file should contain

```
\Entry  $d_1d_2$ {...}{...}{...}{...}{...}
\Xref {\See {string1}}
\Morexref {\See {string5}}
. . .
\Page {...}
\Page {...}
. . .
\Page {...}
\Xref {\See {string1}}
\Morexref {\See {string5}}
. . .
```


Part VII

The Style Files

Introduction

`lamstex.tex` provides default treatments for all the main $\text{L}_\mathcal{M}\mathcal{S}$ - $\text{T}_\text{E}\text{X}$ constructions, while `lamstex.stf` and `lamstex.stb` provide default treatments for front matter and back matter constructions.

In some styles, some of these constructions are a little more complicated, or have to be combined in different ways; and certain features, like running heads, haven't yet been considered at all. So in this Part we discuss the standard $\text{L}_\mathcal{M}\mathcal{S}$ - $\text{T}_\text{E}\text{X}$ style files, although in a somewhat more perfunctory manner than the treatment of previous Parts.

As usual (compare section 27.2), we will be using double horizontal lines for code from these files.

Chapter 40. The paper style

The file `paper.st` begins

```
\catcode'\@=11
```

to allow us to access and create private control sequences, followed by

```
\ifx\paperst@\relax\catcode'\@=\active  
\endinput\else\let\paperst@=\relax\fi
```

which prevents `paper.st` from being read in twice (compare section 1.2), a necessity because of the `\NameHL` commands that occur later (we can't `\NameHL1` heading twice, since that involves `\define\heading` twice). Then we have

```
\let\alloc@=\alloc@@
```

so that we can create new counters, etc., without having them written to the `.log` file (compare page 38).

40.1. Basic settings. Next we set basic parameters that differ from the default style:

```
\hsize=30pc  
\vsize=42pc  
\parindent=1em  
\normallineskiplimit=1pt  
\advance\hoffset 48pt  
\advance\voffset 78pt
```

`\hoffset` and `\voffset` are changed to center these small pages within an $8\frac{1}{2} \times 11$ inch piece of paper.

40.2. Fonts and point sizes. Since parts of the paper will be set in 8 point type, we first have to introduce numerous extra fonts:

```

\font@\ninerm=cmr9
\font@\eightrm=cmr8
\font@\sixrm=cmr6
\font@\eighti=cmmi8   \skewchar\eighti='177
\font@\sixi=cmmi6     \skewchar\sixi='177
\font@\ninesy=cmsy9   \skewchar\ninesy='60
. . .

```

$\mathcal{A}\mathcal{M}\mathcal{S}$ -TEX's `\font@` is basically like the TEX primitive `\font`, except that the font name is added to `\fontlist@` (see section 3.7), for use by $\mathcal{A}\mathcal{M}\mathcal{S}$ -TEX's `\syntax` command. The only slightly surprising things here are the `\ninerm` and `\ninesy`—they are needed only because they are used fleetingly in the definition of `\big` for eight point type (see below).

If the additional AMS fonts have been loaded, using any of

```

\loadmsam
\loadmsbm
\loadeufm

```

then flags `\ifmsamloaded@`, `\ifmsbmloaded@`, and `\ifeufmloaded@`, respectively, will have been set true. And if

```

\loadbold

```

was used to load `cmmib..` and `cmmbsy...` fonts, flags `\ifcmmibloaded@` and `\ifcmsyloaded@` will have been set true.

In each case, we should then load additional fonts for eight point:

```

\ifmsamloaded@
\font@\eightmsa=msam8
\font@\sixmsa=msam6
\fi
. . .

```

```

\ifcmbsyloaded@
\font@\eightcmbsy=cmbsy8 \skewchar\eightcmbsy='60
\font@\sixcmbsy=cmbsy6 \skewchar\sixcmbsy='60
\fi

```

(If even more crazy families are to be loaded, like eufb, eusm, eusb, eurm, eurb, further clauses will have to be added, but I couldn't bear including them.)

Now we want to define `\tenpoint` and `\eightpoint`, following the basic scheme used in Appendix E of *The T_EXbook*. For literal mode, we will want spaces to give different amounts of glue in the two different point sizes, so we first declare a new glue register,

```

\newskip\ttglue@

```

A few macros need to know what point size is currently used, so this information is kept in a control sequence `\pointsize@`. Thus, the definition of `\tenpoint` will be of the form:

```

\def\tenpoint{\def\pointsize@{10}}%
. . .
}

```

After `\pointsize@` is defined, `\normalbaselineskip` is set to 12pt (at the end of the definition, `\normalbaselines` will be stated, which actually sets `\baselineskip` to this value); on the other hand, `\normallineskip` and `\normallineskiplimit` will each have the value 1pt for both 10 point and 8 point type.

Then values of `\abovedisplayskip`, . . . , for the spacing above and below displayed formulas are set.

Next we use

```

\textonlyfont@rm\tenrm
. . .
\textonlyfont@bf\tenbf

```

so that `\rm` means `\tenrm` in 10 point type, etc. $\text{\AA}MS\text{-}\text{\TeX}$'s `\textonlyfont@` construction means that `\rm, \dots`, will give error messages if used in math mode. (However, if the file `pcompat.tex` has been `\input`, and `\plainfonts` has been specified, then `\textonlyfont@` will be redefined so that `\rm, \dots`, will function exactly as in plain \TeX .)

Next we have to set up the families for math fonts, and then `\let\big=\tenbig@`, which is defined separately, since `\big` is different for 10 point type and 8 point type (following Appendix E, however, `\bigg, \dots`, are kept the same).

If the user has typed `\syntax`, so that `\ifsyntax@` is true, then we might as well skip all these font assignments, and we can give `\big` a simpler definition, in which the actual size used is irrelevant,

```

\ifsyntax@
  \def\big##1{{\hbox{${\left##1\right.}$}}}
\else
  \let\big=\tenbig@
  \textfont0=\tenrm . . .
  . . .
  \textfont3=\tenex . . .
  \textfont\itfam=\tenit
  . . .
  \textfont\bffam=\tenbf . . .
  \ifmsamloaded@
  . . .
\fi
. . .
\fi

```

We also `\let\tt=\tentt`, although we don't bother producing a `\tt` math family, and we then use

```
\tt\ttglue@=.5em minus.15em
```

Thus, we switch to this `\tt` font (so that `em` refers to that font), and then assign the value of `\ttglue@`. This switch to `\tt` is temporary, since we will switch to `\rm` at the end.

Before ending, however, we have to set `\box\strutbox`, which determines struts,

```
\setbox\strutbox=\hbox{\vrule\height8.5pt\depth3.5pt\width0pt}
```

as well as `\strutbox@`, which $\mathcal{A}\mathcal{M}\mathcal{S}$ - \TeX uses for its special math struts,

```
\setbox\strutbox@=\hbox{\vrule\height8pt\depth3pt\width0pt}
```

We also set the $\mathcal{A}\mathcal{M}\mathcal{S}$ - \TeX dimen

```
\ex@=.2326ex
```

(as explained in `amstex.doc`, I don't remember why this value is chosen!). And then the definition of `\tenpoint` ends with

```
\normalbaselines
```

so that we have set `\baselineskip`, etc., to their “normal” values, and then

```
\ifmmode\else\rm\fi
```

to switch to `\rm`; the `\ifmmode` test is required in case a construction like

```
$$ \tenpoint ... $$
```

is used, since `\rm` will give an error message in math mode!

After the definition of `\tenpoint`, we still have to give the definition

```
\def\tenbig@#1{\hbox{${\left#1\ vbox to8.5pt{}}
\right.\n@space$}}
```

and then we can switch to `\tenpoint`:

```
\tenpoint
```

NOTICE that we must not say `\tenpoint` until `\tenbig@` has been defined, or `\big` will initially be undefined.

The definition of `\eightpoint` is exactly analogous, and followed by the definition

```
\def\eightbig@#1{\hbox{\textfont0=\ninerm\textfont2=\ninesy
\left#1\vbox to6.5pt{\right.\n@space$}}}
```

which is where `\ninerm` and `\ninesy` get used; then, for example, `\big(` in `\tenpoint` selects the `\ninerm` (.

Then we use

```
{\catcode'\active
\gdef\litcodes@{\def {\allowbreak\hskip\ttglue@}}}
```

so that if literal mode is used within `\tenpoint` or `\eightpoint`, the active space will have the desired width.

