eledmac
A presumptuous attempt to port EDMAC, TABMAC and EDSTANZA to LaTeX[*]

Peter Wilson
Herries Press[†]
Maïeul Rouquette[‡]
based on the original work by
John Lavagnino, Dominik Wujastyk, Herbert Breger and Wayne Sullivan

Abstract

EDMAC, a set of Plain TeX macros, was made at the beginning of 90's for typesetting critical editions in the traditional way, i.e., similar to the Oxford Classical Texts, Teubner, Arden Shakespeare and other series. A separate set of Plain TeX macros, TABMAC, provides for tabular material. Another set of Plain TeX macros, EDSTANZA, assists in typesetting verse.

The eledmac package makes the EDMAC, TABMAC and EDSTANZA facilities available to authors who would prefer to use LaTeX. The principal functions provided by the package are marginal line numbering and multiple series of footnotes and endnotes keyed to line numbers.

In addition to the EDMAC, TABMAC and EDSTANZA functions the package also provides for index entries keyed to both page and line numbers. Multiple series of the familiar numbered footnotes are also available.

Other LaTeX packages for critical editions include EDNOTES, and poemscol for poetical works.

In October 2012, Maïeul Rouquette released the *eledform* package[1]. Based on eledmac, this package provides tools for creating a formal description (formalism) of textual variants.

To report bugs, please go to ledmac's GitHub page and click "New Issue": `https://github.com/maieul/ledmac/issues/`. You must open an account with github.com to access my page (maieul/ledmac). GitHub accounts are free for open-source users.

You can subscribe to the eledmac mail list in:
`https://lists.berlios.de/pipermail/ledmac-users/`

[*]This file (`eledmac.dtx`) has version number v1.5, last revised 2013/07/11.
[†]`herries dot press at earthlink dot net`
[‡]`maieul at maieul dot net`
[1]`http://www.ctan.org/eledform`.

Contents

List of Figures

1 Introduction

The EDMAC macros [LW90] for typesetting critical editions of texts have been available for use with TeX since 90's. Since EDMAC was introduced there has been a small but constant demand for a version of EDMAC that could be used with LaTeX. The eledmac package is an attempt to satisfy that request.

eledmac would not have been possible without the amazing work by John Lavagnino and Dominik Wujastyk, the original authors of EDMAC. I, Peter Wilson, am very grateful for their encouragement and permission to use EDMAC as a base. The majority of both the code and this manual are by these two. The tabular material is based on the TABMAC code [Bre96], by permission of its author, Herbert Breger. The verse-related code is by courtesy of Wayne Sullivan, the author of EDSTANZA [Sul92], who has kindly supplied more than his original macros.

Since 2011's Maïeul Rouquette begun to maintain and extend eledmac. As plain TeX is used by little people, and LaTeX by more people eledmac and original EDMAC are more and more distant.

1.1 Overview

The eledmac package, together with LaTeX, provides several important facilities for formatting critical editions of texts in a traditional manner. Major features include:

- automatic stepped line numbering, by page or by section;
- sub-lineation within the main series of line numbers;
- variant readings automatically keyed to line numbers;
- caters for both prose and verse;
- multiple series of the footnotes and endnotes;
- block or columnar formatting of the footnotes;
- simple tabular material may be line numbered;
- indexing keyed to page and line numbers.

eledmac 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. LaTeX and eledmac will take care of the formatting and visual correlation of all the disparate types of information.

The original EDMAC can be used as a 'stand alone' processor or as part of a process. One example is its use as the formatting engine or 'back end' for the output of an automatic manuscript collation program. COLLATE, written by Peter Robinson, 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. For further details of this and other related work, visit the EDMAC home page at http://www.homepages.ucl.ac.uk/~ucgadkw/edmac/index.html.

Apart from eledmac there are some other LaTeX packages for critical edition typesetting. As Peter Wilson is not an author, or even a prospective one, of any critical edition work he could not provide any opinions on what authors in this area might feel comfortable with or how well any of the packages meet their needs.

EDNOTES [Lüc03], by Uwe Lück and Christian Tapp, is another LaTeX package being developed for critical editions. Unlike eledmac which is based on EDMAC, EDNOTES takes a different (internal) approach and provides a different set of features. For example it provides additional facilities for overlapping lemmas and for handling tables. For more information there is a web site at http://ednotes.sty.de.vu or email to ednotes.sty@web.de.

The poemscol package [Bur01] by John Burt is designed for typesetting critical editions of collections of poems. I do not know how, or whether, poemscol and eledmac will work together.

Critical authors may find it useful to look at EDMAC, EDNOTES, eledmac, and poemscol to see which best meets their needs.

At the time of writing Peter Wilson knows of two web sites, apart from the EDMAC home page, that have information on eledmac, and other programs.

- Jerónimo Leal pointed me to http://www.guit.sssup.it/latex/critical.html. This also mentions another package for critical editions called MauroTeX (http://www.maurolico.unipi.it/mtex/mtex.htm). These sites are both in Italian.

- Dirk-Jan Dekker maintains http://www.djdekker.net/ledmac which is a FAQ for typesetting critical editions and eledmac.

This manual contains a general description of how to use the LaTeX version of EDMAC, namely eledmac(in sections 2 through 16.4); the complete source code for the package, with extensive documentation (in sections 17 and following) ; 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 the earlier sections. 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, we suggest that you should read only the general documentation in sections 2, unless you are particularly interested in the innards of eledmac.

1.2 History

1.2.1 EDMAC

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.[2] 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 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 `\special`s. 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*,[3] an edition of the letters of Nicolaus Copernicus,[4] Simon Bredon's *Arithmetica*,[5] a Latin translation by Plato of Tivoli of an Arabic astrolabe text,[6] a Latin translation of part II of the Arabic *Algebra* by Abū Kāmil Shujā' b. Aslam,[7] the Latin *Rithmachia* of Werinher von Tegernsee,[8] a middle-Dutch romance epic on the Crusades,[9] a seventeenth-century Hungar-

[2] 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).

[3] 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).

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

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

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

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

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

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

ian politico-philosophical tract,[10] an anonymous Latin compilation from Hungary entitled *Sermones Compilati in Studio Gererali Quinqeecclesiensi in Regno Ungarie*,[11] the collected letters and papers of Leibniz,[12] Theodosius's *Spherics*, the German *Algorismus* of Sacrobosco, the Sanskrit text of the *Kāśikāvṛtti* of Vāmana and Jayāditya,[13] and the English texts of Thomas Middleton's collected works.

1.2.2 eledmac

Version 1.0 of `TABMAC` was released by Herbert Breger in October 1996. This added the capability for typesetting tabular material.

Version 0.01 of `EDSTANZA` was released by Wayne Sullivan in June 1992, to help a colleague with typesetting Irish verse.

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). In August 2003 the `TABMAC` functions were added; the starting point for these being version 1.0 of Ocober 1996. The `EDSTANZA` (v0.01) functions were added in February 2004. Sidenotes and regular footnotes in numbered text were added in April 2004.

This port was called *ledmac*.

Since July 2011, ledmac is maintained by Maïeul Rouquette.

Important changes were put in version 1.0, to make eledmac more easily extensible (see 4.3 p.17). They can make some little troubles with old customization. That is why a new name was selected: *eledmac*. To migrate from ledmac to eledmac, please read Appendix Appendix A (p.180).

2 The eledmac package

eledmac is a three-pass package like LaTeX itself. Although your textual apparatus and line numbers will be printed even on the first run, it takes two more passes through LaTeX 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. eledmac 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 LaTeX 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 eledmac's

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

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

[12]Leibniz, *Sämtliche Schriften und Briefe*, series I, III, VII, being edited by Dr. H. Breger, Dr. N. Gädeke and others, at the Leibniz-Archiv, Niedersächsische Landesbibliothek, Hannover. (see `http://www.nlb-hannover.de/Leibniz`)

[13]Being prepared at Poona and Lausanne Universities.

note commands with it: this is appropriate for introductions and other material added by the editor around the edited text.

3 Numbering text lines and paragraphs

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

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

The `\beginnumbering` macro resets the line number to zero, reads an auxiliary file called ⟨*jobname*⟩`.nn` (where ⟨*jobname*⟩ 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 file called ⟨*jobname*⟩`.end` to receive the text of the endnotes. `\endnumbering` closes the ⟨*jobname*⟩`.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. eledmac 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.[14]

`\firstlinenum` `\linenumincrement` By default, `eledmac` numbers every 5th line. There are two counters, `firstlinenum` and `linenumincrement`, that control this behaviour; they can be changed using `\firstlinenum{`⟨*num*⟩`}` and `\linenumincrement{`⟨*num*⟩`}`. `\firstlinenum` specifies the first line that will have a printed number, and `\linenumincrement` is the difference between succesive numbered lines. For example, to start printing numbers at the first line and to have every other line numbered:

```
\firstlinenum{1} \linenumincrement{2}
```

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

`\firstsublinenum`
`\sublinenumincrement`
`\pausenumbering`
`\resumenumbering`

There are similar commands, `\firstsublinenum{`⟨*num*⟩`}` and `\sublinenumincrement{`⟨*num*⟩`}` for controlling sub-line numbering.

eledmac 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 LaTeX may reach its memory limit. There are two solutions to this. The first is to get a larger LaTeX with increased memory. 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` . . . `\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
```

```
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

```
\newcommand{\memorybreak}{\pausenumbering\resumenumbering}
```

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

`\numberpstarttrue`
`\numberpstartfalse`
`\thepstart`

It's possible to insert a number at every `\pstart` command. You must use the `\numberpstarttrue` command to have it. You can stop the numbering with `\numberpstartfalse`. You can redefine the command `\thepstart` to change style. On each `\beginnumbering` the numbering restarts.

