ledmac

A brave but perhaps doomed attempt to port EDMAC to LaTeX*

Peter Wilson†

based on the original work by

John Lavagnino and Dominik Wujastyk

March 2, 2021

Contents

*This file has version number v0.1, last revised 2003/04/01.

†Catholic University of America (Now at peter.r.wilson@boeing.com)

1 Introduction

The EDMAC macros for typesetting critical editions of texts have been available for use with TeX for some years. Since EDMAC was introduced there has been a small but constant demand for a version of EDMAC that could be used with LaTeX. The ledmac package is an attempt to satisfy that request.

The package is very much in an experimental state, and may for ever remain so, although hopefully not.

ledmac would not have been possible without the amazing work by John Lavagnino and Dominik Wujastyk, the original authors of EDMAC. The majority of both the code and this manual are by these two.

I have altered their code and documentation as little as possible. In order to more easily show the debt that I owe, my few contributions are in the font you are now reading. The original work is in the normal roman font.

There are places where I have not supplied some of the original EDMAC facilities, either because they are natively provided by LaTeX (such as font handling), or are available from other LaTeX packages (such as crop marks).

1.1 Overview

The EDMAC macros, together with TEX, provide several important facilities for formatting critical editions of texts in a traditional manner. Major features include:

- automatic stepped line numbering, by page or by chapter;
- sub-lineation within the main series of line numbers;
- variant readings automatically keyed to line numbers;
- multiple series of footnotes and endnotes;
- block or columnar formatting of footnotes.

EDMAC allows the scholar engaged in preparing a critical edition to focus attention wholly on the task of creating the critical text and evaluating the variant readings, text-critical notes and testimonia. TEX and EDMAC will take care of the formatting and visual correlation of all the disparate types of information.

EDMAC works together with the PLAIN TEX format, and with the exception of footnote-related commands, virtually all plain TEX commands are available for use in the normal way. Other languages and fonts (Sanskrit, Greek, Russian, etc.),[1] can be incorporated.

While EDMAC can be used "out of the box", with little or no customization, you may also go to the other extreme and view it as a collection of tools. Critical editions are amongst the most idiosyncratic of books (like their authors), so we have made EDMAC deliberately bland in some ways, while also trying to document it reasonably well so that you can find out how to make it do what you want.

This documentation assumes the "manual" use of EDMAC. But EDMAC is also successfully being used (with TEX, of course) as the formatting engine or "back

[1] See, e.g., *TUGboat* **9** (1988), pp. 131–151.

end" for the output of an automatic manuscript collation program. COLLATE runs on the Apple Macintosh, can collate simultaneously up to a hundred manuscripts of any length, and provides facilities for the scholar to tailor the collation interactively.[2]

This manual contains a general description of how to use the LaTeX version of EDMAC, namely ledmac, (in section 2); the complete source code for the package, with extensive documentation (in sections 3 through 12); a series of examples (in Appendix A); and an Index to the source code. We do not suggest that you need to read the source code for this package in order to use it; we provide this code primarily for reference, and many of our comments on it repeat material that is also found in section 2. But no documentation, however thorough, can cover every question that comes up, and many can be answered quickly by consultation of the code. On a first reading, then, you should skip from the general documentation of section 2 to the examples in Appendix A, unless you are particularly interested in the innards of EDMAC.

1.2 History

The original version of EDMAC was TEXTED.TEX, written by John Lavagnino in late 1987 and early 1988 for formatting critical editions of English plays.

John passed these macros on to Dominik Wujastyk who, in September–October 1988, added the footnote paragraphing mechanism, margin swapping and other changes to suit his own purposes, making the style more like that traditionally used for classical texts in Latin and Greek (e.g., the Oxford Classical Texts series). He also wrote some extra documentation and sent the files out to several people. This version of the macros was the first to be called EDMAC.

The present version was developed in the summer of 1990, with the intent of adding necessary features, streamlining and documenting the code, and further generalizing it to make it easily adaptable to the needs of editors in different disciplines. John did most of the general reworking and documentation, with the financial assistance of the Division of the Humanities and Social Sciences, California Institute of Technology. Dominik adapted the code to the conventions of Frank Mittelbach's doc option, and added some documentation, multiple-column footnotes, cross-references, and crop marks.[3] A description by John and Dominik of this version of EDMAC was published as "An overview of EDMAC: a PLAIN TEX format for critical editions", *TUGboat* **11** (1990), pp. 623–643.

From 1991 through 1994, the macros continued to evolve, and were tested at a number of sites. We are very grateful to all the members of the (now defunct) edmac@mailbase.ac.uk discussion group who helped us with smoothing out bugs and infelicities in the macros. Ron Whitney and our anonymous reviewer at the TUG were both of great help in ironing out last-minute wrinkles, while Ron made

[2] Contact "COLLATE"'s author, Peter Robinson, at The Computers and Manuscripts Project, Oxford University Computing Service, 13 Banbury Road, Oxford OX2 6NN, England. Internet: "peterr@vax.oxford.ac.uk".

[3] This version of the macros was used to format the Sanskrit text in volume I of *Metarules of Pāṇinian Grammar* by Dominik Wujastyk (Groningen: Forsten, 1993).

some important suggestions which may help to make future versions of EDMAC even more efficient. Wayne Sullivan, in particular, provided several important fixes and contributions, including adapting the Mittelbach/Schöpf "New Font Selection Scheme" for use with PLAIN TEX and EDMAC. Another project Wayne has worked on is a DVI post-processor which works with an EDMAC that has been slightly modified to output \specials. This combination enables you to recover to some extent the text of each line, as ASCII code, facilitating the creation of concordances, an *index verborum*, etc.

At the time of writing (1994), we are pleased to be able to say that EDMAC is being used for real-life book production of several interesting editions, such as the Latin texts of Euclid's *Elements*,[4] an edition of the letters of Nicolaus Copernicus,[5] Simon Bredon's *Arithmetica*,[6] a Latin translation by Plato of Tivoli of an Arabic astrolabe text,[7] a Latin translation of part II of the Arabic *Algebra* by Abū Kāmil Shujā' b. Aslam,[8] the Latin *Rithmachia* of Werinher von Tegernsee,[9] a middle-Dutch romance epic on the Crusades,[10] a seventeenth-century Hungarian politico-philosophical tract,[11] an anonymous Latin compilation from Hungary entitled *Sermones Compilati in Studio Gererali Quinqeecclesiensi in Regno Ungarie*,[12] the collected letters and papers of Leibniz,[13] Theodosius's *Spherics*, the German *Algorismus* of Sacrobosco, the Sanskrit text of the *Kāśikāvṛtti* of Vāmana and Jayāditya,[14] and the English texts of Thomas Middleton's collected works, as well as the editions illustrated in Appendix A.

In March 2003 Peter Wilson started an attempt to port EDMAC from TeX to LaTeX. The starting point was EDMAC version 3.16 as documented on 19 July 1994 (available from CTAN).

[4] Gerhard Brey used EDMAC in the production of Hubert L. L. Busard and Menso Folkerts, *Robert of Chester's (?) Redaction of Euclid's* Elements*, the so-called Adelard II Version*, 2 vols., (Basel, Boston, Berlin: Birkhäuser, 1992).

[5] Being prepared at the German Copernicus Research Institute, Munich.

[6] Being prepared by Menso Folkerts *et al.*, at the Institut für Geschichte der Naturwissenschaften in Munich.

[7] Richard Lorch, Gerhard Brey *et al.*, at the same Institute.

[8] Richard Lorch, 'Abū Kāmil on the Pentagon and Decagon' in *Vestigia Mathematica*, ed. M. Folkerts and J. P. Hogendijk (Amsterdam, Atlanta: Rodopi, 1993).

[9] Menso Folkerts, 'Die *Rithmachia* des Werinher von Tegernsee', ibid.

[10] Geert H. M. Claassens, *De Middelnederlandse Kruisvaartromans*, (Amsterdam: Schiphower en Brinkman, 1993).

[11] Emil Hargittay, *Csáky István: Politica philosophiai Okoskodás-szerint való rendes életnek példája (1664–1674)* (Budapest: Argumentum Kiadó, 1992).

[12] Being produced, as was the previous book, by Gyula Mayer in Budapest.

[13] Leibniz, *Sämtliche Schriften und Briefe*, series I, III, VII, being edited by Dr. H. Breger and Dr. N. Gädeke at the Leibniz-Archiv, Niedersächsische Landesbibliothek, Hannover.

[14] Being prepared at Poona and Lausanne Universities.

2 How to use EDMAC

2.1 Introduction

A document that uses EDMAC will use many standard TEX commands, along with special EDMAC commands. This chapter describes the usage of all the EDMAC commands; the name of each will be printed in the margin as it is introduced. We assume a basic familiarity with TEX conventions; if you are not conversant with TEX, you should be able to form an idea of the specific capabilities of EDMAC from this account, but (we repeat) you will not be able to use EDMAC without first learning some TEX.

All you need to do to invoke EDMAC is to include the line `\input EDMAC.DOC` at the top of your document, and to have the file EDMAC.DOC somewhere on your disk that is "visible" to TEX for input. It takes only a few seconds for TEX to read EDMAC.DOC, but if you are going to use it frequently, as will certainly be the case if you are doing a real edition, you will find it convenient to compile it into a TEX format file, loading it after PLAIN.TEX and any other private macros.

EDMAC *is a three-pass system*, like LATEX.[15] Although your textual apparatus and line numbers will be printed even on the first run, it takes two more passes through TEX to be sure that everything gets to its right place. Any changes you make to the input file may similarly require three passes to get everything to the right place, if the changes alter the number of lines or notes. EDMAC will tell you that you need to make more runs, when it notices, but it does not expend the labor to check this thoroughly. If you have problems with a line or two misnumbered at the top of a page, try running TEX once or twice more.

A file may mix *numbered* and *unnumbered* text. Numbered text is printed with marginal line numbers and can include footnotes and endnotes that are referenced to those line numbers: this is how you'll want to print the text that you're editing. Unnumbered text is not printed with line numbers, and you can't use EDMAC's note commands with it: this is appropriate for introductions and other material added by the editor around the edited text.

2.2 General markup

`\beginnumbering` `\endnumbering`

Each section of numbered text must be preceded by `\beginnumbering` and followed by `\endnumbering`:

`\beginnumbering`
⟨*text*⟩
`\endnumbering`

The `\beginnumbering` macro resets the line number to zero, reads an auxiliary file called ⟨*filename*⟩"."⟨*nn*⟩ (where *filename* is the name of the main input file for this job, and *nn* is "1" for the first numbered section, "2" for the second section, and so on), and then creates a new version of this auxiliary file to collect information during this run. The first instance of `\beginnumbering` also opens a

[15] Oh yes it is, if you have a table of contents, lists of figures, etc.

file called ⟨*filename*⟩".end" to receive the text of the endnotes. `\endnumbering` closes the ⟨*filename*⟩"."⟨*nn*⟩ file.

If the line numbering of a text is to be continuous from start to end, then the whole text will be typed between one pair of `\beginnumbering` and `\endnumbering` commands. But your text will most often contain chapter or other divisions marking sections that should be independently numbered, and these will be appropriate places to begin new numbered sections. `EDMAC` has to read and store in memory a certain amount of information about the entire section when it encounters a `\beginnumbering` command, so it speeds up the processing and reduces memory use when a text is divided into a larger number of sections (at the expense of multiplying the number of external files that are generated).

`\pstart` `\pend` Within a numbered section, each paragraph of numbered text must be marked using the `\pstart` and `\pend` commands:

`\pstart`

⟨*paragraph of text*⟩

`\pend`

Text that appears within a numbered section but isn't marked with `\pstart` and `\pend` will not be numbered.

The following example shows the proper section and paragraph markup, and the kind of output that would typically be generated:

```
\beginnumbering
\pstart
This is a sample paragraph, with
lines numbered automatically.
\pend

\pstart
This paragraph too has its
lines automatically numbered.
\pend

The lines of this paragraph are
not numbered.

\pstart
And here the numbering begins
again.
\pend
\endnumbering
```

1 This is a sample paragraph
2 with lines numbered
3 automatically.
4 This paragraph too
5 has its lines automatically
6 numbered.

The lines of this paragraph are not numbered.

7 And here the numbering
8 begins again.

`\autopar` You can use `\autopar` to avoid the nuisance of this paragraph markup and still have every paragraph automatically numbered. The scope of the `\autopar` command needs to be limited by keeping it within a group, as follows:

```
\begingroup
  \beginnumbering
  \autopar

  A paragraph of numbered text.

  Another paragraph of numbered
  text.

  \endnumbering
\endgroup
```

1 A paragraph of numbered
2 text.
3 Another paragraph of
4 numbered text.

`\autopar` fails, however, on paragraphs that start with a `{` or with any other command that starts a new group before it generates any text. Such paragraphs need to be started explicitly, before the new group is opened, using `\indent`, `\noindent`, or `\leavevmode`, or using `\pstart` itself.[16]

`\pausenumbering`
`\resumenumbering`

EDMAC stores a lot of information about line numbers and footnotes in memory as it goes through a numbered section. But at the end of such a section, it empties its memory out, so to speak. If your text has a very long numbered section it is possible that your TEX may reach its memory limit. There are two solutions to this. The first is to get a new TEX with increased memory. There are several BigTEX implementations easily available today, both commercial and public-domain, depending on your operating system. The second solution is to split your long section into several smaller ones. The trouble with this is that your line numbering will start again at zero with each new section. To avoid this problem, we provide `\pausenumbering` and `\resumenumbering` which are just like `\endnumbering` and `\beginnumbering`, except that they arrange for your line numbering to continue across the break. Use `\pausenumbering` only between numbered paragraphs:

```
\beginnumbering
\pstart
Paragraph of text.
\pend
\pausenumbering

\resumenumbering
\pstart
Another paragraph.
\pend
\endnumbering
\bye
```

1 Paragraph of
2 text.
3 Another paragraph.

We have defined these commands as two macros, in case you find it necessary to insert text between numbered sections without disturbing the line numbering. But if you are really just using these macros to save memory, you might as well say

[16] For a detailed study of the reasons for this restriction, see Barbara Beeton, "Initiation rites", *TUGboat* **12** (1991), pp. 257–258.

```
\def\memorybreak{\pausenumbering\resumenumbering}
```

and say `\memorybreak` between the relevant `\pend` and `\pstart`.

2.3 The apparatus

`\text` `\edtext` Within numbered paragraphs, all footnotes and endnotes are generated by forms of the `\text` macro:

`\text{`⟨*lemma*⟩`}`⟨*commands*⟩`/`

The original `\text` macro may look a little unusual to LaTeX users, as its second argument is ended by `/`. A more familiar format is provided by the `\edtext` macro:

`\edtext{`⟨*lemma*⟩`}{`⟨*commands*⟩`}`

The only difference between `\text` and `\edtext` is the method of delineating the second argument.

The ⟨*lemma*⟩ argument is the lemma in the main text: `\text` both prints this as part of the text, and makes it available to the ⟨*commands*⟩ you specify to generate notes. The `/` at the end terminates the command; it is part of the macro's definition so that spaces after the macro will be treated as significant.

For example:

```
I saw my friend \text{Smith}
\Afootnote{Jones C, D.}/
on Tuesday.
```

1 I saw my friend
2 Smith on Tuesday.

2 Smith] Jones C, D.

The lemma `Smith` is printed as part of this sentence in the text, and is also made available to the footnote that specifies a variant, `Jones C, D`. The footnote macro is supplied with the line number at which the lemma appears in the main text.

The ⟨*lemma*⟩ may contain further `\text` commands; this is the other reason why `\text`'s arguments are terminated by `/`. Nesting makes it possible to print an explanatory note on a long passage together with notes on variants for individual words within the passage. For example:

```
\text{I saw my friend
  \text{Smith}\Afootnote{Jones
  C, D.}/ on Tuesday.}
  \Bfootnote{The date was
  July 16, 1954.}
/
```

1 I saw my friend
2 Smith on Tuesday.

2 Smith] Jones C, D.

1–2 I saw my friend Smith on Tuesday.] The date was July 16, 1954.

However, `\text` cannot handle overlapping but unnested notes—for example, one note covering lines 10–15, and another covering 12–18; a `\text` that starts in the ⟨*lemma*⟩ argument of another `\text` must end there, too. (The `\lemma` and `\linenum` commands may be used to generate overlapping notes if necessary.)

Commands used in \text's second argument The second argument of the \text macro, ⟨*commands*⟩, may contain a series of subsidiary commands that generate various kinds of notes.

\Afootnote \Bfootnote \Cfootnote \Dfootnote \Efootnote

Five separate series of footnotes are maintained; when all five are used, the A notes appear in a layer just below the main text, followed by the rest in turn, down to the E notes at the bottom. These are the main macros that you will use to construct the critical apparatus of your text. EDMAC provides five layers of notes in the belief that this will be adequate for the most demanding editions. But it is not hard to add further layers of notes to EDMAC should they be required.

\Aendnote \Bendnote \Cendnote \Dendnote \Eendnote

EDMAC also maintains five separate series of endnotes. Normally, none of them is printed: you must use the \doendnotes macro described below (p. 15) to call for their output at the appropriate point in your document.

\lemma

Sometimes you want to change the lemma that gets passed to the notes. You can do this by using \lemma within the second argument to \text, before the note commands.

\lemma{⟨*alternative lemma*⟩}

The most common use of this command is to abbreviate the lemma that's printed in the notes. For example:

```
\text{I saw my friend
  \text{Smith}\Afootnote{Jones
  C, D.}/ on Tuesday.}
  \lemma{I \dots\ Tuesday.}
  \Bfootnote{The date was
  July 16, 1954.}
/
```

1 I saw my friend
2 Smith on Tuesday.
2 Smith] Jones C, D.
1–2 I . . . Tuesday.]
The date was July 16, 1954.

\linenum

You can use \linenum to change the line numbers passed to the notes. The notes are actually given seven parameters: the page, line, and sub-line number for the start of the lemma; the same three numbers for the end of the lemma; and the font specifier for the lemma. As argument to \linenum, you specify those seven parameters in that order, separated by vertical bars (the | character). However, you can retain the value computed by EDMAC for any number by simply omitting it; and you can omit a sequence of vertical bars at the end of the argument. For example, \linenum{|||23} changes one number, the ending page number of the current lemma.

This command doesn't change the marginal line numbers in any way; it just changes the numbers passed to the footnotes. Its use comes in situations that \text has trouble dealing with for whatever reason. If you need notes for overlapping passages that aren't nested, for instance, you can use \lemma and \linenum to generate such notes despite the limitations of \text. If the ⟨*lemma*⟩ argument to \text is extremely long, you may run out of memory; here again you can specify a note with an abbreviated lemma using \lemma and \linenum. The numbers used in \linenum need not be entered manually; you can use the “”x-”” symbolic cross-referencing commands below (p. 15) to compute them automatically.

Similarly, being able to manually change the lemma's font specifier in the notes might be important if you were using multiple scripts or languages. The form of

the font specifier is three separate codes separated by / characters, giving the family, series, and shape codes as defined within NFSS.

Changing the names of these commands The commands for generating the apparatus have been given rather bland names, because editors in different fields have widely divergent notions of what sort of notes are required, where they should be printed, and what they should be called. But this doesn't mean you have to type `\Afootnote` when you'd rather say something you find more meaningful, like `\variant`. We recommend that you create a series of such aliases and use them instead of the names chosen here; all you have to do is put commands of this form at the start of your file:

```
\let\variant=\Afootnote
\let\explanatory=\Bfootnote
\let\trivial=\Aendnote
\let\testimonia=\Cfootnote
```

It is also possible to define aliases for `\text`, which can be easier to type. You can make a single character substitute for `\text` by saying this:

```
\catcode`\<=\active
\let<=\text
```

Then you might say `<{Smith}\variant{Jones}/`. This of course destroys the ability to use `<` in any new macro definitions, so long as it remains in effect; hence it should be used with care.

Changing the character at the end of the command requires more work:

```
\catcode`\<=\active
\def\xtext#1#2>{\text{#1}{#2}/}
\let<=\xtext
```

This allows you to say `<{Smith}\Afootnote{Jones}>`.

Aliases for `\text` of the first kind shown here also can't be nested—that is, you can't use the alias in the text that forms the first argument to `\text`. (See section 6 to find out why.) Aliases of the second kind may be nested without any problem.

2.4 Lineation commands

`\lineation` EDMAC can number lines either by page or by section; you specify this using the `\lineation{`⟨*arg*⟩`}` macro, where ⟨*arg*⟩ is either `page` or `section`. You may only use this command at places where numbering is not in effect; you can't change the lineation system within a section. You can change it between sections: they don't all have to use the same lineation system. The line-of-section system is EDMAC's standard setting.

\linenummargin The marginal line numbers will be printed in the `left`, `right`, `inner`, or `outer` margin, depending on which you specify as argument to the `\linenummargin` command: for example, `\linenummargin{inner}`. Normally, line numbers appear in the left margin. You can change this whenever you're not in the middle of making a paragraph.

In most cases, you will not want a number printed for every single line of the text. Four LaTeX `counters` control the printing of marginal numbers. `firstlinenum` specifies the number of the first line in a section to number, and `linenumincrement` is the increment between numbered lines. `firstsublinenum` and `sublinenumincrement` do the same for sub-lines. Initially, all these counters are set equal to 5.

\leftlinenum \rightlinenum \linenumsep When a marginal line number is to be printed, there are a lot of ways to display it. You can redefine `\leftlinenum` and `\rightlinenum` to change the way marginal line numbers are printed in the left and right margins respectively; the initial versions print the number in font `\numlabfont` (described below) at a distance `\linenumsep` (initially set to one pica) from the text.

2.5 Changing the line numbers

Normally the line numbering starts at 1 for the first line of a section and steps up by one for each line thereafter. There are various common modifications of this system, however; the commands described here allow you to put such modifications into effect.

\startsub \endsub You insert the `\startsub` and `\endsub` commands in your text to turn sub-lineation on and off. In plays, for example, stage directions are often numbered with sub-line numbers: as line 10.1, 10.2, 10.3, rather than as 11, 12, and 13. Titles and headings are sometimes numbered with sub-line numbers as well.

When sub-lineation is in effect, the line number counter is frozen and the sub-line counter advances instead. If one of these commands appears in the middle of a line, it doesn't take effect until the next line; in other words, a line is counted as a line or sub-line depending on what it started out as, even if that changes in the middle.

\startlock \endlock The `\startlock` command, used in running text, locks the line number at its current value, until you say `\endlock`. It can tell for itself whether you are in a patch of line or sub-line numbering. One use for line-number locking is in printing poetry: there the line numbers should be those of verse lines rather than of printed lines, even when a verse line requires several printed lines.

\lockdisp When line-number locking is used, several printed lines may have the same line number, and you have to specify whether you want the number attached to the first printed line or the last, or whether you just want the number printed by them all. (This assumes that, on the basis of the settings of the previous parameters, it is necessary to display a line number for this line.) You specify your preference using `\lockdisp`; its argument is a word, either `first`, `last`, or `all`. EDMAC initially sets this to `first`.

\setline \advanceline In some cases you may want to modify the line numbers that are automatically calculated: if you are printing only fragments of a work but want to print

line numbers appropriate to a complete version, for example. The \setline and \advanceline commands may be used to change the current line's number (or the sub-line number, if sub-lineation is currently on). They change both the marginal line numbers and the line numbers passed to the notes. \setline takes one argument, the value to which you want the line number set; it must be 0 or greater. \advanceline takes one argument, an amount that should be added to the current line number; it may be positive or negative.

2.6 Alternate footnote formatting

If you just launch into EDMAC using the commands outlined above, you will get a standard layout for your text and notes. You may be happy to accept this at the very beginning, while you get the hang of things, but the standard layout is not particularly pretty, and you will certainly want to modify it in due course. EDMAC provides ways of changing the fonts and layout of your text, but these are not aimed at being totally comprehensive. They are enough to deal with simple variations from the norm, and to exemplify how you might go on to make more swingeing changes.

\footparagraph
\foottwocol
\footthreecol

All footnotes will normally be formatted as a series of separate paragraphs in one column. But there are three other formats available for notes, and using these macros you can select a different format for a series of notes. \footparagraph formats all the footnotes of a series as a single paragraph (see figs. 3 and 5, pp. 92 and 100); \foottwocol formats them as separate paragraphs, but in two columns (see bottom notes in fig. 4, p. 96); \footthreecol, in three columns (see second layer of notes in fig.2, p. 88). Each of these macros takes one argument: a letter (between A and E) for the series of notes you want changed. So a text with three layers of notes might begin thus:

```
\footnormal{A}
\footthreecol{B}
\footparagraph{C}
```

This would make the A-notes ordinary, B-notes would be in three columns, and the bottom layer of notes would be formed into a paragraph on each page.

\interparanoteglue

If you use paragraphed footnotes, the macro \interparanoteglue defines the glue appearing in between footnotes in the paragraph. It is a macro whose argument is the glue you want, and its initial setting is (see p. 69):

```
\interparanoteglue{1em plus .4em minus .4em}
```

You should set the \textwidth for the text, and the \baselineskip of the footnotes (this is done for you if you use the standard \notefontsetup), before you call any of these macros, because their action depends on those values; too much or too little space will be allotted for the notes on the page if these macros use the wrong values.[17]

[17] There is one tiny proviso about using paragraphed notes: you shouldn't force any explicit

2.7 Fonts

One of the most important features of the appearance of the notes, and indeed of your whole document, will be the fonts used. We will first describe the commands that give you control over the use of fonts in the different structural elements of the document, especially within the notes, and then in subsequent sections specify how these commands are used.

For those who are setting up EDMAC for a large job, here is a list of the complete set of EDMAC macros relating to fonts that are intended for manipulation by the user: \endashchar, \fullstop, \notefontsetup, \notenumfont, \numlabfont, and \rbracket.

\notefontsetup The \notefontsetup macro defines the standard size of the fonts for all your footnotes; ledmac initially chooses \footnotesize.

\notenumfont The \notenumfont macro specifies the font used for the line numbers printed in notes. This will typically be a command like \bfseries (ledmac's initial value is \normalfont) that selects a distinctive style for the note numbers, but leaves the choice of a size up to \notefontsetup.

\numlabfont Line numbers for the main text are usually printed in a smaller font in the margin. The \numlabfont macro is provided as a standard name for that font: it is initially set to be a \scriptsize normal font. You might wish to use a different font if, for example, you preferred to have these line numbers printed using old-style numerals.

\select@lemmafont We will briefly discuss \select@lemmafont here because it is important to know about it now, although it is not one of the macros you would expect to change in the course of a simple job. Hence it is "protected" by having the @-sign in its name.

When you use the \text macro to mark a word in your text as a lemma, that word will normally be printed again in your apparatus. If the word in the text happens to be in a font such as italic or bold you would probably expect it to appear in the apparatus in the same font. This becomes an absolute necessity if the font is actually a different script, such as Arabic or Cyrillic. \select@lemmafont does the work of decoding EDMAC's data about the fonts used to print the lemma in the main text and calling up those fonts for printing the lemma in the note.