With the fonts now declared, we can properly print the \LaTeX logo; for best effects somewhat different choices are required at different point sizes:

```
\def\LamSTeX{L\kern-.4em\raise.3ex\hbox{\$ssize\Cal A$}%
\def\next@{10}\ifx\next@\pointsize@
\kern-.25em\else\kern-.3em\fi
\lower.4ex\hbox{\def\next@{10}\ifx\next@\pointsize@
\eighty\else\sixsy\fi M}%
\kern-.1em{\$Cal S$}-\TeX}
```

Although `\claimformat` is supposed to be the same as in the default style, most people don't have an 8 point "caps and small caps" font, so `\tenpoint` and `\eightpoint` don't define `\smc`, which simply has the default `lamstex.tex` meaning of `\tensmc` (choosing the font `cmcsc10`). So we substitute `\rm\uppercase` instead:

```
\def\claimformat@#1#2#3{\medbreak\noindent@
\def\next@{8}\ifx\next@\pointsize@\next@
\rm\uppercase{#1 {\claim@@F#2} #3}%
\punct@{\null.}\addspace@enspace}
```

```

\else
  \smc#1 {\claim@@@F#2} #3\punct@{\null.}%
  \addspace@\enspace
\fi
\sl}

```

Finally, since we have the smaller `\eightpoint` fonts, we might as well use them in the `\windex@` routine for printing index entries at the right margin, for proofing; we change the strut so that it gives only 10pt between lines.

```

\def\windex@{\ifindexing@
  \expandafter\unmacro@\meaning\stari@\unmacro@
  \edef\macdef@{\string"\macdef@\string"}%
  \edef\next@{\write\ndx@{\macdef@}}\next@
  \write\ndx@{\number\pageno}{\page@N}{\page@P}{\page@Q}}%
\fi
\ifindexproofing@
  \ifx\stariii@\empty\else
    \expandafter\unmacro@\meaning\stariii@\unmacro@\fi
    \insert\margin@{\hbox{\eightpoint
  \vrule\height7\p@\depth3\p@\width\z@\starii@
  \ifx\stariii@\empty\else\tt\macdef@\fi}}\fi}

```

40.3. The .toc levels. Since `paper.st` has so many different heading levels,

```

\HL1 alias \chapter
\h11 alias \section
\h12 alias \subsection
\h13 alias \topic
\h14 alias \subtopic

```

we want to allow `\toclevel` to indicate how many levels should be written to the `.toc` file: `\toclevel1` should indicate that only `\chapter`'s are written, `\toclevel2` that `\chapter`'s and `\section`'s are written, etc. We might as well allow `\toclevel0` to mean nothing is written (although then it was fairly silly to specify `\tocfile` to begin with).

We do this in terms of a counter `\toclevel@`, initially with the value 3 (so that `\chapter's`, `\section's` and `\subsection's`—the only numbered heading levels in this style—are written):

```
\newcount\toclevel@
\toclevel@=3
\def\toclevel#1{\toclevel@=#1\relax}
```

To get this to work, we have to restate the definitions of `\HLtoc@` and `\hltoc@` from `lamstex.tex`. The only difference in the definition of `\HLtoc@` is that we don't do anything if `\toclevel@` is less than 1:

```
\def\HLtoc@{%
  \iftoc@
    \ifnum\toclevel@ < 1
    \else
      (old code for \HLtoc)
    \fi
  \fi}
```

Similarly, `\hltoc@` only writes to the `.toc` file when `\hllevel` is less than `\toclevel@` (so if `\toclevel@` is 1, no `\hl` is written, if `\toclevel@` is 2, only `\hl1` is written, etc., as desired):

```
\def\hltoc@{%
  \iftoc@
    \ifnum\hllevel@ < \toclevel@
      (old code for \hltoc@)
    \fi
  \fi}
```

40.4. Setting up heading levels. Next we have to set up the heading levels. As mentioned on page 217, we want to make sure that the basic `\NameHL's` in this style file don't write information to the `.toc` file even if `\tocfile` has been specified before the `\docstyle` line. So we define

```
\def\notocwrite@#1#2#3{\iftoc@\test@true\else\test@false\fi
\toc@false#1{#2}#3\iftest@\toc@true\fi}
```

For example,

```
\notocwrite@\NameHL1\heading
```

first stores the value of `\iftoc@` in `\iftest@`, declares `\tocfalse@`, calls `\NameHL1\heading`, and restores `\toc@true` if `\iftoc@` was initially true. The `\NameHL` and `\Namehl` constructions don't involve `\iftest@`, so this is safe. (The definition of `\notocwrite@` encloses argument #2 in braces in case we have something like `\NameHL{10}`....)

Next we state

```
\newfontstyle\heading{\smc}
```

Thus, `\HL1`, *alias* `\heading`, has numbers set in `\smc` (which will also be used for the heading title, in the definitions to follow).

One additional feature of the paper style is that a `\subsection` (`\h12`) is only allowed within a `\section` (`\h11`), not directly within a `\heading`. So we need a flag

```
\newif\ifinsection@
```

which `\heading` will set false and `\section` will set true. Aside from this, `\heading` is handled by defining '`\HL@1`' almost exactly as in the default style,

```
\expandafter\def\csname HL@1\endcsname#1\endHL-{%
\global\insection@false
\bigbreak\medskip
{\locallabel@
\global\setbox1=\vbox{\Let@\tabskip\hss@
\halign to\hsize{\smc\hfil\ignorespaces##\unskip\hfil\cr
\expandafter\ifx\csname HL@W1\endcsname\empty\else
\csname HL@W1\endcsname\space\fi
```

```
{\HL@F\ifx\thelabel@\empty\else\thelabel@\space\fi}%
\ignorespaces#1\crr}}}%
\unvbox1
\nobreak\medskip}
```

merely adding some extra space before the heading, and setting the heading in `\smc`.

One of the interesting features of the paper style is the definition

```
\def\appendices{%
\NameHL1\appendix
\Reset\appendix1%
\newnumstyle\appendix\Alph
\newword\appendix{Appendix}%
}
```

This means that after `\appendices` has appeared, `\appendix` may be used (and `\heading` may no longer be used—compare page 218); the new headings created by `\appendix` will be numbered ‘A’, ‘B’, . . . , and the headings will now say ‘Appendix A . . .’, etc.

Notice that when `\appendices` is used, `\NameHL1\appendix` *will* be written to the `.toc` file. As we will see in section 8, this will enable `\maketoc` to properly process any `\appendix` lines that appear afterwards.

Next we have

```
\notocwrite@\Namehl1\section
\newstyle\section#1{#1\null.}
```

so that `\section` can be used for `\hl1`; in addition, we now have a period printed after the `\section` number (the default style omits the period).

Although each `\heading` normally resets `\section` numbers to 1 (this will be handled by ‘`\HL@I1`’, to be defined shortly), we will allow

```
\keepsection
```

analogous to `\keepitem`, to be typed before `\heading`, to prevent this. `\keepsection` will merely set a flag

```
\newif\ifcontinuesection@
\def\keepsection{\global\continuesection@true}
```

which each `\section` should set false (with this arrangement, if `\keepsection` is typed before a `\heading` with no `\section's`, then it carries over to the next `\heading`).

The definition of `\section`, via '`hl@1`', also sets `\ifinsection@` to be true, and the formatting is somewhat different from the default style: instead of running the `\section` title into the text, we start a new paragraph, after a `\smallskip`:

```
\expandafter\def\csname hl@1\endcsname#1{%
  \global\insection@true
  \global\continuesection@false
  \medbreak\noindent@@
  {\locallabel@
   \bf{\hl@F\ifx\thelabel@@\empty\else\thelabel@@\space\fi}%
   \ignorespaces#1\unskip\punct@{\null.}}%
  \par\nobreak\smallskip}
```

For `hl2`, eventually to be named `\subsection`, we must first declare counters and default values:

```
\expandafter\newcount\csname hl@C2\endcsname
\csname hl@C2\endcsname=0
\expandafter\def\csname hl@S2\endcsname#1{#1\null.}
\expandafter\let\csname hl@N2\endcsname=\arabic
\expandafter\let\csname hl@P2\endcsname=\empty
\expandafter\let\csname hl@Q2\endcsname=\empty
\expandafter\def\csname hl@F2\endcsname{\bf}
\expandafter\let\csname hl@W2\endcsname=\empty
```

Then we can

```
\notocwrite@Namehl2\subsection
```

[recall (page 215) that `\Nameh12` must be used after `'\h1@C2'`, ... have been defined]. Then we define `\subsection`, via `'\h1@2'`, giving an error message if the flag `\ifinsection@` is false:

```

\expandafter\def\csname h1@2\endcsname#1{%
  \ifinsection@
    \smallbreak
    \noindent@
    {\locallabel@
      {\h1@F\ifx\thelabel@@\empty\else
        \thelabel@@\space\fi}\bf
      \ignorespaces#1\unskip\punct@{\null.}%
      \addspace@\enspace}%
    \else
      \Err@{\noexpand\subsection not in a \string\section}%
    \fi}

```

The declaration of counters and default values for `\h13` and `\h14` are analogous, except that we can even

```

\expandafter\let\csname h1@F3\endcsname=\empty
\expandafter\let\csname h1@F4\endcsname=\empty

```

since the numbers for these heading levels aren't even going to be printed.