With the `\sidepstartnumtrue` command, the number of `\pstart` will be printed in side. In this case, the line number will be not printed.

With the `\labelpstarttrue` command, a `\label` added just after a `\pstart` will refer to the number of this pstart.

3.1 Lineation commands

`\numberlinefalse`
`\numberlinetrue`
`\lineation`

Line numbering can be disabled with `\numberlinefalse`. It can be enabled again with `\numberlinetrue`. Lines can be numbered either by page, by pstart or by section; you specify this using the `\lineation{`⟨*arg*⟩`}` macro, where ⟨*arg*⟩ is either `page`, `pstart` 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 package's standard setting is `\lineation{section}`. If the lineation is by pstart, the pstart number will be printed before the line number in the notes.

`\linenummargin` The command `\linenummargin`⟨*location*⟩ specifies the margin where the line (or pstart) numbers will be printed. The permissable value for ⟨*location*⟩ is one out of the list `left`, `right`, `inner`, or `outer`, for example `\linenummargin{inner}`. The package's default setting is

```
\linenummargin{left}
```

to typeset the numbers in the left hand margin. You can change this whenever you're not in the middle of making a paragraph.

More precisely, the value of `\linenummargin` used is that in effect at the `\pend` of a numbered paragraph. Apart from an initial setting for `\linenummargin`, only change it after a `\pend`, whereupon it will apply to all following numbered paragraphs, until changed again (changing it between a `\pstart` and `\pend` pair will apply the change to all the current paragraph).

`\firstlinenum` `\linenumincrement` `\firstsublinenum` `\sublinenumincrement` 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 and they can be set by the macros `\firstlinenum{`⟨*num*⟩`}`, etc. `\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 are set to 5 (e.g., `\firstlinenum{5}`.

`\linenumberlist` You can define `\linenumberlist` to specify a non-uniform distribution of printed line numbers. For example:

```
\def\linenumberlist{1,2,3,5,7,11,13,17,19,23,29}
```

to have numbers printed on prime-numbered lines only. There must be no spaces within the definition which consists of comma-separated decimal numbers. The numbers can be in any order but it is easier to read if you put them in numerical order. Either omitting the definition of `\linenumberlist` or following the vacuous definition

```
\def\linenumberlist{}
```

the standard numbering sequence is applied. The standard sequence is that specified by the combination of the `firstlinenum`, `linenumincrement`, `firstsublinenum` and `linenumincrement` counter values.

`\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.

3.2 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{`⟨*arg*⟩`}`; its argument is a word, either `first`, `last`, or `all`. The package initially sets this as `\lockdisp{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{`⟨*num*⟩`}` and `\advanceline{`⟨*num*⟩`}` 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.

`\setlinenum` The `\setline` and `\advanceline` macros should only be used within a `\pstart...\pend` group. The `\setlinenum{`⟨*num*⟩`}` command can be used outside such a group, for example between a `pend` and a `\pstart`. It sets the line number to ⟨*num*⟩. It has no effect if used within a `\pstart...\pend` group

`\linenumberstyle` `\sublinenumberstyle` Line numbers are nomally printed as arabic numbers. You can use `\linenumberstyle{`⟨*style*⟩`}` to change the numbering style. ⟨*style*⟩ must be one of:

`Alph`	Uppercase letters (A...Z).
`alph`	Lowercase letters (a...z).
`arabic`	Arabic numerals (1, 2, ...)
`Roman`	Uppercase Roman numerals (I, II, ...)
`roman`	Lowercase Roman numerals (i, ii, ...)

Note that with the `Alph` or `alph` styles, 'numbers' must be between 1 and 26 inclusive.

Similarly `\sublinenumberstyle{`⟨*style*⟩`}` can be used to change the numbering style of sub-line numbers, which is normally arabic numerals.

`\skipnumbering` When inserted into a numbered line the macro `\skipnumbering` causes the numbering of that particular line to be skipped; that is, the line number is unchanged and no line number will be printed.

4 The apparatus

4.1 Commands

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

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

The ⟨*lemma*⟩ argument is the lemma in the main text: `\edtext` both prints this as part of the text, and makes it available to the ⟨*commands*⟩ you specify to generate notes.

For example:

```
I saw my friend \edtext{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 `\edtext` commands. 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:

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

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

`\Afootnote` `\Bfootnote` `\Cfootnote` `\Dfootnote` `\Efootnote` Five separate series of the footnotes are maintained; each macro taking one argument like `\Afootnote{`⟨*text*⟩`}`. 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. The package 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 should they be required.

An optional argument can be added before the text of the footnote. Its value is a comma separated list of options. The available options are:

- `nonum` to disable line numbering for this note.
- `nosep` to disable the lemma separator for this note.

Example: `\Afootnote[nonum]{`⟨*text*⟩`}`.

`\Aendnote` `\Bendnote` `\Cendnote` `\Dendnote` `\Eendnote` The package also maintains five separate series of endnotes. Like footnotes each macro takes a single argument like `\Aendnote{`⟨*text*⟩`}`. Normally, none of them are printed: you must use the `\doendnotes` macro described below (p. 25) to call for their output at the appropriate point in your document.

By default, no paragraph can be made in the notes of critical apparatus. You can allow it by adding the options `parapparatus` when loading the package :

```
\usepackage[parapparatus]{eledmac}
```

`\lemma` If you want to change the lemma that gets passed to the notes, you can do this by using `\lemma{`⟨*alternative*⟩`}` within the second argument to `\edtext`, before the note commands. The most common use of this command is to abbreviate the lemma that's printed in the notes. For example:

```
\edtext{I saw my friend
  \edtext{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{`⟨*arg*⟩`}` 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 the 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 eledmac 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 `\edtext` 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 `\edtext`. If the ⟨*lemma*⟩ argument to `\edtext` 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. 25) 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
```

Formalism for textual criticism If your notes are for textual criticism, you should use the *eledform* package[15].

This package provides tools to describes the textual variants in a formal way. It is based on eledmac for the typographical aspect.

4.2 Alternate footnote formatting

If you just launch into eledmac 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. The package 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 signifiant changes.

`\footparagraph` `\foottwocol` `\footthreecol` By default, all footnotes are formatted as a series of separate paragraphs in one column. Three other formats are also available for notes, and using these macros you can select a different format for a series of notes.

[15] `http://www.ctan.org/pkg/eledform`.

- `\footparagraph` formats all the footnotes of a series as a single paragraph;
- `\foottwocol` formats them as separate paragraphs, but in two columns;
- `\footthreecol`, in three columns.

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.

4.3 Display options

Since version 1.0, some commands can be used to change the display of the footnotes. All can have an optional argument `[`⟨*s*⟩`]`, which is the letter of the series — or a list of letters separated by comma — depending on which option is applied.

When a length, noted ⟨*l*⟩, is used, it can be stretchable: `a minus b minus c`. The final length `m` is calculated by LaTeX to have: $b - a \leq m \leq b + c$. If you use relative unity[16], it will be relative to fontsize of the footnote.

4.3.1 Control line number printing

`\numberonlyfirstinline` By default, the line number is printed in every note. If you want to print it only the first time for a value (i.e one time for line 1, one time for line 2 etc.), you can use `\numberonlyfirstinline[`⟨*s*⟩`]`. Use `\numberonlyfirstinline[`⟨*s*⟩`][`⟨*false*⟩`]` to cancel it (`<s>` can be empty if you want to disable it for every series).

`umberonlyfirstintwolines` Suppose you have a lemma on line 2 and a lemma between line 2 and line 3. With `\numberonlyfirstinline`, the second lemma is considered to be on the same line as the first lemma. But if you add `\numberonlyfirstintwolines[`⟨*s*⟩`]`, the distinction is made. Use `\numberonlyfirstintwolines[`⟨*s*⟩`][`⟨*false*⟩`]` to cancel it (`<s>` can be empty if you want to disable it for every series).

`\symlinenum` For setting a particular symbol in place of the line number, you can use `\symlinenum[`⟨*s*⟩`]{`⟨*symbol*⟩`}` in combination with `\numberonlyfirstinline[`⟨*s*⟩`]`. From the second lemma of the same line, the symbol will be used instead of line number.

`\nonumberinfootnote` You can use `\nonumberinfootnote[`⟨*s*⟩`]` if you don't want to have the line number in a footnote. To cancel it, use `\nonumberinfootnote[`⟨*s*⟩`][`⟨*false*⟩`]`.

`\pstartinfootnote` You can use `\pstartinfootnote[`⟨*s*⟩`]` if you want to print the pstart number in the footnote, before the line and subline number. Use `\pstartinfootnote[`⟨*s*⟩`][`⟨*false*⟩`]` to cancel it (`<s>` can be empty if you want to disable it for every series). Note that when you change the lineation system, the option is automatically switched :

[16]Like `em` which is the width of a M.

- If you use lineation by pstart, the option is enabled.
- If you use lineation by section or by page, the option is disabled.

`\onlypstartinfootnote` In combination with `\pstartinfootnote`, you can use `\onlypstartinfootnote[⟨s⟩]` if you want to print only the pstart number in the footnote, and not the line and subline number. Use `\onlypstartinfootnote[⟨s⟩][⟨false⟩]` to cancel it (`<s>` can be empty if you want to disable it for every series).

`\beforenumberinfootnote` With `\beforenumberinfootnote[⟨s⟩]{⟨l⟩}`, you can add some space before the line number in a footnote. If the line number is not printed, the space is not either. The default value is 0 pt.

`\afternumberinfootnote` With `\afternumberinfootnote[⟨s⟩]{⟨l⟩}` you can add some space after the line number in a footnote. If the line number is not printed, the space is not either. The default value is 0.5 em.

`\nonbreakableafternumber` By default, the space defined by `\afternumberinfootnote` is breakable. With `\afternumberinfootnote[⟨s⟩]` it becomes non breakable. Use `\afternumberinfootnote[⟨s⟩][⟨false⟩]` to cancel it (`<s>` can be empty if you want to disable it for every series).

`\beforesymlinenum` With `\beforesymlinenum[⟨s⟩]{⟨l⟩}` you can add some space before the line symbol in a footnote. The default value is value set by `\beforenumberinfootnote`.

`\aftersymlinenum` With `\aftersymlinenum[⟨s⟩]{⟨l⟩}` you can add some space before the line symbol in a footnote. The default value is value set by `\afternumberinfootnote`.

`\inplaceofnumber` If no number or symbolic line number is printed, you can add a space, with `\inplaceofnumber[⟨s⟩]{⟨l⟩}`. The default value is 1 em.

`\boxlinenum` It could be useful to put the line number inside a fixed box: the content of the note will be printed after this box. You can use `\boxlinenum[⟨s⟩]{⟨l⟩}` to do that. To subsequently disable this feature, use `\boxlinenum` with length equal to 0 pt. One use of this feature is to print line number in a column, and the note in an other column:

```
\Xhangindent{1em}
\afternumberinfootnote{0em}
\boxlinenum{1em}
```

`\boxsymlinenum` `\boxsymlinenum[⟨s⟩]{⟨l⟩}` is the same as `\boxlinenum` but for the line number symbol.

4.3.2 Separator between the lemma and the note content

`\lemmaseparator` By default, in a footnote, the separator between the lemma and thenote is a right bracket (`\rbracket`). You can use `\lemmaseparator[⟨s⟩]{⟨lemmaseparator⟩}` to change it. The optional argument can be used to specify in which series it is applied. Note that there is a non-breakable space between lemma and separator, but **breakable** space between separator and lemma.

`\beforelemmaseparator` Using `\beforelemmaseparator[⟨s⟩]{⟨l⟩}` you can add some space between lemma and separator. If your lemma separator is empty, this space won't be printed. The default value is 0 em.

`\afterlemmaseparator` Using `\afterlemmaseparator[`⟨*s*⟩`]{`⟨*l*⟩`}` you can add some space between separator and note. If your lemma separator is empty, this space won't be printed. The default value is 0.5 em.

`\nolemmaseparator` You can suppress the lemma separator, using `\nolemmaseparator[`⟨*s*⟩`]`, which is simply a alias of `\lemmaseparator[`⟨*s*⟩`]{}`.

`\inplaceoflemmaseparator` With `\inplaceoflemmaseparator[`⟨*s*⟩`]{`⟨*l*⟩`}` you can add a space if no lemma separator is printed. The default value is 1 em.

4.3.3 Font style

`Xnotenumfont` `\Xnotenumfont[`⟨*s*⟩`]{`⟨*command*⟩`}` is used to change the font style for line numbers in critical footnotes ; ⟨*command*⟩ must be one (or more) switching command, like `\bfseries`.

`Xendnotenumfont` `\Xendnotenumfont[`⟨*s*⟩`]{`⟨*command*⟩`}` is used to change the font style for line numbers in critical footnotes. ⟨*command*⟩ must be one (or more) switching command, like `\bfseries`.

`notenumfontX` `\notenumfontX[`⟨*s*⟩`]{`⟨*command*⟩`}` is used to change the font style for note numbers in familiar footnotes. ⟨*command*⟩ must be one (or more) switching command, like `\bfseries`.

`\Xnotefontsize` `\Xnotefontsize[`⟨*s*⟩`]{`⟨*command*⟩`}` is used to define the font size of critical footnotes of the series. The default value is `\footnotesize`. The ⟨*command*⟩ must not be a size in pt, but a standard LaTeX size, like `\small`.

`\notefontsizeX` `\notefontsizeX[`⟨*s*⟩`]{`⟨*command*⟩`}` is used to define the font size of critical footnotes of the series. The default value is `\footnotesize`. The ⟨*command*⟩ must not be a size in pt, but a standard LaTeX size, like `\small`.

`\Xendnotefontsize` `\Xendnotefontsize[`⟨*s*⟩`]{`⟨*l*⟩`}` is used to define the font size of end critical footnotes of the series. The default value is `\footnotesize`. The ⟨*command*⟩ must not be a size in pt, but a standard LaTeX size, like `\small`.

4.3.4 Styles of notes content

`\Xhangindent` For critical notes NOT paragraphed you can define an indent with `\Xhangindent[`⟨*s*⟩`]{`⟨*l*⟩`}`, which will be applied in the second line of notes. It can help to make distinction between a new note and a break in a note. The default value is 0 pt.

`\hangindentX` For familiar notes NOT paragraphed you can define an indent with `\Xhangindent[`⟨*s*⟩`]{`⟨*l*⟩`}`, which will be applied in the second line of notes. It can help to make a distinction between a new note and a break in anote.

4.3.5 Arbitrary code at the beginninging of notes

The three next commands add an arbitrary code at the beginning of notes. As the name's space is local to the notes, you can use it to redefine some style inside the notes. For example, if you don't want the pstart number to be in bold, use :

```
\bhookXnote{\renewcommand{\thepstart}{\arabic{pstart}.}}
```

`\bhookXnote` `\bhookXnote[⟨series⟩]{⟨code⟩}` is to be used at the beginning of the critical footnotes.

`\bhooknoteX` `\bhooknoteX[⟨series⟩]{⟨code⟩}` is to be used at the beginning of the familiar footnotes.

`\bhookXendnote` `\bhookXendnote[⟨series⟩]{⟨code⟩}` is to be used at the beginning of the endnotes.

4.3.6 Options for notes in columns

For the following four macros, be careful that the columns are made from right to left.

`\hsizetwocol` `\hsizetwocol[⟨s⟩]{⟨l⟩}` is used to change width of a column when critical notes are displaying in two columns. Defaut value is `.45 \hsize`.

`\hsizethreecol` `\hsizethreecol[⟨s⟩]{⟨l⟩}` is used to change width of a column when critical notes are displaying in three columns. Defaut value is `.3 \hsize`.

`\hsizetwocolX` `\hsizetwocol[⟨s⟩]{⟨l⟩}` is used to change width of a column when familiar notes are displaying in two columns. Defaut value is `.45 \hsize`.

`\hsizethreecolX` `\hsizethreecolX[⟨s⟩]{⟨l⟩}` is used to change width of a column when familiar notes are displaying in three columns. Defaut value is `.3 \hsize`.

4.3.7 Options for paragraphed footnotes

`\afternote` You can add some space after a note by using `\afternote[⟨s⟩]{⟨l⟩}`. The default value is `1em plus.4em minus.4em`.

`\parafootsep` For paragraphed footnotes (see below), you can chooce the separator between each note by `\parafootsep[⟨s⟩]{⟨l⟩}`. A common separator is double pipe (`$||$`), which you can set by `\parafootsep$||$`.

4.3.8 Options for block of notes

`\txtbeforeXnotes` You can add some text before critical notes with `\textbeforeXnotes[⟨s⟩]{⟨text⟩}`.

`\beforeXnotes` You can change the vertical space printed before the rule of the critical notes with `\beforeXnotes[⟨s⟩]{⟨l⟩}`. The default value is `1.2em plus .6em minus .6em`.

`\beforenotesX` You can change the vertical space printed before the rule of the familiar notes with `\beforenotesX[⟨s⟩]{⟨l⟩}`. The default value is `1.2em plus .6em minus .6em`.

`\preXnotes` You can set the space before the first series of critical notes printed on each page and set a different amount of space for subsequent the series on the page. You can do it with `\preXnotes{⟨l⟩}`. You can disable this feature by setting the length to 0 pt.

`\prenotesX` You can want the space before the first printed (in a page) series of familiar notes not to be the same as before other series. You can do it with `\prenotesX{⟨l⟩}`. You can disable this feature by setting the length to 0 pt.

`\maxhXnotes` By default, one series of critical notes can take 80% of the page size, before being broken to the next page. If you want to change the size use `\maxhXnotes[⟨s⟩]{⟨l⟩}`. Be careful : the length can't be flexible, and is relative to the the current font. For exemple, if you want that note takes, at most, 33 of the text height, do `\maxhnotes{33\textheight}`.

`\maxhnotesX` `\maxhnotesX[`⟨*s*⟩`]{`⟨*l*⟩`}` is the same as previous, but for familiar footnotes.

Be careful with the two previous commands. Actually, for technical purposes, one paragraphed note is considered as one block. Consequently, it can't be broken between two pages, even if you used these commands. The debug is in the todolist.

4.4 Page layout

You should set up the page layout parameters, and in particular 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 these; too much or too little space will be allotted for the notes on the page if these macros use the wrong values.[17]

4.5 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 for a large job, here is a list of the complete set of eledmac macros relating to fonts that are intended for manipulation by the user: `\endashchar`, `\fullstop`, `\numlabfont`, and `\rbracket`.

`\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 defined as