\select@lemmafont is a macro that takes one long argument—the cluster of line numbers passed to the note commands. This cluster ends with a code indicating what fonts were in use at the start of the lemma. \select@lemmafont selects the appropriate font for the note using that font specifier.

EDMAC uses \select@lemmafont in a standard footnote format macro called \normalfootfmt. The footnote formats for each of the layers A to E are \let equal to \normalfootfmt. So all the layers of footnotes are formatted in the same way.

But it is also likely that you might want to have different fonts for just, say, the note numbers in layers A and B of your apparatus. To do this, make two

line-breaks inside such notes: do not use \par, \break, or \penalty=-10000. If you must have a line-break for some obscure reason, just suggest the break very strongly: \penalty=-9999 will do the trick. Page 68 explains why this restriction is necessary.

copies of the \normalfootfmt macro (see p. 61)—or \twocolfootfmt, or the other appropriate macro ending in -footfmt, depending on what footnote format you have selected—and give these macros the names \Afootfmt and \Bfootfmt. Then, in these new macros, change the font specifications (and spacing, or whatever) to your liking.

\endashchar \fullstop \rbracket

A relatively trivial matter relates to punctuation. In your footnotes, there will sometimes be spans of line numbers like this: 12–34, or lines with sub-line numbers like this: 55.6. The en-dash and the full stop are taken from the same font as the numbers, and it all works nicely. But what if you wanted to use old-style numbers, like 12 and 34? These look nice in an edition, but when you use the fonts provided by PLAIN TEX they are taken from a math font which does not have the en-dash or full stop in the same places as a text font. If you (or your macros) just typed $\oldstyle 12--34$ or $\oldstyle 55.6$ you would get "12″34"and "55▷6". So we define \endashchar and \fullstop, which produce an en-dash and a full stop respectively from the \rm font, whatever font you are using for the numbers. These two macros are used in the macros which format the line numbers in the margins and footnotes, instead of explicit punctuation. We also define an \rbracket macro for the right square bracket printed at the end of the lemma in many styles of textual notes (including EDMAC's standard style).

Here are some examples of how you might define some of the font macros.

```
\renewcommand*{\notefontsetup}{\small}
\renewcommand*{\notenumfont}{\sffamily}
```

These commands select \small fonts for the notes, and choose a sans font for the line numbers within notes.

2.8 Crop marks

The ledmac package does not provide crop marks. These are available with either the memoir class of the crop package.

2.9 Endnotes

\doendnotes \endprint

\doendnotes closes the .end file that contains the text of the endnotes, if it's open, and prints one series of endnotes, as specifed by a series-letter argument, e.g., \doendnotes{A}. \endprint is the macro that's called to print each note. It uses \notenumfont, \select@lemmafont, and \notefontsetup to select fonts, just as the footnote macros do (see p. 14 above).

\noendnotes

If you aren't going to have any endnotes, you can say \noendnotes in your file, before the first \beginnumbering, to suppress the generation of an unneeded .end file.

2.10 Cross referencing

EDMAC provides a simple cross-referencing facility that allows you to mark places in the text with labels, and generate page and line number references to those

places elsewhere in the text using those labels.

\edlabel First you place a label in the text using the command \edlabel{foo}. “”foo”” can be almost anything you like, including letters, numbers, punctuation, or a combination—anything but spaces; you might say \edlabel{toves-3}, for example.[18]

\edpageref \lineref \sublineref Elsewhere in the text, either before or after the \edlabel, you can refer to its location by saying \edpageref{foo}, or \lineref{foo}, or \sublineref{foo}. These commands will produce, respectively, the page, line and sub-line on which the \edlabel{foo} command occurred.

A \edlabel command may appear in the main text, or in the first argument of \text, but not in the apparatus itself. But \edpageref, \lineref and \sublineref commands can also be used in the apparatus to refer to \edlabels in the text.

The \edlabel command works by writing macros to an .lxb file (which will only be created if you are actually using some of these commands). Clearly, then, you will need to process your document through LaTeX twice in order for the references to be resolved.

You will be warned if you say \edlabel{foo} and foo has been used as a label before. The ref commands will return references to the last place in the file marked with this label. You will also be warned if a reference is made to an undefined label. (This will also happen the first time you process a document after adding a new \edlabel command: the auxiliary file will not have been updated yet.)

If you want to refer to a word inside a \text{...}/ command, the \edlabel should be defined inside the first argument, e.g.,

```
The \text{creature\edlabel{elephant} was quite
  unafraid}\Afootnote{Of the mouse, that is.}/
```

\xpageref \xlineref \xsublineref However, there are situations in which you'll want EDMAC to return a number without displaying any warning messages about undefined labels or the like: if you want to use the reference in a context where TEX is looking for a number, such a warning will lead to a complaint that the number is missing. This is the case for references used within the argument to \linenum, for example. For this situation, three variants of the reference commands, with the x prefix, are supplied: \xpageref, \xlineref, and \xsublineref. The only operations they perform are ones that TEX can do in its “mouth”. They have these limitations: they will not tell you if the label is undefined, and they must be preceded in the file by at least one of the four other cross-reference commands—e.g., a \label{foo} command, even if you never refer to that label—since those commands can all do the necessary processing of the .aux file, and these cannot.

\xxref The macros \xxref and \edmakelabel let you manipulate numbers and labels in ways which you may find helpful in tricky situations.

The \xxref command generates a reference to a sequence of lines, for use in the second argument of \text. It takes two arguments, both of which are

[18] More precisely, you should stick to characters in the TEX categories of “letter” and “other”.

labels: e.g., `\xxref{mouse}{elephant}`. It calls `\linenum` (q.v., p. 10 above) and sets the beginning page, line, and sub-line numbers to those of the place where `\edlabel{mouse}` was placed, and the ending numbers to those where `\edlabel{elephant}` occurs.

`\edmakelabel` Sometimes the `\edlabel` command cannot be used to specify exactly the page and line desired—for example, if you want to refer to a page and line number in another volume of your edition. In such cases, you can use the `\edmakelabel` macro so that you can "roll your own" label. For example, if you say "`\edmakelabel{elephant}{10|25|0}`" you will have created a new label, and a later call to `\edpageref{elephant}` would print "10" and `\lineref{elephant}` would print "25". The sub-line number here is zero. It is usually best to collect your `\edmakelabel` statements near the top of your document, so that you can see them at a glance.

2.11 Miscellaneous

Generally, you should set the `\vsize` and `\hsize` of you document at the top, before any `EDMAC` commands are issued, since `EDMAC` uses these values to work out some things, like crop marks and footnote spacing. As *The TEXbook* says (p. 251), "It's best not to monkey with `\hsize` and `\vsize` except at the very beginning of a job...".

Any changes you make to `\leftskip` or `\rightskip` will apply to the main body text, but not to the footnotes. This is how you can get a narrow-set text on a wider base of notes, a common style for critical editions. Footnote material is always set `\hsize` wide.

`\extensionchars` When `EDMAC` assembles the name of the auxiliary file for a section, it prefixes `\extensionchars` to the section number. This is initially defined to be empty, but you can add some characters to help distinguish these files if you like; what you use is likely to be system-dependent. If, for example, you said `\def\extensionchars{!}`, then you would get temporary files called `jobname.!1`, `jobname.!2`, etc.

2.12 Known bugs

The PLAIN TEX `\footnote` command will work only within unnumbered text; within numbered text it will wreak havoc. In general, `EDMAC`'s system for adding marginal line numbers breaks anything that makes direct use of the TEX insert system.

`\parshape` cannot be used within numbered text, except in a very restricted way (see p. 50).

`\ballast` `EDMAC` is a three-pass system, but even after a document has been processed three times, there are some tricky situations in which the page breaks decided by TEX never settle down. At each successive run of TEX, `EDMAC` oscillates between two different sets of page decisions. To stop this happening, should it arise, Wayne Sullivan suggested the inclusion of the quantity `\ballast`. The amount of `\ballast` will be subtracted from the penalties which apply to the page breaks

calculated on the *previous* run through TEX, thus reinforcing these breaks. So if you find your page breaks oscillating, say `\ballast=100` or some such figure, and with any luck the page breaks will settle down. Luckily, this problem doesn't crop up at all often.

The restriction on explicit line-breaking in paragraphed footnotes, mentioned in footnote 17, p. 13, and described in more detail on p. 68, really is a nuisance if that's something you need to do. There are some possible solutions, described by Michael Downes, but this area remains unsatisfactory.

Help, suggestions and corrections will be gratefully received.

3 Implementation overview

Commentary presented in this font is by PRW.

We present the EDMAC code in roughly the order in which it's used during a run of TEX. The order is *exactly* that in which it's read when you load the EDMAC package, because the same file is used to generate this book and to generate the TEX input file. Most of what follows consists of macro definitions, but there are some TEX commands that are executed immediately—especially at the start of the code. The documentation generally describes the code from the point of view of what happens when the macros are executed, though. As each macro is introduced, its name is printed in the margin.

We begin with the commands you use to start and stop line numbering in a section of text (Section 4). Next comes the machinery for writing and reading the auxiliary file for each section that helps us count lines, and for creating list macros encoding the information from that file (Section 5); this auxiliary file will be read at the start of each section, to create those list macros, and a new version of the file will be started to collect information from the body of the section.

Next are commands for marking sections of the text for footnotes (Section 6), followed by the macros that take each paragraph apart, attach the line numbers and insertions, and send the result to the vertical list (Section 7). The footnote commands (Section 8) and output routine (Section 9) finish the main part of the processing; cross-referencing (Section 10) and endnotes (Section 11) complete the story.

In what follows, macros with an @ in their name are more internal to the workings of EDMAC than those made up just of ordinary letters, just as in PLAIN TEX (see *The TEXbook*, p. 344). You are meant to be able to make free with ordinary macros, but the "@" ones should be treated with more respect, and changed only if you are pretty sure of what you are doing.

4 Preliminaries

I'll try and use `l@d` in macro names to help avoid name clashes, but this is not a hard and fast rule. For example, if an original EDMAC macro includes `edmac` I'll simply change that to `ledmac`.

Announce the name and version of the package, which is targetted for LaTeX2e.

```
⟨*code⟩
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{ledmac}[2003/04/01 v0.0 LaTeX port of EDMAC]

```

Replace as many `\def`'s by `\newcommand`'s as possible to avoid overwriting LaTeX macros.

Replace user-level TeX counts by LaTeX counters.

Use the LaTeX font handling mechanisms.

Use LaTeX messaging and file facilities.

`\@l@dtempcnta` `\@l@dtempcntb` In imitation of LaTeX, we create a couple of scratch counters, `\@l@dtempcnta` and `\@l@dtempcntb`, that we can use like `\count255` (see *The TeXbook*, p. 122).

```
\newcount\@l@dtempcnta \newcount\@l@dtempcntb
```

`\ledmac@warning` Write a warning message. Changed to use LaTeX capabilities.

```
\newcommand{\ledmac@warning}[1]{\PackageWarning{ledmac}{#1}}
```

4.1 Sectioning commands

`\section@num` You use `\beginnumbering` and `\endnumbering` to begin and end a line-numbered section of the text; the pair of commands may be used as many times as you like within one document to start and end multiple, separately line-numbered sections. TeX will maintain and display a "section number" as a counter named `\section@num` that counts how many `\beginnumbering` and `\resumenumbering` commands have appeared; it needn't be related to the logical divisions of your text.

`\extensionchars` Each section will read and write an associated "line-list file", containing information used to do the numbering; the file will be called ⟨*jobname*⟩"."⟨*nn*⟩, where *nn* is the section number. However, you may direct that an extra string be added before the *nn* in that filename, in order to distinguish these temporary files from others: that string is called `\extensionchars`. Initially it's empty, since different operating systems have greatly varying ideas about what characters are permitted in file names. So `\def\extensionchars{-}` gives temporary files called `jobname.-1`, `jobname.-2`, etc.

```
\newcount\section@num
\section@num=0
\let\extensionchars=\empty
```

`\ifnumbering` `\numberingtrue` `\numberingfalse` The `\ifnumbering` flag is set to `true` if we're within a numbered section (that is, between `\beginnumbering` and `\endnumbering`). You can use `\ifnumbering` in your own code to check whether you're in a numbered section, but don't change the flag's value.

```
\newif\ifnumbering
```

\beginnumbering

\beginnumbering begins a section of numbered text. When it's executed we increment the section number, initialize our counters, send a message to your terminal, and call macros to start the lineation machinery and endnote files.

The initializations here are trickier than they look. \line@list@stuff will use all of the counters that are zeroed here when it assembles the line-list and other lists of information about the lineation. But it will do all of this locally and within a group, and when it's done the lists will remain but the counters will return to zero. Those same counters will then be used as we process the text of this section, but the assignments will be made globally. These initializations actually apply to both uses, though in all other respects there should be no direct interaction between the use of these counters and variables in the two processing steps.

```
%\def\beginnumbering{%
\newcommand*{\beginnumbering}{%
  \ifnumbering
     \errmessage{Numbering has already been started}%
     \endnumbering
  \fi
  \global\numberingtrue
  \global\advance\section@num by 1
  \global\absline@num=0
  \global\line@num=0
  \global\subline@num=0
  \global\@lock=0
  \global\sub@lock=0
  \global\sublines@false
  \global\let\next@page@num=\relax
  \global\let\sub@change=\relax
  \message{Section \the\section@num }%
  \line@list@stuff{\jobname.\extensionchars\the\section@num}%
  \end@stuff}
```

\endnumbering

\endnumbering must follow the last text for a numbered section. It takes care of notifying you when changes have been noted in the input that require running the file through again to move everything to the right place.

```
\def\endnumbering{%
  \ifnumbering
    \global\numberingfalse
    \normal@pars
    \ifx\insertlines@list\empty\else
      \global\noteschanged@true
    \fi
    \ifx\line@list\empty\else
      \global\noteschanged@true
    \fi
    \ifnoteschanged@
      \immediate\write\sixt@@n{EDMAC reminder: }%
      \immediate\write\sixt@@n{ The number of footnotes in this section
                               has changed since the last run.}%
```

```
      \immediate\write\sixt@@n{ You will need to run TeX two more times
                               before the footnote placement}%
      \immediate\write\sixt@@n{ and line numbering in this section are
                               correct.}%
    \fi
  \else
    \errmessage{Numbering was not started}%
  \fi}
```

\pausenumbering
\resumenumbering

The \pausenumbering macro is just the same as \endnumbering, but with the \ifnumbering flag set to true, to show that numbering continues across the gap.[19]

```
%\def\pausenumbering{%
\newcommand{\pausenumbering}{%
  \endnumbering\global\numberingtrue}
```

The \resumenumbering macro is a bit more involved, but not much. It does most of the same things as \beginnumbering, but without resetting the various counters. Note that no check is made by \resumenumbering to ensure that \pausenumbering was actually invoked.

```
%\def\resumenumbering{%
\newcommand*{\resumenumbering}{%
  \ifnumbering
     \global\advance\section@num by 1
     \message{Section \the\section@num\space
                                (continuing the previous section)}%
     \line@list@stuff{\jobname.\extensionchars\the\section@num}%
     \end@stuff
  \else
     \errmessage{Numbering should already have been started.}%
     \endnumbering
     \beginnumbering
  \fi}

```

5 Line counting

5.1 Choosing the system of lineation

Sometimes you want line numbers that start at 1 at the top of each page; other times you want line numbers that start at 1 at the start of each section and increase regardless of page breaks. EDMAC can do it either way, and you can switch from one to the other within one work. But you have to choose one or the other for all line numbers and line references within each section. Here we will define internal codes for these systems and the macros you use to select them.

\ifbypage@
\bypage@true
\bypage@false

The \ifbypage@ flag specifies the current lineation system: true for line-of-section, false for line-of-page. EDMAC will use the line-of-section system unless

[19] Our thanks to Wayne Sullivan, who suggested the idea behind these macros.

instructed otherwise.

```
\newif\ifbypage@
```

`\lineation` `\lineation` is the macro you use to select the lineation system. Its argument is a string: either `page` or `section`.

```
%\def\lineation#1{{%
\newcommand*{\lineation}[1]{{%
  \ifnumbering
    \errmessage{You can't use \string\lineation\space
                           within a numbered section}%
  \else
    \def\@tempa{#1}\def\@tempb{page}%
    \ifx\@tempa\@tempb
        \global\bypage@true
    \else
       \def\@tempb{section}%
       \ifx\@tempa\@tempb
           \global\bypage@false
       \else
           \ledmac@warning{Bad \string\lineation\space argument.}%
       \fi
    \fi
  \fi}}

```

`\linenummargin` `\line@margin` You call `\linenummargin` to specify which margin you want your line numbers in; it takes one argument, a string. You can put the line numbers in the same margin on every page using `left` or `right`; or you can use `inner` or `outer` to get them in the inner or outer margins. (These last two options assume that even-numbered pages will be on the left-hand side of every opening in your book.) You can change this within a numbered section, but the change may not take effect just when you'd like; if it's done between paragraphs nothing surprising should happen.

The selection is recorded in `\line@margin`: 0 for left, 1 for right, 2 for outer, and 3 for inner.

```
\newcount\line@margin
%\def\linenummargin#1{{%
\newcommand*{\linenummargin}[1]{{%
  \def\@tempa{#1}\def\@tempb{left}%
  \ifx\@tempa\@tempb
      \global\line@margin=0
  \else
     \def\@tempb{right}%
     \ifx\@tempa\@tempb
         \global\line@margin=1
     \else
       \def\@tempb{outer}%
       \ifx\@tempa\@tempb
```

```
            \global\line@margin=2
        \else
          \def\@tempb{inner}%
          \ifx\@tempa\@tempb
              \global\line@margin=3
          \else
              \ledmac@warning{Bad \string\linenummargin\space argument.}%
          \fi
        \fi
      \fi
  \fi}}

```

`\c@firstlinenum` `\c@linenumincrement`

The following parameters tell EDMAC which lines should be printed with line numbers. `\firstlinenum` is the number of the first line in each section that gets a number; `\linenumincrement` is the difference between successive numbered lines. The initial values of these counters produce labels on lines 5, 10, 15, etc. `\linenumincrement` must be at least 1. I have changed the original counts into LaTeX counters.

```
%\newcount\firstlinenum
%\newcount\linenumincrement
%\firstlinenum=5
%\linenumincrement=5
\newcounter{firstlinenum}
  \setcounter{firstlinenum}{5}
\newcounter{linenumincrement}
  \setcounter{linenumincrement}{5}
```

`\c@firstsublinenum` `\c@sublinenumincrement`

The following parameters are just like `\firstlinenum` and `\linenumincrement`, but for sub-line numbers. `\sublinenumincrement` must be at least 1. I have changed the original counts into LaTeX counters.

```
%\newcount\firstsublinenum
%\newcount\sublinenumincrement
%\firstsublinenum=5
%\sublinenumincrement=5
\newcounter{firstsublinenum}
  \setcounter{firstsublinenum}{5}
\newcounter{sublinenumincrement}
  \setcounter{sublinenumincrement}{5}

```

`\lockdisp` `\lock@disp`

When line locking is being used, the `\lockdisp` macro specifies whether a line number—if one is due to appear—should be printed on the first printed line or on the last, or by all of them. Its argument is a word, either `first`, `last`, or `all`. Initially, it is set to `first`.

`\lock@disp` encodes the selection: 0 for first, 1 for last, 2 for all.

```
\newcount\lock@disp
%\def\lockdisp#1{{%
```

```
\newcommand{\lockdisp}[1]{{%
  \def\@tempa{#1}\def\@tempb{first}%
  \ifx\@tempa\@tempb
      \global\lock@disp=0
  \else
     \def\@tempb{last}%
     \ifx\@tempa\@tempb
         \global\lock@disp=1
     \else
       \def\@tempb{all}%
       \ifx\@tempa\@tempb
           \global\lock@disp=2
       \else
           \ledmac@warning{Bad \string\lockdisp\space argument.}%
       \fi
     \fi
  \fi}}
```

`\sublockdisp` `\sublock@disp` The same questions about where to print the line number apply to sub-lines, and these are the analogous macros for dealing with the problem.

```
\newcount\sublock@disp
%\def\sublockdisp#1{{%
\newcommand{\sublockdisp}[1]{{%
  \def\@tempa{#1}\def\@tempb{first}%
  \ifx\@tempa\@tempb
      \global\sublock@disp=0
  \else
     \def\@tempb{last}%
     \ifx\@tempa\@tempb
         \global\sublock@disp=1
     \else
       \def\@tempb{all}%
       \ifx\@tempa\@tempb
           \global\sublock@disp=2
       \else
           \ledmac@warning{Bad \string\sublockdisp\space argument.}%
       \fi
     \fi
  \fi}}

```

`\leftlinenum` `\rightlinenum` `\linenumsep` `\numlabfont` `\leftlinenum` and `\rightlinenum` are the macros that are called to print marginal line numbers on a page, for left- and right-hand margins respectively. They're made easy to access and change, since you may often want to change the styling in some way. These standard versions illustrate the general sort of thing that will be needed; they're based on the `\leftheadline` macro in *The TEXbook*, p. 416.

Whatever these macros output gets printed in a box that will be put into the appropriate margin without any space between it and the line of text. You'll

generally want a kern between a line number and the text, and `\linenumsep` is provided as a standard way of storing its size. Line numbers are usually printed in a smaller font, and `\numlabfont` is provided as a standard name for that font. When called, these macros will be executed within a group, so font changes and the like will remain local.

The original `\numlabfont` specification is equivalent to the LaTeX `\scriptsize` for a 10pt document.

```
\newdimen\linenumsep
\linenumsep=1pc
%\ifx\selectfont\undefined
%  \let\numlabfont=\sevenrm
%\else
%  \def\numlabfont{\fontsize{7}{8pt}\rm}
  \newcommand{\numlabfont}{\normalfont\scriptsize}
%\fi
%\def\leftlinenum{\numlabfont\the\line@num
\newcommand*{\leftlinenum}{\numlabfont\the\line@num
              \ifsublines@
                \ifnum\subline@num>0
                     \unskip\fullstop\the\subline@num
                \fi
              \fi
              \kern\linenumsep}
%\def\rightlinenum{\kern\linenumsep \numlabfont\the\line@num
\newcommand*{\rightlinenum}{\kern\linenumsep \numlabfont\the\line@num
              \ifsublines@
                \ifnum\subline@num>0
                     \unskip\fullstop\the\subline@num
                \fi
              \fi}

```

5.2 List macros

Reminder: compare these with the LaTeX list macros in case they would be suitable instead.

We will make heavy use of lists of information, which will be built up and taken apart by the following macros; they are adapted from *The TEXbook*, pp. 378–379, which discusses their use in more detail.

These macros consume a large amount of the run-time of this code. We intend to replace them in a future version, and in anticipation of doing so have defined their interface in such a way that it is not sensitive to details of the underlying code.

`\list@create` The `\list@create` macro creates a new list. In this version of `EDMAC` this macro doesn't do anything beyond initializing an empty list macro, but in future versions it will do more.

```
%\def\list@create#1{\global\let#1=\empty}
```

```
\newcommand*{\list@create}[1]{\global\let#1=\empty}
```

`\list@clear` The `\list@clear` macro just initializes a list to the empty list; in this version of EDMAC it is no different from `\list@create`.

```
%\def\list@clear#1{\global\let#1=\empty}
\newcommand*{\list@clear}[1]{\global\let#1=\empty}
```

`\xright@appenditem` `\@toksa` `\@toksb` `\xright@appenditem` expands an item and appends it to the right end of a list macro. We want the expansion because we'll often be using this to store the current value of a counter. It creates global control sequences, like `\xdef`, and uses two temporary token-list registers, `\@toksa` and `\@toksb`.

```
\newtoks\@toksa \newtoks\@toksb
\global\@toksa={\\}
\long\def\xright@appenditem#1\to#2{%
  \global\@toksb=\expandafter{#2}%
  \xdef#2{\the\@toksb\the\@toksa\expandafter{#1}}%
  \global\@toksb={}}
```

`\xleft@appenditem` `\xleft@appenditem` expands an item and appends it to the left end of a list macro; it is otherwise identical to `\xright@appenditem`.

```
\long\def\xleft@appenditem#1\to#2{%
  \global\@toksb=\expandafter{#2}%
  \xdef#2{\the\@toksa\expandafter{#1}\the\@toksb}%
  \global\@toksb={}}
```

`\gl@p` The `\gl@p` macro removes the leftmost item from a list and places it in a control sequence. You say `\gl@p\l\to\z` (where `\l` is the list macro, and `\z` receives the left item). `\l` is assumed nonempty: say `\ifx\l\empty` to test for an empty `\l`. The control sequences created by `\gl@p` are all global.

```
\def\gl@p#1\to#2{\expandafter\gl@poff#1\gl@poff#1#2}
\long\def\gl@poff\\#1#2\gl@poff#3#4{\gdef#4{#1}\gdef#3{#2}}

```

5.3 Line-number counters and lists

Footnote references using line numbers rather than symbols can't be generated in one pass, because we don't know the line numbers till we ship out the pages. It would be possible if footnotes were never keyed to more than one line; but some footnotes gloss passages that may run for several lines, and they must be tied to the first line of the passage glossed. And even one-line passages require two passes if we want line-per-page numbering rather than line-per-section numbering.

So we run TEX over the text several times, and each time save information about page and line numbers in a "line-list file" to be used during the next pass. At the start of each section—whenever `\beginnumbering` is executed—the line-list file for that section is read, and the information from it is encoded into a few list macros.

We need first to define the different line numbers that are involved in these macros, and the associated counters.

\line@num The \line@num counter stores the line number that's used in marginal line numbering and in notes: counting either from the start of the page or from the start of the section, depending on your choice for this section. This may be qualified by \subline@num.

```
211 \newcount\line@num
```

\subline@num The \subline@num counter stores a sub-line number that qualifies \line@num. For example, line 10 might have sub-line numbers 1, 2 and 3, which might be printed as lines 10.1, 10.2, 10.3.

```
212 \newcount\subline@num
```

\ifsublines@ \sublines@true \sublines@false We maintain an associated flag, \ifsublines@, to tell us whether we're within a sub-line range or not.

You may wonder why we don't just use the value of \subline@num to determine this—treating anything greater than 0 as an indication that sub-lineation is on. We need a separate flag because sub-lineation can be used together with line-number locking in odd ways: several pieces of a logical line might be interrupted by pieces of sub-lineated text, and those sub-line numbers should not return to zero until the next change in the major line number. This is common in the typesetting of English Renaissance verse drama, in which stage directions are given sub-line numbers: a single line of verse may be interrupted by several stage directions.

```
213 \newif\ifsublines@
```

\absline@num The \absline@num counter stores the absolute number of lines since the start of the section: that is, the number we've actually printed, no matter what numbers we attached to them. This value is never printed on an output page, though \line@num will often be equal to it. It is used internally to keep track of where notes are to appear and where new pages start: using this value rather than \line@num is a lot simpler, because it doesn't depend on the lineation system in use.

```
214 \newcount\absline@num
```

We'll be calling \absline@num numbers "absolute" numbers, and \line@num and \subline@num numbers "visible" numbers.