We can then state

```

\notocwrite@\Nameh13\topic

```

and

```

\notocwrite@\Nameh14\subtopic

```

The definition of `\topic`, via `'\h1@3'`, is

```

\expandafter\def\csname h1@3\endcsname#1{%
  \smallbreak

```

```
\noindent@
{\locallabel@
 \bf\ignorespaces#1\unskip\punct@{\null.}%
 \addspace@\enspace}}
```

while `\subtocic`, defined via `'\hl@4'`, starts as an ordinary paragraph:

```
\expandafter\def\csname hl@4\endcsname#1{%
 \smallbreak
 {\locallabel@\bf\ignorespaces#1\unskip
 \punct@{\null.}\addspace@\enspace}}
```

The initializations are fairly simple:

```
\expandafter\def\csname HL@I1\endcsname{\ifcontinuesection@
 \else\Reset\hl11\fi}
\expandafter\def\csname hl@I1\endcsname{\Reset\hl21%
 \ifx\pref\empty\newpre\hl2{}\else\new\hl2{\pref.}\fi}
\expandafter\def\csname hl@I2\endcsname{\Reset\hl31}
\expandafter\def\csname hl@I3\endcsname{\Reset\hl41}
```

Thus, `\section` numbers are reset to 1 after each `\heading`, unless `\keepsection` preceded it; `\subsection` numbers are reset to 1 at each `\section`, and the n^{th} subsection number is printed as $m.n$, where m is the `\section` number, unless empty (because of a `\section""` command); and `\subsection` resets `\topic` numbers to 1, while `\topic` resets `\subtopic` numbers to 1, even though these numbers aren't printed (they can still be referred to by `\nref`, if the `\topic` or `\subtopic` is given a `\label`).

40.5. Footnotes. The definition of `\vfootnote@` is modified so that the footnotes will be printed in 8 point type:

```
\def\vfootnote@#1{\insert\footins
 \bgroup
 \floatingpenalty2000
 \interlinepenalty=\interfootnotelinepenalty}
```



```

\leftskip=0pt \rightskip=0pt \spaceskip=0pt \xspaceskip=0pt
\eightpoint
\splittopskip=\ht\strutbox \splitmaxdepth=\dp\strutbox
\locallabel@{\noindent@{\foottext@F#1}\modifyfootnote@
\footstrut\futurelet\next\fo@t}

```

Note that the `\strutbox` will now have the value set by `\eightpoint`.

40.6. Additional “top matter” and “end matter” constructions. In addition to `\title`, `\author`, `\affil`, and `\date`, which already appear in the default style, the paper style also has several other constructions that go at the beginning of the paper, and a few that go at the end.

For `\abstract... \endabstract`, as for `\title` and `\author`, we declare a box

```

\newbox\abstractbox@

```

to hold the abstract. Because we are printing a word ‘Abstract’ at the beginning, followed by punctuation and spacing, we then

```

\rightadd@\abstract\to\nofrillslist@
\def\abstract@W{Abstract}

```

The `\box\abstractbox@` is going to be set with

```

\leftskip=24pt \rightskip=\leftskip

```

rather than as a `\vbox` with smaller `\hsize`, so that we can simply `\unvbox` it at the proper time, instead of having to include it in `\centerline`, and worrying about spacing before and after it. We will increase the `\tolerance` to 800 for these smaller lines.

Unfortunately, these values of `\leftskip` and `\rightskip` *do not* influence `\displaywidth` and `\displayindent`, which determine the positioning of displayed formulas within the `\abstract`; the latter are influenced only by `\parshape`—see *The T_EXbook*, page 188. Moreover, we can’t simply assign them the desired values, because, as explained on that page, T_EX assigns them values immediately after the `$$` that begins a display, based on the value

of `\parshape`. Instead we will have to change `\everydisplay` (which, recall from section 16.1, is already non-empty in \LaTeX) to include the desired changes.

```

\def\abstract{\begingroup
  \global\setbox\abstractbox=\vbox\bgroup
  \eightpoint\leftskip=24pt \rightskip=\leftskip
  \everydisplay{\advance\displaywidth by -48pt
    \displayindent=24pt \csname displaymath \endcsname}%
  \tolerance=800
  \noindent@@
  \ifx\abstract@W\empty\else\abstract@W\punct@{\null.}%
    \addspace@\enspace\fi}
\def\endabstract{\egroup\endgroup}

```

`\thanks`, `\keywords`, and `\subjclass` are handled like `\date`:

```

\let\thanks@=\relax
\long\def\thanks#1{\gdef\thanks@{\ignorespaces#1}}

\let\keywords@=\relax
\def\keywords#1{\gdef\keywords@{\ignorespaces#1}}

\let\subjclass@=\relax
\def\subjclass#1{\gdef\subjclass@{\ignorespaces#1}}

```

The `\long` allows several paragraphs of thanks. Extra group is included in the definitions of `\keywords` and `\subjclass` in case a font change is included; but, as we will see, that is unnecessary for `\thanks`. None of these need `\unskip` at the end, because they will all eventually be ended by a `\par`.

`\address` is somewhat different, because it can be used arbitrarily many times. A counter `\addresscount@` is used to keep track of the number, and each use of `\address#1` creates a new control sequence to store the argument:

```

\newcount\addresscount@
\addresscount@=0

```

```

\long\def\address#1{\global\advance\addresscount@ by 1
\expandafter\gdef
\csname address\number\addresscount@\endcsname
  {\ignorespaces#1}}

```

Since `\keywords`, `\subjclass`, and `\address's` are printed at the end of the paper in the paper style, we redefine `\bye` to include this material before adding the `\vfill\supereject\end`:

```

\def\bye{\par
\nobreak
\vskip12pt minus6pt
\eightpoint
\ifx\keywords@\relax\else
\noindent@{\it Keywords.\enspace}\keywords@\par\fi
\ifx\subjclass@\relax\else
\noindent@ 1980 {\it Mathematics subject
classifications\/}\colon@\space\subjclass@\par\fi
\ifnum\addresscount@ > 0
\nobreak
\vskip12pt minus6pt
\loop\ifnum\addresscount@>0
\csname address\number\addresscount@\endcsname\endgraf
\global\advance\addresscount@ by -1
\repeat
\fi
\vfill\supereject
\end}

```

Notice that instead of a `:` we used `\colon@` (page 162). No `\keywords@W` or `\subjclass@W` has been provided, to change ‘Keywords’ or ‘Mathematics subject classifications’, since those are presumably somewhat standard terms; but they could easily be added if needed. In the `\loop` for the `\address's`, we needed `\endgraf` instead of `\par`, since `\loop` isn’t `\long`.

And then we make `\enddocument` synonymous with this new `\bye`:

```
\let\enddocument=\bye
```

The remaining constructions, which occur at the top, are placed by `\maketitle`, just as in the default style. We also declare initially empty elements

```
\let\pretitle=\empty
\let\preauthor=\empty
\let\preaffil=\empty
\let\predate=\empty
\let\preabstract=\empty
\let\prepaper=\empty
```

which can be redefined to add additional elements:

```
\def\maketitle{\hrule\height 0pt \vskip-\topskip
\pretitle
\vskip24pt plus12pt minus12pt
\unvbox\titlebox@
\preauthor
\ifvoid\authorbox@\else
\vskip12pt plus6pt minus3pt \unvbox\authorbox@\fi
\preaffil
\ifvoid\affilbox@\else
\vskip10pt plus5pt minus2pt \unvbox\affilbox@\fi
\predate
\ifx\date@\relax\else
\vskip6pt plus2pt minus2pt\centerline{\rm\date@}%
\let\date@=\relax\fi
\preabstract
\ifx\thanks@\relax\else
\vfootnote@{\}\thanks@
\let\thanks@=\relax\fi
```

```

\ifvoid\abstractbox@ \else
  \vskip15pt plus12pt minus12pt
  \unvbox\abstractbox@
\fi
\prepaper
\vskip18pt plus12pt minus6pt}

```

Notice that the `\thanks` is treated like the text of a footnote, but with no number.

40.7. Bibliography. The only modification required for `\makebib` in the paper style is to switch to 8 point type:

```

\def\makebib{\begingroup\eightpoint
  \bigbreak
  \centerline{\smc\makebib@W}\nobreak\medskip
  \sfcode'\.=1000 \everypar{}\parindent=0pt
  \def\nopunct{\nopunct@true}\def\nospace{\nospace@true}%
  \nopunct@false\nospace@false
  \def\lkerns@{\null\kern-1sp\kern1sp}%
  \def\nkerns@{\null\kern-2sp\kern2sp}%
}

```

For the case where `\UseBibTeX` has been specified, so that the

```

\makebib
. . .
\endmakebib

```

region is replaced by a

```

\bibliography{...}

```

line, we redefine `\beginthebibliography@` so that it will also print in 8 point type:

```

\def\beginthebibliography@#1{\eightpoint
\setbox0=\hbox{#1\ } \bibindent@=\wd0
\bigbreak\centerline{\smc\bibliography@W}\nobreak\medskip
\sfcode'\.=1000 \everypar{ }\parindent=0pt}

```

40.8. \maketoc. Finally, the `\maketoc` command is special, because it is used right in the paper style, rather than in a “front matter file”. So most of the definitions of `lamstex.stf` are also required in `paper.st`:

```

\def\dotleaders ...
\def\Page@ ...
\long\def\widerthanhsiz@
\long\def\setentry@##1##2##3##4{%
. . .
\else
\hbox to\hsiz@{\kern24pt} ...}
\fi}
\def\endstrut@
\newdimen\thehang@
\long\def\longentry@##1##2##3##4{. . .
. . .
\hangafter1 \hangindent\thehang@ \leftskip=24pt
. . .}
\newif\ifemptynumber@
\def\Style@ ... . . . \def\Style@@@ ...
\newdimen\digits

```

with the difference that all entries will be indented 24 points from the left margin (the same amount of indentation that occurs for the `\abstract`).