```
\newcommand{\numlabfont}{\normalfont\scriptsize}
```

You might wish to use a different font if, for example, you preferred to have these line numbers printed using old-style numerals.

`\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 normal document 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

[17]There is one tiny proviso about using paragraphed notes: you shouldn't force any explicit 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 103 explains why this restriction is necessary.

many styles of textual notes (including eledmac's standard style). For polyglossia, when the lemma is RTL, the bracket automatically switches to a left bracket.

`\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 `\edtext` 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 eledmac'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.

eledmac 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 the footnotes are formatted in the same way.

4.6 Create a new series

If you need more than 5 series of critical footnotes you can create extra series, using `\newseries` command. For example to create G and H series `\newseriesG,H`.

5 Verse

In 1992 Wayne Sullivan[18] wrote the EDSTANZA macros [Sul92] for typesetting verse in a critical edition. More specifically they were for handling poetry stanzas which use indentation to indicate rhyme or metre.

With Wayne Sullivan's permission the majority of this section has been taken from [Sul92]. I have made a few changes to enable his macros to be used in the LaTeX eledmac package.

`\stanza` `\&` Use `\stanza` at the start of a stanza. Each line in a stanza is ended by an ampersand (&), and the stanza itself is ended by putting `\&` at the end of the last line.

`\stanzaindentbase` Lines within a stanza may be indented. The indents are integer multiples of the length `\stanzaindentbase`, whose default value is 20pt.

`\setstanzaindents` In order to use the stanza macros, one must set the indentation values. First the value of `\stanzaindentbase` should be set, unless the default value 20pt is desired. Every stanza line indentation is a multiple of this.

[18]Department of Mathematics, University College, Dublin 4, Ireland

To specify these multiples one invokes, for example
`\setstanzaindents{3,1,2,1,2}`.

The numerical entries must be whole numbers, 0 or greater, separated by commas without embedded spaces. The first entry gives the hanging indentation to be used if the stanza line requires more than one print line. If it is known that each stanza line will fit on a single print line, then this first entry should be 0; TEX does less work in this case, but no harm ensues if the hanging indentation is not 0 but is never used. Enumeration is by stanza lines, not by print lines. In the above example the lines are indented one unit, two units, one unit, two units, with 3 units of hanging indentation in case a stanza line is too long to fit on one print line.

Since version 0.13, if the indentation is repeated every *n* verses of the stanza, you can define only the *n* first indentations, and say they are repeated, defining the value of the `stanzaindentsrepetition` counter at *n*. For example:

```
\setstanzaindents{0,1,0}
\setcounter{stanzaindentsrepetition}{2}
```

is like

```
\setstanzaindents{0,1,0,1,0,1,0,1,0,1,0}
```

If you don't use the `stanzaindentsrepetition` counter, make sure you have at least one more numerical entry in `\setstanzavalues` than the number of lines in the stanza. The macros make no restriction on the number of lines in a stanza. Stanza indentation values (and penalty values) obey TEX's grouping conventions, so if one stanza among several has a different structure, its indentations (penalties) may be set within a group; the prior values will be restored when the group ends.

`\setstanzapenalties` When the stanzas run over several pages, often it is desirable that page breaks should arise between certain lines in the stanza, so a facility for including penalties after stanza lines is provided. If you are satisfied with the page breaks, you need not set the penalty values.

The command
`\setstanzapenalties{1,5000,10100,5000,0}`
results in a penalty of 5000 being placed after the first and third lines of the stanza, and a penalty of −100 after the second.

The first entry "1" is a control value. If it is zero, then no penalties are passed on to TEX, which is the default. Values between 0 and 10000 are penalty values; values between 10001 and 20000 have 10000 subtracted and the result is given as a negative penalty. The mechanism used for indentations and penalties requires unsigned values less than 32768. No penalty is placed after the last line, so the final `,0` in then example above could be omitted. The control sequence `\endstanzaextra` can be defined to include a penalty. A penalty of 10000 will prevent a page break; such a penalty is included automatically where there is stanza hanging indentation. A penalty of −10000 (corresponding to the entry value

20000 in this context) forces a page break. Values in between act as suggestions as to the desirability of a page break at a given line. There is a subtle interaction between penalties and *glue*, so it may take some adjustment of skips and penalties to achieve the best results.

`\ampersand` If you need to print an & symbol in a stanza, use the `\ampersand` macro, not `\&` which will end the stanza.

`\endstanzaextra` The macro `\endstanzaextra`, if it is defined, is called at the end of a stanza. You could define this, for example, to add extra space between stanzas (by default there is no extra space between stanzas); if you are using the memoir class, it provides a length `\stanzaskip` which may come in handy.

`\startstanzahook` Similarly, if `\startstanzahook` is defined, it is called by `\stanza` at the start. This can be defined to do something.

`\flagstanza` Putting `\flagstanza[`⟨*len*⟩`]{`⟨*text*⟩`}` at the start of a line in a stanza (or elsewhere) will typeset ⟨*text*⟩ at a distance ⟨*len*⟩ before the line. The default ⟨*len*⟩ is `\stanzaindentbase`.

For example, to put a verse number before the first line of a stanza you could proceed along the lines:

```
\newcounter{stanzanum}
\setcounter{stanzanum}{0}
\newcommand*{\startstanzahook}{\refstepcounter{stanzanum}}
\newcommand{\numberit}{\flagstanza{\thestanzanum}}
...
\stanza
\numberit First line...&
   rest of stanza\&

\stanza
\numberit First line, second stanza...
```

5.1 Hanging symbol

`\hangingsymbol` It's possible to insert a symbol on each line of verse's hanging, as in French typography for '['. To insert in eledmac, redefine macro `\hangingsymbol` with this code:

```
\renewcommand{\hangingsymbol}{[\,}
```

6 Grouping

In a `minipage` environment LaTeX changes `\footnote` numbering from arabic to alphabetic and puts the footnotes at the end of the minipage.

`minipage` You can put numbered text with critical footnotes in a minipage and the footnotes are set at the end of the minipage.

You can also put familiar footnotes (see section 11) in a minipage but unlike with `\footnote` the numbering scheme is unaltered.

`ledgroup` Minipages, of course, aren't broken across pages. Footnotes in a `ledgroup` environment are typeset at the end of the environment, as with minipages, but the environment includes normal page breaks. The environment makes no change to the textwidth so it appears as normal text; it just might be that footnotes appear in the middle of a page, with text above and below.

`ledgroupsized` The `ledgroupsized` environment is similar to `ledroup` except that you must specify a width for the environment, as with a minipage.
`\begin{ledgroupsized}[`⟨*pos*⟩`]{`⟨*width*⟩`}`.

The required ⟨*width*⟩ argument is the text width for the environment. The optional ⟨*pos*⟩ argument is for positioning numbered text within the normal textwidth. It may be one of the characters:

l (left) numbered text is flush left with respect to the normal textwidth. This is the default.

c (center) numbered text is in the center of the textwidth.

r (right) numbered text is flush right with respect to the normal textwidth.

Note that normal text, footnotes, and so forth are all flush left.

`\begin{ledgroupsized}{\textwidth}` is effectively the same as `\begin{ledgroup}`

7 Crop marks

The eledmac package does not provide crop marks. These are available with either the memoir class [Wil02] or the crop package.

8 Endnotes

`\doendnotes` `\endprint` `\printnpnum` `\doendnotes{`⟨*letter*⟩`}` 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 `\select@lemmafont` to select fonts, just as the footnote macros do (see p. 91 above).

As endnotes may be printed at any point in the document they always start with the page number of where they were specified. The macro `\printnpnum{`⟨*num*⟩`}` is used to print these numbers. Its default definition is:
`\newcommand*{\printnpnum}[1]{p.#1) }`

`\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.

9 Cross referencing

The package 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 using those labels.

`\edlabel` First you place a label in the text using the command `\edlabel{`⟨*lab*⟩`}`. ⟨*lab*⟩ can be almost anything you like, including letters, numbers, punctuation, or a combination—anything but spaces; you might say `\edlabel{toves-3}`, for example.[19]

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

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

The `\edlabel` command works by writing macros to the LaTeX `.aux` file. 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 an `\edtext{...}{...}` command, the `\edlabel` should be defined inside the first argument, e.g.,

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

`\xpageref` `\xlineref` `\xsublineref` However, there are situations in which you'll want eledmac 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 LaTeX 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`. 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 `\edlabel{foo}` command, even if you never refer to that label—since those commands can all do the necessary processing of the `.aux` file, and the `\x...` ones 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{`⟨*lab1*⟩`}{`⟨*lab2*⟩`}` command generates a reference to a sequence of lines, for use in the second argument of `\edtext`. It takes two arguments, both of which are labels: e.g., `\xxref{mouse}{elephant}`. It calls `\linenum` (q.v.,

[19] More precisely, you should stick to characters in the TeX categories of 'letter' and 'other'.

p. 15 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{`⟨*lab*⟩`}{`⟨*numbers*⟩`}` macro so that you can 'roll your own' label. For example, if you say '`\edmakelabel{elephant}{10|25|0}`' you will create 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.

`\label` `\ref` `\pageref` The normal `\label`, `\ref` and `\pageref` macros may be used within numbered text, and operate in the familiar fashion.

10 Side notes

The `\marginpar` command does not work in numbered text. Instead the package provides for non-floating sidenotes in either margin.

`\ledleftnote` `\ledrightnote` `\ledsidenote` `\sidenotemargin` `\ledleftnote{`⟨*text*⟩`}` will put ⟨*text*⟩ into the left margin level with where the command was issued. Similarly, `\ledrightnote{`⟨*text*⟩`}` puts ⟨*text*⟩ in the right margin. `\ledsidenote{`⟨*text*⟩`}` will put ⟨*text*⟩ into the margin specified by the current setting of `\sidenotemargin{`⟨*location*⟩`}`. The permissable value for ⟨*location*⟩ is one out of the list `left`, `right`, `inner`, or `outer`, for example `\sidenotemargin{outer}`. The package's default setting is
`\sidenotemargin{right}`
to typeset `\ledsidenote`s in the right hand margin. This is the opposite to the default margin for line numbers. The style for a `\ledsidenote` follows that for a `\ledleftnote` or a `\ledrightnote` depending on the margin it is put in.

If two, say, `\ledleftnote`, commands are called in the same line the second ⟨*text*⟩ will obliterate the first. There is no problem though with having both a left and a right sidenote on the same line.

`\ledlsnotewidth` `\ledrsnotewidth` The left sidenote text is put into a box of width `\ledlsnotewidth` and the right text into a box of width `\ledrsnotewidth`. These are initially set to the value of `\marginparwidth`.

`\rightnoteupfalse` `\leftnoteupfalse` By default, Sidenotes are placed to align with the last line of the note to which it refers. If you want they to be placed to align with the first line of the note to which it refers, use `\leftnoteupfalse` (for left note) and/or `\rightnoteupfalse` (for right note).

`\ledlsnotesep` `\ledrsnotesep` The texts are put a distance `\ledlsnotesep` (or `\ledrsnotesep`) into the left (or right) margin. These lengths are initially set to the value of `\linenumsep`.

`\ledlsnotefontsetup` `\ledrsnotefontsetup` These macros specify how the sidenote texts are to be typeset. The initial definitions are:

```
\newcommand*{\ledlsnotefontsetup}{\raggedleft\footnotesize}% left
\newcommand*{\ledrsnotefontsetup}{\raggedright\footnotesize}% right
```

These can of course be changed to suit.

`\sidenotesep` If you have two or more sidenotes for the same line, they are separated by a comma. But if you want to change this separator, you can redefine the macro `\sidenotesep`.

11 Familiar footnotes

The footmisc package [Fai03] by Robin Fairbairns has an option whereby sequential footnote marks in the text can be separated by commas[3,4] like so. As a convenience eledmac provides this automatically.

`\multfootsep` `\multfootsep` is used as the separator between footnote markers. Its default definition is:

```
\providecommand*{\multfootsep}{\textsuperscript{\normalfont,}}
```

and can be changed if necessary.

`\footnoteA` `\footnoteB` `\footnoteC` `\footnoteD` `\footnoteE` As well as the standard LaTeX footnotes generated via `\footnote`, the package also provides three series of additional footnotes called `\footnoteA` through `\footnoteE`. These have the familiar marker in the text, and the marked text at the foot of the page can be formated using any of the styles described for the critical footnotes. Note that the 'regular' footnotes have the series letter at the end of the macro name whereas the critical footnotes have the series letter at the start of the name.

`\footnormalX` `\footparagraphX` `\foottwocolX` `\footthreecolX` Each of the `\foot...X` macros takes one argument which is the series letter (e.g., `B`). `\footnormalX` is the typical footnote format. With `\footparagraphX` the series is typeset a one paragraph, with `\foottwocolX` the notes are in two columns, and are in three columns with `\foothreecolX`.

`\thefootnoteA` `\bodyfootmarkA` `\footfootmarkA` As well as using the `\foot...X` macros to specify the general footnote arrangement for a series, each series uses a set of macros for styling the marks. The mark numbering scheme is defined by the `\thefootnoteA` macro; the default is:

```
\renewcommand*{\thefootnoteA}{\arabic{footnoteA}}
```

The appearance of the mark in the text is controlled by `\bodyfootmarkA` which is defined as:

```
\newcommand*{\bodyfootmarkA}{%
  \hbox{\textsuperscript{\normalfont\thefootnoteA}}}
```

The command `\footfootmarkA` controls the appearance of the mark at the start of the footnote text. It is defined as:

```
\newcommand*{\footfootmarkA}{\textsuperscript{\thefootnoteA}}
```

There are similar command triples for the other series.

Additional footnote series can be easily defined: you just have to use `\newseries`, defined above (see 4.6 p.22).

12 Indexing

`\edindex` LaTeX provides the `\index{`⟨*item*⟩`}` command for specifying that ⟨*item*⟩ and the current page number should be added to the raw index (`idx`) file. The `\edindex{`⟨*item*⟩`}` macro can be used in numbered text to specify that ⟨*item*⟩ and the current page & linenumber should be added to the raw index file.

If the memoir class or the imakeidx package is used then the macro takes an optional argument, which is the name of a raw index file. For example `\edindex[line]{item}` will use `line.idx` as the raw file instead of `\jobname.idx`.

The minimal version of imakeidx package to be used is the version 1.3a uploaded on CTAN on 2013/07/11.

Be careful with the order of package loading and index declaration. You must use this order:

1. Load imakeidx.
2. Load eledmac.
3. Declare the index with the macro `\makeindex` of imakeidx.

`\pagelinesep` The page & linenumber combination is written as `page\pagelinesep line`, where the default definition is `\newcommand{\pagelinesep}{-}` so that an item on page 3, line 5 will be noted as being at `3-5`. You can renew `\pagelinesep` to get a different separator (but it just so happens that `-` is the default separator used by the MakeIndex program).

`\edindexlab` The `\edindex` process uses a `\label`/`\ref` mechanism to get the correct line number. It automatically generates labels of the form `\label{\edindexlab N}`, where N is a number, and the default definition of `\edindexlab` is:
`\newcommand*{\edindexlab}{$&}`
in the hopes that this will not be used by any other labels (`\edindex`'s labels are like `\label{$&27}`). You can change `\edindexlab` to something else if you need to.

13 Tabular material

LaTeX's normal `tabular` and `array` environments cannot be used where line numbering is being done; more precisely, they can be used but with odd results, so don't use them. However, eledmac provides some simple tabulation environments that can be line numbered. The environments can also be used in normal unnumbered text.

`edarrayl` `edarrayc` `edarrayr` `edtabularl` `edtabularc` `edtabularr` There are six environments; the `edarray*` environments are for math and `edtabular*` for text entries. The final `l`, `c`, or `r` in the environment names indicate that the entries will be flushleft (`l`), centered (`c`) or flushright (`r`). There is no means of specifying different formats for each column, nor for specifying a fixed width for a column. The environments are centered with respect to the surrounding text.

```
\begin{edtabularc}
1 & 2 & 3 \\
a & bb & ccc \\
AAA & BB & C
\end{edtabularc}
```

1	2	3
a	bb	ccc
AAA	BB	C

Entries in the environments are the same as for the normal `array` and `tabular` environments but there must be no ending `\\` at the end of the last row. *There must be the same number of column designators (the &) in each row.* There is no equivalent to any line drawing commands (such as `\hline`). However, unlike the normal environments, the `ed...` environments can cross page breaks.

Macros like `\edtext` can be used as part of an entry.

For example:

```
\beginnumbering
\pstart
\begin{edtabularl}
\textbf{\Large I} & wish I was a little bug\edindex{bug} &
\textbf{\Large I} & eat my peas with honey\edindex{honey} \\
 & With whiskers \edtext{round}{\Afootnote{around}} my tummy &
 & I’ve done it all my life. \\
 & I’d climb into a honey\edindex{honey} pot &
 & It makes the peas taste funny \\
 & And get my tummy gummy.\edindex{gummy} &
 & But it keeps them on the knife.
\end{edtabularr}
\pend
\endnumbering
```

produces the following parallel pair of verses.

I	wish I was a little bug	**I**	eat my peas with honey
	With whiskers round my tummy		I’ve done it all my life.
	I’d climb into a honey pot		It makes the peas taste funny
	And get my tummy gummy.		But it keeps them on the knife.

`\edtabcolsep` The distance between the columns is controlled by the length `\edtabcolsep`.

`\spreadmath` `\spreadtext` `\spreadmath{`⟨*math*⟩`}` typesets `{`⟨*math*⟩`}` but the `{`⟨*math*⟩`}` has no effect on the calculation of column widths. `\spreadtext{`⟨*text*⟩`}` is the analagous command for use in `edtabular` environments.

```
\begin{edarrayl}
1 & 2  &  3  &  4 \\
  & \spreadmath{F+G+C} & & \\
a & bb & ccc & dddd
\end{edarrayl}
```

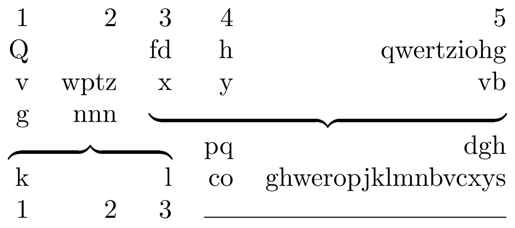
1	2	3	4
	$F+G+C$		
a	bb	ccc	$dddd$

`\edrowfill` The macro `\edrowfill{`⟨*start*⟩`}{`⟨*end*⟩`}{`⟨*fill*⟩`}` fills columns number ⟨*start*⟩ to ⟨*end*⟩ inclusive with ⟨*fill*⟩. The ⟨*fill*⟩ argument can be any horizontal ‘fill’. For example `\hrulefill` or `\upbracefill`.

Note that every row must have the same number of columns, even if some would not appear to be necessary.

The `\edrowfill` macro can be used in both tabular and array environments. The typeset appearance of the following code is shown below.