\@lock \sub@lock The \@lock and \sub@lock counters tell us the state of line-number and sub-line-number locking. 0 means we're not within a locked set of lines; 1 means we're at the first line in the set; 2, at some intermediate line; and 3, at the last line.

```
215 \newcount\@lock
216 \newcount\sub@lock
```

\line@list \insertlines@list \actionlines@list \actions@list Now we can define the list macros that will be created from the line-list file. We will maintain the following lists:

- \line@list: the page and line numbers for every lemma marked by \text. There are seven pieces of information, separated by vertical bars:
  1. the starting page,

2. line, and
3. sub-line numbers, followed by the
4. ending page,
5. line, and
6. sub-line numbers, and then the
7. font specifier for the lemma.

These line numbers are all visible numbers. Thus a lemma that started on page 23, line 35 and went on until page 24, line 3 (with no sub-line numbering), and was typeset in a font from `\fam0` would have a line list entry like this: `23|35|0|24|3|0|0`. That assumes the use of the Plain font selection scheme, which uses the family number as its font specifier. When NFSS is used, the font specifier is a set of four codes for font encoding, family, series, and shape, separated by `/` characters; in that case, the line-list entry would be `23|35|0|24|3|0|OT1/cmr/m/n`.

There is one item in this list for every lemma marked by `\text`, even if there are several notes to that lemma, or no notes at all. `\text` reads the data in this list, making it available for use in the text of notes.

- `\insertlines@list`: the line numbers of lines that have footnotes or other insertions. These are the absolute numbers where the corresponding lemmas begin. This list contains one entry for every footnote in the section; one lemma may contribute no footnotes or many footnotes. This list is used by `\add@inserts` within `\do@line`, to tell it where to insert notes.
- `\actionlines@list`: a list of absolute line numbers at which we are to perform special actions; these actions are specified by the `\actions@list` list defined below.
- `\actions@list`: action codes corresponding to the line numbers in `\actionlines@list`. These codes tell `EDMAC` what action it's supposed to take at each of these lines. One action, the page-start action, is generated behind the scenes by `EDMAC` itself; the others, for specifying sub-lineation, line-number locking, and line-number alteration, are generated only by explicit commands in your input file. The page-start and line-number-alteration actions require arguments, to specify the new values for the page or line numbers; instead of storing those arguments in another list, we have chosen the action-code values so that they can encode both the action and the argument in these cases. Action codes greater than -1000 are page-start actions, and the code value is the page number; action codes less than -5000 specify line numbers, and the code value is a transformed version of the line number; action codes between these two values specify other actions which require no argument.

  Here is the full list of action codes and their meanings:

  Any number greater than -1000 is a page-start action: the line number associated with it is the first line on a page, and the action number is the page number. (The cutoff of -1000 is chosen because negative page-number values are used by some macro packages; we assume that page-number values

less than −1000 are not common.) Page-start action codes are added to the list by the `\page@action` macro, which is (indirectly) triggered by the workings of the `\page@start` macro; that macro should always be called in the output routine, just before the page contents are assembled. `EDMAC` calls it in `\pagecontents`.

The action code −1001 specifies the start of sub-lineation: meaning that, starting with the next line, we should be advancing `\subline@num` at each start-of-line command, rather than `\line@num`.

The action code −1002 specifies the end of sub-lineation. At the next start-of-line, we should clear the sub-line counter and start advancing the line number. The action codes for starting and ending sub-lineation are added to the list by the `\sub@action` macro, as called to implement the `\startsub` and `\endsub` macros.

The action code −1003 specifies the start of line number locking. After the number for the current line is computed, it will remain at that value through the next line that has an action code to end locking.

The action code −1004 specifies the end of line number locking.

The action code −1005 specifies the start of sub-line number locking. After the number for the current sub-line is computed, it will remain at that value through the next sub-line that has an action code to end locking.

The action code −1006 specifies the end of sub-line number locking.

The four action codes for line and sub-line number locking are added to the list by the `\do@lockon` and `\do@lockoff` macros, as called to implement the `\startlock` and `\endlock` macros.

An action code of −5000 or less sets the current visible line number (either the line number or the sub-line number, whichever is currently being advanced) to a specific positive value. The value of the code is $-(5000 + n)$, where n is the value (always ≥ 0) assigned to the current line number. Action codes of this type are added to the list by the `\set@line@action` macro, as called to implement the `\advanceline` and `\setline` macros: this action only occurs when the user has specified some change to the line numbers using those macros. Normally `EDMAC` computes the visible line numbers from the absolute line numbers with reference to the other action codes and the settings they invoke; it doesn't require an entry in the action-code list for every line.

Here are the commands to create these lists:

```
\list@create{\line@list}
\list@create{\insertlines@list}
\list@create{\actionlines@list}
\list@create{\actions@list}
```

\page@num
\endpage@num
\endline@num
\endsubline@num

We'll need some counters while we read the line-list, for the page number and the ending page, line, and sub-line numbers. Some of these will be used again later on, when we are acting on the data in our list macros.

```
\newcount\page@num
\newcount\endpage@num
\newcount\endline@num
\newcount\endsubline@num
```

\ifnoteschanged@
\noteschanged@true
\noteschanged@false

If the number of footnotes in a section is different from what it was during the last run, or if this is the very first time you've run TEX on this file, the information from the line-list used to place the notes will be wrong, and some notes will probably be misplaced. When this happens, we prefer to give a single error message for the whole section rather than messages at every point where we notice the problem, because we don't really know where in the section notes were added or removed, and the solution in any case is simply to run TEX two more times; there's no fix needed to the document. The `\ifnoteschanged@` flag is set if such a change in the number of notes is discovered at any point.

```
\newif\ifnoteschanged@
```

5.4 Reading the line-list file

\read@linelist
\@inputcheck

`\read@linelist` is the control sequence that's called by `\beginnumbering` (via `\line@list@stuff`) to open and process a line-list file; its argument is the name of the file, which will be opened on stream `\@inputcheck` to check for its existence. The first thing we do is initialize all the lists we just described.

```
\newread\@inputcheck
%\def\read@linelist#1{%
\newcommand*{\read@linelist}[1]{%
  \list@clear{\line@list}%
  \list@clear{\insertlines@list}%
  \list@clear{\actionlines@list}%
  \list@clear{\actions@list}%
```

Try to open the line-list file, as a check on whether it exists.

```
  \openin\@inputcheck=#1
  \ifeof\@inputcheck
    \ledmac@warning{Can't find line-list file #1}%
    \global\noteschanged@true
  \else
    \global\noteschanged@false
    \closein\@inputcheck
```

The file's there. We start a new group and make some special definitions we'll need to process it: it's a sequence of TEX commands, but they require a few special settings. We make [and] become grouping characters: they're used that way in the line-list file, because we need to write them out one at a time rather than in balanced pairs, and it's easier to just use something other than real braces. @ must become a letter, since this is run in the ordinary TEX context. We ignore carriage

returns, since if we're in horizontal mode they can get interpreted as spaces to be printed.

```
\begingroup
   \catcode`\[=1 \catcode`\]=2
   \makeatletter \catcode`\^^M=9
```

Our line, page, and line-locking counters were already zeroed by `\line@list@stuff` if this is being called from within `\beginnumbering`; sub-lineation will be turned off as well in that case. On the other hand, if this is being called from `\resumenumbering`, those things should still have the values they had when `\pausenumbering` was executed.

Now, after these preliminaries, we start interpreting the file.

```
   \input #1
\endgroup
\fi
```

When the `\input` is done, we're all through with the line-list file. All the information we needed from it will now be encoded in our list macros.

Finally, we initialize the `\next@actionline` and `\next@action` macros, which specify where and what the next action to be taken is.

```
\global\page@num=-1
\ifx\actionlines@list\empty
    \gdef\next@actionline{1000000}%
\else
    \gl@p\actionlines@list\to\next@actionline
    \gl@p\actions@list\to\next@action
\fi}

```

This version of `\read@linelist` creates list macros containing data for the entire section, so they could get rather large. It would be no more difficult to read the line-list file incrementally rather than all at once: we could read, at the start of each paragraph, only the commands relating to that paragraph, using `\read`. But this would require that we have two line-lists open at once, one for reading, one for writing, and on systems without version numbers we'd have to do some file renaming outside of TEX for that to work. We've retained this slower approach to avoid that sort of hacking about, but have provided the `\pausenumbering` and `\resumenumbering` macros to help you if you run into macro memory limitations (see p. 8 above).

5.5 Commands within the line-list file

This section defines the commands that can appear within a line-list file. They all have very short names because we are likely to be writing very large numbers of them out. One macro, `\@l`, is especially short, since it will be written to the line-list file once for every line of text in a numbered section. (Another of these commands, `\@lab`, will be introduced in a later section, among the cross-referencing commands it is associated with.)

When these commands modify the various page and line counters, they deliberately do not say `\global`. This is because we want them to affect only the counter values within the current group when nested calls of `\@ref` occur. (The code assumes throughout that the value of `\globaldefs` is zero.)

The macros with `action` in their names contain all the code that modifies the action-code list: again, this is so that they can be turned off easily for nested calls of `\@ref`.

`\@l` `\@l` does everything related to the start of a new line of numbered text.

First increment the absolute line-number, and perform deferred actions relating to page starts and sub-lines.

```
%\def\@l{\advance\absline@num by 1
\newcommand*{\@l}{\advance\absline@num by 1
        \ifx\next@page@num\relax \else
            \page@action
            \let\next@page@num=\relax
        \fi
        \ifx\sub@change\relax \else
           \ifnum\sub@change>0
              \sublines@true
           \else
              \sublines@false
           \fi
           \sub@action
           \let\sub@change=\relax
        \fi
```

Fix the lock counters, if necessary. A value of 1 is advanced to 2; 3 advances to 0; other values are unchanged.

```
        \ifcase\@lock
           \or
              \@lock=2
           \or \or
              \@lock=0
        \fi
        \ifcase\sub@lock
           \or
              \sub@lock=2
           \or \or
              \sub@lock=0
        \fi
```

Now advance the visible line number, unless it's been locked.

```
        \ifsublines@
             \ifnum\sub@lock<2
               \advance\subline@num by 1
             \fi
        \else
             \ifnum\@lock<2
               \advance\line@num by 1 \subline@num=0
```

```
289               \fi
290          \fi}
291
```

`\@page` `\@page` marks the start of a new output page; its argument is the number of that page.

First we reset the visible line numbers, if we're numbering by page, and store the page number itself in a counter.

```
292 %\def\@page#1{\ifbypage@
293 \newcommand*{\@page}[1]{\ifbypage@
294              \line@num=0 \subline@num=0
295            \fi
296            \page@num=#1
```

And we set a flag that tells `\@l` that a new page number is to be set, because other associated actions shouldn't occur until the next line-start occurs.

```
297            \def\next@page@num{#1}}
298
```

`\sub@on` `\sub@off` The `\sub@on` and `\sub@off` macros turn sub-lineation on and off: but not directly, since such changes don't really take effect until the next line of text. Instead they set a flag that notifies `\@l` of the necessary action.

```
299 %\def\sub@on{\ifsublines@
300 \newcommand*{\sub@on}{\ifsublines@
301      \let\sub@change=\relax
302   \else
303      \def\sub@change{1}%
304   \fi}
305 %\def\sub@off{\ifsublines@
306 \newcommand*{\sub@off}{\ifsublines@
307      \def\sub@change{-1}%
308   \else
309      \let\sub@change=\relax
310   \fi}
311
```

`\@adv` The `\@adv` macro advances the current visible line number by the amount specified as its argument. This is used to implement `\advanceline`.

```
312 %\def\@adv#1{\ifsublines@
313 \newcommand*{\@adv}[1]{\ifsublines@
314        \advance\subline@num by #1
315        \ifnum\subline@num<0
316           \ledmac@warning{\string\advanceline\space produced
317               a sub-line number less than zero.}%
318           \subline@num=0
319        \fi
320   \else
321        \advance\line@num by #1
322        \ifnum\line@num<0
```

```
          \ledmac@warning{\string\advanceline\space produced
              a line number less than zero.}%
          \line@num=0
       \fi
  \fi
  \set@line@action}
```

`\@set` The `\@set` macro sets the current visible line number to the value specified as its argument. This is used to implement `\setline`.

```
%\def\@set#1{\ifsublines@
\newcommand*{\@set}[1]{\ifsublines@
    \subline@num=#1
  \else
    \line@num=#1
  \fi
  \set@line@action}
```

`\page@action` `\page@action` adds an entry to the action-code list to change the page number.

```
%\def\page@action{%
\newcommand*{\page@action}{%
  \xright@appenditem{\the\absline@num}\to\actionlines@list
  \xright@appenditem{\next@page@num}\to\actions@list}
```

`\set@line@action` `\set@line@action` adds an entry to the action-code list to change the visible line number.

```
%\def\set@line@action{%
\newcommand*{\set@line@action}{%
  \xright@appenditem{\the\absline@num}\to\actionlines@list
  \ifsublines@
       \@l@dtempcnta=-\subline@num
  \else
       \@l@dtempcnta=-\line@num
  \fi
  \advance\@l@dtempcnta by -5000
  \xright@appenditem{\the\@l@dtempcnta}\to\actions@list}
```

`\sub@action` `\sub@action` adds an entry to the action-code list to turn sub-lineation on or off, according to the current value of the `\ifsublines@` flag.

```
%\def\sub@action{%
\newcommand*{\sub@action}{%
  \xright@appenditem{\the\absline@num}\to\actionlines@list
  \ifsublines@
      \xright@appenditem{-1001}\to\actions@list
  \else
      \xright@appenditem{-1002}\to\actions@list
  \fi}
```

\lock@on
\do@lockon

\lock@on adds an entry to the action-code list to turn line number locking on. The current setting of the sub-lineation flag tells us whether this applies to line numbers or sub-line numbers.

Adding commands to the action list is slow, and it's very often the case that a lock-on command is immediately followed by a lock-off command in the line-list file, and therefore really does nothing. We use a look-ahead scheme here to detect such pairs, and add nothing to the line-list in those cases.

```
%\def\lock@on{\futurelet\next\do@lockon}
\newcommand*{\lock@on}{\futurelet\next\do@lockon}
%\def\do@lockon{%
\newcommand*{\do@lockon}{%
  \ifx\next\lock@off
     \global\let\lock@off=\skip@lockoff
  \else
     \xright@appenditem{\the\absline@num}\to\actionlines@list
     \ifsublines@
       \xright@appenditem{-1005}\to\actions@list
       \ifcase\sub@lock
          \sub@lock=1
       \else
          \sub@lock=0
       \fi
     \else
       \xright@appenditem{-1003}\to\actions@list
       \ifcase\@lock
          \@lock=1
       \else
          \@lock=0
       \fi
     \fi
  \fi}
```

\lock@off
\do@lockoff
\skip@lockoff

\lock@off adds an entry to the action-code list to turn line number locking off.

```
%\def\do@lockoff{%
\newcommand*{\do@lockoff}{%
  \xright@appenditem{\the\absline@num}\to\actionlines@list
  \ifsublines@
    \xright@appenditem{-1006}\to\actions@list
    \ifnum\sub@lock=2
       \sub@lock=3
    \else
       \sub@lock=0
    \fi
  \else
    \xright@appenditem{-1004}\to\actions@list
    \ifnum\@lock=2
       \@lock=3
    \else
       \@lock=0
```

```
    \fi
  \fi}
%\def\skip@lockoff{\global\let\lock@off=\do@lockoff}
\newcommand*{\skip@lockoff}{\global\let\lock@off=\do@lockoff}
\global\let\lock@off=\do@lockoff

```

\@ref
\insert@count

`\@ref` marks the start of a passage, for creation of a footnote reference. It takes two arguments:

- `#1`, the number of entries to add to `\insertlines@list` for this reference. This value, here and within `\text`, which computes it and writes it to the line-list file, will be stored in the `\insert@count` counter.

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

- `#2`, a sequence of other line-list-file commands, executed to determine the ending line-number. (This may also include other `\@ref` commands, corresponding to uses of `\text` within the first argument of another instance of `\text`.)

\dummy@ref

When nesting of `\@ref` commands does occur, it's necessary to temporarily redefine `\@ref` within `\@ref`, so that we're only doing one of these at a time.

```
%\def\dummy@ref#1#2{#2}
\newcommand*{\dummy@ref}[2]{#2}
```

The first thing `\@ref` itself does is to add the specified number of items to the `\insertlines@list` list.

```
%\def\@ref#1#2{%
\newcommand*{\@ref}[2]{%
  \global\insert@count=#1
  \loop\ifnum\insert@count>0
    \xright@appenditem{\the\absline@num}\to\insertlines@list
    \global\advance\insert@count by -1
  \repeat
```

Next, process the second argument to determine the page and line numbers for the end of this lemma. We temporarily equate `\@ref` to a different macro that just executes its argument, so that nested `\@ref` commands are just skipped this time. Some other macros need to be temporarily redefined to suppress their action.

```
  \begingroup
    \let\@ref=\dummy@ref
    \let\page@action=\relax
    \let\sub@action=\relax
    \let\set@line@action=\relax
    \let\@lab=\relax
    #2
    \global\endpage@num=\page@num
    \global\endline@num=\line@num
```

```
    \global\endsubline@num=\subline@num
  \endgroup
```

Now store all the information about the location of the lemma's start and end in `\line@list`.

```
    \xright@appenditem%
      {\the\page@num|\the\line@num|%
       \ifsublines@ \the\subline@num \else 0\fi|%
       \the\endpage@num|\the\endline@num|%
       \ifsublines@ \the\endsubline@num \else 0\fi}\to\line@list
```

Finally, execute the second argument of `\@ref` again, to perform for real all the commands within it.

```
  #2}

```

5.6 Writing to the line-list file

We've now defined all the counters, lists, and commands involved in reading the line-list file at the start of a section. Now we'll cover the commands that EDMAC uses within the text of a section to write commands out to the line-list.

`\linenum@out` The file will be opened on output stream `\linenum@out`.

```
\newwrite\linenum@out
```

`\iffirst@linenum@out@` `\first@linenum@out@true` `\first@linenum@out@false` Once any file is opened on this stream, we keep it open forever, or else switch to another file that we keep open. The reason is that we want the output routine to write the page number for every page to this file; otherwise we'd have to write it at the start of every line. But it's not very easy for the output routine to tell whether an output stream is open or not. There's no way to test the status of a particular output stream directly, and the asynchronous nature of output routines makes the status hard to determine by other means.

We can manage pretty well by means of the `\iffirst@linenum@out@` flag; its inelegant name suggests the nature of the problem that made its creation necessary. It's set to be `true` before any `\linenum@out` file is opened. When such a file is opened for the first time, it's done using `\immediate`, so that it will at once be safe for the output routine to write to it; we then set this flag to `false`.

```
\newif\iffirst@linenum@out@
  \first@linenum@out@true
```

`\line@list@stuff` The `\line@list@stuff` macro, which is called by `\beginnumbering`, performs all the line-list operations needed at the start of a section. Its argument is the name of the line-list file.

```
%\def\line@list@stuff#1{%
\newcommand*{\line@list@stuff}[1]{%
```

First, use the commands of the previous section to interpret the line-list file from the last run.

```
  \read@linelist{#1}%
```

Now close the current output line-list file, if any, and open a new one. The first time we open a line-list file for output, we do it using `\immediate`, and clear the `\iffirst@linenum@out@` flag.

```
\iffirst@linenum@out@
   \immediate\closeout\linenum@out
   \global\first@linenum@out@false
   \immediate\openout\linenum@out=#1
\else
```

If we get here, then this is not the first line-list we've seen, so we don't open or close the files immediately. We also need to insert a `\@page` command, since this might begin in the middle of a page.

```
   \closeout\linenum@out
   \openout\linenum@out=#1
   \page@start
\fi}

```

`\new@line` The `\new@line` macro sends the `\@l` command to the line-list file, to mark the start of a new text line.

```
%\def\new@line{\write\linenum@out{\string\@l}}
\newcommand*{\new@line}{\write\linenum@out{\string\@l}}
```

`\flag@start` `\flag@end` We enclose a lemma marked by `\text` in `\flag@start` and `\flag@end`: these send the `\@ref` command to the line-list file. `\text` is responsible for setting the value of `\insert@count` appropriately; it actually gets done by the various footnote macros.

```
%\def\flag@start{%
\newcommand*{\flag@start}{%
  \edef\next{\write\linenum@out{%
                  \string\@ref[\the\insert@count][}}%
  \next}
%\def\flag@end{\write\linenum@out{]}}
\newcommand*{\flag@end}{\write\linenum@out{]}}
```

`\page@start` `\page@start` writes a command to the line-list file noting the current page number. When used within an output routine, this should be called so as to place its `\write` within the box that gets shipped out, and as close to the top of that box as possible.

```
%\def\page@start{%
\newcommand*{\page@start}{%
  \iffirst@linenum@out@ \else
    \write\linenum@out{\string\@page[\the\pageno]}%
  \fi}

```

`\startsub` `\endsub` `\startsub` and `\endsub` turn sub-lineation on and off, by writing appropriate instructions to the line-list file. When sub-lineation is in effect, the line number counter is frozen and the sub-line counter advances instead. If one of these commands appears in the middle of a line, it doesn't take effect until the next line; in

other words, a line is counted as a line or sub-line depending on what it started out as, even if that changes in the middle.

We tinker with `\lastskip` because a command of either sort really needs to be attached to the last word preceding the change, not the first word that follows the change. This is because sub-lineation will often turn on and off in mid-line—stage directions, for example, often are mixed with dialogue in that way—and when a line is mixed we want to label it using the system that was in effect at its start. But when sub-lineation begins at the very start of a line we have a problem, if we don't put in this code.

```
%\def\startsub{\dimen0\lastskip
\newcommand*{\startsub}{\dimen0\lastskip
  \ifdim\dimen0>0pt \unskip \fi
  \write\linenum@out{\string\sub@on}%
  \ifdim\dimen0>0pt \hskip\dimen0 \fi}
\def\endsub{\dimen0\lastskip
  \ifdim\dimen0>0pt \unskip \fi
  \write\linenum@out{\string\sub@off}%
  \ifdim\dimen0>0pt \hskip\dimen0 \fi}

```

`\advanceline` You can use `\advanceline` in running text to advance the current visible line-number by a specified value, positive or negative.

```
%\def\advanceline#1{\write\linenum@out{\string\@adv[#1]}}
\newcommand*{\advanceline}[1]{\write\linenum@out{\string\@adv[#1]}}

```

`\setline` You can use `\setline` in running text to set the current visible line-number to a specified positive value.

```
%\def\setline#1{%
\newcommand*{\setline}[1]{%
  \ifnum#1<0
    \ledmac@warning{Bad setline argument.}%
  \else
    \write\linenum@out{\string\@set[#1]}%
  \fi}

```

`\startlock` `\endlock` You can use `\startlock` or `\endlock` in running text to start or end line number locking at the current line. They decide whether line numbers or sub-line numbers are affected, depending on the current state of the sub-lineation flags.

```
%\def\startlock{\write\linenum@out{\string\lock@on}}
\newcommand*{\startlock}{\write\linenum@out{\string\lock@on}}
\def\endlock{\write\linenum@out{\string\lock@off}}

```

6 Marking text for notes

The `\text` macro is used to create all footnotes and endnotes, as well as to print the portion of the main text to which a given note or notes is keyed. The idea is

to have that lemma appear only once in the `.tex` file: all instances of it in the main text and in the notes are copied from that one appearance.

`\text` requires two arguments. At any point within numbered text, you use it by saying:

```
\text{#1}{#2}/
```

where

- `#1` is the piece of the main text being glossed; it gets added to the main text, and is also used as a lemma for notes to it.
- `#2` is a series of subsidiary macros that generate various kinds of notes. The `/` after `#2` *must* appear: it marks the end of the macro. (*The TEXbook*, p. 204, points out that when additional text to be matched follows the arguments like this, spaces following the macro are not skipped, which is very desirable since this macro will never be used except within text. Having an explicit terminator also helps keep things straight when nested calls to `\text` are used.) Braces around `#2` are optional.

The `\text` macro may be used (somewhat) recursively; that is, `\text` may be used within its own first argument. The code would be much simpler without this feature, but nested notes will commonly be necessary: it's quite likely that we'll have an explanatory note for a long passage and notes on variants for individual words within that passage. The situation we can't handle is overlapping notes that aren't nested: for example, one note covering lines 10–15, and another covering 12–18. You can handle such cases by using the `\lemma` and `\linenum` macros within `#2`: they alter the copy of the lemma and the line numbers that are passed to the notes, and hence allow you to overcome any limitations of this system, albeit with extra effort.

The recursive operation of `\text` will fail if you try to use a copy that is called something other than `\text`. In order to handle recursion, `\text` needs to redefine its own definition temporarily at one point, and that doesn't work if the macro you are calling is not actually named `\text`. There's no problem as long as `\text` is not invoked in the first argument. If you want to call `\text` something else, it is best to create instead a macro that expands to an invocation of `\text`, rather than copying `\text` and giving it a new name; otherwise you will need to add an appropriate definition for your new macro to `\morenoexpands`.

Side effects of our line-numbering code make it impossible to use the usual footnote macros directly within a paragraph whose lines are numbered (see comments to `\do@line`, p. 49). Instead, the appropriate note-generating command is appended to the list macro `\inserts@list`, and when `\pend` completes the paragraph it inserts all the notes at the proper places.

Note that we don't provide previous-note information, although it's often wanted; your own macros must handle that. We can't do it correctly without keeping track of what kind of notes have gone past: it's not just a matter of remembering the line numbers associated with the previous invocation of `\text`,

because that might have been for a different kind of note. It is preferable for your footnote macros to store and recall this kind of information if they need it.

An example where some "memory" of line numbers might be required is where there are several variant readings per line of text, and you do not wish the line number to be repeated for each lemma in the notes. After the first occurrence of the line number, you might want the symbol "‖" instead of further occurrences, for instance. This can easily be done by a macro like `\printlines`, if it saves the last value of `\@nums` that *it* saw, and then performs a simple conditional test to see whether to print a number or a "‖".

6.1 \text itself

`\end@lemmas` The various note-generating macros might want to request that commands be executed not at once, but in close connection with the start or end of the lemma. For example, footnote numbers in the text should be connected to the end of the lemma; or, instead of a single macro to create a note listing variants, you might want to use several macros in series to create individual variants, which would each add information to a private macro or token register, which in turn would be formatted and output when all of `#2` for the lemma has been read.

To accomodate this, we provide a list macro to which macros may add commands that should subsequently be executed at the end of the lemma when that lemma is added to the text of the paragraph. A macro should add its contribution to `\end@lemmas` by using `\xleft@appenditem`. (Anything that needs to be done at the *start* of the lemma may be handled using `\aftergroup`, since the commands specified within `\text`'s second argument are executed within a group that ends just before the lemma is added to the main text.)