We also need to have the definition

```

\def\maketoc@W{Contents}

```

so that `\newword\maketoc` can be used before `\maketoc` itself.

Then the definition of \maketoc starts

```
\def\maketoc{\par
\begingroup
\eightpoint \tolerance=800
\unlabel@\noset@
\let\nopunct=\relax \let\nospace=\relax
\let\overlong=\relax
\everypar={}\parindent=0pt
\lineskiplimit=0pt
```

The \checkmainfile@ in the lamstex.stf definition is naturally deleted, and the \par\vfill\break is replaced by a \par. We use \begingroup, because we will be making redefinitions of \HL, etc., and the old definitions will have to be restored after the table of contents is finished; and the \lineskiplimit=0pt, which is so important for the entries of the table of contents (compare section 37.2), will also be confined to this group.

And then come the various redefinitions that also occur in \maketoc in lamstex.stf:

```
\def\HL ...
\setbox0=\hbox{0.00}
\digits=\wd0
\def\hl ...
\def\NameHL ...
\def\Namehl ...
```

The definitions of \HL and \hl are somewhat different, reflecting both the larger number of heading levels, and somewhat different formatting in the table of contents. Then we have to add

```
\NameHL1\heading \Namehl1\section \Namehl2\subsection
\Namehl3\topic \Namehl4\subtopic
```

so that \heading, ..., \subtopic will be suitably redefined while making the table of contents. Naturally we can't add \NameHL1\appendix at the same time, but \appendices will add this to the .toc file, so that \appendix will be suitably redefined at the necessary time when printing the Contents.

Finally, we print the properly centered heading, diminish `\hsize` by 24 points, (so that the right margin will match that of the `\abstract`), `\input` the file `\jobname.toc`, end the group, and add some vertical space:

```
\centerline{\smc\maketoc@W}  
\nobreak  
\vskip18pt plus12pt minus6pt  
\advance\hsize by -24pt  
\input \jobname.toc  
\endgroup  
\vskip 12pt plus8pt minus4pt
```

The definitions involving `\island's` from `lamstex.stf` are not needed, since `paper.st` does not print a list of Figures, Tables, etc., even when a table of contents is made.

And nothing from `lamstex.stb` is required, since `paper.st` does not provide for an index.

Chapter 41. The book style

41.1. Basic settings. The book style has

```
\ifx\bookst@\relax\catcode'\@=\active
\endinput\else\let\bookst@=\relax\fi
```

analogous to the code in `paper.st`, and then some slightly different basic settings,

```
\hsize=30pc
\vsizer=42pc
```

Then, more significantly, it changes `\makeheadline` so that running heads will be separated from the main body of the text by a different amount of space; in addition, the running head is not made into a `\vbox to 0pt`, as in plain TeX—the running heads are a measurable part of the page (the final pages might even be printed with “crop marks” on them, which should naturally take into account the presence of the running heads). `\makefootline` is also changed, although that will only effect the page numbers at the bottoms of special pages, like the initial pages of each chapter:

```
\def\makeheadline{\hbox{%
\botsmash{\line{\vbox to 8.5pt{} \the\headline}}}%
\nointerlineskip\vskip 20pt}
\def\makefootline{\baselineskip=2.5pc
\line{\the\footline}}
```

Just for good measure, we make things like `\title`, `\author`, `\affil`, `\date`, `\undefined`, since they should appear only in the “front matter file” for the book:

```
\let\title=\undefined
\let\author=\undefined
\let\affil=\undefined
\let\date=\undefined
```

and, in particular, we

```
\let\makebib=\undefined
```

since `\makebib` should only occur in the “back matter file”.

41.2. Fonts and point sizes. Exactly the same new fonts are declared as in the paper style, and `\tenpoint` and `\eightpoint` are defined in the same way.

41.3. The .toc levels. `\toclevel` is defined just as in the paper style, and `\HLtoc@` and `\hltoc@` are redefined in exactly the same way.

41.4. Flushing out figures. Both `\part` and `\chapter` are going to flush out any figures from previous `\chapter`'s. As discussed in section 36.7, this means that we will have to modify the `\plainoutput` routine so that empty pages produced at the ends of chapters won't increase the page numbers, and won't get printed. Since this additional process is memory intensive, however, we don't want it to be invoked unless a flag

```
\newif\ifflush@
```

is true, where `\FlushedFigs` and `\NoFlushedFigs` are used to set the flag:

```
\def\FlushedFigs{\global\flush@true}
\def\NoFlushedFigs{\global\flush@false}
```

(The savvy user will specify `\FlushedFigs` near the end of a `\chapter` with `\Aplace`'s and add `\NoFlushedFigs` at the beginning of the next `\chapter`.)

It might seem that `\part` and `\chapter` will flush out figures from a previous `\chapter` if their definitions are of the form

```
\par\vfill\supereject
. . .
{\locallabel@ . . . }
. . .
```

(using the `\par\vfill\supereject` that occurs in the definition of `\bye`). Unfortunately, that may not be sufficient when we have a situation like

```

. . .
\Aplace{...}
\chapter

```

because the `\Aplace`'d material may not get printed until after the remaining aspects of `\chapter` have already set things up for preparing pages for this next `\chapter`. For example, flushed out figures from Chapter 9 might end up having heading levels saying 'Chapter 10 ...'!—which is even worse than having extra blank pages.

Therefore `\part` and `\chapter` will, in all situations, use something like

```
\par\vfill\break\null\kern-\topskip\nobreak\vfill\supereject
```

(essentially the construction that appears in the plain definition of `\dosupereject`), so that we can be sure that any held-over figures are flushed out before the remaining aspects of `\chapter` are brought into play. The problem with this approach is that a blank page will now be produced when there are no figures to be flushed out. However, we will take care of that problem later, during the `\output` routine. We will need a flag

```
\newif\ifSflush@
```

to indicate such special pages, and then we define

```
\def\flush@{\par\vfill\break\null\kern-\topskip\nobreak
\global\Sflush@true\vfill\supereject}
```

41.5. `\part and \chapter`. The book style has a new heading level `\HLO`, *alias* `\part`. We will need a new font for setting it,

```
\font\BF=cmbx10 scaled \magstep3
```

and then we set up things for `\HLO`:

```

\expandafter\newcount\csname HL@C0\endcsname
\csname HL@C0\endcsname=0
\expandafter\def\csname HL@S0\endcsname#1{#1\}/}
\expandafter\let\csname HL@N0\endcsname=\Roman
\expandafter\let\csname HL@P0\endcsname=\empty
\expandafter\let\csname HL@Q0\endcsname=\empty
\expandafter\def\csname HL@F0\endcsname{\BF}
\expandafter\def\csname HL@W0\endcsname{Part}

```

So \HLO levels will be numbered I, II, The word preceding the number will actually be 'PART' rather than 'Part', but we will use \uppercase for this, rather than defining '\HL@W0' as PART, so that Part will appear in the .toc file; this allows the style file for the front matter to print things differently, if desired.

Once these definitions have been made, we can then \NameHLO\part (see page 215). As with the paper style, however, we need to have

```

\def\notocwrite@#1#2#3{\iftoc@\test@true\else\test@false\fi
\toc@false#1{#2}#3\iftest@\toc@true\fi}

```

so that we can

```

\notocwrite@\NameHLO\part

```

We introduce a new flag

```

\newif\ifpart@

```

since \part pages will be special (they will have no number at the bottom and also no running head). Then we define \part, via '\HL@0', which involves quite a few tricky maneuvers.

Our definition of \csname HL@0\endcsname begins with

```

\flush@
\global\part@true
\ifodd\pageno\else\advancepageno\fi

```

where the `\flush@` (section 4) flushes out any figures remaining from the previous `\chapter`, and the third clause insures that `\part`'s always begin on odd-numbered pages.