```
\begin{edtabularr}
1                                & 2    & 3  & 4  & 5 \\
Q                                &      & fd & h  & qwertziohg \\
v                                & wptz & x  & y  & vb \\
g                                & nnn  & \edrowfill{3}{5}{\upbracefill} & & \\
\edrowfill{1}{3}{\downbracefill} &      &    & pq & dgh \\
k                                &      & l  & co & ghweropjklmnbvcxys \\
1                                & 2 & 3 & \edrowfill{4}{5}{\hrulefill} &
\end{tabularr}
```

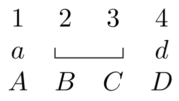


You can also define your own 'fill'. For example:

```
\newcommand*{\upbracketfill}{%
  \vrule height 4pt depth 0pt\hrulefill\vrule height 4pt depth 0pt}
```

is a fill like `\upbracefill` except it has the appearance of a (horizontal) bracket instead of a brace. It can be used like this:

```
\begin{edarrayc}
1 & 2                                & 3 & 4 \\
a & \edrowfill{2}{3}{\upbracketfill} &   & d \\
A & B                                & C & D
\end{edarrayc}
```



`\edatleft` `\edatright` `\edatleft[`⟨*math*⟩`]{`⟨*symbol*⟩`}{`⟨*halfheight*⟩`}` typesets the math ⟨*symbol*⟩ as `\left<symbol>` with the optional ⟨*math*⟩ centered before it. The ⟨*symbol*⟩ is twice ⟨*halfheight*⟩ tall. The `\edatright` macro is similar and it typesets `\right<symbol>` with ⟨*math*⟩ centered after it.

```
\begin{edarrayc}
   & 1 & 2 & 3 &  \\
   & 4 & 5 & 6 &  \\
\edatleft[left =]{\{}{1.5\baselineskip}
   & 7 & 8 & 9 &
\edatright[= right]{)}{1.5\baselineskip}
\end{edarrayc}
```

$$left = \left\{ \begin{array}{ccc} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{array} \right) = right$$

`\edbeforetab` `\edaftertab` \edbeforetab{⟨*text*⟩}{⟨*entry*⟩}, where ⟨*entry*⟩ is an entry in the leftmost column, typesets ⟨*text*⟩ left justified before the ⟨*entry*⟩. Similarly **\edaftertab{**⟨*entry*⟩**}{**⟨*text*⟩**}**, where ⟨*entry*⟩ is an entry in the rightmost column, typesets ⟨*text*⟩ right justified after the ⟨*entry*⟩.

For example:

```
\begin{edarrayl}
                     A  & 1 & 2 & 3 \\
\edbeforetab{Before}{B} & 1 & 3 & 6 \\
                     C  & 1 & 4 & \edaftertab{8}{After} \\
                     D  & 1 & 5 & 0
\end{edarrayl}
```

	A	1	2	3	
Before	B	1	3	6	
	C	1	4	8	After
	D	1	5	0	

`\edvertline` `\edvertdots` The macro \edvertline{⟨*height*⟩} draws a vertical line ⟨*height*⟩ high (contrast this with \edatright where the size argument is half the desired height).

```
\begin{edarrayr}
 a & b & C & d   & \\
 v & w & x & y   & \\
 m & n & o & p   & \\
 k &   & L & cvb & \edvertline{4pc}
\end{edarrayr}
```

$$\left. \begin{array}{rrrr} a & b & C & d \\ v & w & x & y \\ m & n & o & p \\ k & & L & cvb \end{array} \right|$$

The \edvertdots macro is similar to \edvertline except that it produces a vertical dotted instead of a solid line.

14 Sectioning commands

The standard sectioning command (`\chapter`, `\section` etc.) can be used inside a numbered text. But the line which contains it won't be numbered, and you can't add critical notes inside.

However, eledmac provides the following commands :

- `\ledchapter[`⟨*text*⟩`]{`⟨*critical text*⟩`}`
- `\ledchapter*`
- `\ledsection[`⟨*text*⟩`]{`⟨*critical text*⟩`}`
- `\ledsection*`
- `\ledsubsection[`⟨*text*⟩`]{`⟨*critical text*⟩`}`
- `\ledsubsection*`
- `\ledsubsubsection[`⟨*text*⟩`]{`⟨*critical text*⟩`}`
- `\ledsubsubsection*`

Which are the equivalent of the standard LaTeX commands, but be careful. Note the following points :

1. All these commands close a `\pstart`, and open a new one. The content of the command itself is between `\pstart` and `\pend`.
2. Don't try to make `\let\chapter\ledchapter`, or other things like it: the `\led`section commands call the standard commands.
3. For the non-starred sections, use the optional argument ⟨*text*⟩ to provide the text to the table of contents.
4. The `\ledchapter` doesn't open a new page. You must use `\beforeledchapter` before. This also closes a `\pstart` and opens a new.

`\ledsectnotoc` You can create a table of contents that indexes only the titles that appear on the left side of the edition: for instance, titles from the original language, not the translation. You could use `\ledsectnotoc` at the beginning of the side environnment :

```
\begin{Rightside}
\ledsectnotoc
...
\end{Rightside}
```

15 Quotation environments

The `quotation` and `quote` environment can be used so that same definition/note appears both inside and outside a numbered section. The typographical consequences will resemble the outside numbered sections, based on the styles of the *book* class. However, if you use a package that redefines these environments, these redefinitions won't be available inside the numbering section. You must open the quotation environments inside a `\start-\pend` block, not outside.

In some case, you don't want these environments be redefined in numbered section. You can load the package with the option `noquotation` to prevent this redefinition.

16 Miscellaneous

`\extensionchars` When the package 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 `\renewcommand{\extensionchars}{!}`, then you would get temporary files called `jobname.!1`, `jobname.!2`, etc.

`\ifledfinal` The package can take options. The option 'final', which is the default is for final typesetting; this sets `\ifledfinal` to TRUE. The other option, 'draft', may be useful during earlier stages and sets `\ifledfinal` to FALSE.

`\showlemma` The lemma within the text is printed via `\showlemma{lemma}`. Normally, or with the 'final' option, the definition of `\showlemma` is:
`\newcommand*{\showlemma}[1]{#1}`
so it just produces its argument. With the 'draft' option it is defined as
`\newcommand*{\showlemma}[1]{\textit{#1}}`
so that its argument is typeset in an italic font, which may make it easier to check that all lemmas have been treated.

If you would prefer some other style, you could put something like this in the preamble:

```
\ifledfinal\else
   \renewcommand{\showlemma}[1]{\textbf{#1}}% or simply ...[1]{#1}
\fi
```

16.1 Known and suspected limitations

In general, eledmac's system for adding marginal line numbers breaks anything that makes direct use of the LaTeX insert system, which includes marginpars, footnotes and floats.

However, you can use both `\footnote` and the familiar footnote series notes in numbered text. A `\marginpar` in numbered text will throw away its contents and send a warning message to the terminal and log file, but will do no harm.

`\parshape` cannot be used within numbered text, except in a very restricted way.

`\ballast` LaTeX 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, eledmac may oscillate 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

```
\setcounter{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 a footnote 17, p. 21, and described in more detail on p. 102, 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.

LaTeX has a reputation for putting things in the wrong margin after a page break. The eledmac package does nothing to improve the situation — in fact it just makes it more obvious if numbered text crosses a page (or column) boundary and the numbers are meant to flip from side to side. Try and keep the numbers in the same margin all the time. Another aspect of TeX's page breaking mechanism is that when numbering lines by the page, the first few numbers after a page break may continue as though the lines were still on the previous page.

`\pageparbreak` If you can't resist flipping the numbers or numbering by the page, then you might find that judicious use of `\pageparbreak` may help if numbering goes awry across a page (or column) break. It tries to force TeX into partitioning the current paragraph into two invisibly joined paragraphs with a page break between them. Insert the command between the last word on one page and the first word on the next page. If later you change something earlier in the document the natural page break may be in a different place, and you will have to adjust the location of `\pageparbreak` accordingly.

`\footfudgefiddle` For paragraphed footnotes TEX has to estimate the amount of space required. If it underestimates this then the notes may get too long and run off the bottom of the text block. `\footfudgefiddle` can be increased from its default 64 (say to 68) to increase the estimate. You have to use `\renewcommand` for this, like:

```
\renewcommand{\footfudgefiddle}{68}
```

Help, suggestions and corrections will be gratefully received.

16.2 Use with other packages

Because of eledmac's complexity it may not play well with other packages. In particular eledmac is sensitive to commands in the arguments to the `\edtext` and `\*footnote` macros (this is discussed in more detail in section 21, and in particular the discussion about `\no@expands` and `\morenoexpands`). You will have to see what works or doesn't work in your particular case.

It is possible that eledmac and the hyperref package may work together. I have not tried this combination but past experience with hyperref suggests that cooperation is unlikely; hyperref changes many LaTeX internals and eledmac does things that are not normally seen in LaTeX.

If you want to use the option *bottom* of the footmisc package, you must load this package *before* the eledmac package.

`\morenoexpands` You can define the macro `\morenoexpands` to modify macros that you call within `\edtext`. Because of the way eledmac numbers the lines the arguments to `\edtext` can be processed more than once and in some cases a macro should only be processed once. One example is the `\colorbox` macro from the color package, which you might use like this:

```
... \edtext{\colorbox{mycolor}{lemma}}{\Afootnote{...\colorbox...}}
```

If you actually try this[20] you will find LaTeX whinging '`Missing { inserted`', and then things start to fall apart. The trick in this case is to specify either:

```
\newcommand{\morenoexpands}{\let\colorbox=0}
```

or

```
\makeatletter
\newcommand{\morenoexpands}{\let\colorbox\@secondoftwo}
\makeatother
```

(`\@secondoftwo` is an internal LaTeX macro that takes two arguments and thows away the first one.) The first incantation lets color show in both the main text and footnotes whereas the second one shows color in the main text but kills it in the lemma and footnotes. On the other hand if you use `\textcolor` instead, like

```
... \edtext{\textcolor{mycolor}{lemma}}{\Afootnote{...\textcolor...}}
```

there is no need to fiddle with `\morenoexpands` as the color will naturally be displayed in both the text and footnotes. To kill the color in the lemma and footnotes, though, you can do:

```
\makeatletter
\newcommand{\morenoexpands}{\let\textcolor\@secondoftwo}
\makeatother
```

It took me a little while to discover all this. If you run into this sort of problem you may have to spend some time experimenting before hitting on a solution.