`\end@lemmas` is intended for the few things that need to be associated with the end of the lemma, like footnote numbers. Such numbers are not implemented in the current version, and indeed no use is currently made of `\end@lemmas` or of the `\aftergroup` trick. The general approach would be to define a macro to be used within the second argument of `\text` that would add the appropriate command to `\end@lemmas`.

Commands that are added to this list should always take care not to do anything that adds possible line-breaks to the output; otherwise line numbering could be thrown off.

```
489 \list@create{\end@lemmas}
```

`\dummy@text` We now need to define a number of macros that allow us to weed out nested instances of `\text`, and other problematic macros, from our lemma. This is similar to what we did in reading the line-list file using `\dummy@ref` and various redefinitions—and that's because nested `\text` macros create nested `\@ref` entries in the line-list file.

Here's a macro that takes the same arguments as `\text` but merely returns the first argument and ignores the second.

```
490 \long\def\dummy@text#1#2/{#1}
```

\dummy@edtext LaTeX users are not used to delimeted arguments, so I provide a \edtext macro as well.

```
\newcommand{\dummy@edtext}[2]{#1}
```

\@gobble We're going to need another macro that takes one argument and ignores it entirely. This one is identical to the LaTeX macro of the same name. Therefore I don't define it.

```
%\def\@gobble#1{}
```

\no@expands \morenoexpands We need to turn off macro expansion for certain sorts of macros we're likely to see within the lemma and within the notes.

The first class is font-changing macros. We suppress expansion for them by letting them become equal to zero.[20] This is done because we want to pass into our notes the generic commands to change to roman or whatever, and not their expansions that will ask for a particular style at a specified size. The notes may well be in a smaller font, so the command should be expanded later, when the note's environment is in effect.

A second sort to turn off includes a few of the accent macros. Most are not a problem: an accent that's expanded to an `\accent` command may be harder to read but it works just the same. The ones that cause problems are: those that use alignments—TeX seems to get confused about the difference between alignment parameters and macro parameters; those that use temporary control sequences; and those that look carefully at what the current font is.

(The `\copyright` macro defined in Plain TeX has this sort of problem as well, but isn't used enough to bother with. That macro, and any other that causes trouble, will get by all right if you put a `\noexpand` in front of it in your file—or a `\protect` if you're using NFSS.)

We also need to eliminate all `EDMAC` macros like `\label` and `\setline` that write things to auxiliary files: that writing should be done only once. And we make `\text` itself, if it appears within its own argument, do nothing but copy its first argument.

Finally, we execute `\morenoexpands`. The version of `\morenoexpands` defined here does nothing; but you may define a version of your own when you need to add more expansion suppressions as needed with your macros. That makes it possible to make such additions without needing to copy or modify the standard `EDMAC` code. If you define your own `\morenoexpands`, you must be very careful about spaces: if the macro adds any spaces to the text when it runs, extra space will appear in the main text when `\text` is used.

(A related problem, not addressed by these two macros, is that of characters whose category code is changed by any the macros used in the arguments to `\text`. Since the category codes are set when the arguments are scanned, macros that depend on changing them will not work. We have most often encountered this with characters that are made "active" within text in some, but not all, of the languages used within the document. One way around the problem, if it takes

[20] Since "control sequences equivalent to characters are not expandable"—*The TeXbook*, answer to Exercise 20.14.

this form, is to ensure that those characters are *always* active; within languages that make no special use of them, their associated control sequences should simply return the proper character.)

```
%\def\no@expands{\let\rm=0\let\it=0\let\sl=0\let\bf=0\let\tt=0%
\newcommand*{\no@expands}{\let\rm=0\let\it=0\let\sl=0\let\bf=0\let\tt=0%
  \let\b=0\let\c=0\let\d=0\let\t=0%
  \let\select@@lemmafont=0%
  \def\protect{\noexpand\protect\noexpand}%
  \let\startsub=\relax  \let\endsub=\relax
  \let\startlock=\relax \let\endlock=\relax
  \let\edlabel=\@gobble   \let\edpageref=\@gobble
  \let\lineref=\@gobble \let\sublineref=\@gobble
  \let\setline=\@gobble \let\advanceline=\@gobble
  \let\text=\dummy@text
  \let\edtext=\dummy@edtext
  \morenoexpands}
\let\morenoexpands=\relax

```

`\text` Now we begin `\text` itself. The definition requires a `/` after the arguments: this eliminates the possibility of problems about knowing where `#2` ends. This also changes the handling of spaces following an invocation of the macro: normally such spaces are skipped, but in this case they're significant because `#2` is a "delimited parameter". Since `\text` is always used in running text, it seems more appropriate to pay attention to spaces than to skip them.

When executed, `\text` first ensures that we're in horizontal mode.

```
\long\def\text#1#2/{\leavevmode
```

`\@tag` Our normal lemma is just argument `#1`; but that argument could have further invocations of `\text` within it. We get a copy of the lemma without any `\text` macros within it by temporarily redefining `\text` to just copy its first argument and ignore the other, and then expand `#1` into `\@tag`, our lemma.

This is done within a group that starts here, in order to get the original `\text` restored; within this group we've also turned off the expansion of those control sequences commonly found within text that can cause trouble for us.

```
  \begingroup
    \no@expands
    \xdef\@tag{#1}%
```

`\@nums` Prepare more data for the benefit of note-generating macros: the line references and font specifier for this lemma go to `\@nums`.

```
    \set@line
```

`\insert@count` will be altered by the note-generating macros: it counts the number of deferred footnotes or other insertions generated by this instance of `\text`.

```
    \global\insert@count=0
```

Now process the note-generating macros in argument #2 (i.e., \Afootnote, \lemma, etc.). \ignorespaces is here to skip over any spaces that might appear at the start of #2; otherwise they wind up in the main text. Footnote and other macros that are used within #2 should all end with \ignorespaces as well, to skip any spaces between macros when several are used in series.

```
    \ignorespaces #2\relax
```

Finally, we're ready to admit the first argument into the current paragraph.

It's important that we generate and output all the notes for this chunk of text *before* putting the text into the paragraph: notes that are referenced by line number should generally be tied to the start of the passage they gloss, not the end. That should all be done within the expansion of #2 above, or in \aftergroup commands within that expansion.

```
    \flag@start
  \endgroup
  #1%
```

Finally, we add any insertions that are associated with the *end* of the lemma. Footnotes that are identified by symbols rather than by where the lemma begins in the main text need to be done here, and not above.

```
  \ifx\end@lemmas\empty \else
    \gl@p\end@lemmas\to\x@lemma
    \x@lemma
    \global\let\x@lemma=\relax
  \fi
  \flag@end}
```

Here's the promised undelimeted LaTeX version of \text.

\edtext

```
\newcommand{\edtext}[2]{\leavevmode
  \begingroup
    \no@expands
    \xdef\@tag{#1}%
    \set@line
    \global\insert@count=0
    \ignorespaces #2\relax
    \flag@start
  \endgroup
  #1%
  \ifx\end@lemmas\empty \else
    \gl@p\end@lemmas\to\x@lemma
    \x@lemma
    \global\let\x@lemma=\relax
  \fi
  \flag@end}

```

\set@line The \set@line macro is called by \text to put the line-reference field and font specifier for the current block of text into \@nums.

One instance of \text may generate several notes, or it may generate none—it's legitimate for argument #2 to \text to be empty. But \flag@start and \flag@end induce the generation of a single entry in \line@list during the next run, and it's vital to also remove one and only one \line@list entry here.

```
%\def\set@line{%
\newcommand*{\set@line}{%
```

If no more lines are listed in \line@list, something's wrong—probably just some change in the input. We set all the numbers to zeros, following an old publishing convention for numerical references that haven't yet been resolved.

```
  \ifx\line@list\empty
    \global\noteschanged@true
    \xdef\@nums{000|000|000|000|000|000|\edfont@info}%
```

All's well; our reference is there.

```
 \else
   \gl@p\line@list\to\@tempb
   \xdef\@nums{\@tempb|\edfont@info}%
   \global\let\@tempb=\undefined
 \fi}

```

\edfont@info The macro \edfont@info returns coded information about the current font; the coding depends on the font selection scheme in use. See section 8.1 for more on font selection schemes.

```
%\ifx\selectfont\undefined       % we're using Plain fonts
%  \def\edfont@info{\the\fam}
%\else                           % we're using NFSS
%  \def\edfont@info{\f@encoding/\f@family/\f@series/\f@shape}
  \newcommand*{\edfont@info}{\f@encoding/\f@family/\f@series/\f@shape}
%\fi

```

6.2 Substitute lemma

\lemma The \lemma macro allows you to change the lemma that's passed on to the notes.

```
%\def\lemma#1{\xdef\@tag{#1}\ignorespaces}
\newcommand*{\lemma}[1]{\xdef\@tag{#1}\ignorespaces}
```

6.3 Substitute line numbers

\linenum The \linenum macro can change any or all of the page and line numbers that are passed on to the notes.

As argument \linenum takes a set of seven parameters separated by vertical bars, in the format used internally for \@nums (see p. 27): the starting page, line, and sub-line numbers, followed by the ending page, line, and sub-line numbers, and then the font specifier for the lemma. However, you can omit any parameters you don't want to change, and you can omit a string of vertical bars at the end of

the argument. Hence `\linenum{18|4|0|18|7|1|0}` is an invocation that changes all the parameters, but `\linenum{|3}` only changes the starting line number, and leaves the rest unaltered.

We use `\\` as an internal separator for the macro parameters.

```
%\def\linenum#1{%
\newcommand*{\linenum}[1]{%
  \xdef\@tempa{#1|||||||\noexpand\\\@nums}%
  \global\let\@nums=\empty
  \expandafter\line@set\@tempa|\\\ignorespaces}
```

`\line@set` `\linenum` calls `\line@set` to do the actual work; it looks at the first number in the argument to `\linenum`, sets the corresponding value in `\@nums`, and then calls itself to process the next number in the `\linenum` argument, if there are more numbers in `\@nums` to process.

```
\def\line@set#1|#2\\#3|#4\\{%
  \gdef\@tempb{#1}%
  \ifx\@tempb\empty
        \@add{#3}%
  \else
        \@add{#1}%
  \fi
  \gdef\@tempb{#4}%
  \ifx\@tempb\empty\else
      \@add{|}\line@set#2\\#4\\%
  \fi}
```

`\@add` `\line@set` uses `\@add` to tack numbers or vertical bars onto the right hand end of `\@nums`.

```
%\def\@add#1{\xdef\@nums{\@nums#1}}
\newcommand{\@add}[1]{\xdef\@nums{\@nums#1}}

```

7 Paragraph decomposition and reassembly

In order to be able to count the lines of text and affix line numbers, we add an extra stage of processing for each paragraph. We send the paragraph into a box register, rather than straight onto the vertical list, and when the paragraph ends we slice the paragraph into its component lines; to each line we add any notes or line numbers, add a command to write to the line-list, and then at last send the line to the vertical list. This section contains all the code for this processing.

7.1 Boxes, counters, `\pstart` and `\pend`

`\raw@text` `\ifnumberedpar@` `\numberedpar@true` `\numberedpar@false` `\num@lines` `\one@line` `\par@line`

Here are numbers and flags that are used internally in the course of the paragraph decomposition.

When we first form the paragraph, it goes into a box register, `\raw@text`, instead of onto the current vertical list. The `\ifnumberedpar@` flag will be `true`

while a paragraph is being processed in that way. \num@lines will store the number of lines in the paragraph when it's complete. When we chop it up into lines, each line in turn goes into the \one@line register, and \par@line will be the number of that line within the paragraph.

```
\newbox\raw@text
\newif\ifnumberedpar@
\newcount\num@lines
\newbox\one@line
\newcount\par@line
```

\pstart \pstart starts the paragraph by clearing the \inserts@list list and other relevant variables, and then arranges for the subsequent text to go into the \raw@text box. \pstart needs to appear at the start of every paragraph that's to be numbered; the \autopar command below may be used to insert these commands automatically.

Beware: everything that occurs between \pstart and \pend is happening within a group; definitions must be global if you want them to survive past the end of the paragraph.

```
%\def\pstart{\ifnumbering \else
\newcommand*{\pstart}{\ifnumbering \else
    \errmessage{\string\pstart\space must be used
                within a numbered section}%
    \beginnumbering
  \fi
  \ifnumberedpar@
    \errmessage{\string\pstart\space encountered while another
                \string\pstart\space was in effect}%
    \pend
  \fi
  \list@clear{\inserts@list}%
  \global\let\next@insert=\empty
  \begingroup\normal@pars
  \global\setbox\raw@text=\vbox\bgroup
  \numberedpar@true}

```

\pend \pend must be used to end a numbered paragraph.

```
%\def\pend{\ifnumbering \else
\newcommand*{\pend}{\ifnumbering \else
  \errmessage{\string\pend\space must be used
                within a numbered section}%
  \fi
  \ifnumberedpar@ \else
    \errmessage{\string\pend\space must follow a \string\pstart}%
  \fi
```

We set all the usual interline penalties to zero and then immediately call \endgraf to end the paragraph; this ensures that there'll be no large interline penalties to prevent us from slicing the paragraph into pieces. These penalties revert to the

values that you set when the group for the \vbox ends. Then we call \do@line to slice a line off the top of the paragraph, add a line number and footnotes, and restore it to the page; we keep doing this until there aren't any more lines left.

```
\brokenpenalty=0 \clubpenalty=0
\displaywidowpenalty=0 \interlinepenalty=0 \predisplaypenalty=0
\postdisplaypenalty=0 \widowpenalty=0
\endgraf\global\num@lines=\prevgraf\egroup
\global\par@line=0
\loop\ifvbox\raw@text
   \do@line
\repeat
```

Deal with any leftover notes, and then end the group that was begun in the \pstart.

```
\flush@notes
\endgroup
\ignorespaces}

```

\autopar In most cases it's only an annoyance to have to label the paragraphs to be numbered with \pstart and \pend. \autopar will do that automatically, allowing you to start a paragraph with its first word and no other preliminaries, and to end it with a blank line or a \par command. The command should be issued within a group, after \beginnumbering has been used to start the numbering; all paragraphs within the group will be affected.

A few situations can cause problems. One is a paragraph that begins with a begin-group character or command: \pstart will not get invoked until after such a group beginning is processed; as a result the character that ends the group will be mistaken for the end of the \vbox that \pstart creates, and the rest of the paragraph will not be numbered. Such paragraphs need to be started explicitly using \indent, \noindent, or \leavevmode—or \pstart, since you can still include your own \pstart and \pend commands even with \autopar on.

Prematurely ending the group within which \autopar is in effect will cause a similar problem. You must either leave a blank line or use \par to end the last paragraph before you end the group.

The functioning of this macro is more tricky than the usual \everypar: we don't want anything to go onto the vertical list at all, so we have to end the paragraph, erase any evidence that it ever existed, and start it again using \pstart. We remove the paragraph-indentation box using \lastbox and save the width, and then skip backwards over the \parskip that's been added for this paragraph. Then we start again with \pstart, restoring the indentation that we saved, and locally change \par so that it'll do our \pend for us.

```
%\def\autopar{\ifnumbering \else
\newcommand*{\autopar}{\ifnumbering \else
   \errmessage{\string\autopar\space must be used
                  within a numbered section}%
   \beginnumbering
\fi
```

```
\everypar={\setbox0=\lastbox
  \endgraf \vskip-\parskip
  \pstart \noindent \kern\wd0
  \let\par=\pend}%
\ignorespaces}
```

`\normal@pars` We also define a macro which we can rely on to turn off the `\autopar` definitions at various important places, if they are in force. We'll want to do this within footnotes, for example.

```
%\def\normal@pars{\everypar={}\let\par\endgraf}
\newcommand*{\normal@pars}{\everypar={}\let\par\endgraf}

```

7.2 Processing one line

`\do@line` The `\do@line` macro is called by `\pend` to do all the processing for a single line of text.

```
%\def\do@line{%
\newcommand*{\do@line}{%
```

First, pull one line off the top of `\raw@text`, which contains the remaining unprocessed lines of the paragraph. `\vbadness` must be cranked up to suppress `Underfull vbox` errors from `\vsplit`; `\splittopskip` will be inserted at the top of `\one@line`, so we zero it. (This skip will appear in the final vertical list, just before every `\baselineskip`.)

```
{\vbadness=10000 \splittopskip=0pt
\global\setbox\one@line=\vsplit\raw@text to\baselineskip}%
```

`\one@line` comes out of `\vsplit` as a `vbox`; we now convert it to an `hbox`.

This operation breaks if there's an insert connected to the line. In that case, the content of the `vbox` `\one@line` before this operation is not just an `hbox`: it's an `hbox` followed by an insert. After the `\unvbox`, the last thing on the vertical list is not the `hbox` but the insert. The result is that our line heads prematurely onto the vertical list—with incorrect interline spacing, because there's still a level of boxing that should be undone—and `\one@line` is the void box, because the last thing on the vertical list wasn't a box. The subsequent code consequently prints a blank line.

All this is why insertions need to be kept out of the paragraph until this point; our footnote macros add all insertions to list macros, and the `\add@inserts` macro below puts them onto the vertical list at the proper time.

```
\unvbox\one@line \global\setbox\one@line=\lastbox
```

Calculate the line and page number for this line.

```
\getline@num
```

Now we'll add the line to the vertical list, with a line number attached if necessary.

The `\hfil\hbox to \wd\one@line` is necessary to position a hangindented line correctly: without it, `\one@line` gets stretched out to `\hsize` in width and

the indentation disappears. This is because hanging indentation is done by setting a nonzero "shift" value for the `hbox` that contains the line within the `vbox`, and that shift vanishes, like the penalties, when we slice up the paragraph; one can examine the `\ht` or `\wd` of a box within TEX, but it provides no way of examining the `\shift`, though it would be a trivial modification of the TEX program to add that function. (`\parshape` also works by setting a nonzero shift, but this fix isn't good enough there, because the total width of the lines is also varied in that case; our algorithm will push all the lines of text over to the right margin.)

We put the `\new@line` start-of-line marker in the output list at this point too: putting it within the `\hbox` here ensures that it comes before any of the text of the line in the vertical list, but cannot be broken away from it at a page break.

```
\hbox to \hsize{\affixline@num{%
  \hfil\hbox to \wd\one@line{\new@line\unhbox\one@line}}}%
```

Now we pull the footnotes and insertions for this line out of the `\inserts@list` list macro and attach them.

```
\add@inserts
```

Penalties get stripped off by this slicing process; the following macro puts them back in as the last step.

```
\add@penalties}

```

7.3 Line and page number computation

`\getline@num` The `\getline@num` macro determines the page and line numbers for the line we're about to send to the vertical list.

```
%\def\getline@num{%
\newcommand*{\getline@num}{%
  \global\advance\absline@num by 1
  \do@actions
  \do@ballast
  \ifsublines@
     \ifnum\sub@lock<2
       \global\advance\subline@num by 1
     \fi
  \else
     \ifnum\@lock<2
       \global\advance\line@num by 1
       \global\subline@num=0
     \fi
  \fi}

```

`\do@ballast` The real work in the macro above is done in `\do@actions`, but before we plunge into that, let's get `\do@ballast` out of the way. This macro looks to see if there is an action to be performed on the *next* line, and if it is going to be a page break action, `\do@ballast` decreases the `\ballast@count` counter by the amount of

\ballast. This means, in practice, that when \add@penalties assigns penalties at this point, TEX will be given extra encouragement to break the page here (see p. 57).

\ballast
\ballast@count

First we set up the required counters; they are initially set to zero, and will remain so unless you say \ballast=⟨*some figure*⟩ in your document.

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

And here is \do@ballast itself. It advances \absline@num within the protection of a group to make its check for what happens on the next line.

```
%\def\do@ballast{\global\ballast@count=0
\newcommand*{\do@ballast}{\global\ballast@count=0
  \begingroup
    \advance\absline@num by 1
    \ifnum\next@actionline=\absline@num
      \ifnum\next@action>-1001
        \global\advance\ballast@count by -\ballast
       \fi
     \fi
  \endgroup}
```

\do@actions
\do@actions@next

The \do@actions macro looks at the list of actions to take at particular absolute line numbers, and does everything that's specified for the current line.

It may call itself recursively, and to do this efficiently (using TEX's optimization for tail recursion), we define a control-sequence called \do@actions@next that is always the last thing that \do@actions does. If there could be more actions to process for this line, \do@actions@next is set equal to \do@actions; otherwise it's just \relax.

```
%\def\do@actions{%
\newcommand*{\do@actions}{%
  \global\let\do@actions@next=\relax
  \ifnum\absline@num<\next@actionline\else
```

First, page number changes, which will generally be the most common actions. If we're restarting lineation on each page, this is where it happens.

```
    \ifnum\next@action>-1001
       \global\page@num=\next@action
       \ifbypage@
             \global\line@num=0 \global\subline@num=0
       \fi
```

Next, we handle commands that change the line-number values. (We subtract 5001 rather than 5000 here because the line number is going to be incremented automatically in \getline@num.)

```
    \else
       \ifnum\next@action<-4999
          \@l@dtempcnta=-\next@action
          \advance\@l@dtempcnta by -5001
```

```
688          \ifsublines@
689             \global\subline@num=\@l@dtempcnta
690          \else
691             \global\line@num=\@l@dtempcnta
692          \fi
```

It's one of the fixed codes. We rescale the value in `\@l@dtempcnta` so that we can use a case statement.

```
693       \else
694          \@l@dtempcnta=-\next@action
695          \advance\@l@dtempcnta by -1000
696          \ifcase\@l@dtempcnta
```

Commands that turn sub-lineation on and off.

```
697          \or
698             \global\sublines@true
699          \or
700             \global\sublines@false
```

Line locking. We ignore these indications when they don't appear at the right times: a start-lock should appear only when locking is entirely off, and an end-lock should only appear when locking is in the "middle".

```
701          \or
702             \ifcase\@lock
703               \global\@lock=1
704             \else
705               \global\@lock=0
706             \fi
707          \or
708             \ifnum\@lock=2
709               \global\@lock=3
710             \else
711               \global\@lock=0
712             \fi
```

Sub-line locking. Same comments as for line locking.

```
713          \or
714             \ifcase\sub@lock
715               \global\sub@lock=1
716             \else
717               \global\sub@lock=0
718             \fi
719          \or
720             \ifnum\sub@lock=2
721               \global\sub@lock=3
722             \else
723               \global\sub@lock=0
724             \fi
```

If we get here, some unknown action code has been encountered.

```
725          \else
```

```
            \ledmac@warning{Bad action code,
                                value \next@action.}%
          \fi
       \fi
    \fi
```

Now we get information about the next action off the list, and then set `\add@inserts@next` so that we'll call ourself recursively: the next action might also be for this line.

There's no warning if we find `\actionlines@list` empty, since that will always happen near the end of the section.

```
    \ifx\actionlines@list\empty
         \gdef\next@actionline{1000000}%
    \else
         \gl@p\actionlines@list\to\next@actionline
         \gl@p\actions@list\to\next@action
         \global\let\do@actions@next=\do@actions
    \fi
  \fi
```

Make the recursive call, if necessary.

```
\do@actions@next}

```

7.4 Line number printing

`\affixline@num` `\affixline@num` takes a single argument, a series of commands for printing the line just split off by `\do@line`; it puts that line back on the vertical list, and adds a line number if necessary.

To determine whether we need to affix a line number to this line, we compute the following:

$$n = int((linenum - firstlinenum)/linenumincrement)$$
$$m = firstlinenum + (n \times linenumincrement)$$

(where *int* truncates a real number to an integer). m will be equal to *linenum* only if we're to paste a number on here. However, the formula breaks down for the first line to number (and any before that), so we check that case separately: if `\line@num` $\leq$ `\firstlinenum`, we compare the two directly instead of making these calculations.

We compute, in the scratch counter `\@l@dtempcnta`, the number of the next line that should be printed with a number (m in the above discussion), and move the current line number into the counter `\@l@dtempcntb` for comparison.

Remember that some counts are now counters!

First, the case when we're within a sub-line range.

```
%\def\affixline@num#1{%
\newcommand*{\affixline@num}[1]{%
  \ifsublines@
```

```
%    \@l@dtempcntb=\subline@num
%    \ifnum\subline@num>\firstsublinenum
%      \@l@dtempcnta=\subline@num
%      \advance\@l@dtempcnta by-\firstsublinenum
%      \divide\@l@dtempcnta by\sublinenumincrement
%      \multiply\@l@dtempcnta by\sublinenumincrement
%      \advance\@l@dtempcnta by\firstsublinenum
%    \else
%      \@l@dtempcnta=\firstsublinenum
%    \fi
    \@l@dtempcntb=\subline@num
    \ifnum\subline@num>\c@firstsublinenum
      \@l@dtempcnta=\subline@num
      \advance\@l@dtempcnta by-\c@firstsublinenum
      \divide\@l@dtempcnta by\c@sublinenumincrement
      \multiply\@l@dtempcnta by\c@sublinenumincrement
      \advance\@l@dtempcnta by\c@firstsublinenum
    \else
      \@l@dtempcnta=\c@firstsublinenum
    \fi
```

That takes care of computing the values for comparison, but if line number locking is in effect we have to make a further check. If this check fails, then we disable the line-number display by setting the counters to arbitrary but unequal values.

```
    \ifcase\sub@lock
      \or
        \ifnum\sublock@disp=1
           \@l@dtempcntb=0 \@l@dtempcnta=1
        \fi
      \or
        \ifnum\sublock@disp=2 \else
           \@l@dtempcntb=0 \@l@dtempcnta=1
        \fi
      \or
        \ifnum\sublock@disp=0
           \@l@dtempcntb=0 \@l@dtempcnta=1
        \fi
    \fi
```

Now the line number case, which works the same way.

```
  \else
    \@l@dtempcntb=\line@num
%    \ifnum\line@num>\firstlinenum
%       \@l@dtempcnta=\line@num
%       \advance\@l@dtempcnta by-\firstlinenum
%       \divide\@l@dtempcnta by\linenumincrement
%       \multiply\@l@dtempcnta by\linenumincrement
%       \advance\@l@dtempcnta by\firstlinenum
%    \else
```

```
%       \@l@dtempcnta=\firstlinenum
%    \fi
    \ifnum\line@num>\c@firstlinenum
       \@l@dtempcnta=\line@num
       \advance\@l@dtempcnta by-\c@firstlinenum
       \divide\@l@dtempcnta by\c@linenumincrement
       \multiply\@l@dtempcnta by\c@linenumincrement
       \advance\@l@dtempcnta by\c@firstlinenum
    \else
       \@l@dtempcnta=\c@firstlinenum
    \fi
```

A locking check for sub-lines, just like the version for line numbers above.

```
    \ifcase\@lock
       \or
         \ifnum\lock@disp=1
            \@l@dtempcntb=0 \@l@dtempcnta=1
         \fi
       \or
         \ifnum\lock@disp=2 \else
            \@l@dtempcntb=0 \@l@dtempcnta=1
         \fi
       \or
         \ifnum\lock@disp=0
            \@l@dtempcntb=0 \@l@dtempcnta=1
         \fi
    \fi
  \fi
```

The following test is true if we need to print a line number.

```
  \ifnum\@l@dtempcnta=\@l@dtempcntb
```

If we got here, we're going to print a line number; so now we need to calculate a number that will tell us which side of the page will get the line number. We start from `\line@margin`, which asks for one side always if it's less than 2; and then if the side does depend on the page number, we simply add the page number to this side code—because the values of `\line@margin` have been devised so that this produces a number that's even for left-margin numbers and odd for right-margin numbers.

```
    \@l@dtempcntb=\line@margin
    \ifnum\@l@dtempcntb>1
      \advance\@l@dtempcntb by\page@num
    \fi
```

Now print the line (`#1`) with its page number.

```
    \ifodd\@l@dtempcntb
      #1\rlap{{\rightlinenum}}%
    \else
      \llap{{\leftlinenum}}#1%
    \fi
  \else
```

If no line number is to be appended, we just print the line as is.

```
    #1%
  \fi
```

Now fix the lock counters, if necessary. A value of 1 is advanced to 2; 3 advances to 0; other values are unchanged.

```
  \ifcase\@lock
  \or
    \global\@lock=2
  \or \or
    \global\@lock=0
  \fi
  \ifcase\sub@lock
  \or
    \global\sub@lock=2
  \or \or
    \global\sub@lock=0
  \fi}

```

7.5 Add insertions to the vertical list

`\inserts@list` `\inserts@list` is the list macro that contains the inserts that we save up for one paragraph.

```
\list@create{\inserts@list}
```

`\add@inserts` `\add@inserts@next` `\add@inserts` is the penultimate macro used by `\do@line`; it takes insertions saved in a list macro and sends them onto the vertical list.