And then we want something like

```
{\locallabel@
 \global\setbox1=\vbox{\Let@
 \baselineskip=21pt
 \halign{\BF\ignorespaces##\unskip\hfil\cr
 \expandafter\ifx\csname HL@W0\endcsname\empty
 \else\csname HL@W0\endcsname\space\space\fi
 {\HL@@F\thelabel@@}\cr
 \noalign{\vskip30pt}}%
 \uppercase{\ignorespaces#1\cr}}}
```

printing something like 'PART II' on the first line, an extra 30 points of vertical space, and then the title #1 in uppercase letters.

But we want to eliminate the first line and the extra 30 points if both the `<word>` and the number happen to be empty. Moreover,

```
\uppercase{\csname HL@W0\endcsname}
```

won't uppercase the letters in `\csname HL@W0\endcsname`, and even

```
\uppercase\expandafter{\csname HL@W0\endcsname}
```

won't do the trick, so we need to resort to a strategy used before (compare pages 218 and 223),

```
\expandafter\def\csname HL@0\endcsname#1\endHL{%-
 \flush@
 \global\part@true
 \ifodd\pageno\else\advancepageno\fi
 {\locallabel@
 \global\setbox1=\vbox{\Let@
 \baselineskip=21pt
 \halign{\BF\ignorespaces##\unskip\hfil\cr
```

```

\test@false
\expandafter\ifx\csname HL@W0\endcsname\empty
\ifx\thelabel@@\empty\global\let\Next@=T\fi\fi
\ifx\Next@ T%
\else
\expandafter\ifx\csname HL@W0\endcsname\empty
\else
\def\next@{\let\nextii@=}
\expandafter\next@\csname HL@W0\endcsname
\uppercase\expandafter{\nextii@}\space\space\fi
{\HL@@F\thelabel@@}%
\fi
\cr
\noalign{\ifx\Next@ T\vskip-\baselineskip\else\vskip30pt\fi}%
\uppercase{\ignorespaces#1}\crr}}}%
\def\aftertoc@{\vfill\break\advancepageno\global\part@false}%
\hrule\height0pt\mark{}\vskip1.25in\unvbox 1
}

```

The `\hrule` keeps the `\vskip1.25in` from disappearing at the top of the page, and the `\mark{}` clears any marks from previous `\section's` (see below). Notice that `\aftertoc@` causes the page number to jump to the next odd number after the `\part` page, but the proper page for the `\part` will appear in the `.toc` file (compare page 213); `\advancepageno` requires less work from TeX than `\Offset\Page2`, which could also have been used.

We needed the globally defined `\Next@` here, since each line of the `\halign` is already within a group; although `\Next@` is used in various other \LaTeX routines, none of them will be involved when we typeset (the `\uppercase'd`) `\HL@W` (for greater security, we could always use some new scratch token instead).

Next we have

```

\notocwrite@\NameHL1\chapter
\newword\chapter{Chapter}
\newif\iffirstchapterpage@

```

We need the new flag `\iffirstchapterpage@`, because initial pages of chapters will be treated specially, with no running heads, and page numbers at the bottom. As with the paper style, we also have the flag

```
\newif\ifinsection@
```

Even-numbered pages will normally have the `\chapter` title at the top. However, we will allow

```
\runningchapter{...}
```

to be used before `\chapter` to specify a different running chapter (this change from version 1, where `\runningchapter` was supposed to occur after the `\chapter`, makes things much easier). We need a new flag to alert `\chapter` to the fact that `\runningchapter` has been used; the argument of `\runningchapter` will be stored in a token list, for reasons that will soon become apparent:

```
\newif\ifrunningchapter@
\newtoks\runningchapterertoks@
\def\runningchapter#1{\global\runningchapter@true
\runningchapterertoks@={#1}}
```

The running heads at the top of odd-numbered pages will normally contain the most recent `\section` title, an arrangement that will require the use of `\mark's`, except that we will use the `\chapter` title if no `\section` has appeared. `\chapter` is going to start a new page, issue an empty `\mark{}`, to clear `\mark's` from the previous `\section's`, and then define `\thechapter@` to be the desired running head on even-numbered pages.

Just in case there are pages before a `\chapter`, we begin with

```
\let\thechapter@=\relax
```

We then define `\chapter`, via `'\HL@1'`, by

```

\expandafter\def\csname HL@1\endcsname#1\endHL{%
  \flush@
  \global\insection@false
  \ifrunningchapter@\else\runningchapter@={#1}\fi
  \global\runningchapter@false
  {\noexpands@
   \xdef\thechapter@{\ifx\Thepref@\empty\else
    \Thepref@\null. \fi\the\runningchapter@}}%
  \global\firstchapterpage@true
  {\locallabel@
   \global\setbox1=\vbox{\Let@\tabskip\hss@
   \halign to\hsize{\bf\hfil\ignorespaces##\unskip\hfil\cr
   \expandafter\ifx\csname HL@W1\endcsname\empty\else
    \csname HL@W1\endcsname\space\fi
   {\HL@F\ifx\thelabel@\empty\else\thelabel@\space\fi}%
   \ignorespaces#1\crr}}%
  }%
  \unvbox1 \mark{}}%
  \nobreak\vskip\baselineskip
  %\firstparflush@
}

```

Note that:

(1) As with ‘\HL@0,’ we need to flush out any figures from the previous \chapter.

(2) The tokens #1 are stored in the token list \runningchapter@, unless \runningchapter has already been used, and has thus already stored something in \runningchapter@. After this decision for \runningchapter@ has been made, we can reset the flag \ifrunningchapter@ to be false.


(3) Then \thechapter@ is created by an \xdef, inside a group containing \noexpands@, so that appropriate control sequences in \Thepref@ are left alone. Even though we have an \xdef, our chapter title (or abbreviated title, specified by \runningchapter) appears as originally typed, since it is the value of \the\runningchapter@—thus, we do not have to worry about expansion of control sequences in \thechapter@, and do not have to resort

to the `\unmacro@` meaning trick (this works because `\thechapter@` is not going to be written to a file, but merely typeset at the appropriate time).

(4) At the very end of the definition the line

```
\firstparflush@
```

is designed to start the next paragraph unindented; it has been commented out, since it is not used in the book style, but other book styles might want to use it.

 The definition of `\firstparflush@`, for those styles that want it, is given immediately afterwards:

```
\def\firstparflush@{\parindent=0pt
\everypar{\global\parindent=10pt \global\everypar{}}}
```

Thus, the first paragraph will be set with `\parindent` equal to 0pt, but this paragraph will set `\everypar`, for all successive paragraphs, to

```
\global\parindent=10pt \global\everypar{}
```

This means that `\parindent` will be 10pt for the second paragraph, and, moreover, `\everypar` will be back to the empty list for the third paragraph. The `\global`'s are needed because a paragraph might start with something like

```
{\it Consequently}, we see that ...
```

Having defined `\part` and `\chapter`, which set special flags `\ifpart@` and `\iffirstchapterpage@`, we also define `\footline`, which uses them, printing nothing at the bottom of `\part` pages, but a centered page number, in 9 point type at the bottom of the first page of each chapter. In essence we want

```
\ninepoint\folio
```

except that there is no `\ninepoint` command, merely `\ninerm` to switch to the 9 point roman font. So we have to use

```

\footline={\ifpart@\hfil
\else
\iffirstchapterpage@\hfil
\inerm\page@S{\page@P\page@N{\number\page@C}\page@Q}\hfil
\fi
\global\firstchapterpage@false
\fi}

```

—thus, `\newfontstyle\page` won't effect the page numbers at the bottom of initial chapter pages.

At this point we also want to prevent any `\A`place'd material from occurring on the first page of each `\chapter`, by modifying `\advancedimtopins@`, as indicated on page 402:

```

\def\advancedimtopins@{%
\iffirstchapterpage@
\else
\advance\dimen@ by \dimen\topins
\global\dimen\topins=\dimen@
\fi}

```

41.6. `\plainoutput`. The `\part` and `\chapter` heading levels are the only ones that flush out figures, so now is the time to consider the necessary modifications to `\plainoutput`, as already indicated in section 4.

We are going to need a new flag and a new box,

```

\newif\ifblankpage@
\newbox\topinsbox@

```

and also a counter in which we store the special penalty

```
-1073741824 (= -'10000000000)
```

that T_EX inserts when it sees `\end` (page 462):

```
\newcount\endpenalty@
\endpenalty@=-'10000000000
```

Most of our extra manipulations will be done only when `\ifflush@` is true. However, we will also be doing extra manipulations when `\ifSflush@` is true and `\insertpenalties` is 0 (which is when the additional

```
\null\kern-\topskip\nobreak\vfill
```

will create a blank page). So we first use

```
\test@false
\ifflush@\test@true
\else\ifSflush@\global\Sflush@false
\ifnum\insertpenalties=0 \test@true\fi
\fi\fi
```

to set `\iftest@` true in either of these cases, and reset `\ifSflush@` to false.