[20]Reported by Dirk-Jan Dekker in the CTT thread 'Incompatibility of "color" package' on 2003/08/28.

16.3 Parallel typesetting

Peter Wilson have developed the Ledpar package as an adjunct to eledmac specifically for parallel typesetting of critical texts. This also cooperates with the babel package for typesetting in multiple languages. The package is called *eledpar* since september 2012.

He also developed the ledarab package for handling parallel arabic text in critical editions. Howerer, this package is not maintened by Maïeul Rouquette. You should use the possibility of modern TeX processor, like Xe(La)TeX

16.4 Notes for EDMAC users

If you have never used `EDMAC`, ignore this section. If you have used `EDMAC` and are starting on a completely new document, ignore this section. Only read this section if you are converting an original `EDMAC` document to use eledmac.

The package still provides the original `\text` command, but it is (a) deprecated, and (b) its name has been changed[21] to `\critext`; use the `\edtext` macro instead. However, if you do use `\critext` (the new name for `\text`), the following is a reminder.

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

`\critext{`⟨*lemma*⟩`}`⟨*commands*⟩`/`

The ⟨*lemma*⟩ argument is the lemma in the main text: `\critext` 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 \critext{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 `\critext` commands. 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:

```
\critext{I saw my friend
  \critext{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.

[21]A name like `\text` is likely to be defined by other LaTeX packages (it certainly is by the AMS packages) and it seems sensible to try and avoid clashes with other definitions.

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

The second argument of the `\critext` macro, ⟨*commands*⟩, is the same as the second argument to the `\edtext` macro.

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

```
\catcode`\<=\active
\let<=\critext
```

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>{\critext{#1}{#2}/}
\let<=\xtext
```

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

Aliases for `\critext` 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 `\critext`. (See section 21 to find out why.) Aliases of the second kind may be nested without any problem.

If you really have to use `\critext` in any of the tabular or array environments, then `\edtext` must not be used in the same environment. If you use `\critext` in one of these environments then you have to issue the declaration `\usingcritext` beforehand. The declaration `\usingedtext` must be issued to revert to the default assumption that `\edtext` will be used.

17 Implementation overview

We present the eledmac 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 eledmac package, because the same file is used to generate this manual and to generate the LaTeX package file. Most of what follows consists of macro definitions, but there are some 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 18). 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 20); 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 21), followed by the macros that take each paragraph apart, attach the line numbers and insertions, and send the result to the vertical list (Section 22). The footnote commands (Section 23) and output routine (Section 26) finish the main part of the processing; cross-referencing (Section 27) and endnotes (Section 28) complete the story.

In what follows, macros with an @ in their name are more internal to the workings of eledmac 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.

18 Preliminaries

We 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` We will simply change that to `eledmac`.

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

```
⟨*code⟩
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{eledmac}[2013/07/11 v1.5 LaTeX port of EDMAC]
```

Generally, these are the modifications to the original. `EDMAC` code:

- 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.

`\ifledfinal` `\ifparapparatus@` Use this to remember which option is used, set and execute the options with final as the default.

```
\newif\ifledfinal
\newif\ifparapparatus@
\newif\ifnoquotation@
\parapparatus@false
\DeclareOption{noquotation}{\noquotation@true}
\DeclareOption{final}{\ledfinaltrue}
\DeclareOption{draft}{\ledfinalfalse}
\DeclareOption{parapparatus}{\parapparatus@true}
\ExecuteOptions{final}
```

Use the starred form of `\ProcessOptions` which executes options in the order listed in the source file: class options, then listed package options, so a package option can override a class option with the same name. This was suggested by Dan Luecking in the `ctt` thread *Class/package option processing*, on 27 February 2004.

```
\ProcessOptions*\relax

```

Loading package *xargs* to declare commands with optional arguments. *Etoolbox* is also used to make code clearer - for example, in dynamic command names (which can replace `\csname` etc.). Use *suffix* to declare commands with a starred version, xstring to work with strings and *iflutex* to test if LuaLaTeX is running.

```
\RequirePackage{xargs}
\RequirePackage{etoolbox}
\reserveinserts{32}
\RequirePackage{suffix}
\RequirePackage{xstring}
\RequirePackage{ifluatex}
```

`\if@RTL` The `\if@RTL` is defined by the bidi package, which is sometimes loaded by *polyglossia*. But we define it if the bidi package is not loaded.

```
\ifcsdef{if@RTL}{}{\newif\if@RTL}
```

`\showlemma` `\showlemma{`⟨*lemma*⟩`}` typesets the lemma text in the body. It depends on the option.

```
\ifledfinal
  \newcommand*{\showlemma}[1]{#1}
\else
  \newcommand*{\showlemma}[1]{\underline{#1}}
\fi

```

`\linenumberlist` The code for the `\linenumberlist` mechanism was given to Peter Wilson by Wayne Sullivan on 2004/02/11.

Initialize it as `\empty`

```
\let\linenumberlist=\empty

```

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

In imitation of LaTeX, we create a couple of scratch counters.

LaTeX already defines `\@tempcnta` and `\@tempcntb` but Peter Wilson have found in the past that it can be dangerous to use these (for example one of the AMS packages did something nasty to the ccaption package's use of one of these).

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

`\ifl@dmemoir`

Define a flag for if the memoir class has been used.

```
\newif\ifl@dmemoir
\@ifclassloaded{memoir}{\l@dmemoirtrue}{\l@dmemoirfalse}

```

`\ifl@imakeidx`

Define a flag for if the imakeidx package has been used.

```
\newif\ifl@imakeidx
\@ifpackageloaded{imakeidx}{\l@imakeidxtrue}{\l@imakeidxfalse}
```

18.1 Messages

All the messages are grouped here as macros. This saves TeX's memory when the same message is repeated and also lets them be edited easily.

`\eledmac@warning`

Write a warning message.

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

`\eledmac@error`

Write an error message.

```
\newcommand{\eledmac@error}[2]{\PackageError{eledmac}{#1}{#2}}
```

`\led@err@NumberingStarted`
`d@err@NumberingNotStarted`
`umberingShouldHaveStarted`

```
\newcommand*{\led@err@NumberingStarted}{%
  \eledmac@error{Numbering has already been started}{\@ehc}}
\newcommand*{\led@err@NumberingNotStarted}{%
  \eledmac@error{Numbering was not started}{\@ehc}}
\newcommand*{\led@err@NumberingShouldHaveStarted}{%
  \eledmac@error{Numbering should already have been started}{\@ehc}}
```

`\led@mess@NotesChanged`

```
\newcommand*{\led@mess@NotesChanged}{%
  \typeout{eledmac reminder: }%
  \typeout{ The number of the footnotes in this section
            has changed since the last run.}%
  \typeout{ You will need to run LaTeX two more times
            before the footnote placement}%
  \typeout{ and line numbering in this section are
            correct.}}
```

\led@mess@SectionContinued

```
\newcommand*{\led@mess@SectionContinued}[1]{%
  \message{Section #1 (continuing the previous section)}}
```

\led@err@LineationInNumbered

```
\newcommand*{\led@err@LineationInNumbered}{%
  \eledmac@error{You can’t use \string\lineation\space within
                a numbered section}{\@ehc}}
```

\led@warn@BadLineation
\led@warn@BadLinenummargin
\led@warn@BadLockdisp
\led@warn@BadSublockdisp

```
\newcommand*{\led@warn@BadLineation}{%
  \eledmac@warning{Bad \string\lineation\space argument}}
\newcommand*{\led@warn@BadLinenummargin}{%
  \eledmac@warning{Bad \string\linenummargin\space argument}}
\newcommand*{\led@warn@BadLockdisp}{%
  \eledmac@warning{Bad \string\lockdisp\space argument}}
\newcommand*{\led@warn@BadSublockdisp}{%
  \eledmac@warning{Bad \string\sublockdisp\space argument}}
```

\led@warn@NoLineFile

```
\newcommand*{\led@warn@NoLineFile}[1]{%
  \eledmac@warning{Can’t find line-list file #1}}
```

\led@warn@BadAdvancelineSubline
\led@warn@BadAdvancelineLine

```
\newcommand*{\led@warn@BadAdvancelineSubline}{%
  \eledmac@warning{\string\advanceline\space produced a sub-line
                  number less than zero.}}
\newcommand*{\led@warn@BadAdvancelineLine}{%
  \eledmac@warning{\string\advanceline\space produced a line
                  number less than zero.}}
```

\led@warn@BadSetline
\led@warn@BadSetlinenum

```
\newcommand*{\led@warn@BadSetline}{%
  \eledmac@warning{Bad \string\setline\space argument}}
\newcommand*{\led@warn@BadSetlinenum}{%
  \eledmac@warning{Bad \string\setlinenum\space argument}}
```