It may call itself recursively, and to do this efficiently (using TEX's optimization for tail recursion), we define a control-sequence called `\add@inserts@next` that is always the last thing that `\add@inserts` does. If there could be more inserts to process for this line, `\add@inserts@next` is set equal to `\add@inserts`; otherwise it's just `\relax`.

```
%\def\add@inserts{%
\newcommand*{\add@inserts}{%
  \global\let\add@inserts@next=\relax
```

If `\inserts@list` is empty, there aren't any more notes or insertions for this paragraph, and we needn't waste our time.

```
  \ifx\inserts@list\empty \else
```

The `\next@insert` macro records the number of the line that receives the next footnote or other insert; it's empty when we start out, and just after we've affixed a note or insert.

```
  \ifx\next@insert\empty
    \ifx\insertlines@list\empty
      \global\noteschanged@true
      \gdef\next@insert{100000}%
    \else
```

```
      \gl@p\insertlines@list\to\next@insert
    \fi
  \fi
```

If the next insert's for this line, tack it on (and then erase the contents of the insert macro, as it could be quite large). In that case, we also set `\add@inserts@next` so that we'll call ourself recursively: there might be another insert for this same line.

```
  \ifnum\next@insert=\absline@num
    \gl@p\inserts@list\to\@insert
    \@insert
    \global\let\@insert=\undefined
    \global\let\next@insert=\empty
    \global\let\add@inserts@next=\add@inserts
  \fi
\fi
```

Make the recursive call, if necessary.

```
\add@inserts@next}

```

7.6 Penalties

`\add@penalties` `\add@penalties` is the last macro used by `\do@line`. It adds up the club, widow, and interline penalties, and puts a single penalty of the appropriate size back into the paragraph; these penalties get removed by the `\vsplit` operation. `\displaywidowpenalty` and `\brokenpenalty` are not restored, since we have no easy way to find out where we should insert them.

In this code, `\num@lines` is the number of lines in the whole paragraph, and `\par@line` is the line we're working on at the moment. The counter `\@l@dtempcnta` is used to calculate and accumulate the penalty; it is initially set to the value of `\ballast@count`, which has been worked out in `\do@ballast` above (p. 50). Finally, the penalty is checked to see that it doesn't go below −10000.

```
%\def\add@penalties{\@l@dtempcnta=\ballast@count
\newcommand*{\add@penalties}{\@l@dtempcnta=\ballast@count
  \ifnum\num@lines>1
    \global\advance\par@line by 1
    \ifnum\par@line=1
      \advance\@l@dtempcnta by \clubpenalty
    \fi
    \@l@dtempcntb=\par@line \advance\@l@dtempcntb by 1
    \ifnum\@l@dtempcntb=\num@lines
      \advance\@l@dtempcnta by \widowpenalty
    \fi
    \ifnum\par@line<\num@lines
      \advance\@l@dtempcnta by \interlinepenalty
    \fi
```

```
\fi
  \ifnum\@l@dtempcnta=0
    \relax
  \else
    \ifnum\@l@dtempcnta>-10000
      \penalty\@l@dtempcnta
    \else
      \penalty -10000
    \fi
  \fi}
```

7.7 Printing leftover notes

`\flush@notes` The `\flush@notes` macro is called after the entire paragraph has been sliced up and sent on to the vertical list. If the number of notes to this paragraph has increased since the last run of TEX, then there can be leftover notes that haven't yet been printed. An appropriate error message will be printed elsewhere; but it's best to go ahead and print these notes somewhere, even if it's not in quite the right place. What we do is dump them all out here, so that they should be printed on the same page as the last line of the paragraph. We can hope that's not too far from the proper location, to which they'll move on the next run.

```
%\def\flush@notes{%
\newcommand*{\flush@notes}{%
  \@xloop
    \ifx\inserts@list\empty \else
      \gl@p\inserts@list\to\@insert
      \@insert
      \global\let\@insert=\undefined
  \repeat}
```

`\@xloop` `\@xloop` is a variant of the PLAIN TEX `\loop` macro, useful when it's hard to construct a positive test using the TEX `\if` commands—as in `\flush@notes` above. One says `\@xloop ... \if ... \else ... \repeat`, and the action following `\else` is repeated as long as the `\if` test fails. (This macro will work wherever the PLAIN TEX `\loop` is used, too, so we could just call it `\loop`; but it seems preferable not to change the definitions of any of the standard macros.)

This variant of `\loop` was introduced by Alois Kabelschacht in *TUGboat* **8** (1987), pp. 184–5.

```
\def\@xloop#1\repeat{%
  \def\body{#1\expandafter\body\fi}%
  \body}
```

8 Footnotes

The footnote macros are adapted from those in PLAIN TEX, but they differ in these respects: the outer-level commands must add other commands to a list macro rather than doing insertions immediately; there are five separate levels of footnotes, not just one; and there are options to reformat footnotes into paragraphs or into multiple columns.

8.1 Fonts

Before getting into the details of formatting the notes, we set up some font macros. It is the notes that present the greatest challenge for our font-handling mechanism, because we need to be able to take fragments of our main text and print them in different forms: it is common to reduce the size, for example, without otherwise changing the fonts used.

I have deleted all Plain Font-related code, just keeping the code for NFSS font handling.

`\notefontsetup` The font setup defined in `\notefontsetup` defines the standard fonts for the text of the footnotes. Parts of the footnote, such as the line number references and the lemma, are enclosed in groups, with their own font macros, so a note in plain roman can still have line numbers in bold, say, and the lemma in the same font encoding, family, series, and shape of font as in the main text. Typically this definition should specify only a size. It should always end with `\selectfont` or with a command like `\rm` that does a `\selectfont`.

The original font for `\notefontsetup` effectively maps to LaTeX `\footnotesize` for a 10pt document.

```
% \def\notefontsetup{\fontsize{8}{9pt}\selectfont}
\newcommand*{\notefontsetup}{\footnotesize}
```

`\notenumfont` The line numbers will be printed using the font selected by executing `\notenumfont`.

The original font for `\notenumfont` maps to LaTeX `\scriptsize` for a 10pt document. However, the description in the user guide does not seem to match the definition (the usage guide says that the size is `\notefontsetup`).

```
% \def\notenumfont{\fontsize{7}{8pt}\rm}
\newcommand*{\notenumfont}{\normalfont}
```

`\select@lemmafont` `\select@@lemmafont` `\select@lemmafont` is provided to set the right font for the lemma in a note. This macro extracts the font specifier from the line and page number cluster, and issues the associated font-changing command, so that the lemma is printed in its original font.

```
\def\select@lemmafont#1|#2|#3|#4|#5|#6|#7|{\select@@lemmafont#7|}
\def\select@@lemmafont#1/#2/#3/#4|%
  {\fontencoding{#1}\fontfamily{#2}\fontseries{#3}\fontshape{#4}%
   \selectfont}

```

8.2 Outer-level footnote commands

\Afootnote The outer-level footnote commands will look familiar: they're just called \Afootnote, \Bfootnote, etc., instead of plain \footnote. What they do, however, is quite different, since they have to operate in conjunction with \text when numbering is in effect.

If we're within a line-numbered paragraph, then, we tack this note onto the \inserts@list list, and increment the deferred-page-bottom-note counter.

```
%\def\Afootnote#1{%
\newcommand*{\Afootnote}[1]{%
  \ifnumberedpar@
    \xright@appenditem{\noexpand\vAfootnote{A}%
                  {{\@nums}{\@tag}{#1}}}\to\inserts@list
    \global\advance\insert@count by 1
```

Within free text, there's no need to put off making the insertion for this note. No line numbers are available, so this isn't generally that useful; but you might want to use it to get around some limitation of EDMAC.

```
  \else
    \vAfootnote{A}{{0|0|0|0|0|0|0}{}{#1}}%
  \fi\ignorespaces}
```

\Bfootnote We need similar commands for the other footnote series.
\Cfootnote
\Dfootnote
\Efootnote

```
%\def\Bfootnote#1{%
\newcommand*{\Bfootnote}[1]{%
  \ifnumberedpar@
    \xright@appenditem{\noexpand\vBfootnote{B}%
                  {{\@nums}{\@tag}{#1}}}\to\inserts@list
    \global\advance\insert@count by 1
  \else
    \vBfootnote{B}{{0|0|0|0|0|0|0}{}{#1}}%
  \fi\ignorespaces}

%\def\Cfootnote#1{%
\newcommand*{\Cfootnote}[1]{%
  \ifnumberedpar@
    \xright@appenditem{\noexpand\vCfootnote{C}%
                  {{\@nums}{\@tag}{#1}}}\to\inserts@list
    \global\advance\insert@count by 1
  \else
    \vCfootnote{C}{{0|0|0|0|0|0|0}{}{#1}}%
  \fi\ignorespaces}

%\def\Dfootnote#1{%
\newcommand*{\Dfootnote}[1]{%
  \ifnumberedpar@
    \xright@appenditem{\noexpand\vDfootnote{D}%
                  {{\@nums}{\@tag}{#1}}}\to\inserts@list
    \global\advance\insert@count by 1
  \else
    \vDfootnote{D}{{0|0|0|0|0|0|0}{}{#1}}%
```

```
  \fi\ignorespaces}
%\def\Efootnote#1{%
\newcommand*{\Efootnote}[1]{%
  \ifnumberedpar@
    \xright@appenditem{\noexpand\vEfootnote{E}%
                   {{\@nums}{\@tag}{#1}}}\to\inserts@list
    \global\advance\insert@count by 1
  \else
    \vEfootnote{E}{{0|0|0|0|0|0|0}{}{#1}}%
  \fi\ignorespaces}
```

8.3 Normal footnote formatting

The processing of each note is done by four principal macros: the `vfootnote` macro takes the text of the footnote and does the `\insert`; it calls on the `footfmt` macro to select the right fonts, print the line number and lemma, and do any other formatting needed for that individual note. Within the output routine, the two other macros, `footstart` and `footgroup`, are called; the first prints extra vertical space and a footnote rule, if desired; the second does any reformatting of the whole set of footnotes in this series for this page—such as paragraphing or division into columns—and then sends them to the page.

These four macros, and the other macros and parameters shown here, are distinguished by the "series letter" that indicates which set of footnotes we're dealing with—"A", `B`, `C`, `D`, or `E`. The series letter always precedes the string `foot` in macro and parameter names. Hence, for the `A` series, the four macros are called `\vAfootnote`, `\Afootfmt`, `\Afootstart`, and `\Afootgroup`.

`\normalvfootnote` We now begin a series of commands that do "normal" footnote formatting: a format much like that implemented in Plain TEX, in which each footnote is a separate paragraph.

`\normalvfootnote` takes the series letter as `#1`, and the entire text of the footnote is `#2`. It does the `\insert` for this note, calling on the `footfmt` macro for this note series to format the text of the note.

```
%\def\normalvfootnote#1#2{\insert\csname #1footins\endcsname\bgroup
\newcommand*{\normalvfootnote}[2]{\insert\csname #1footins\endcsname\bgroup
  \notefontsetup
  \interlinepenalty\csname inter#1footnotelinepenalty\endcsname
  \splittopskip\ht\strutbox
  \splitmaxdepth\dp\strutbox \floatingpenalty\@MM
  \leftskip\z@skip \rightskip\z@skip
  \spaceskip\z@skip \xspaceskip\z@skip
  \csname #1footfmt\endcsname #2\egroup}
```

`\normalfootfmt` `\normalfootfmt` is a "normal" macro to take the footnote line and page number information (see p. 27), and the desired text, and output what's to be printed. Argument `#1` contains the line and page number information and lemma font

specifier; #2 is the lemma; #3 is the note's text. This version is very rudimentary—it uses \printlines to print just the range of line numbers, followed by a square bracket, the lemma, and the note text; it's intended to be copied and modified as necessary.

\par should always be redefined to \endgraf within the format macro (this is what \normal@pars does), to override any tricky stuff which might be done in the main text to get the lines numbered automatically (as set up by \autopar, for example).

```
%\def\normalfootfmt#1#2#3{%
\newcommand*{\normalfootfmt}[3]{%
  \normal@pars
  \parindent=0pt \parfillskip=0pt plus 1fil
  {\notenumfont\printlines#1|}\strut\enspace
      {\select@lemmafont#1|#2}\rbracket\enskip#3\strut\par}

```

\endashchar
\fullstop
\rbracket

The fonts that are used for printing notes might not have the character mapping we expect: for example, the Computer Modern font that contains old-style numerals does not contain an en-dash or square brackets, and its period and comma are in odd locations. To allow use of the standard footnote macros with such fonts, we use the following macros for certain characters.

The \endashchar macro is simply an en-dash from a \rm font that is immune to changes in the surrounding font. The same goes for the full stop. These two are used in \printlines. The right bracket macro is the same again; it crops up in \normalfootfmt and the other footnote macros for controlling the format of footnotes.

```
%\def\endashchar{{\rm --}}
%\def\fullstop{{\rm .}}
%\def\rbracket{{\rm\thinspace ]}}
\def\endashchar{\textnormal{--}}
\newcommand*{\fullstop}{\textnormal{.}}
\newcommand*{\rbracket}{\textnormal{\thinspace]}}

```

\printlines

The \printlines macro prints the line numbers for a note—which, in the general case, is a rather complicated task. The seven parameters of the argument are the line numbers as stored in \@nums, in the form described on page 27: the starting page, line, and sub-line numbers, followed by the ending page, line, and sub-line numbers, and then the font specifier for the lemma.

\@pnum
\@ssub
\@elin
\@esl
\@dash

To simplify the logic, we use a lot of counters to tell us which numbers need to get printed (using 1 for yes, 0 for no, so that \ifodd tests for "yes"). The counter assignments are:

- \@pnum for page numbers;
- \@ssub for starting sub-line;
- \@elin for ending line;

- `\@esl` for ending sub-line; and
- `\@dash` for the dash between the starting and ending groups.

There's no counter for the line number because it's always printed.

```
\newcount\@pnum \newcount\@ssub \newcount\@elin
\newcount\@esl  \newcount\@dash
```

First of all, we print the page numbers only if: 1) we're doing the lineation by page, and 2) the ending page number is different from the starting page number.

```
\def\printlines#1|#2|#3|#4|#5|#6|#7|{\begingroup
  \@pnum=0 \@dash=0
  \ifbypage@
     \ifnum#4=#1 \else
        \@pnum=1
        \@dash=1
     \fi
  \fi
```

We print the ending line number if: 1) we're printing the ending page number, or 2) it's different from the starting line number.

```
  \@elin=\@pnum
  \ifnum#2=#5 \else
      \@elin=1
      \@dash=1
  \fi
```

We print the starting sub-line if it's nonzero.

```
  \@ssub=0
  \ifnum#3=0 \else
      \@ssub=1
  \fi
```

We print the ending sub-line if it's nonzero and: 1) it's different from the starting sub-line number, or 2) the ending line number is being printed.

```
  \@esl=0
  \ifnum#6=0 \else
      \ifnum#6=#3
         \@esl=\@elin
      \else
         \@esl=1
         \@dash=1
      \fi
  \fi
```

Now we're ready to print it all, based on our counter values. The only subtlety left here is when to print a period between numbers. But the only instance in which this is tricky is for the ending sub-line number: it could be coming after the starting sub-line number (in which case we want only the dash) or after an ending line number (in which case we need to insert a period).

```
  \ifodd\@pnum #1\fullstop\fi
```

```
  #2%
  \ifodd\@ssub \fullstop #3\fi
  \ifodd\@dash \endashchar\fi
  \ifodd\@pnum #4\fullstop\fi
  \ifodd\@elin #5\fi
  \ifodd\@esl \ifodd\@elin\fullstop\fi #6\fi
\endgroup}

```

`\normalfootstart` `\normalfootstart` is a standard footnote-starting macro, called in the output routine whenever there are footnotes of this series to be printed: it skips a bit and then draws a rule.

Any `footstart` macro must put onto the page something that takes up space exactly equal to the `\skip\footins` value for the associated series of notes. TEX makes page computations based on that `\skip` value, and the output pages will suffer from spacing problems if what you add takes up a different amount of space.

The `\leftskip` and `\rightskip` values are both zeroed here. Similarly, these skips are cancelled in the `vfootnote` macros for the various types of notes. Strictly speaking, this is necessary only if you are using paragraphed footnotes, but we have put it here and in the other `vfootnote` macros too so that the behavior of `EDMAC` in this respect is general across all footnote types (you can change this). What this means is that any `\leftskip` and `\rightskip` you specify applies to the main text, but not the footnotes. The footnotes continue to be of width `\hsize`.

```
%\def\normalfootstart#1{%
\newcommand*{\normalfootstart}[1]{%
  \vskip\skip\csname #1footins\endcsname
  \leftskip0pt \rightskip0pt
  \csname #1footnoterule\endcsname}
```

`\normalfootnoterule` `\normalfootnoterule` is a standard footnote-rule macro, for use by a `footstart` macro: just the same as the PLAIN TEX footnote rule.

```
\let\normalfootnoterule=\footnoterule
```

`\normalfootgroup` `\normalfootgroup` is a standard footnote-grouping macro: it sends the contents of the footnote-insert box to the output page without alteration.

```
%\def\normalfootgroup#1{\unvbox\csname #1footins\endcsname}
\newcommand*{\normalfootgroup}[1]{\unvbox\csname #1footins\endcsname}

```

8.4 Standard footnote definitions

`\footnormal` We can now define all the parameters for the five series of footnotes; initially they use the "normal" footnote formatting, which is set up by calling `\footnormal`. You can switch to another type of formatting by using `\footparagraph`, `\foottwocol`, or `\footthreecol`.

Switching to a variation of "normal" formatting requires changing the quantities defined in `\footnormal`. The best way to proceed would be to make a copy

of this macro, with a different name, make your desired changes in that copy, and then invoke it, giving it the letter of the footnote series you wish to control.

(We have not defined baseline skip values like `\abaselineskip`, since this is one of the quantities set in `\notefontsetup`.)

What we want to do here is to say something like the following for each footnote series. (This is an example, not part of the actual `EDMAC` code.)

```
\newinsert\Afootins
\newcount\interAfootnotelinepenalty \interAfootnotelinepenalty=100
\skip\Afootins=12pt plus5pt minus5pt
\count\Afootins=1000
\dimen\Afootins=0.8\vsize
\let\vAfootnote=\normalvfootnote  \let\Afootfmt=\normalfootfmt
\let\Afootstart=\normalfootstart  \let\Afootgroup=\normalfootgroup
\let\Afootnoterule=\normalfootnoterule
```

Instead of repeating ourselves, we define a `\footnormal` macro that makes all these assignments for us, for any given series letter. This also makes it easy to change from any different system of formatting back to the `normal` setting.

We begin by defining the five new insertion classes, and some `count` registers; these are `\outer` operations that can't be done inside `\footnormal`.

```
\newinsert\Afootins \newinsert\Bfootins
\newinsert\Cfootins \newinsert\Dfootins
\newinsert\Efootins
\newcount\interAfootnotelinepenalty
\newcount\interBfootnotelinepenalty
\newcount\interCfootnotelinepenalty
\newcount\interDfootnotelinepenalty
\newcount\interEfootnotelinepenalty
```

Now we set up the `\footnormal` macro itself. It takes one argument: the footnote series letter.

```
%\def\footnormal#1{%
\newcommand*{\footnormal}[1]{%
  \csname inter#1footnotelinepenalty\endcsname=100
  \expandafter\let\csname #1footstart\endcsname=\normalfootstart
  \expandafter\let\csname v#1footnote\endcsname=\normalvfootnote
  \expandafter\let\csname #1footfmt\endcsname=\normalfootfmt
  \expandafter\let\csname #1footgroup\endcsname=\normalfootgroup
  \expandafter\let\csname #1footnoterule\endcsname=%
                                          \normalfootnoterule
  \count\csname #1footins\endcsname=1000
  \dimen\csname #1footins\endcsname=0.8\vsize
  \skip\csname #1footins\endcsname=12pt plus6pt minus6pt}

```

Some of these values deserve comment: the `\dimen` setting allows 80% of the page to be occupied by notes; the `\skip` setting is deliberately flexible, since pages with lots of notes attached to many of the lines can be a bit hard for TEX to make.

And finally, we initialize the formatting for all the footnote series to be normal.

```
\footnormal{A}
\footnormal{B}
\footnormal{C}
\footnormal{D}
\footnormal{E}
```

8.5 Paragraphed footnotes

The paragraphed-footnote option reformats all the footnotes of one series for a page into a single paragraph; this is especially appropriate when the notes are numerous and brief. The code is based on *The TEXbook*, pp. 398–400, with alterations for our environment. This algorithm uses a considerable amount of save-stack space: a TEX of ordinary size may not be able to handle more than about 100 notes of this kind on a page.

`\footparagraph` The `\footparagraph` macro sets up everything for one series of footnotes so that they'll be paragraphed; it takes the series letter as argument. We include the setting of `\count\footins` to 1000 for the footnote series just in case you are switching to paragraphed footnotes after having columnar ones, since they change this value (see below).

It is important to call `\footparagraph` only after `\hsize` has been set for the pages that use this series of notes; otherwise TEX will try to put too many or too few of these notes on each page. If you need to change the `\hsize` within the document, call `\footparagraph` again afterwards to take account of the new value. The argument of `\footparagraph` is the letter ("A"–"E") denoting the series of notes to be paragraphed.

```
%\def\footparagraph#1{%
\newcommand*{\footparagraph}[1]{%
  \expandafter\let\csname #1footstart\endcsname=\parafootstart
  \expandafter\let\csname v#1footnote\endcsname=\para@vfootnote
  \expandafter\let\csname #1footfmt\endcsname=\parafootfmt
  \expandafter\let\csname #1footgroup\endcsname=\para@footgroup
  \count\csname #1footins\endcsname=1000
  \para@footsetup{#1}}
```

`\para@footsetup` `\footparagraph` calls the `\para@footsetup` macro to calculate a special fudge factor, which is the ratio of the `\baselineskip` to the `\hsize`. We assume that the proper value of `\baselineskip` for the footnotes (normally 9 pt) has been set already, in `\notefontsetup`. The argument of the macro is again the note series letter.

```
%\def\para@footsetup#1{{\notefontsetup
\newcommand*{\para@footsetup}[1]{{\notefontsetup
  \dimen0=\baselineskip
  \multiply\dimen0 by 1024
  \divide \dimen0 by \hsize \multiply\dimen0 by 64
```

```
\expandafter
\xdef\csname #1footfudgefactor\endcsname{%
  \expandafter\en@number\the\dimen0 }}}

```

`\en@number` The `\en@number` macro extracts the numerical part from a `dimen` register and strips off the ""pt"" (see *The TEXbook*, p. 375). It turns `10pt` into `10` so that you can use it in arithmetic.

```
{\catcode`p=12 \catcode`t=12 \gdef\en@number#1pt{#1}}

```

`\parafootstart` `\parafootstart` is the same as `\normalfootstart`, but we give it again to ensure that `\rightskip` and `\leftskip` are zeroed (this needs to be done before `\para@footgroup` in the output routine). You might have decided to change this for other kinds of note, but here it should stay as it is. The size of paragraphed notes is calculated using a fudge factor which in turn is based on `\hsize`. So the paragraph of notes needs to be that wide.

The argument of the macro is again the note series letter.

```
%\def\parafootstart#1{%
\newcommand*{\parafootstart}[1]{%
  \rightskip=0pt \leftskip=0pt \parindent=0pt
  \vskip\skip\csname #1footins\endcsname
  \csname #1footnoterule\endcsname}

```

`\para@vfootnote` `\para@vfootnote` is a version of the `\vfootnote` command that's used for paragraphed notes. It gets appended to the `\inserts@list` list by an outer-level footnote command like `\Afootnote`. The first argument is the note series letter; the second is the full text of the printed note itself, including line numbers, lemmata, and footnote text.

The initial model for this insertion is, of course, the `\insert\footins` definition in *The TEXbook*, p. 398. There, the footnotes are first collected up in hboxes, and these hboxes are later unpacked and stuck together into a paragraph.

However, Michael Downes has pointed out that because text in hboxes gets typeset in restricted horizontal mode, there are some undesirable side-effects if you later want to break such text across lines. In restricted horizontal mode, where TEX does not expect to have to break lines, it does not insert certain items like `\discretionary`s. If you later unbox these hboxes and stick them together, as the *TEXbook* macros do to make these footnotes, you lose the ability to hyphenate after an explicit hyphen. This can lead to overfull `\hbox`es when you would not expect to find them, and to the uninitiated it might be very hard to see why the problem had arisen.[21]

Wayne Sullivan pointed out to us another subtle problem that arises from the same cause: TEX also leaves the `\language` whatsit nodes out of the horizontal list.[22] So changes from one language to another will not invoke the proper hy-

[21] Michael Downes, "Line Breaking in `\unhbox`ed Text", *TUGboat* **11** (1990), pp. 605–612.
[22] See *The TEXbook*, p. 455 (editions after January 1990).

phenation rules in such footnotes. Since critical editions often do deal with several languages, especially in footnotes, we really ought to get this bit of code right.

To get around these problems, Wayne suggested emendations to the *TEXbook* versions of these macros which are broadly the same as those described by Michael: the central idea (also suggested by Donald Knuth in a letter to Michael) is to avoid collecting the text in an `\hbox` in the first place, but instead to collect it in a `\vbox` whose width is (virtually) infinite. The text is therefore typeset in unrestricted horizontal mode, as a paragraph consisting of a single long line. Later, there is an extra level of unboxing to be done: we have to unpack the `\vbox`, as well as the hboxes inside it, but that's not too hard. For details, we refer you to Michael's article, where the issues are clearly explained.[23] Michael's unboxing macro is called `\unvxh`: unvbox, extract the last line, and unhbox it.