Then, if this test is passed, we first test if `\box\footins` is void and also `\box\topins` is void or has height 0pt; and if that test is true, we then use the code on page 464 to test whether `box255` is essentially blank, reducing the page number by 1, and setting `\ifblankpage@` true, if it is:

```
\def\plainoutput{%
\test@false
\ifflush@\test@true
\else\ifSflush@\global\Sflush@false
\ifnum\insertpenalties=0 \test@true\fi
\fi\fi
\iftest@
\test@false
\ifvoid\footins
\ifvoid\topins\test@true
\else\ifdim\ht\topins=0pt\test@true
\fi\fi
```

```

\fi
\iftest@
\setbox0=\vbox{\unvcopy255 \unskip\unpenalty\unkern
\global\setbox1=\lastbox\unskip}%
\ifdim\ht0=0pt \ifdim\ht1=0pt
\global\advance\pageno by -1
\global\blankpage@true
\fi\fi
\fi
\fi

```

Then we could continue with the commands from the standard `\plainoutput` routine:

```

\specialsplit@false\ifvoid\topins\else\ifdim\ht\topins=0pt
\specialsplit@true\advance\minpagesize by -\skip\topins\fi\fi
\fliptopins@
\setbox\outbox@=\vbox{\makeheadline\pagebody\makefootline}%
{\noexpands@ \let\style=\relax
\shipout@\box\outbox@
}%
\advancepageno
\resetdimtopins@
\ifvoid 255 \else \unvbox 255 \penalty\outputpenalty\fi
\ifnum\outputpenalty>-20000 \else\dosupereject\fi}

```

That would insure that the page numbers don't increase for the spurious blank pages. But we can do better than that. If we replace the

```
\shipout@\box\outbox@
```

with

```

\ifblankpage@ \global\blankpage@false
\else\shipout@\box\outbox@\fi

```

then the spurious blank pages shouldn't even be printed.

This seems to work fine at the end of all \chapter's except when there are figures to be flushed out at the end of the last \chapter, before the \bye or \enddocument. In that case, after each round of the \output routine, which now increases \deadcycles by 1, since nothing is shipped out, T_EX keeps adding

```
\line{} \vfill \penalty -'10000000000
```

so that we eventually get an error message like

```
! Output loop---25 consecutive dead cycles.
```

Page 264 of *The T_EXbook* says “When T_EX sees an \end command, it terminates the job only if the main vertical list has been entirely output and if \deadcycles=0. Otherwise it inserts the equivalent of

```
\line{} \vfill \penalty-'10000000000
```

into the main vertical list, and prepares to read the '\end' token again.” Apparently, the main vertical list has not been “entirely output” because \box\topins hasn't been shipped out. Emptying out \box\topins with something like

```
\setbox0=\box\topins
```

doesn't seem to help, so I resorted to

```
\ifblankpage@
  \ifnum\outputpenalty=\endpenalty@
    \shipout\vbox{\hrule\width1pt\height0pt\box\topins}%
  \fi
  \global\blankpage@false
\else
  \shipout@\box\outbox@
\fi
```

which seems to work—the only adverse side-effect is the possibility of extra, totally blank, pages at the end.

There is also one other modification that we need: instead of always using

```
\setbox\outbox@=\vbox{\makeheadline\pagebody\makefootline}
```

we use

```
\ifblankpage@ \setbox\outbox@=\pagebody\else
\setbox\outbox@=\vbox{\makeheadline\pagebody\makefootline}\fi
```

This takes care of spurious blank pages that might be created by the first `\part` or `\chapter`, since `\makeheadline` or `\makefootline` might involve quantities that haven't even been determined yet.

So the whole definition is:

```
\def\plainoutput{%
\test@false
\ifflush@\test@true
\else\ifSflush@\global\Sflush@false
\ifnum\insertpenalties=0 \test@true\fi
\fi\fi
\iftest@
\test@false
\ifvoid\footins
\ifvoid\topins\test@true
\else\ifdim\ht\topins=0pt \test@true
\fi\fi
\fi
\iftest@
\setbox0=\vbox{\unvcopy255 \unskip\unpenalty\unkern
\global\setbox1=\lastbox\unskip}%
\ifdim\ht0=0pt \ifdim\ht1=0pt
\global\advance\pageno by -1
\global\blankpage@true
\fi\fi
\fi
\fi
```

```

\specialsplit@false
\ifvoid\topins\else
  \ifdim\ht\topins=0pt
    \specialsplit@true
    \advance\minpagesize by -\skip\topins
  \fi\fi
\fliptopins@
\ifblankpage@
  \setbox\outbox@=\pagebody\else
  \setbox\outbox@=\vbox{\makeheadline
  \pagebody\makefootline}%
\fi
{\noexpands@ \let\style=\relax
\ifblankpage@
  \ifnum\outputpenalty=\endpenalty@
  \shipout\vbox{\hrule\width1pt\height0pt
  \box\topins}%
\fi
\global\blankpage@false
\else
  \shipout@\box\outbox@
\fi}%
\advancepageno
\resetdimtopins@
\ifvoid 255 \else \unvbox 255 \penalty\outputpenalty\fi
\ifnum\outputpenalty>-20000 \else \dosupereject \fi}

```

bd For the clause

```

\ifblankpage@
  \setbox\outbox@=\pagebody\else
  \setbox\outbox@=\vbox{\makeheadline
  \pagebody\makefootline}
\fi

```

I originally had simply

```
\ifblankpage@else
  \setbox\outbox@=\vbox{\makeheadline
    \pagebody\makefootline}
\fi
```

But this led to the dead cycles error message, though I don't understand why.

⚡ Since the extra test in `\plainoutput` always regards a “blank” page as a mistake, actual blank pages can't be produced with something like

```
\newpage\null\vfill\break or \pagebreak\null\vfill\break
```

but we can use something like

```
\newpage\blankpage
```

where we have defined

```
\def\blankpage{\null\null\vfill\break}
```

just in case that's needed. [Of course

```
\Aplace{\Figure\Hbyw{...}\endFigure}
```

possibly with a `\caption`, would be used if a blank page were required for a picture, the most usual reason for such a request. Also,

```
\NoFlushedFigs
\newpage\null\vfill\break
```

would work.]

41.7. Other heading levels. `\appendices` is treated exactly as in the paper style, so we won't repeat the definition here.

Next we have

```
\notocwrite@Namehl1\section
\newstyle\section#1{#1\ull.}
```

as in the paper style.

The running heads at the top of odd-numbered pages will normally contain the most recent `\section` title, except that we will use the `\chapter` title if no `\section` has yet appeared. We also want to allow `\runningsection` to be used before `\section` (again a change from version 1), which is treated just like `\runningchapter`:

```
\newif\ifrunningsection@
\newtoks\runningsectionontoks@
\def\runningsection#1{\global\runningsection@true
\runningsectionontoks@{#1}}
```

In the definition of `\section`, via `'\hl@1'`, instead of defining `\thechapter@`, we need to make an appropriate `\mark`; since `\mark`'s are expanded, this requires a little more care, using now familiar tricks:

```
{\noexpands@
\edef\next@{\toks@={\ifx\Thepref@empty\else
\Thepref@\null. \fi
\the\runningsectionontoks@}}%
\next@
\mark{\the\toks@}}
```

Here the `\edef\next@` makes `\next@` mean

```
\toks@=(\section number). (\section title)
```

so that `\next@` actually assigns `\toks@` that value, and then

```
\mark{\the\toks@}
```

produces the `\mark`, with no further expansions. The whole definition is

```

\expandafter\def\csname hl@1\endcsname#1{%
  \global\insection>true
  \medbreak
  \ifrunningsection@\else\runningsectiontoks@{#1}\fi
  \global\runningsection>false
  \noindent@
  {\noexpands@
   \edef\next@{\toks@{\ifx\Thepref@\empty\else
    \Thepref@\null. \fi
   \the\runningsectiontoks@}}\next@
  \mark{\the\toks@}}%
  {\locallabel@
   {\hl@F\ifx\thelabel@\empty\else\thelabel@\space\fi}\bf
  \ignorespaces#1\unskip\punct@{\null.}\addspace@\enspace}%
  \par\nobreak\smallskip}

```

`\section` is the only heading level that will produce `\mark's`, except for `\chapter`, which emits `\mark{}`; so we should now define `\headline`. On odd-numbered pages this will usually involve `\botmark`, unless `\botmark` is empty, in which case we will use `\thechapter@`. However, `\botmark` being “empty” cannot be checked using the test `\ifx\botmark\empty`, which is never true, since `\botmark` is a primitive that is not equivalent to `\empty`. So we first define a routine

```

\def\BotOrChap@#1\BotOrChap@{%
  \def\next@{#1}\ifx\next@\empty\thechapter@
  \else\ignorespaces#1\unskip\fi}

```

which will be used after an `\expandafter` in the definition

```

\headline={\unlabel@ \noset@
  \def\{\unskip\space\ignorespaces}%
  \ifpart@\hfil
  \else
  \iffirstchapterpage@\hfil

```

```

\else
\ifodd\pageno
\hfil
\smc\expandafter\BotOrChap@\botmark\BotOrChap@
\hfil
\llap{\tenpoint\folio}%
\else
\rlap{\tenpoint\folio}%
\hfil\smc\thechapter@
\hfil
\fi
\fi\fi}

```

The `\unlabel` and `\noset` are used in case some `\Reset` or `\Offset` or some `\label` or `\pagelabel` appeared within a `\chapter` or `\section` title. The definition of `\\` is provided for an unedited `\\` in a `\chapter` title.