Doing things this way has an important consequence: as Michael pointed out, you really can't put an explicit line-break into a note built in a `\vbox` the way we are doing.[24] In other words, be very careful not to say `\break`, or `\penalty-10000`, or any equivalent inside your para-footnote. If you do, most of the note will probably disappear. You *are* allowed to make strong suggestions; in fact `\penalty-9999` will be quite okay. Just don't make the break mandatory. We haven't applied any of Michael's solutions here, since we feel that the problem is exiguous, and `EDMAC` is quite baroque enough already. If you think you are having this problem, look up Michael's solutions.

One more thing; we set `\leftskip` and `\rightskip` to zero. This has the effect of neutralizing any such skips which may apply to the main text (cf. p. 64 above). We need to do this, since `footfudgefactor` is calculated on the assumption that the notes are `\hsize` wide.

So, finally, here is the modified foot-paragraph code, which sets the footnote in vertical mode so that language and discretionary nodes are included.

```
%\def\para@vfootnote#1#2{\insert\csname #1footins\endcsname
\newcommand*{\para@vfootnote}[2]{\insert\csname #1footins\endcsname
  \bgroup
    \notefontsetup
    \interlinepenalty=\csname inter#1footnotelinepenalty\endcsname
    \splittopskip=\ht\strutbox
    \splitmaxdepth=\dp\strutbox \floatingpenalty=\@MM
    \leftskip0pt \rightskip0pt
    \setbox0=\vbox{\hsize=\maxdimen
      \noindent\csname #1footfmt\endcsname#2}%
    \setbox0=\hbox{\unvxh0}%
    \dp0=0pt
    \ht0=\csname #1footfudgefactor\endcsname\wd0
```

Here we produce the contents of the footnote from box 0, and add a penalty of 0 between boxes in this insert.

```
    \box0
```

[23] Wayne supplied his own macros to do this, but since they were almost identical to Michael's, we have used the latter's `\unvxh` macro since it is publicly documented.

[24] "Line Breaking", p. 610.

```
    \penalty0
  \egroup}
```

The final penalty of 0 was added here at Wayne's suggestion to avoid a weird page-breaking problem, which occurs on those occasions when TEX attempts to split foot paragraphs. After trying out such a split (see *The TEXbook*, p. 124), TEX inserts a penalty of −10000 here, which nearly always forces the break at the end of the whole footnote paragraph (since individual notes can't be split) even when this leads to an overfull vbox. The change above results in a penalty of 0 instead which allows, but doesn't force, such breaks. This penalty of 0 is later removed, after page breaks have been decided, by the `\unpenalty` macro in `\makehboxofhboxes`. So it does not affect how the footnote paragraphs are typeset (the notes still have a penalty of −10 between them, which is added by `\parafootfmt`).

`\unvxh` Here is Michael's definition of `\unvxh`, used above. Michael's macro also takes care to remove some unwanted penalties and glue that TEX automatically attaches to the end of paragraphs. When TEX finishes a paragraph, it throws away any remaining glue, and then tacks on the following items: a `\penalty` of 10000, a `\parfillskip` and a `\rightskip` (*The TEXbook*, pp. 99–100). `\unvxh` cancels these unwanted paragraph-final items using `\unskip` and `\unpenalty`.

```
%\def\unvxh#1{%
\newcommand*{\unvxh}[1]{%
  \setbox0=\vbox{\unvbox#1%
    \global\setbox1=\lastbox}%
  \unhbox1
  \unskip            % remove \rightskip,
  \unskip            % remove \parfillskip,
  \unpenalty         % remove \penalty of 10000,
  \hskip\ipn@skip}   % but add the glue to go between the notes
```

`\interparanoteglue` `\ipn@skip` Close observers will notice that we snuck some glue called `\ipn@skip` onto the end of the hbox produced by `\unvxh` in the above macro.

We want to be able to have some glue between our paragraphed footnotes. But since we are initially setting our notes in internal vertical mode, as little paragraphs, any paragraph-final glue will get discarded. Since `\unvxh` is already busy fiddling with glue and penalties at the end of these paragraphs, we take advantage of the opportunity to provide our inter-note spacing.

We collect the value of the inter-parafootnote glue value as the parameter of a macro called—wait for it—`\interparanoteglue`. We put this value into the value of a glue register `\ipn@skip` (inter-para-note-skip) making sure first to set the current font to the value normally used in footnotes so that the value of an `em` will be taken from the right font.

```
\newskip\ipn@skip
%\def\interparanoteglue#1{%
\newcommand*{\interparanoteglue}[1]{%
```

```
1105              {\notefontsetup\global\ipn@skip=#1 \relax}}
1106 \interparanoteglue{1em plus.4em minus.4em}
1107
```

There is a point to be careful about regarding the `\interparanoteglue`. Remember that in `\para@vfootnote` we do some measurements on the footnote box, and use the resulting size to make an estimate of how much the note will contribute to the height of our final footnote paragraph. This information is used by the output routine to allocate the right amount of vertical space on the page for the notes (*The TEXbook*, pp. 398–399).

The length of the footnote includes the natural size of the glue specified by `\interparanoteglue`, but not its stretch or shrink components, since at this point the note has no need to stretch or shrink. Later, when the paragraph is actually composed by `\parafootgroup` in the output routine, TEX will almost certainly do some stretching and shrinking of this glue in order to make the paragraph look nice. Probably the stretching and shrinking over the whole paragraph will cancel each other out. But if not, the actual vertical size of the paragraph may not match the size the output routine had been told to expect, and you may get an overfull/underfull `\vbox` message from the output routine. To minimize the risk of this, you can do two things: keep the `plus` and `minus` components of `\interparanoteglue` small compared with its natural glue, and keep them the same as each other. As a general precaution, keep the size and flexibility of the `\skip\footins` glue on the high side too: because the reckoning is approximate, footnote blocks may be up to a line bigger or smaller than the output routine allows for, so keep some flexible space between the text and the notes.

`\parafootfmt` `\parafootfmt` is `\normalfootfmt` adapted to do the special stuff needed for paragraphed notes—leaving out the `\endgraf` at the end, sticking in special penalties and kern, and leaving out the `\footstrut`. The first argument is the line and page number information, the second is the lemma, and the third is the text of the footnote.

```
1108 %\def\parafootfmt#1#2#3{%
1109 \newcommand*{\parafootfmt}[3]{%
1110   \normal@pars
1111   \parindent=0pt \parfillskip=0pt plus1fil
1112   {\notenumfont\printlines#1|}\enspace
1113   {\select@lemmafont#1|#2}\rbracket\enskip
1114   #3\penalty-10 }
```

Note that in the above definition, the penalty of -10 encourages a line break between notes, so that notes have a slight tendency to begin on new lines.

`\para@footgroup` This `footgroup` code is modelled on the macros in *The TEXbook*, p. 399. The only difference is the `\unpenalty` in `\makehboxofhboxes`, which is there to remove the penalty of 0 which was added to the end of each footnote by `\para@vfootnote`.

The call to `\notefontsetup` is to ensure that the correct `\baselineskip` for the footnotes is used. The argument is the note series letter.

```
1115 %\def\para@footgroup#1{%
```

```
\newcommand*{\para@footgroup}[1]{%
  \unvbox\csname #1footins\endcsname \makehboxofhboxes
  \setbox0=\hbox{\unhbox0 \removehboxes}%
  \notefontsetup
  \noindent\unhbox0\par}
%\def\makehboxofhboxes{\setbox0=\hbox{}%
\newcommand*{\makehboxofhboxes}{\setbox0=\hbox{}%
  \loop
    \unpenalty
    \setbox2=\lastbox
  \ifhbox2
    \setbox0=\hbox{\box2\unhbox0}
  \repeat}
%\def\removehboxes{\setbox0=\lastbox
\newcommand*{\removehboxes}{\setbox0=\lastbox
  \ifhbox0{\removehboxes}\unhbox0 \fi}

```

8.6 Columnar footnotes

\rigidbalance
\dosplits
\splitoff
\@h
\@k

We will now define macros for three-column notes and two-column notes. Both sets of macros will use `\rigidbalance`, which splits a box (`#1`) into into a number (`#2`) of columns, each with a space (`#3`) between the top baseline and the top of the `\vbox`. The `\rigidbalance` macro is taken from *The TEXbook*, p. 397, with a slight change to the syntax of the arguments so that they don't depend on white space. Note also the extra unboxing in `\splitoff`, which allows the new `\vbox` to have its natural height as it goes into the alignment.

The LaTeX `\line` macro has no relationship to the TeX `\line`. The LaTeX equivalent is `\@@line`.

```
\newcount\@k \newdimen\@h
%\def\rigidbalance#1#2#3{\setbox0=\box#1 \@k=#2 \@h=#3
\newcommand*{\rigidbalance}[3]{\setbox0=\box#1 \@k=#2 \@h=#3
%  \line{\splittopskip=\@h \vbadness=\@M \hfilneg
  \@@line{\splittopskip=\@h \vbadness=\@M \hfilneg
  \valign{##\vfil\cr\dosplits}}}
%\def\dosplits{\ifnum\@k>0 \noalign{\hfil}\splitoff
\newcommand*{\dosplits}{\ifnum\@k>0 \noalign{\hfil}\splitoff
  \global\advance\@k-1\cr\dosplits\fi}
%\def\splitoff{\dimen0=\ht0
\newcommand*{\splitoff}{\dimen0=\ht0
  \divide\dimen0 by\@k \advance\dimen0 by\@h
  \setbox2 \vsplit0 to \dimen0
  \unvbox2 }

```

Three columns

\footthreecol

You say `\footthreecol{A}` to have the `A` series of footnotes typeset in three

columns. It is important to call this only after \hsize has been set for the document.

```
%\def\footthreecol#1{%
\newcommand*{\footthreecol}[1]{%
  \expandafter\let\csname v#1footnote\endcsname=\threecolvfootnote
  \expandafter\let\csname #1footfmt\endcsname=\threecolfootfmt
  \expandafter\let\csname #1footgroup\endcsname=\threecolfootgroup
  \threecolfootsetup{#1}}
```

The \footstart and \footnoterule macros for these notes assume the normal values (p. 64 above).

\threecolfootsetup The \threecolfootsetup macro calculates and sets some numbers for three-column footnotes.

We set the \count of the foot insert to 333. Each footnote can be thought of as contributing only one third of its height to the page, since the footnote insertion has been made as a long narrow column, which then gets trisected by the \rigidbalance routine (inside \threecolfootgroup). These new, shorter columns are saved in a box, and then that box is *put back* into the footnote insert, replacing the original collection of footnotes. This new box is, therefore, only about a third of the height of the original one.

The \dimen value for this note series has to change in the inverse way: it needs to be three times the actual limit on the amount of space these notes are allowed to fill on the page, because when TEX is accumulating material for the page and checking that limit, it doesn't apply the \count scaling.

```
%\def\threecolfootsetup#1{%
\newcommand*{\threecolfootsetup}[1]{%
  \count\csname #1footins\endcsname 333
  \multiply\dimen\csname #1footins\endcsname by 3 }
```

\threecolvfootnote \threecolvfootnote is the \vfootnote command for three-column notes. The call to \notefontsetup ensures that the \splittopskip and \splitmaxdepth take their values from the right \strutbox: the one used in footnotes. Note especially the importance of temporarily reducing the \hsize to 0.3 of its normal value. This determines the widths of the individual columns. So if the normal \hsize is, say, 10 cm, then each column will be $0.3 \times 10 = 3$ cm wide, leaving a gap of 1 cm spread equally between columns (i.e., .5 cm between each).

The arguments are 1) the note series letter and 2) the full text of the note (including numbers, lemma and text).

```
%\def\threecolvfootnote#1#2{%
\newcommand*{\threecolvfootnote}[2]{%
  \insert\csname #1footins\endcsname\bgroup
  \notefontsetup
  \interlinepenalty=\csname inter#1footnotelinepenalty\endcsname
  \floatingpenalty=20000
  \splittopskip=\ht\strutbox \splitmaxdepth=\dp\strutbox
  \rightskip=0pt \leftskip=0pt
  \csname #1footfmt\endcsname #2\egroup}
```

\threecolfootfmt

\threecolfootfmt is the command that formats one note. It uses \raggedright, which will usually be preferable with such short lines. Setting the \parindent to zero means that, within each individual note, the lines begin flush left.

The arguments are 1) the line numbers, 2) the lemma and 3) the text of the -footnote command.

```
%\def\threecolfootfmt#1#2#3{%
\newcommand*{\threecolfootfmt}[3]{%
  \normal@pars
  \hsize .3\hsize
  \parindent=0pt
  \tolerance=5000
  \raggedright
  \leavevmode
  \strut{\notenumfont\printlines#1|}\enspace
  {\select@lemmafont#1|#2}\rbracket\enskip
  #3\strut\par\allowbreak}
```

\threecolfootgroup

And here is the footgroup macro that's called within the output routine to regroup the notes into three columns. Once again, the call to \notefontsetup is there to ensure that it is the right \splittopskip—the one used in footnotes—which is used to provide the third argument for \rigidbalance. This third argument ("h") is the topskip for the box containing the text of the footnotes, and does the job of making sure the top lines of the columns line up horizontally. In *The TEXbook*, p. 398, Donald Knuth suggests retrieving the ouput of \rigidbalance, putting it back into the insertion box, and then printing the box. Here, we just print the \line which comes out of \rigidbalance directly, without any re-boxing.

```
%\def\threecolfootgroup#1{{\notefontsetup
\newcommand*{\threecolfootgroup}[1]{{\notefontsetup
  \splittopskip=\ht\strutbox
  \expandafter
  \rigidbalance\csname #1footins\endcsname 3 \splittopskip }}

```

Two columns

\foottwocol

You say \foottwocol{A} to have the A series of footnotes typeset in two columns. It is important to call this only after \hsize has been set for the document.

```
%\def\foottwocol#1{%
\newcommand*{\foottwocol}[1]{%
  \expandafter\let\csname v#1footnote\endcsname=\twocolvfootnote
  \expandafter\let\csname #1footfmt\endcsname=\twocolfootfmt
  \expandafter\let\csname #1footgroup\endcsname=\twocolfootgroup
  \twocolfootsetup{#1}}
```

\twocolfootsetup
\twocolvfootnote
\twocolfootfmt
\twocolfootgroup

Here is a series of macros which are very similar to their three-column counterparts. In this case, each note is assumed to contribute only a half a line of text. And

the notes are set in columns `0.45\hsize` wide, giving a gap between them of one tenth of the `\hsize`.

```
%\def\twocolfootsetup#1{%
\newcommand*{\twocolfootsetup}[1]{%
  \count\csname #1footins\endcsname 500
  \multiply\dimen\csname #1footins\endcsname by 2 }

%\def\twocolvfootnote#1#2{\insert\csname #1footins\endcsname\bgroup
\newcommand*{\twocolvfootnote}[2]{\insert\csname #1footins\endcsname\bgroup
  \notefontsetup
  \interlinepenalty=\csname inter#1footnotelinepenalty\endcsname
  \floatingpenalty=20000
  \splittopskip=\ht\strutbox \splitmaxdepth=\dp\strutbox
  \rightskip=0pt \leftskip=0pt
  \csname #1footfmt\endcsname #2\egroup}

%\def\twocolfootfmt#1#2#3{%
\newcommand*{\twocolfootfmt}[3]{%
  \normal@pars
  \hsize .45\hsize
  \parindent=0pt
  \tolerance=5000
  \raggedright
  \leavevmode
  \strut{\notenumfont\printlines#1|}\enspace
  {\select@lemmafont#1|#2}\rbracket\enskip
  #3\strut\par\allowbreak}

%\def\twocolfootgroup#1{{\notefontsetup
\newcommand*{\twocolfootgroup}[1]{{\notefontsetup
  \splittopskip=\ht\strutbox
  \expandafter
  \rigidbalance\csname #1footins\endcsname 2 \splittopskip}}

```

9 Output routine

Now we begin the output routine and associated things.

I have deleted all the crop mark code.

There are a couple of macros from plain TeX that we need (at least for now).

`\pageno` `\advancepageno`

`\pageno` is a page number, starting at 1, and `\advancepageno` increments the number.

```
\countdef\pageno=0 \pageno=1
\newcommand*{\advancepageno}{\ifnum\pageno<\z@ \global\advance\pageno\m@ne
  \else\global\advance\pageno\@ne\fi}

```

`\output` `\edmac@output` `\pagecontents`

This is probably the trickiestt part of moving from TeX to LaTeX. The original code is below, but we need something very different.

This is a new output routine, with changes to handle printing all our footnotes. Those changes have not been added directly, but are in macros that get called here: that should make it easier to see what would need to be taken over to a different output routine. We continue to use the `\pagebody`, `\makeheadline`, `\makefootline`, and `\dosupereject` macros of PLAIN TEX; for those macros, and the original version of `\output`, see *The TEXbook*, p. 364.

```
%\output{\edmac@output}
%\def\edmac@output{\shipout\vbox{\normal@pars
%    \vbox{\makeheadline\pagebody\makefootline}%
%  }%
%  \advancepageno
%  \ifnum\outputpenalty>-\@MM\else\dosupereject\fi}
%

%\def\pagecontents{\page@start
%\newcommand*{\pagecontents}{\page@start
%  \ifvoid\topins\else\unvbox\topins\fi
%  \dimen@=\dp\@cclv \unvbox\@cclv % open up \box255
%  \do@feet
%  \ifr@ggedbottom \kern-\dimen@ \vfil \fi}
%
```

`\do@feet` `\do@feet` ships out all the footnotes. Standard `EDMAC` has only five feet, but there is nothing in principal to prevent you from creating an arachnoid or centipedal edition; straightforward modifications of `EDMAC` are all that's required. However, the myriapedal edition is ruled out by TEX's limitations: the number of insertion classes is limited to 255.

```
%\def\do@feet{%
%\newcommand*{\do@feet}{%
%  \ifvoid\footins\else
%    \vskip\skip\footins
%    \footnoterule
%    \unvbox\footins
%  \fi
%  \ifvoid\Afootins\else
%    \Afootstart{A}\Afootgroup{A}%
%  \fi
%  \ifvoid\Bfootins\else
%    \Bfootstart{B}\Bfootgroup{B}%
%  \fi
%  \ifvoid\Cfootins\else
%    \Cfootstart{C}\Cfootgroup{C}%
%  \fi
%  \ifvoid\Dfootins\else
%    \Dfootstart{D}\Dfootgroup{D}%
%  \fi
%  \ifvoid\Efootins\else
%    \Efootstart{E}\Efootgroup{E}%
%  \fi}
```

```
%
```

The code that now follows is for the LaTeX output routine.

With luck we might only have to change `\@makecol`. The kernel definition of this, and perhaps some other things, is:

```
\gdef \@makecol {%
  \ifvoid\footins
    \setbox\@outputbox \box\@cclv
  \else
    \setbox\@outputbox \vbox {%
      \boxmaxdepth \@maxdepth
      \@tempdima\dp\@cclv
      \unvbox \@cclv
      \vskip \skip\footins
      \color@begingroup
        \normalcolor
        \footnoterule
        \unvbox \footins
      \color@endgroup
      }%
  \fi
  \xdef\@freelist{\@freelist\@midlist}%
  \global \let \@midlist \@empty
  \@combinefloats
  \ifvbox\@kludgeins
    \@makespecialcolbox
  \else
    \setbox\@outputbox \vbox to\@colht {%
      \@texttop
      \dimen@ \dp\@outputbox
      \unvbox\@outputbox
      \vskip -\dimen@
      \@textbottom
      }%
  \fi
  \global \maxdepth \@maxdepth
}

\gdef \@reinserts{%
  \ifvoid\footins\else\insert\footins{\unvbox\footins}\fi
  \ifvbox\@kludgeins\insert\@kludgeins{\unvbox\@kludgeins}\fi
}
```

`\@makecol` This is a partitioned version of the 'standard' `\@makecol`, with the initial code put into another macro.

```
\gdef \@makecol {%
  \dofootinsert
  \xdef\@freelist{\@freelist\@midlist}%
```

```
    \global \let \@midlist \@empty
    \@combinefloats
    \ifvbox\@kludgeins
      \@makespecialcolbox
    \else
      \setbox\@outputbox \vbox to\@colht {%
        \@texttop
        \dimen@ \dp\@outputbox
        \unvbox\@outputbox
        \vskip -\dimen@
        \@textbottom
        }%
    \fi
    \global \maxdepth \@maxdepth
 }

```

\dofootinsert This macro essentially holds the initial portion of the kernel \@makecol code.

```
\newcommand*{\dofootinsert}{%
```

The package starts off with calling \page@start

```
    \page@start
```

then continues with the kernel code

```
    \ifvoid\footins
      \setbox\@outputbox \box\@cclv
    \else
      \setbox\@outputbox \vbox {%
        \boxmaxdepth \@maxdepth
        \@tempdima\dp\@cclv
        \unvbox \@cclv
        \vskip \skip\footins
        \color@begingroup
          \normalcolor
          \footnoterule
          \unvbox \footins
        \color@endgroup
        }%
    \fi
```

That's the end of the copy of the kernel code. We finally call a macro to handle all the additional EDMAC feet.

```
    \doxtrafeet
}

```

\doxtrafeet \doxtrafeet is the code extending \@makecol to cater for the extra EDMAC feet. NOTE: the code is likely to be 'featurefull'.

```
\newcommand*{\doxtrafeet}{%
  \setbox\@outputbox \vbox{%
    \unvbox\@outputbox
```

```
    \ifvoid\Afootins\else\Afootstart{A}\Afootgroup{A}\fi
    \ifvoid\Bfootins\else\Bfootstart{B}\Bfootgroup{B}\fi
    \ifvoid\Cfootins\else\Cfootstart{C}\Cfootgroup{C}\fi
    \ifvoid\Dfootins\else\Dfootstart{D}\Dfootgroup{D}\fi
    \ifvoid\Efootins\else\Efootstart{E}\Efootgroup{E}\fi
  }}

```

Reminder: we may have to fiddle with `\@reinserts` as well. Check with various floats (pages).

10 Cross referencing

Reminder: Have to try and write this to use the LaTeX .aux file (which also handles includes).

You can mark a place in the text using a command of the form `\label{foo}`, and later refer to it using the label `foo` by saying `\pageref{foo}`, or `\lineref{foo}` or `\sublineref{foo}`. These reference commands will produce, respectively, the page, line and sub-line on which the `\label{foo}` command occurred.

The reference macros warn you if a reference is made to an undefined label. If `foo` has been used as a label before, the `\label{foo}` command will issue a complaint; subsequent `\pageref` and `\lineref` commands will refer to the latest occurrence of `\label{foo}`. When any of these commands are used, an additional auxiliary file, called ⟨*filename*⟩".aux", is created.

`\labelref@list` Set up a new list, `\labelref@list`, to hold the page, line and sub-line numbers for each label.

```
\list@create{\labelref@list}
```

`\@aux` `\@aux` is the output stream number for our auxiliary file that contains labeling commands.

```
\newwrite\@aux
```

`\do@labelsfile` The `\do@labelsfile` macro opens the auxiliary labels file and executes it; the file should consist of a stream of `\make@labels` commands. Then it opens a new auxiliary file, and redefines itself so that it isn't called again. This will be called by the first `\label` or reference command that appears in the text—and not at all if there are no labels or references, so we won't bother at all with an `.aux` file if there's no need for it.

I have (temporarily?) changed the label file from `.aux` to `.lxb`.

```
%\def\do@labelsfile{%
\newcommand*{\do@labelsfile}{%
%   \openin\@inputcheck=\jobname.aux
   \openin\@inputcheck=\jobname.lxb
   \ifeof\@inputcheck \else
      \closein\@inputcheck
      \begingroup
```

```
          \makeatletter \catcode`\^^M=9
%          \input\jobname.aux
          \input\jobname.lxb
       \endgroup
   \fi
%   \immediate\openout\@aux=\jobname.aux
   \immediate\openout\@aux=\jobname.lxb
   \global\let\do@labelsfile=\relax}

```

`\edlabel` `\zz@@@` The `\label` command first writes a `\@lab` macro to the `\linenum@out` file. It then checks to see that the `\labelref@list` actually has something in it (if not, it creates a dummy entry), and pops the next value for the current label, storing it in `\label@refs`. Finally it defines the label to be `\empty` so that any future check will turn up the fact that it has been used.[25]

```
%\def\zz@@@{000|000|000} % set three counters to zero in one go
\newcommand*{\zz@@@}{000|000|000} % set three counters to zero in one go

```

LaTeX already defines a `\label` macro, which is different. I have renamed the original `\label` as `\edlabel`.

```
%\def\label#1{\do@labelsfile
\newcommand*{\edlabel}[1]{\do@labelsfile
  \write\linenum@out{\string\@lab}%
  \ifx\labelref@list\empty
    \xdef\label@refs{\zz@@@}%
  \else
    \gl@p\labelref@list\to\label@refs
  \fi
  \edef\next{\write\@aux{\string\make@labels\label@refs|{#1}}}%
  \next}

```

`\make@labels` The `\make@labels` macro gets executed when the labels file is read. For each label it defines a macro, whose name is made up partly from the label you supplied, that contains the page, line and sub-line numbers. But first it checks to see whether the label has already been used (and complains if it has).

```
\newcommand*{\make@labels}{}
\def\make@labels#1|#2|#3|#4{%
  \expandafter\ifx\csname the@label#4\endcsname \relax\else
    \ledmac@warning{Duplicate definition of label `#4'
                    on page \number\pageno.}%
  \fi
  \expandafter\gdef\csname the@label#4\endcsname{#1|#2|#3}%
  \ignorespaces}

```

[25] The remaining macros in this section were kindly revised by Wayne Sullivan, who substantially improved their efficiency and flexibility.

\@lab The \@lab command, which appears in the \linenum@out file, appends the current values of page, line and sub-line to the \labelref@list. These values are defined by the earlier \@page, \@l, and the \sub@on and \sub@off commands appearing in the \linenum@out file.

LaTeX uses the page counter for page numbers.

```
%\def\@lab{\xright@appenditem
\newcommand*{\@lab}{\xright@appenditem
%  {\the\page@num|\the\line@num|%
  {\space\thepage|\the\line@num|%
     \ifsublines@ \the\subline@num \else 0\fi}\to\labelref@list}

```

\edpageref \xpageref If the specified label exists, \pageref gives its page number. For this reference command, as for the other two, a special version with prefix x is provided for use in places where the command is to be scanned as a number, as in \linenum. These special versions have two limitations: they don't print error messages if the reference is unknown, and they can't appear as the first label or reference command in the file; you must ensure that a \label or a normal reference command appears first, or these x-commands will always return zeros. LaTeX already defines a \pageref, so changing the name to \edpageref.

```
%\def\pageref#1{\ref@undefined{#1}\getref@num{1}{#1}}
%\def\xpageref#1{\getref@num{1}{#1}}
\newcommand*{\edpageref}[1]{\ref@undefined{#1}\getref@num{1}{#1}}
\newcommand*{\xpageref}[1]{\getref@num{1}{#1}}

```

\lineref \xlineref If the specified label exists, \lineref gives its line number.

```
%\def\lineref#1{\ref@undefined{#1}\getref@num{2}{#1}}
%\def\xlineref#1{\getref@num{2}{#1}}
\newcommand*{\lineref}[1]{\ref@undefined{#1}\getref@num{2}{#1}}
\newcommand*{\xlineref}[1]{\getref@num{2}{#1}}

```

\sublineref \xsublineref If the specified label exists, \sublineref gives its sub-line number.

```
%\def\sublineref#1{\ref@undefined{#1}\getref@num{3}{#1}}
%\def\xsublineref#1{\getref@num{3}{#1}}
\newcommand*{\sublineref}[1]{\ref@undefined{#1}\getref@num{3}{#1}}
\newcommand*{\xsublineref}[1]{\getref@num{3}{#1}}

```

\ref@undefined The next three macros are used by the referencing commands above, and do the job of extracting the right numbers from the label macro that contains the page, line, and sub-line number. The \ref@undefined macro is called when you refer to a label with the normal referencing macros. Its argument is a label, and it just checks that the labels file has indeed been opened and read (if not, it does so), and that the label is defined (if not, it squeals). (It is these checks which the x-forms of the reference macros leave out.)

```
%\def\ref@undefined#1{\do@labelsfile
\newcommand*{\ref@undefined}[1]{\do@labelsfile
  \expandafter\ifx\csname the@label#1\endcsname\relax
  \ledmac@warning{Reference `#1'
            on page \the\pageno\space undefined. Using `000'.}%
        \fi}

```

`\getref@num` Next, `\getref@num` fetches the number we want. It has two arguments: the first is simply a digit, specifying whether to fetch a page ("1"), line ("2") or sub-line ("3") number. (This switching is done by calling `\label@parse`.) The second argument is the label-macro, which because of the `\@lab` macro above is defined to be a string of the type `123|456|789`.

```
%\def\getref@num#1#2{%
\newcommand*{\getref@num}[2]{%
  \expandafter
  \ifx\csname the@label#2\endcsname \relax
    000%
  \else
    \expandafter\expandafter\expandafter
    \label@parse\csname the@label#2\endcsname|#1%
  \fi}

```

`\label@parse` Notice that we slipped another `|` delimiter into the penultimate line of `\getref@num`, to keep the "switch-number" separate from the reference numbers. This `|` is used as another parameter delimiter by `\label@parse`, which extracts the appropriate number from its first arguments. The `|`-delimited arguments consist of the expanded label-macro (three reference numbers), followed by the switch-number (`1`, `2`, or `3`) which defines which of the earlier three numbers to pick out. (It was earlier given as the first argument of `\getref@num`.)

```
\newcommand*{\label@parse}{}
\def\label@parse#1|#2|#3|#4{%
  \ifcase #4\relax
  \or #1%
  \or #2%
  \or #3%
  \fi}

```

`\xxref` The `\xxref` command takes two arguments, both of which are labels, e.g., `\xxref{mouse}{elephant}`. It first does some checking to make sure that the labels do exist (if one doesn't, those numbers are set to zero). Then it calls `\linenum` and sets the beginning page, line, and sub-line numbers to those of the place where `\label{mouse}` was placed, and the ending numbers to those at `\label{elephant}`. The point of this is to be able to manufacture footnote line references to passages which can't be specified in the normal way as the first argument to `\text` for one reason or another. Using `\xxref` in the second argument

of \text lets you set things up at least semi-automatically.

```
%\def\xxref#1#2{%
\newcommand*{\xxref}[2]{%
  {\expandafter\ifx\csname the@label#1\endcsname
   \relax \expandafter\let\csname the@label#1\endcsname\zz@@@\fi
   \expandafter\ifx\csname the@label#2\endcsname \relax
   \expandafter\let\csname the@label#2\endcsname\zz@@@\fi
   \linenum{\csname the@label#1\endcsname|%
    \csname the@label#2\endcsname}}}

```

\edmakelabel Sometimes the \edlabel command cannot be used to specify exactly the page and line desired; you can use the \edmakelabel macro make your own label. For example, if you say "\edmakelabel{elephant}{10|25|0}" you will have created a new label, and a later call to \edpageref{elephant} would print "10" and \lineref{elephant} would print "25". The sub-line number here is zero. \edmakelabel takes a label, followed by a page and a line number(s) as arguments. LaTeX defines a \makelabel macro which is used in lists. I've changed the name to \edmakelabel.

```
%\def\makelabel#1#2{\expandafter\xdef\csname the@label#1\endcsname{#2}}
\newcommand*{\edmakelabel}[2]{\expandafter\xdef\csname the@label#1\endcsname{#2}}

```

(If you are only going to refer to such a label using \xxref, then you can omit entries in the same way as with \linenum (see pp. 45 and 27), since \xxref makes a call to \linenum in order to do its work.)

11 Endnotes

\@end \ifend@ \end@true \end@false Endnotes of all varieties are saved up in a file, typically named ⟨*filename*⟩".end". \@end is the output stream number for this file, and \ifend@ is a flag that's true when the file is open.

```
\newwrite\@end
\newif\ifend@

```

\end@open \end@close \end@open and \end@close are the macros that are used to open and close the endnote file. Note that all our writing to this file is \immediate: all page and line numbers for the endnotes are generated by the same mechanism we use for the footnotes, so that there's no need to defer any writing to catch information from the output routine.

```
\def\end@open#1{\end@true\immediate\openout\@end=#1\relax}
\def\end@close{\end@false\immediate\closeout\@end}

```

\end@stuff \end@stuff is used by \beginnumbering to do everything that's necessary for the endnotes at the start of each section: it opens the \@end file, if necessary, and writes the section number to the endnote file.

```
\def\end@stuff{%
 \ifend@\relax\else
   \end@open{\jobname.end}%
 \fi
 \immediate\write\@end{\string\@section{\the\section@num}}}
```

`\Aendnote` `\Bendnote` `\Cendnote` `\Dendnote` `\Eendnote` The following five macros each function to write one endnote to the `.end` file. Like the footnotes, these endnotes come in five series, `A` through `E`. We change `\newlinechar` so that in the file every space becomes the start of a new line; this generally ensures that a long note doesn't exceed restrictions on the length of lines in files.

```
%\def\Aendnote#1{{\newlinechar='40
\newcommand*{\Aendnote}[1]{{\newlinechar='40
       \immediate\write\@end{\string\Aend%
                {\ifnumberedpar@\@nums\fi}%
                {\ifnumberedpar@\@tag\fi}{#1}}}\ignorespaces}

%\def\Bendnote#1{{\newlinechar='40
\newcommand*{\Bendnote}[1]{{\newlinechar='40
       \immediate\write\@end{\string\Bend%
                {\ifnumberedpar@\@nums\fi}%
                {\ifnumberedpar@\@tag\fi}{#1}}}\ignorespaces}

%\def\Cendnote#1{{\newlinechar='40
\newcommand*{\Cendnote}[1]{{\newlinechar='40
       \immediate\write\@end{\string\Cend%
                {\ifnumberedpar@\@nums\fi}%
                {\ifnumberedpar@\@tag\fi}{#1}}}\ignorespaces}

%\def\Dendnote#1{{\newlinechar='40
\newcommand*{\Dendnote}[1]{{\newlinechar='40
       \immediate\write\@end{\string\Dend%
                {\ifnumberedpar@\@nums\fi}%
                {\ifnumberedpar@\@tag\fi}{#1}}}\ignorespaces}

%\def\Eendnote#1{{\newlinechar='40
\newcommand*{\Eendnote}[1]{{\newlinechar='40
       \immediate\write\@end{\string\Eend%
                {\ifnumberedpar@\@nums\fi}%
                {\ifnumberedpar@\@tag\fi}{#1}}}\ignorespaces}
```

`\Aend` `\Bend` `\Cend` `\Dend` `\Eend` `\endprint` `\@gobblethree` `\@section` `\Aendnote` and the like write commands called `\Aend` and so on to the endnote file; these are analogous to the various `footfmt` commands above, and they take the same arguments. When we process this file, we'll want to pick out the notes of one series and ignore all the rest. To do that, we equate the `end` command for the series we want to `\endprint`, and leave the rest equated to `\@gobblethree`, which just skips over its three arguments. The `\endprint` here is nearly identical in its functioning to `\normalfootfmt`.

The endnote file also contains `\@section` commands, which supply the section numbers from the main text; standard `EDMAC` does nothing with this information, but it's there if you want to write custom macros to do something with it.

```
\def\endprint#1#2#3{{\notefontsetup{\notenumfont\printlines#1|}%
      \enspace{\select@lemmafont#1|#2}\enskip#3\par}}
%\def\@gobblethree#1#2#3{}
\newcommand*{\@gobblethree}[3]{}
\let\Aend=\@gobblethree
\let\Bend=\@gobblethree
\let\Cend=\@gobblethree
\let\Dend=\@gobblethree
\let\Eend=\@gobblethree
\let\@section=\@gobble

```

`\doendnotes` `\doendnotes` is the command you use to print one series of endnotes; it takes one argument, the series letter of the note series you want to print.

```
%\def\doendnotes#1{\end@close
\newcommand*{\doendnotes}[1]{\end@close
   \begingroup
      \makeatletter
      \expandafter\let\csname #1end\endcsname=\endprint
      \input\jobname.end
   \endgroup}
```

`\noendnotes` You can say `\noendnotes` before the first `\beginnumbering` in your file if you aren't going to be using any of the endnote commands: this will suppress the creation of an `.end` file. If you do have some lingering endnote commands in your file, the notes will be written to your terminal and to the TEX log file.

```
%\def\noendnotes{\global\let\end@stuff=\relax
\newcommand*{\noendnotes}{\global\let\end@stuff=\relax
                \global\chardef\@end=16 }
```

12 The End

⟨/code⟩

A Examples

This section presents some documents. The first one, in section A.1, was written natively in LaTeX, while the others were originally written for TeX. I have done some limited conversions of the later examples so that they look more like LaTeX code. In particular wherever possible I have replaced `\def` commands by either `\newcommand` or `\renewcommand` as appropriate. I have also replaced the original TeX font handling commands by the LaTeX font commands.

A.1 Simple example

This made-up example, `ledeasy.tex`, is included to show how simple it can be to use `EDMAC` in a LaTeX document. The code is given below and the result is shown in Figure 1.

```
⟨*easy⟩
% ledeasy.tex simple example of the ledmac package
\documentclass{article}
\usepackage{ledmac}
%% number every line
\setcounter{firstlinenum}{1}
\setcounter{linenumincrement}{1}
\title{Simple Example}
\author{Peter Wilson\thanks{Standing on the shoulders of giants.}}
\date{}
\begin{document}
\maketitle
\tableofcontents
\section{First}
   This is a simple example of using the \textsf{ledmac}
package wth ordinary LaTeX constructs.

\subsection{Example text}\label{subsec}

\beginnumbering
\pstart
The \textsf{ledmac} package lets you do some unusual things in
a LaTeX document. For example you can have lines numbered and
there are
\edtext{several}{\Afootnote{This is an ‘A’ footnote.}}
\edtext{levels}{\Bfootnote{This is a ‘B’ level footnote.}}
of footnotes.
You can label lines within the numbered text and refer to them
outside. Do not try and use any normal LaTeX footnoting or
\edtext{exotica}{\Afootnote{Like floats.}}
within the numbered portions of the text\edlabel{line}.
\pend
\endnumbering

```

Simple Example

Peter Wilson*

Contents

1 First

This is a simple example of using the ledmac package with ordinary LaTeX constructs.

1.1 Example text

1 The ledmac package lets you do some unusual things in a LaTeX document.
2 For example you can have lines numbered and there are several levels of foot-
3 notes. You can label lines within the numbered text and refer to them outside. Sidenotes are OK
4 Do not try and use any normal LaTeX marginpars[1] or exotica within the num-
5 bered portions of the text.

2 Last

I forgot to mention that you can use ordinary footnotes[2,3] outside the numbered text. You can also[a] have[b] formatted footnotes[c] in normal[d] text.

There are 5 numbered lines in the example shown in section 1.1.

*Standing on the shoulders of giants.
[1]You will get a warning but no text.
[2]An ordinary footnote
[3]And another

[a]Additionally [b]Specify [c]Like this [d]Text that does not have line numbers

2 several] This is an 'A' footnote.
4 exotica] Like floats.

2 levels] This is a 'B' level footnote.

Figure 1: Output from `ledeasy.tex`.

```
\section{Last}

    I forgot to mention that you can use ordinary
footnotes\footnote{An ordinary footnote}
outside the numbered text. There are
\lineref{line} numbered lines in the example shown
in section~\ref{subsec}.

\end{document}
```
⟨/easy⟩

A.2 General example of features

This made-up example, `ledfeat.tex`, is included purely to illustrate some of EDMAC's main features. It is hard to find real-world examples that actually use as many layers of notes as this, so we made one up. The example is a bit tricky to read, but close study and comparison with the output (Figure 2) will be illuminating.

I have converted the original TeX code to look more like LaTeX code.

⟨∗features⟩
```
% ledfeat.tex  Small test file for ledmac package
\documentclass{article}
\usepackage{ledmac}

 \makeatletter
 % I'd like a spaced out colon after the lemma:
 \newcommand{\spacedcolon}{{\rmfamily\thinspace:\thinspace}}
 \renewcommand*{\normalfootfmt}[3]{%
   \normal@pars
   \parindent=0pt \parfillskip=0pt plus 1fil
   {\notenumfont\printlines#1|}\strut\enspace
   {\select@lemmafont#1|#2}\spacedcolon\enskip#3\strut\par}

 % And I'd like the 3-col notes printed with a hanging indent:
 \renewcommand*{\threecolfootfmt}[3]{%
   \normal@pars
   \hsize .3\hsize
   \parindent=0pt
   \tolerance=5000       % high, but not infinite
   \raggedright
   \hangindent1.5em \hangafter1
   \leavevmode
   \strut\hbox to 1.5em{\notenumfont\printlines#1|\hfil}\ignorespaces
   {\select@lemmafont#1|#2}\rbracket\enskip
   #3\strut\par\allowbreak}

 % And I'd like the 2-col notes printed with a double colon:
 \newcommand*{\doublecolon}{{\rmfamily\thinspace::\thinspace}}
 \renewcommand*{\twocolfootfmt}[3]{%
```

This is an example of some text with variant readings recorded as ‘A’ footnotes. From here on, though, we shall have ‘C’. For spice, let us mark a longer passage, but give a different lemma for it, so that we don’t get a huge amount of text in a note. Finally, we shouldn’t forget the paragraphed notes, which are so useful when there are a great number of short notes to be recorded.

This is a second paragraph, giving more *examples* of text with variant readings recorded as ‘A’ footnotes. From here on, though, we shall have ‘B’ notes in the text. For spice, let us mark a longer passage, but give a different lemma for it, so that we don’t get a *huge* amount of text in a note. Finally, we shouldn’t forget the column notes, which are so useful when there are many short notes to be recorded.

1 example :: eximemple C, D.
1 variant :: alternative, A, B.
2 though :: however α, β
6 *examples* :: eximples L, M.
6 variant :: alternative, A, B.

2 ‘C’] B, *pace* the text
7 though] however α, β
7 ‘B’] B, as correctly stated in the text
9 Finally] In the end X, Y
9 we] we here K
9 shouldn’t] ought not to L, M
10 forget the] omit to mention the §, ¶
10 column] blocked M, N
10 notes] variants H
10 useful] very, very useful L, P
10 many] lots of Z
11 recorded] recorded and put down: M (repetition)

2–4 For spice ... note: The note here is type ‘C’
8–9 For spice, ... note: This is a rogue note of type ‘C’.

3 huge: vast E, F; note that this is a ‘D’ note to section of text within a longer lemma
9 *huge*: vast E, F; note that this is a ‘D’ note to text within a longer lemma.

4 Finally: in the end X, Y **4** we: us K **4** shouldn’t: ought not to L, M **4** forget the: omit to mention the §, ¶ **4** paragraphed: blocked M, N **4** notes: variants HH, KK **5** useful: truly useful L, P **5** a great number of: many, many (preferably) **5** recorded: noted: repetition

Figure 2: Output from `ledfeat.tex`.

```
  \normal@pars
  \hsize .45\hsize
  \parindent=0pt
  \tolerance=5000
  \raggedright
  \leavevmode
  \strut{\notenumfont\printlines#1|}\enspace
  {\select@lemmafont#1|#2}\doublecolon\enskip
  #3\strut\par\allowbreak}

% And in the paragraphed footnotes, I'd like a colon too:
\renewcommand*{\parafootfmt}[3]{%
  \normal@pars
  \parindent=0pt \parfillskip=0pt plus 1fil
  {\notenumfont\printlines#1|}\enspace
  {\select@lemmafont#1|#2}\spacedcolon\enskip
  #3\penalty-10 }
\makeatother

% I'd like the line numbers picked out in bold.
\renewcommand{\notenumfont}{\bfseries}
\lineation{page}
\linenummargin{inner}
\setcounter{firstlinenum}{3}       % just because I can
\setcounter{linenumincrement}{1}
\foottwocol{A}
\footthreecol{B}
\footparagraph{E}
% I've changed \normalfootfmt, so invoke it again for C and D notes.
\footnormal{C}
\footnormal{D}

\begin{document}

\beginnumbering

\pstart
This is an \text{example}
  \Afootnote{eximemple C, D.}/
of some %\footnote{A normal footnote}
text with \text{variant}
  \Afootnote{alternative, A, B.}/
readings recorded as `A' footnotes.  From here on, \text{though}
  \Afootnote{however $\alpha$, $\beta$}/,
we shall have \text{`C'}
  \Bfootnote{B, \textit{pace} the text}/.
\text{For spice, let us mark a longer passage, but give a different
  lemma for it, so that we don't get a \text{huge}
    \Dfootnote{vast E, F; note that this is
    a `D' note to section of text within a longer lemma}/
```

```
  amount of text in a note}\lemma{For spice \dots\ note}
  \Cfootnote{The note here is type ‘C’}/.
\text{Finally}
  \Efootnote{in the end X, Y}/,
\text{we}
  \Efootnote{us K}/
\text{shouldn’t}
  \Efootnote{ought not to L, M}/
\text{forget the}
  \Efootnote{omit to mention the \S, \P}/
\text{paragraphed}
  \Efootnote{blocked M, N}/
\text{notes}
  \Efootnote{variants HH, KK}/,
which are so \text{useful}
  \Efootnote{truly useful L, P}/
when there are \text{a great number of}
  \Efootnote{many, many (preferably)}/
short notes to be \text{recorded}
  \Efootnote{noted: repetition}/.
\pend

```

For comparison purposes, the second paragraph uses the \edtext macro rather than the \text macro.

```
\pstart
This is a second paragraph, giving more \textit{\edtext{examples}{
  \Afootnote{eximples L, M.}}}
of text with \edtext{variant}{
  \Afootnote{alternative, A, B.}}
readings recorded as ‘A’ footnotes.  From here on, \edtext{though}{
  \Bfootnote{however $\alpha$, $\beta$}},
we  shall have \edtext{‘B’}{
  \Bfootnote{B, as correctly stated in the text}} notes in the text.
\edtext{For spice, let us mark a longer passage, but give a different
  lemma for it, so that we don’t get a \textit{\edtext{huge}{
    \Dfootnote{vast E, F; note that this is
    a ‘D’ note to text within a longer lemma.}}}
  amount of text in a note}{\lemma{For spice, \dots\ note}
  \Cfootnote{This is a rogue note of type ‘C’.}}.
\edtext{Finally}{
  \Bfootnote{In the end X, Y}},
\edtext{we}{
  \Bfootnote{we here K}}
\edtext{shouldn’t}{
  \Bfootnote{ought not to L, M}}
\edtext{forget the}{
  \Bfootnote{omit to mention the \S, \P}}
\edtext{column}{
  \Bfootnote{blocked M, N}}
```

```
\edtext{notes}{
  \Bfootnote{variants H}},
which are so \edtext{useful}{
  \Bfootnote{very, very useful L, P}}
when there are \edtext{many}{
  \Bfootnote{lots of Z}}
short notes to be \edtext{recorded}{
  \Bfootnote{recorded and put down: M (repetition)}}.
\pend

\endnumbering
\end{document}
⟨/features⟩
```

A.3 Gascoigne

The first real-life example is taken from an edition of George Gascoigne's *A Hundreth Sundrie Flowres* that is being prepared by G. W. Pigman III, at the California Institute of Technology. Figure 3 shows the result of setting the text with EDMAC.

I have LaTeXified the original code, and removed all the code related to the main document layout, relying on the standard LaTeX layout parameters..

```
⟨*ioc⟩
%% ledioc.tex
\documentclass{article}
\usepackage{ledmac}

\noendnotes
\makeatletter

\newcommand{\os}{\scriptsize}
\setcounter{firstsublinenum}{1000}
\frenchspacing \parskip=0pt \hyphenpenalty=1000

% Say \nolinenums if you want no line numbers in the notes.
\newif\ifnolinenums
\newcommand{\nolinenums}{\global\nolinenumstrue}
\newcommand{\linenums}{\global\nolinenumsfalse}

\renewcommand{\rightlinenum}{\ifbypage@\ifnum\line@num<10\kern.5em\fi\else
\ifnum\line@num<10\kern1em\else\ifnum\line@num<100
  \kern.5em\fi\fi\fi\kern.5em\numlabfont\the\line@num
  \ifnum\subline@num>0:\the\subline@num\fi}

\renewcommand{\leftlinenum}{\numlabfont\the\line@num
  \ifnum\subline@num>0:\the\subline@num\fi \kern.5em}
\linenummargin{outer}
```

Oedipus entreth.
Or that with wrong the right and doubtlesse heire,
Shoulde banisht be out of his princely seate.
Yet thou O queene, so fyle thy sugred toung,
And with suche counsell decke thy mothers tale,
That peace may bothe the brothers heartes inflame,
And rancour yelde, that erst possest the same.
Eteocl. Mother, beholde, youre hestes for to obey,
In person nowe am I resorted hither:
In haste therefore, fayne woulde I knowe what cause
With hastie speede, so moued hath your mynde
To call me nowe so causelesse out of tyme,
When common wealth moste craues my onely ayde:
Fayne woulde I knowe, what queynt commoditie
Persuades you thus to take a truce for tyme,
And yelde the gates wide open to my foe,
The gates that myght our stately state defende,
And nowe are made the path of our decay.
„ *Ioca.* Represse deare son, those raging stormes of wrath,
„That so bedimme the eyes of thine intente,
„As when the tongue (a redy Instrument)
„Would fayne pronounce the meaning of the minde,
„It cannot speake one honest seemely worde.
„But when disdayne is shrunke, or sette asyde,
„And mynde of man with leysure can discourse
„What seemely woordes his tale may best beseeme,
„And that the toung vnfoldes without affectes
„Then may proceede an answere sage and graue,
„And euery sentence sawst with sobernesse:
Wherefore vnbende thyne angrie browes deare chylde,
And caste thy rolling eyes none other waye,
That here doost not *Medusaes* face beholde,
But him, euen him, thy blood and brother deare.
And thou beholde, my *Polinices* eke,
Thy brothers face, wherin when thou mayst see
Thine owne image, remember therwithall,
That what offence thou woldst to him were done,

0.1 entreth] *intrat* MS 20–22 As ... worde.] *not in* 73 20 the] thie MS 21 fayne pronounce] faynest tell MS 21 the minde] thy minde MS 22 It ... worde.] Thie swelling hart puft vp with wicked ire / Can scarce pronounce one inward louing thought. MS 31 *Medusaes*] One of the furies. 75m

Figure 3: Output from `ledioc.tex`.

```
\lineation{page}

\newcommand{\ggfootfmt}[3]{%
  \notefontsetup
  \let\par=\endgraf
  \rightskip=0pt \leftskip=0pt
  \parindent=0pt \parfillskip=0pt plus 1fil
  \ifnolinenums\relax\else
    \begingroup \os \printlines#1|\endgroup
    \enskip
  \fi
  {\rmfamily #2\def\@tempa{#2}\ifx\@tempa\empty
    \else]\enskip\fi#3\penalty-10 }}

% Now reset the \Afootnote parameters and macros:
\footparagraph{A}
\let\Afootfmt=\ggfootfmt
\dimen\Afootins=\vsize
\skip\Afootins=3pt plus9pt
\newcommand*{\ggfootstart}[1]{\vskip\skip\Afootins}
\let\Afootstart=\ggfootstart

\newcommand*{\stage}[1]{\pstart\startsub\parindent=0pt
  \hangindent=3em\hangafter=0
  {\itshape #1}\let\par=\finishstage}
\newcommand{\finishstage}{\pend\endsub}
\newcommand{\sen}{\leavevmode\lower1ex\hbox{\textrm{''}}}
\newcommand{\senspeak}[1]{\pstart\obeylines\setbox0=\hbox{\textrm{''}}%
  \leavevmode
  \lower1ex\copy0\kern-\wd0\hskip1em{\textit{#1}}%
  \hbox to1ex{}\ignorespaces}
\newcommand*{\speak}[1]{\pstart\obeylines\hskip1em{\textit{#1}}%
  \hbox to1ex{}\ignorespaces}
\def\nospeaker{\parindent=0em\pstart\let\par=\pend}
\newcommand*{\nospeak}{\pstart\obeylines}
\makeatother

\begin{document}

\setlength{\parindent}{0pt}

\beginnumbering

\stage{Oedipus \edtext{entreth}{\Afootnote{\textit{intrat} MS}}.}

\nospeak
Or that with wrong the right and doubtlesse heire,
Shoulde banisht be out of his princely seate.
Yet thou O queene, so fyle thy sugred toung,
And with suche counsell decke thy mothers tale,
```

```
That peace may bothe the brothers heartes inflame,
And rancour yelde, that erst possest the same.
\pend

\speak{Eteocl.} Mother, beholde, youre hestes for to obey,
In person nowe am I resorted hither:
In haste therefore, fayne woulde I knowe what cause
With hastie speede, so moued hath your mynde
To call me nowe so causelesse out of tyme,
When common wealth moste craues my onely ayde:
Fayne woulde I knowe, what queynt commoditie
Persuades you thus to take a truce for tyme,
And yelde the gates wide open to my foe,
The gates that myght our stately state defende,
And nowe are made the path of our decay.
\pend

\senspeak{Ioca.}Represse deare son, those raging stormes of wrath,
\sen That so bedimme the eyes of thine intente,
\edtext{\sen As when \edtext{the}{\Afootnote{thie MS}} tongue %
  (a redy Instrument)
\sen Would \edtext{fayne pronounce}{\Afootnote{faynest tell MS}} %
  the meaning of \edtext{the minde}{\Afootnote{thy minde MS}},
\sen \edtext{It}{\lemma{It \dots\ worde.}\Afootnote{Thie %
  swelling hart puft vp with wicked ire / Can scarce pronounce %
  one inward louing thought. MS}} cannot speake one honest %
  seemely worde.}{\lemma{As \dots\ worde.}\Afootnote{\textit{not %
  in} \os73}}
\sen But when disdayne is shrunke, or sette asyde,
\sen And mynde of man with leysure can discourse
\sen What seemely woordes his tale may best beseeme,
\sen And that the toung vnfoldes without affectes
\sen Then may proceede an answere sage and graue,
\sen And euery sentence sawst with sobernesse:
Wherefore vnbende thyne angrie browes deare chylde,
And caste thy rolling eyes none other waye,
That here doost not \edtext{\textit{Medusaes}}{%
\Afootnote{One of the furies. {\os75}m}} face beholde,
But him, euen him, thy blood and brother deare.
And thou beholde, my \textit{Polinices} eke,
Thy brothers face, wherin when thou mayst see
Thine owne image, remember therwithall,
That what offence thou woldst to him were done,
\pend
\endnumbering

\end{document}

⟨/ioc⟩
```

A.4 Shakespeare

The following text illustrates another input file of moderate complexity, with two layers of annotation in use. The example is taken from the Arden *Merchant of Venice*.