`\subsection`, `\topic` and `\subtopic` are handled just as in the paper style, so we won't repeat the definitions here.

The initializations for `\chapter` are a little more complicated than in the paper style:

```

\expandafter\def\csname HL@I1\endcsname{\Reset\hl11%
\Reset\tag1\Reset\claim1\Reset\Figure1\Reset\Table1%
\ifx\pref\empty\newpre\section{}\def\tag@P{}}%
\def\claim@P{}\def\island@P{}%
\else
\newpre\section{\pref.}\edef\tag@P{\pref.%
\edef\claim@P{\pref.}\edef\island@P{\pref.%
\fi
}

```

As before, we reset the `\section` counter to 1 (there is no `\keepsection` command for this style). Moreover, we also want the numbering for displayed formulas, `\claim's`, `\Figure's` and `\Table's` to be reset to 1. If `\pref` is `\empty` (only because the user has typed something like `\chapter""`), then the "pre" material for `\section`, `\tag`, `\claim` and all types of `\island`

numbers should be empty; otherwise it should be the `\chapter` number followed by a period. Notice that we explicitly use `\edef\tag@P{\pref.}`, ..., rather than `\newpre`, since that takes less TeX processing, but we definitely want `\newpre\section`, since that takes care of `\newpre\h11` automatically.

The initializations for other heading levels are the same as in the paper style.

41.8. Footnotes. `\vfootnote@` is redefined exactly as in the paper style, to get footnotes in 8 point type.

41.9. Bibliography. Although `\makebib` isn't allowed in the book style, `\UseBibTeX` is allowed, causing an `.aux` file to be written, on which BIBTeX can operate, thereby creating a `.bbl` file that can be used by the "back matter file". The definition from `lamstex.tex` has to be changed so that `\bibliography{...}` only writes `\bibdata{...}` to the `.aux` file, without trying to read the `.bbl` file. Also, `\bibliography@W` shouldn't be defined, since

```
\newword\bibliography
```

shouldn't be allowed in the main file, only in the back matter file.

```
\def\UseBibTeX{%
  \immediate\openout\auxwrite@=\jobname.aux
  \let\cite=\BTcite@
  \def\nocite##1{\immediate\write\auxwrite@
    {\string\citation{##1}}}%
  \def\bibliographystyle##1{\immediate\write\auxwrite@
    {\string\bibstyle{##1}}}%
  \def\bibliography##1{%
    \immediate\write\auxwrite@{\string\bibdata{##1}}}
```

41.10. book.stf. The front matter style file `book.stf` does not require a mechanism to prevent it from being read in twice. Aside for this, it begins with the same basic settings as `book.st` itself, including the redefinition of `\makeheadline` and `\makefootline`. It also specifies

```
\newnumstyle\page\roman
```

and then loads the same fonts as book.st, and defines `\tenpoint` and `\eightpoint` as in that file.

Two new fonts

```
\font\Bf=cmbx10 scaled \magstep1
\font\BF=cmbx10 scaled \magstep3
```

are also loaded; `\BF` is used for the 'CONTENTS' headings, etc., while `\Bf` is used for the Part entries in the table of contents (Chapter entries will be printed in ordinary `\bf` and other heading level entries in `\rm`).

Pages with `\BF` headings are handled specially, with no running heads, and page numbers at the bottom, and we have the flag

```
\newif\ifspecialpage@
```

to indicate such pages.

The material for running heads will be stored in `\headline@`, initialized by

```
\let\headline@=\relax
```

The `\makepiece` command is supplied for special pieces like a Preface:

```
\def\makepiece#1{\par\vfill\break
\global\specialpage@true
\gdef\headline@{\ignorespaces#1\unskip}%
\centerline{\BF\uppercase{\ignorespaces#1\unskip}}%
\vskip30pt plus10pt minus 10pt}
```

The argument #1 is stored in `\headline@`, which will be used in the running heads of this piece, except for the first page. Here we assume that the heading fits on a single line. A variant

```
\makepiece#1\endmakepiece
```

command, allowing `\\` to indicate line breaks, could be specified in an obvious way. If this were allowed, then there should probably be a

```
\runningpiece
```

command to specify a shortened form for the running heads. This would work like the `\runningchapter` command.

As with `paper.st`, for `\maketoc` we now how to include many of the definitions from `lamstex.stf`:

```
\def\dotleaders ...
\def\Page@ ...
\long\def\widerthanhsz@ ...
\long\def\setentry@ ...
\def\endstrut@ ...
\newdimen\thehang@
\long\def\longentry@ ...
\newif\ifemptynumber@
\def\Style@ ... . . . \def\Style@@@ ...
```

`\digits` isn't used, because the `\hl` entries are set differently.

Because the `\Style@ . . . \Style@@@` complex involves `\style`, which will be set to `\HL@@S` or `\hl@@S`, we need to provide definitions for all cases that will occur:

```
\expandafter\def\csname HL@S0\endcsname#1{#1\}/}
\expandafter\def\csname HL@S1\endcsname#1{#1\null.}
\expandafter\def\csname hl@S1\endcsname#1{#1\null.}
\expandafter\def\csname hl@S2\endcsname#1{#1\null.}
\expandafter\def\csname hl@S3\endcsname#1{#1\}/}
\expandafter\def\csname hl@S4\endcsname#1{#1\}/}
```

As in `paper.st`, we

```
\def\maketoc@W{Contents}
```

We will also need a new flag

```
\newif\ifbib@
```

for one part of the contents.

The definition of `\maketoc` is something of a cross between the definitions in `lamstex.stf` (page 484) and `paper.st` (section 40.8), adding extra information needed for running heads and `\footline`:

```
\def\maketoc{\checkmainfile@\par\vfill\break
\begingroup
\unlabel@ . . . \lineskiplimit=0pt
\def\HL ... \def\hl ...
\def\NameHL ... \def\Namehl ...
\NameHLO\part . . . \Namehl4\subtopic
<extra material to set any
\makebib and \makeindex entries>
\global\specialpage@true
\gdef\headline@{\maketoc@W}%
\centerline{\BF\uppercase\expandafter\maketoc@W}}%
\vskip30pt plus10pt minus10pt
\input\mainfile@.toc
\endgroup}
```

The `<extra material>` starts with

```
\def\makebib##1\Page##2##3##4##5{\bigbreak\bigskip\bigskip
\setentry@{\Bf##1}{-}{\dotleaders}%
{\Page@{##2}{##3}{##4}{##5}}%
\bib@true}%
```

This definition is added for the situation where a `\makebib` in the “back matter file” (see the next section) creates an entry in the `.toc` file for the back matter file, which is then transferred to the `.toc` file for the main file.

Then we

```
\let\bibliography=\makebib
```

in case `\bibliography` appears instead of `\makebib`; this is for the situation where `\UseBibTeX` appeared in the main file, so that a `.bbl` file was created, and the corresponding entry in the `.toc` file for the back matter file was transferred to the `.toc` file for the main file.

Similarly, we add

```
\def\makeindex##1\Page##2##3##4##5{\ifbib@\bigskip\else
\bigbreak\bigskip\bigskip\fi
\setentry@{\Bf##1}{-}{\dotleaders}%
{\Page@{##2}{##3}{##4}{##5}}}%
```

in case a `\makeindex` in the `.toc` file for the back matter file is transferred to the `.toc` file for the main file. Note that the spacing before this entry depends on whether or not a `\makebib` or `\bibliography` has already appeared.

The definition of `\HL` in the above definition has an extra clause for `\part's`:

```
\ifnum\HLlevel@=0
\bigbreak\bigskip
\begingroup
\def\{\unskip\space\ignorespaces}%
\setentry@{\Bf\def\next@{##2}\ifx\next@\empty\else
\uppercase{##2} \fi
\let\style=\HL@S\hbox to35pt{\Style@##3\Style@\hss}\ifemptynumber@
\nobreak\hskip-35pt\fi
\uppercase{\ignorespaces##4\unskip}}{\hfil{}}%
\endgroup
\nobreak\smallskip
\else
\ifnum\HLlevel@=1
. . .
\else
\Err@ ...
\fi}
```

So the first argument of `\setentry@` contains the

PART I PART TITLE

with the Part title, in `\uppercase`, starting 35 points from the word 'PART' (but if there is no part number, we `\hskip` back 35 points). We use `\uppercase{#2}`, since, as mentioned on page 562, the .toc file has '`\HL@WO`', which is 'Part', rather than 'PART', allowing other front matter style files to handle things differently. The second argument (which normally would be the part title) is empty; the third argument, normally `\dotleaders` is `\hfil`, and the fourth argument, normally `\Page...` is also empty, so that 'Part' entries are printed without dot leaders or page numbers.