I have roughly converted the original TeX file to a LaTeX file. The file is below and the result of LaTeXing it is shown in Figure 4.

```
⟨*arden⟩
%% ledarden.tex
\documentclass{article}
\usepackage{ledmac}

\makeatletter
 \newcommand{\stage}[1]{\rlap{\hbox to \the\linenumsep{%
                         \hfil\llap{[\textit{#1}]}}}}

 \newcommand{\speaker}[1]{\pstart\hangindent2em\hangafter1
   \leavevmode\textit{#1}\enspace\ignorespaces}

 \newcommand{\exit}[1]{\hfill\stage{#1}}

 % EDMAC customizations:
 \noendnotes
 \vsize 40pc
 \hsize 23pc
 \parindent 0pt
 \linenumsep=.4in
 \rightskip\linenumsep

 \renewcommand{\interparanoteglue}{1em plus.5em minus.1em}

 \catcode`\<=\active
 \def\xtext#1#2>{\text{#1}{#2}/}
 \let<=\xtext

 \newcommand{\scf}{\tiny}
 \let\Afootnoterule=\relax \let\Bfootnoterule=\relax

 \renewcommand{\rightlinenum}{\numlabfont\llap{\the\line@num}}
 \frenchspacing

 % Footnote formats:
 % \nonumparafootfmt is a footnote format without line numbers.
 \newcommand{\nonumparafootfmt}[3]{%
   \normal@pars
   \rightskip=0pt
   \parindent=0pt \parfillskip=0pt plus 1fil
   \select@lemmafont#1|#2\rbracket\enskip
   \it#3\penalty-10 }
```

[SCENE III.—*Venice*.]

Enter JESSICA *and* [LAUNCELOT] *the clown.*

Jes. I am sorry thou wilt leave my father so,
Our house is hell, and thou (a merry devil)
Didst rob it of some taste of tediousness,—
But fare thee well, there is a ducat for thee,
And Launcelot, soon at supper shalt thou see
Lorenzo, who is thy new master's guest,
Give him this letter,—do it secretly,—
And so farewell: I would not have my father
See me in talk with thee.

Laun. Adieu! tears exhibit my tongue, most beautiful pagan, most sweet Jew!—if a Christian do not play the knave and get thee, I am much deceived; but adieu! these foolish drops do something drown my manly spirit: adieu! [*Exit.*]

Jes. Farewell good Launcelot.
Alack, what heinous sin is it in me
To be ashamed to be my father's child!

Scene III] *Capell; om. Q, F;* Scene IV *Pope. Venice*] *om. Q, F; Shylock's house Theobald; The same. A Room in Shylock's House Capell.* Launcelot] *Rowe; om. Q, F.* 1. I am] *Q, F;* I'm *Pope.* 9. in] *Q; om. F.* 10. *Laun.*] *Q2; Clowne. Q, F.* 10. Adieu!] Adiew, *Q, F.* 11. Jew!] Iewe, *Q, F.* do] *Q, F;* did *F2.* 12. adieu!] adiew, *Q, F.* 12. something] *Q;* somewhat *F.* 13. adieu!] adiew. *Q, F.* S. D.] *Q2, F; om. Q; after l. 15 Capell.* 16. child!] child, *Q, F;* Child? *Rowe.*

5. *soon*] early.

10. *exhibit*] Eccles paraphrased "My tears serve to express what my tongue should, if sorrow would permit it," but probably it is Launcelot's blunder for prohibit (Halliwell) or inhibit (Clarendon).

10. *pagan*] This may have a scurrilous undertone: cf. *2 H 4,* II. ii. 168.

11. *do*] Malone upheld the reading of Qq and F by comparing II. vi. 23: "When you shall please to play the thieves for wives"; Launcelot seems fond of hinting at what is going to happen (cf. II. v. 22–3). If F2's "did" is accepted, *get* is used for beget, as in III. v. 9.

12–13. *foolish. . . spirit*] "tears do not become a man" (*AYL.*, III. iv. 3); cf. also *H 5,* IV. vi. 28–32.

Figure 4: Output from `ledarden.tex`.

```
1818
1819 \newcommand{\newparafootfmt}[3]{%
1820   \normal@pars
1821   \parindent=0pt \parfillskip=0pt plus 1fil
1822   {\notenumfont\printlines#1|}\fullstop\enspace
1823   {\select@lemmafont#1|#2}\rbracket\enskip
1824   \it#3\penalty-10 }
1825
1826 \newcommand{\newtwocolfootfmt}[3]{%
1827   \normal@pars
1828   \hsize .48\hsize
1829   \tolerance=5000
1830   \rightskip=0pt \leftskip=0pt \parindent=5pt
1831   \strut\notenumfont\printlines#1|\fullstop\enspace
1832   \it#2\/\rbracket\penalty100\hskip .5em plus .5em
1833   \rm#3\strut\goodbreak}
1834
1835 % Footnote style selections etc. (done last):
1836 \footparagraph{A}
1837 \foottwocol{B}
1838 \let\Afootfmt=\newparafootfmt
1839 \let\Bfootfmt=\newtwocolfootfmt
1840 \let\collation=\Afootnote
1841 \let\note=\Bfootnote
1842 \lineation{section}
1843 \linenummargin{right}
1844 \makeatother
1845
1846 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
1847
1848 \begin{document}
1849 \pagestyle{empty}
1850
1851 % Initially, we don't want line numbers.
1852 \let\Afootfmt=\nonumparafootfmt
1853
1854 \beginnumbering
1855 \pstart
1856 \centerline{[<{SCENE III}
1857   \lemma{Scene III}
1858   \collation{Capell; om. Q, F; \textnormal{Scene IV} Pope.}>.---%
1859   <{\textit{Venice}}
1860   \collation{om. Q, F; Shylock's house Theobald; The same.
1861   A Room in Shylock's House Capell.}>.]}
1862 \pend
1863 \bigskip
1864
1865 \pstart
1866 \centerline{\textit{Enter} JESSICA \textit{and}
1867   [<{LAUNCELOT}
```

```
  \lemma{Launcelot}
  \collation{Rowe; om. Q, F.}>] \textit{the clown.}} \pend \bigskip

\let\Afootfmt=\newparafootfmt % we do want line numbers from now

 \setline{0}%

\speaker{Jes.}<{I am}
  \collation{Q, F; \textnormal{I'm} Pope.}>
                       sorry thou wilt leave my father so,\\
Our house is hell, and thou (a merry devil)\\
Didst rob it of some taste of tediousness,---\\
But fare thee well, there is a ducat for thee,\\
And Launcelot, <{soon}
  \note{early.}>
                        at supper shalt thou see\\
Lorenzo, who is thy new master's guest,\\
Give him this letter,---do it secretly,---\\
And so farewell: I would not have my father\\
See me <{in}
  \collation{Q; om. F.}>
             talk with thee.
\pend

\speaker{Laun.}
  <{}\lemma{\it Laun.}\collation{Q2; Clowne. Q, F.}>%
<{Adieu!}
  \collation{\textnormal{Adiew}, Q, F.}>
tears <{exhibit}
  \note{Eccles paraphrased ``My tears serve to express what my
  tongue should, if sorrow would permit it,'' but probably it is
  Launce\-lot's blunder for prohibit (Halliwell) or inhibit
  (Clarendon).}>
my tongue, most beautiful <{pagan}
  \note{This may have a scurrilous undertone: cf. \textit{2 H 4,}
  {\scf II.} \textrm{ii. 168.}}>%
, most sweet <{Jew!}
  \collation{\textnormal{Iewe}, Q, F. \quad \textnormal{do]} Q, F;
             \textnormal{did} F2.}>%
---if a Christian <{do}
  \note{Malone upheld the reading of Qq and F by comparing {\scf II.}
   vi. 23: ``When you shall please to play the thieves for
  wives''; Launcelot seems fond of hinting at what is going to
  happen (cf. {\scf II.} v. 22--3). If F2's ``did'' is accepted,
  \textit{get} is used for beget, as in {\scf III.} v. 9.}>
not play the knave and get thee, I am much deceived; but <{adieu!}
  \collation{\textnormal{adiew}, Q, F.}>
these <{foolish drops do \text{something}
  \collation{Q; \textnormal{somewhat} F.}/
drown my manly spirit}
```

```
  \lemma{foolish\textnormal{\dots}spirit}
  \note{``tears do not become a man'' (\textit{AYL.}, {\scf III.}
  iv. 3); cf. also \textit{H 5,} {\scf IV.} vi. 28--32.}>%
: <{adieu!}
  \collation{\textnormal{adiew}. Q, F. \quad \textnormal{S. D.]} Q2, F; om. Q;
  after l. 15 Capell.}>
\exit{Exit.}
\pend

\speaker{Jes.}
Farewell good Launcelot.\\
Alack, what heinous sin is it in me\\
To be ashamed to be my father's <{child!}
  \collation{\textnormal{child}, Q, F; \textnormal{Child?} Rowe.}>
\pend
\endnumbering

\end{document}

⟨/arden⟩
```

A.5 Classical text edition

The next example, which was extracted from a longer file kindly supplied by Wayne Sullivan, University College, Dublin, Ireland, illustrates the use of EDMAC to produce a Latin text edition, the *Periphyseon*, with Greek passages.[26] The Greek font used is that prepared by Silvio Levy and described in *TUGboat*.[27] The output of this file is shown in Figure 5. Note the use of two layers of footnotes to record testimonia and manuscript readings respectively.

I have converted the original EDMAC example file from TeX to something that looks more like LaTeX.

```
⟨*periph⟩
% ledmixed.tex
\documentclass{article}
\usepackage{ledmac}

 \overfullrule0 pt
 \lefthyphenmin=3

```

The LaTeX version uses the lgreek package to access Silvio Levy's greek font. The delims package option subverts[28] the normal meaning of $ to switch in and out of

[26] The bibliographic details of the forthcoming book are: Iohannis Scotti Erivgenae, *Periphyseon* (*De Diuisione Naturae*) Liber Qvartvs [Scriptores Latini Hiberniae vol. xii], (Dublin: School of Celtic Studies, Dublin Institute for Advanced Studies, forthcoming 1992).

[27] *TUGboat* **9** (1988), pp. 20–24.

[28] It actually changes its category code.

Incipit Quartus ΠΕΡΙΦΥΣΕΩΝ 741C

ΑΝΑΚΕΦΑΛΙΟΣΙΣ NATVRARVM

NVTRITOR. Prima nostrae Physiologiae intentio praecipuaque materia erat quod ΥΠΕΡΟΥΣΙΑΔΕΣ (hoc est superessentialis) natura sit causa creatrix existentium et non existentium omnium, a nullo creata, unum principium, una origo, unus et uniuersalis uniuersorum fons, a nullo manans, dum ab eo manant omnia, trinitas coessentialis in tribus substantiis, ΑΝΑΡΧΟΣ (hoc est sine principio), principium et finis, una bonitas, deus unus, ΟΜΟΥΣΙΟΣ et ΥΠΕΡΟΥΣΙΟΣ (id est coessentialis et superessentialis). Et, ut ait sanctus Epifanius, episcopus Constantiae Cypri, in ΑΓΚΥΡΑΤΩ sermone de fide: *Tria sancta, tria consancta, tria agentia, tria coagentia, tria formantia, tria conformantia, tria operantia, tria cooperantia, tria subsistentia, tria consubsistentia sibi inuicem* 742C *coexistentia. Trinitas haec sancta uocatur: tria existentia, una consonantia, una deitas eiusdem essentiae, eiusdem uirtutis, eiusdem subsistentiae, similia similiter aequalitatem gratiae operantur patris et filii et sancti spiritus. Quomodo autem sunt, ipsis relinquitur docere: 'Nemo enim nouit patrem nisi filius, neque filium nisi pater, et cuicumque filius reuelauerit'; reuelatur autem per spiritum sanctum. Non ergo haec tria existentia aut ex ipso aut per ipsum aut ad ipsum in unoquoque digne intelliguntur,* $|R, 264^{\mathrm{r}}|$ *sicut ipsa reuelant:* ΦΩΣ, ΠΥΡ, ΠΝΕΥΜΑ (hoc est lux, ignis, spiritus).

Haec, ut dixi, ab Epifanio tradita, ut quisquis interrogatus quae tria et quid unum in sancta trinitate debeat credere, sana fide$|\ J, 1^{\mathrm{v}}|$ respondere ualeat, aut ad fidem accedens sic erudiatur. Et mihi uidetur spiritum pro calore posuisse, 743A quasi dixisset in similitudine: lux, ignis, calor. Haec enim tria unius essentiae sunt. Sed cur lucem primo dixit, non est mirum. Nam et pater lux est et ignis et calor; et filius est lux, ignis, calor; et spiritus sanctus lux, ignis, calor. Illuminat enim pater, illuminat filius, illuminat spiritus sanctus: ex ipsis enim omnis scientia et sapientia donatur.

15–16 Matth. 11, 27 19 EPIPHANIVS, *Ancoratus* 67; PG 43, 137C–140A; GCS 25, p. 82, 2–12

1 incipit . . . ΠΕΡΙΦΥΣΕΩΝ] *om. R*, incipit quartus *M* 2 ΑΝΑΚΕΦΑΛΙΟΣΙΣ] *FJP, lege* ἀνακεφαλαίωσις 2 physiologiae] phisiologiae *P*, physeologiae *R* 3 quod] *p.* natura *transp. MR* 3 ΥΠΕΡΟΥΣΙΑΔΕΣ] *codd. Vtrum* ὑπερουσιώδης (hoc est superessentialis) natura *cum Gale (p.160) an* ὑπερουσιότης (hoc est superessentialis natura) *cum Floss (PL 122,741C) intelligendum sit, ambigitur* 7 ΟΜΟΥΣΙΟΣ] *codd., lege* ὁμοούσιος 7 **et**] R^1, *om.* R^0 9 ΑΓΚΥΡΑΤΩ] anchurato *MR* 9 de fide] Glo⟨ssa⟩: Ita enim uocatur sermo eius de fide ΑΓΚΥΡΑΤΟΣ, id est procuratus *mg. add. FJP* 10 *agentia*] *actiua MR* 10 *formantia*] *formatiua MR* 11 *operantia*] *operatiua MR* 13 *eiusdem*] *eiusdemque M* 13 *eiusdem uirtutis, eiusdem subsistentiae*] *om. M* 13 *subsistentiae*] *substantiae R* 14 *similiter*] *ex simili MR* 15 *sunt*] *om. M* 25 spiritus sanctus] sanctus spiritus *R*

Figure 5: Output from `ledmixed.tex`.

math mode. We have to save the original meaning of $ before calling the package. Later, we use \Ma and \aM for math mode switching.

```
\let\Ma=$
\let\aM=$
\usepackage[delims]{lgreek}

% We need an addition to \no@expands since the \active $ in lgreek
% causes problems:
\newcommand{\morenoexpands}{\let$=0}

\makeatletter

\newbox\lp@rbox

\newcommand{\ffootnote}[1]{%
  \ifnumberedpar@
    \xright@appenditem{\noexpand\vffootnote{f}{{\@nums}{\@tag}{#1}}}%
                                                \to\inserts@list
    \global\advance\insert@count by 1
%  \else         %% may be used only in numbered text
%    \vffootnote{f}{{0|0|0|0|0|0|0}{}{#1}}%
  \fi\ignorespaces}

\newcommand{\gfootnote}[1]{%
  \ifnumberedpar@
    \xright@appenditem{\noexpand\vgfootnote{g}{#1}}%
                                                \to\inserts@list
    \global\advance\insert@count by 1
%  \else         %% may be used only in numbered text
%    \vgfootnote{g}{#1}%
  \fi\ignorespaces}

\newcommand{\setlp@rbox}[3]{%
  {\parindent\z@\hsize=2.5cm\raggedleft\eightpoint
  \baselineskip 9pt%
  \global\setbox\lp@rbox=\vbox to\z@{\vss#3}}}

\newcommand{\vffootnote}[2]{\setlp@rbox#2}

\newcommand{\vgfootnote}[2]{\def\rd@ta{#2}}

\renewcommand{\do@line}{%
 {\vbadness=10000 \splittopskip=0pt
 \gdef\rd@ta{}% for right margin paragraph->always a few characters
 \global\setbox\one@line=\vsplit\raw@text to\baselineskip}%
 \unvbox\one@line \global\setbox\one@line=\lastbox
 \getline@num
\hbox to\hsize{\affixline@num\add@inserts\hbox to\z@% inserts added here so
  {\hss\box\lp@rbox\kern\linenumsep}%                that margin pars are
```

```
1993    \hfil\hbox to\wd\one@line{\new@line\unhbox\one@line%   included.
1994       \hbox to\z@{\kern\linenumsep\notenumfont\rd@ta\hss}}%
1995  \add@penalties} % margin pars also included in line format
1996
1997 \renewcommand{\affixline@num}{%
1998   \ifsublines@
1999     \@tempcntb=\subline@num
2000     \ifnum\subline@num>\c@firstsublinenum
2001       \@tempcnta=\subline@num
2002       \advance\@tempcnta by-\c@firstsublinenum
2003       \divide\@tempcnta by\c@sublinenumincrement
2004       \multiply\@tempcnta by\c@sublinenumincrement
2005       \advance\@tempcnta by\c@firstsublinenum
2006     \else
2007       \@tempcnta=\c@firstsublinenum
2008     \fi
2009     %
2010     \ifcase\sub@lock
2011       \or
2012         \ifnum\sublock@disp=1
2013            \@tempcntb=0 \@tempcnta=1
2014         \fi
2015       \or
2016         \ifnum\sublock@disp=2 \else
2017            \@tempcntb=0 \@tempcnta=1
2018         \fi
2019       \or
2020         \ifnum\sublock@disp=0
2021            \@tempcntb=0 \@tempcnta=1
2022         \fi
2023     \fi
2024   \else
2025     \@tempcntb=\line@num
2026     \ifnum\line@num>\c@firstlinenum
2027        \@tempcnta=\line@num
2028        \advance\@tempcnta by-\c@firstlinenum
2029        \divide\@tempcnta by\c@linenumincrement
2030        \multiply\@tempcnta by\c@linenumincrement
2031        \advance\@tempcnta by\c@firstlinenum
2032     \else
2033        \@tempcnta=\c@firstlinenum
2034     \fi
2035     \ifcase\@lock
2036        \or
2037          \ifnum\lock@disp=1
2038             \@tempcntb=0 \@tempcnta=1
2039          \fi
2040        \or
2041          \ifnum\lock@disp=2 \else
2042             \@tempcntb=0 \@tempcnta=1
```

```
          \fi
        \or
          \ifnum\lock@disp=0
             \@tempcntb=0 \@tempcnta=1
          \fi
    \fi
  \fi
  %
  \ifnum\@tempcnta=\@tempcntb
    \@tempcntb=\line@margin
    \ifnum\@tempcntb>1
      \advance\@tempcntb by\page@num
    \fi
    \ifodd\@tempcntb
%      #1\rlap{{\rightlinenum}}%
       \xdef\rd@ta{\the\line@num}%
    \else
      \llap{{\leftlinenum}}%#1%
    \fi
  \else
    %#1%
  \fi
  \ifcase\@lock
  \or
    \global\@lock=2
  \or \or
    \global\@lock=0
  \fi
  \ifcase\sub@lock
  \or
    \global\sub@lock=2
  \or \or
    \global\sub@lock=0
  \fi}

\newcommand{\eightpoint}{\scriptsize}

 \lineation{page}
 \linenummargin{right}
 \footparagraph{A}
 \footparagraph{B}

\renewcommand{\notenumfont}{\footnotesize}
\newcommand{\notetextfont}{\footnotesize}

 \let\Afootnoterule=\relax
 \count\Afootins=825
 \count\Bfootins=825

 \newcommand{\Aparafootfmt}[3]{%
```

```
  \normal@pars\eightpoint
  \parindent=0pt \parfillskip=0pt plus1fil
  \notenumfont\printlines#1|\enspace
%      \lemmafont#1|#2\enskip
  \notetextfont
  #3\penalty-10\hskip 1em plus 4em minus.4em\relax}

\newcommand{\Bparafootfmt}[3]{%
  \normal@pars\eightpoint
  \parindent=0pt \parfillskip=0pt plus1fil
  \notenumfont\printlines#1|\enspace
  \select@lemmafont#1|#2\rbracket\enskip
  \notetextfont
  #3\penalty-10\hskip 1em plus 4em minus.4em\relax }
\makeatother

\let\Afootfmt=\Aparafootfmt
\let\Bfootfmt=\Bparafootfmt
\def\lemmafont#1|#2|#3|#4|#5|#6|#7|{\eightpoint}
\parindent=1em

\newcommand{\lmarpar}[1]{\text{}\ffootnote{#1}/}

\newcommand{\rmarpar}[1]{\text{}\gfootnote{#1}/}
\emergencystretch40pt

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\begin{document}

\beginnumbering
\pstart
\rmarpar{741C}
\noindent \text{Incipit Quartus $PERIFUSEWN$}%
\lemma{incipit\ .˜.˜.\ $PERIFUSEWN$}\Bfootnote{\textit{om.\ R},
incipit quartus \textit{M}}/
\pend
\medskip

\pstart
\noindent \text{NVTRITOR}\lemma{$ANAKEFALIOSIS$}\Bfootnote{\textit{
FJP, lege} $<anakefala'iwsis$}/.\lmarpar{$ANAKEFALIOSIS$
NATVRARVM} Prima nostrae
\text{Physiologiae}\lemma{physiologiae}\Bfootnote{phisiologiae
\textit{P}, physeologiae \textit{R}}/ intentio praecipuaque mat\-e\-ria
erat \text{quod}\Bfootnote{\textit{p}.\ natura \textit{transp.\ MR}}/
\text{$UPEROUSIADES$}\Bfootnote{\textit{codd.\ Vtrum}
$<uperousi'wdhs$ (hoc est superessentialis) natura \textit{cum Gale
(p.160) an} $<uperousi'oths$ (hoc est superessentialis natura)
\textit{cum Floss (PL 122,741C) intelligendum sit, ambigitur}}/ (hoc
```

```
est superessentialis) natura sit causa creatrix existentium et
non existentium omnium, a nullo creata, unum principium, una
origo, unus et uniuersalis uniuersorum fons, a nullo manans, dum
ab eo manant omnia, trinitas coessentialis in tribus substantiis,
$ANARQOS$ (hoc est sine principio), principium et finis, una
bonitas, deus unus, \text{$OMOUSIOS$}\Bfootnote{\textit{codd.,
lege} $<omoo'usios$}/ \text{et}\lemma{\textbf{et}}\Bfootnote{\textit{
R}\textsuperscript{1}, \textit{om.\ R}\textsuperscript{0}}/
$UPEROUSIOS$ (id est coessentialis et superessentialis). Et, ut
ait sanctus Epifanius, episcopus Constantiae Cypri, in
\text{$AGKURATW$}\Bfootnote{anchurato \textit{MR}}/ sermone \text{de
fide}\Bfootnote{Glo\Ma\langle\aM ssa\Ma\rangle\aM: Ita
enim uocatur sermo eius de fide $AGKURATOS$, id est procuratus
\textit{mg.\ add.\ FJP}}/:
\begin{itshape}Tria sancta, tria consancta, tria
\text{agentia}\Bfootnote{actiua \textit{MR}}/, tria coagentia, tria
\text{formantia}\Bfootnote{formatiua \textit{MR}}/, tria
conformantia, tria \text{operantia}\Bfootnote{operatiua \textit{MR}}/,
tria cooperantia, tria subsistentia, tria\rmarpar{742C}
consubsistentia sibi inuicem coexistentia. Trinitas haec
sancta uocatur: tria existentia, una consonantia, una deitas
\text{eiusdem}\Bfootnote{eiusdemque \textit{M}}/ essentiae,
\text{eiusdem uirtutis, eiusdem
\text{subsistentiae}{\Bfootnote{substantiae \textit{R}}}/}%
\Bfootnote{\textit{om.\ M}}/, similia \text{similiter}\Bfootnote{ex
simili \textit{MR}}/ aequalitatem gratiae operantur patris et filii
et sancti spiritus. Quomodo autem \text{sunt}\Bfootnote{\textit{om.\
M}}/, ipsis relinquitur docere: \text{`Nemo enim nouit patrem
nisi filius, neque filium nisi pater, et cuicumque filius
reuelauerit'}\Afootnote{Matth.\ 11, 27}/; reuelatur autem per
spiritum sanctum. Non ergo haec tria existentia aut ex ipso aut
per ipsum aut ad ipsum in unoquoque digne intelliguntur,
\Ma\mid\! R, 264^{\rm r}\!\mid\aM\ sicut ipsa reuelant:\end{itshape}
$FWS, PUR, PNEUMA$ \text{(hoc est lux, ignis,
spiritus)}\Afootnote{EPIPHANIVS, \textit{Ancoratus} 67; PG~43,
137C--140A; GCS 25, p.~82, 2--12}/.
\pend

\pstart
Haec, ut dixi, ab Epifanio tradita, ut quisquis interrogatus quae
tria et quid unum in sancta trinitate debeat credere, sana fide
\Ma\!\mid J, 1^{\rm v}\!\mid\aM\ respondere ualeat, aut ad
fidem accedens\rmarpar{743A} sic erudiatur. Et mihi uidetur
spiritum pro calore posuisse, quasi dixisset in similitudine:
lux, ignis, calor. Haec enim tria unius essentiae sunt. Sed cur
lucem primo dixit, non est mirum. Nam et pater lux est et ignis
et calor; et filius est lux, ignis, calor; et \text{spiritus
sanctus}\Bfootnote{sanctus spiritus \textit{R}}/ lux, ignis, calor.
Illuminat enim pater, illuminat filius, illuminat spiritus
sanctus: ex ipsis enim omnis scientia et sapientia donatur.
```

```
\pend
\endnumbering

\end{document}

⟨/periph⟩
```