Similarly, the definition of `\makelist` in `book.stf` is almost the same as that in `lamstex.stf`, except that we add

```
\global\spcialpage@true
\gdef\headline@{\ignorespace#2\unskip}
```

and use

```
\BF\baselineskip=22pt
```

for setting the heading instead of `\bf`.

Finally, `\footline` and `\headline` are reset as in `book.st`, except using the flag `\ifspcialpage@` rather than the flags `\ifpart@` and `\iffirstchapterpage@`.

41.11. book.stb. The back matter style file `book.stb` also begins with the same basic settings as `book.st`, except that redefinition of `\makeheadline` and `\makefootline` is deferred; and then it loads the same fonts as before.

Then we have numerous definitions from `lamstex.stb`:

```
\def\adjustpunct@ ...
\def\ignorepars@ ...
\def\ignorepars@@ ...
\newcount\ctype@
\newcount\Ctype@
\newif\ifleftcolbreak@
\def\cbreak@ ...
\newif\ifletter@
\newtoks\marktoks@i . . . \newtoks\marktoks@v
```

```

\newif\ifentry@
\newcount\dii@
\def\Topage@ ...

```

as well as

```

\newdimen\pageheight@
\pageheight@=\vsize
\newdimen\doublepageheight@
\doublepageheight@=2\pageheight@
\advance\doublepageheight@ by 1pc
\newdimen\pagewidth@
\pagewidth@=\hsize

```

and then we redefine `\makeheadline` and `\makefootline` in terms of `\pagewidth@`:

```

\def\makeheadline{%
  \hbox{\botsmash{\hbox to\pagewidth@
    {\vbox to8.5pt{\the\headline}}}}%
  \nointerlineskip\vskip26pt}
\def\makefootline{\baselineskip=2.5pc\relax
  \hbox to \pagewidth@{\the\footline}}

```

As in `lamstex.stb`, we will need a new flag for the first page of the index,

```

\newif\iffirstindexpage@

```

Recall that in `lamstex.stb` it is the definition of `\combinecolumns@` that actually prints the heading at the top of the first page. The same is true in `book.stb`, so we first

```

\def\makeindex@W{Index}

```

and declare the font

```
\font\Bf=cmbx10 scaled \magstep1
```

Then the definition of `\combinecolumns@` from `lamstex.stb` is modified as follows:

```
\def\combinecolumns@{%
  \setbox\outbox@=\vbox{\makeheadline
  \vbox to\pageheight@{\boxmaxdepth=\maxdepth
  \iffirstindexpage@
  \vbox to30pt{%
    \hbox to\pagewidth@{\hfil\Bf
    \uppercase\expandafter{\makeindex@W}\hfil}}%
    \vfil}}%
  \nointerlineskip
  \fi
  \wd0=\hsize \wd2=\hsize
  \setbox0=\hbox to\pagewidth@{\box0 \kern1pc \box2}}%
  \dimen@=\dp0 \box0 \kern-\dimen@\vfill}}%
  \makefootline}}%
{\noexpands@\let\style=\relax
 \shipout@\box\outbox@
}%
\global\vsiz=\doublepageheight@
\global\firstindexpage@false
\advancepageno}
```

Then we continue with other material from `lamstex.stb`:

```
\newdimen\prevcoldepth@
\def\doublecolumns@ ...
\def\continue@ ...
\newif\ifshortlastcolumn@
\def\balancecolumns@ ...
```

With these definitions—used mainly for the index—out of the way, we first start on the bibliography.

As with `book.stf`, we need

```
\newif\ifspecialpage@
\let\headline@=\relax
```

The definition of `\makebib` is similar to that in `paper.st`, except that we also write to the `.toc` file (compare page 204):

```
\def\makebib{\par\vfill\break
\global\specialpage@true
\gdef\headline@{\makebib@W}%
\begingroup
\eightpoint
\sfcode'\.=1000 \everypar{}\parindent=0pt
\def\nopunct{\nopunct@true} \def\nospace{\nospace@true}%
\nopunct@false\nospace@false
\def\lkerns@{\null\kern-1sp\kern1sp}%
\def\nkerns@{\null\kern-2sp\kern2sp}%
\hbox to\pagewidth@
{\hfil\Bf\uppercase\expandafter{\makebib@W}\hfil}%
\iftoc@
\expandafter\unmacro@\meaning\makebib@W\unmacro@
{\noexpands@
\edef\next@{\write\toc@{\noexpand\noexpand
\noexpand\makebib{\macdef@}}}\next@}%
\write\toc@{\noexpand\Page
{\number\pageno}{\page@N}{\page@P}{\page@Q}^^J}%
\fi
\nobreak\bigskip}
```

If `\UseBibTeX` was specified in the main file, and a `.bbl` file was created, then instead of a `\makebib... \endmakebib` region, the user can simply type

```
\bibliography
```

(but with no argument).

We will need to define

```
\def\bibliography@W{Bibliography}
```

so that

```
\newword\bibliography
```

can be used prior to this (in lamstex.tex this definition is made by \UseBibTeX, which won't appear in the back matter file).

The redefinition of \bibliography, with no argument, will begin with \checkmainfile@, since \mainfile@.bbl is the file we will want to \input:

```
\def\bibliography{\checkmainfile@
\immediate\openin\bbl@=\mainfile@.bbl
\ifeof\bbl@
\W@{No .bbl file}%
\else
\immediate\closein\bbl@
\begin{group}
\input bibtex
\input\mainfile@.bbl
\end{group}
\fi}%
```

Recall that bibtex.tex defines \begin{thebibliography}{...} in terms of \begin{thebibliography}@{...}, which now has to be redefined to give the same sort of information and formatting as \makebib for this file:

```
\def\begin{thebibliography}@#1{\par\vfill\break
\global\specialpage@true
\gdef\headline@{\makebib@W}%
\eightpoint
\setbox0=\hbox{#1\ }\bibindent@=\wd0
\sfcode'\.=1000 \everypar{\parindent=0pt
\hbox to\pagewidth@
{\hfil\Bf\uppercase\expandafter{\bibliography@W}\hfil}}%
```

```

\iftoc@
\expandafter\unmacro@\meaning\makebib@W\unmacro@
{\noexpands@
\edef\next@{\write\toc@{\noexpand\noexpand
\noexpand\bibliography{\macdef@}}\next@}%
\write\toc@{\noexpand\Page
{\number\pageno}{\page@N}{\page@P}{\page@Q}^^J}%
\fi
\nobreak\bigskip}

```

The definition of `\makeindex` is virtually the same as in `lamstex.stb`, except that we again write to the `.toc` file, and set up information for the heading levels and foot line,

```

\def\makeindex{\checkmainfile@\par\vfill\break
\iftoc@
\expandafter\unmacro@\meaning\makeindex@W\unmacro@
{\noexpands@
\edef\next@{\write\toc@{\noexpand\noexpand
\noexpand\makeindex{\macdef@}}\next@}%
\write\toc@{\noexpand\Page
{\number\pageno}{\page@N}{\page@P}{\page@Q}^^J}%
\fi
\global\specialpage@true
\global\firstindexpage@true
\gdef\headline@{\makeindex@W}%
\begingroup
(material from lamstex.stb definition,
starting with \let\asterisk=*, and
ending with \def\shortlastcolumn ... )
\hsize=14pc
\global\vsizer=\doublepageheight@
\maxdepth=\maxdimen
\global\firstindexpage@true
\global\advance\vsizer by -60pt
\everypar{\parindent=0pt

```

```
\eightpoint
\rightskip=0pt plus3em
\spaceskip=.3333em \xspaceskip=.5em
\output={\doublecolumns@}%
\input\mainfile@.xdx
\mark{}%
\output={\balancecolumns@}\vfil\break\endgroup
\global\vsizе=\pageheight@
```

And then we reset `\headline` and `\footline`, as in `book.stf`.

Chapter 42. The letter style

The L^AS-T_EX letter .st is based on the format for business letters discussed at the beginning of Appendix E of *The T_EXbook*, with numerous added features, and some change of notation.

The letter format in *The T_EXbook* has the unfortunate feature that plain T_EX's `\item` command doesn't work well, since it indents by `\parindent`, which is `0pt` in this format. Fortunately, that problem doesn't occur with L^AS-T_EX's `\list \item` command. We do change `\listbi@` to

```
\def\listbi@{\penalty50 \bigskip}
```

(the default value has a `\medskip` instead), since this seems to look better when `\parindent` is `0pt`.

Although there are several interesting features in letter .st, most of them have almost nothing to do with L^AS-T_EX in general, so we won't discuss them here.