\led@err@PstartNotNumbered
\led@err@PstartInPstart
\led@err@PendNotNumbered
\led@err@PendNoPstart
\led@err@AutoparNotNumbered

```
\newcommand*{\led@err@PstartNotNumbered}{%
  \eledmac@error{\string\pstart\space must be used within a
                numbered section}{\@ehc}}
\newcommand*{\led@err@PstartInPstart}{%
  \eledmac@error{\string\pstart\space encountered while another
                \string\pstart\space was in effect}{\@ehc}}
\newcommand*{\led@err@PendNotNumbered}{%
  \eledmac@error{\string\pend\space must be used within a
                numbered section}{\@ehc}}
\newcommand*{\led@err@PendNoPstart}{%
  \eledmac@error{\string\pend\space must follow a \string\pstart}{\@ehc}}
```

```
\newcommand*{\led@err@AutoparNotNumbered}{%
  \eledmac@error{\string\autopar\space must be used within a
                 numbered section}{\@ehc}}
```

\led@warn@BadAction

```
\newcommand*{\led@warn@BadAction}{%
  \eledmac@warning{Bad action code, value \next@action.}}
```

\led@warn@DuplicateLabel
\led@warn@RefUndefined

```
\newcommand*{\led@warn@DuplicateLabel}[1]{%
  \eledmac@warning{Duplicate definition of label ‘#1’ on page \the\pageno.}}
\newcommand*{\led@warn@RefUndefined}[1]{%
  \eledmac@warning{Reference ‘#1’ on page \the\pageno\space undefined.
                   Using ‘000’.}}
```

\led@warn@NoMarginpars

```
\newcommand*{\led@warn@NoMarginpars}{%
  \eledmac@warning{You can’t use \string\marginpar\space in numbered text}}
```

ed@warn@BadSidenotemargin

```
\newcommand*{\led@warn@BadSidenotemargin}{%
  \eledmac@warning{Bad \string\sidenotemmargin\space argument}}
```

\led@warn@NoIndexFile

```
\newcommand*{\led@warn@NoIndexFile}[1]{%
  \eledmac@warning{Undefined index file #1}}
```

\led@err@TooManyColumns
\led@err@UnequalColumns
\led@err@LowStartColumn
\led@err@HighEndColumn
\led@err@ReverseColumns

```
\newcommand*{\led@err@TooManyColumns}{%
  \eledmac@error{Too many columns}{\@ehc}}
\newcommand*{\led@err@UnequalColumns}{%
  \eledmac@error{Number of columns is not equal to the number
                 in the previous row (or \protect\\ \space forgotten?)}{\@ehc}}
\newcommand*{\led@err@LowStartColumn}{%
  \eledmac@error{Start column is too low}{\@ehc}}
\newcommand*{\led@err@HighEndColumn}{%
  \eledmac@error{End column is too high}{\@ehc}}
\newcommand*{\led@err@ReverseColumns}{%
  \eledmac@error{Start column is greater than end column}{\@ehc}}
```

19 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. LaTeX will maintain and display a ‘section number’ as a count 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 `\renewcommand{\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
```

`\ifnumberingR`
`\ifl@dpairing`
`\l@dpairingtrue`
`\l@dpairingfalse`
`\ifpst@rtedL`
`\pst@rtedLtrue`
`\pst@rtedLfalse`
`\l@dnumpstartsL`
`\ifledRcol`

In preparation for the eledpar package, these are related to the 'left' text of parallel texts (when `\ifl@dpairing` is TRUE). They are explained in the eledpar manual.

```
\newif\ifl@dpairing
  \l@dpairingfalse
\newif\ifpst@rtedL
  \pst@rtedLfalse
\newcount\l@dnumpstartsL
\newif\ifledRcol
```

The `\ifnumberingR` flag is set to `true` if we're within a right text numbered section.

```
\newif\ifnumberingR
```

`\beginnumbering`
`\initnumbering@reg`

`\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.

```
\newcommand*{\beginnumbering}{%
  \ifnumbering
    \led@err@NumberingStarted
    \endnumbering
  \fi
  \global\numberingtrue
  \global\advance\section@num \@ne
  \initnumbering@reg
  \message{Section \the\section@num }%
  \line@list@stuff{\jobname.\extensionchars\the\section@num}%
  \l@dend@stuff
  \setcounter{pstart}{1}
  \ifl@dpairing\else\begingroup\fi
  \initnumbering@sectcmd
}
\newcommand*{\initnumbering@reg}{%
  \global\pst@rtedLfalse
  \global\l@dnumpstartsL \z@
  \global\absline@num \z@
  \global\line@num \z@
  \global\subline@num \z@
  \global\@lock \z@
  \global\sub@lock \z@
  \global\sublines@false
  \global\let\next@page@num=\relax
  \global\let\sub@change=\relax
  \resetprevline@
  }

```

\initnumbering@sectcmd
\ledsection
\ledsection*
\ledsubsection
\ledsubsection*
\ledsubsubsection
\ledsubsubsection*
\ledchapter
\ledchapter*
\quotation
\endquotation
\quote
\endquote

`\initnumbering@sectcmd` defines sectioning commands inside numbered section. It also defines quotation environment. Note: this supposes that the user didn't change `\chapter`. If he did, he should redefine `\initnumbering@sectcmd`.

```
\newcommand{\initnumbering@sectcmd}{
    \newcommand{\ledsection}[2][]{%
        \leavevmode\pend\vspace{3.5ex \@plus 1ex \@minus .2ex}\skipnumbering%
        \pstart%
        \leavevmode\section[##1]{##2}\leavevmode\vspace{2.3ex \@plus.2ex}\skipnumbering\pend%
        \vspace{-2\parskip}\vspace{-2\baselineskip}%
        \ifautopar\else\pstart\fi
    }
    \WithSuffix\newcommand\ledsection*[1]{%
        \leavevmode\pend\vspace{3.5ex \@plus 1ex \@minus .2ex}\skipnumbering%
        \pstart%
        \leavevmode\section*{##1}\leavevmode\vspace{2.3ex \@plus.2ex}\skipnumbering\pend%
        \vspace{-2\parskip}\vspace{-2\baselineskip}%
        \ifautopar\else\pstart\fi
    }
    \newcommand{\ledsubsection}[2][]{%
        \leavevmode\pend\vspace{3.5ex \@plus 1ex \@minus .2ex}\skipnumbering%
        \pstart%
        \leavevmode\subsection[##1]{##2}\leavevmode\vspace{1.5ex \@plus .2ex}\skipnumbering\pend%
        \vspace{-2\parskip}\vspace{-2\baselineskip}%
        \ifautopar\else\pstart\fi
```

```
}
\WithSuffix\newcommand\ledsubsection*[1]{%
    \leavevmode\pend\vspace{3.5ex \@plus 1ex \@minus .2ex}\skipnumbering%
    \pstart%
    \leavevmode\subsection*{##1}\leavevmode\vspace{1.5ex \@plus .2ex}\skipnumbering\pe
    \vspace{-2\parskip}\vspace{-2\baselineskip}%
    \ifautopar\else\pstart\fi
}
\newcommand{\ledsubsubsection}[2][]{%
    \leavevmode\pend\vspace{3.5ex \@plus 1ex \@minus .2ex}\skipnumbering%
    \pstart%
    \leavevmode\subsubsection[##1]{##2}\leavevmode\vspace{1.5ex \@plus .2ex}\skipnumbe
    \vspace{-2\parskip}\vspace{-2\baselineskip}%
    \ifautopar\else\pstart\fi
}
\WithSuffix\newcommand\ledsubsubsection*[1]{%
    \leavevmode\pend\vspace{3.5ex \@plus 1ex \@minus .2ex}\skipnumbering%
    \pstart%
    \leavevmode\subsubsection*{##1}\leavevmode\vspace{1.5ex \@plus .2ex}\skipnumbering
    \vspace{-2\parskip}\vspace{-2\baselineskip}%
    \ifautopar\else\pstart\fi
}
\newcommand\ledchapter[2][]{\ifl@dmemoir\gdef\ch@pt@c{##1}\fi~\pend\skipnumbering\psta
\WithSuffix\newcommand\ledchapter*[1]{~\pend\skipnumbering\pstart\chapter*{##1}\pend\p
\patchcmd{\@makeschapterhead}{1\par}{1}{}{}
\pretocmd{\@makeschapterhead}{\par}{}{}
\apptocmd{\@makeschapterhead}{\par}{}{}
\patchcmd{\@makeschapterhead}{\vskip 40\p@}{}{}{}
\patchcmd{\@makechapterhead}{1\par}{1}{}{}
\pretocmd{\@makechapterhead}{\par}{}{}
\apptocmd{\@makechapterhead}{\par}{}{}
\patchcmd{\@makechapterhead}{\vskip 40\p@}{}{}{}
\apptocmd{\@chapter}{\par\leavevmode\vspace{40 \p@}\skipnumbering}{}{}
\apptocmd{\@schapter}{\par\leavevmode\vspace{40 \p@}\skipnumbering}{}{}
\newcommand\beforeledchapter{\pend\cleardoublepage\pstart}
\patchcmd{\chapter}{\cleardoublepage}{\relax}{}{}
\patchcmd{\chapter}{\clearpage}{\relax}{}{}
\ifnoquotation@\else
\renewcommand{\quotation}{\par\leavevmode%
                            \parindent=1.5em%
                            \skipnumbering%
                            \ifautopar%
                                \vskip-\parskip%
                            \else%
                                \vskip\topsep%
                            \fi%
                            \global\leftskip=\leftmargin%
                            \global\rightskip=\leftmargin%
                        }
\renewcommand{\endquotation}{\par%
```

```
                            \global\leftskip=0pt%
                              \global\rightskip=0pt%
                              \leavevmode%
                              \skipnumbering%
                              \ifautopar%
                                  \vskip-\parskip%
                              \else%
                                  \vskip\topsep%
                              \fi%
                              }
    \renewcommand{\quote}{\par\leavevmode%
                              \parindent=0pt%
                              \skipnumbering%
                              \ifautopar%
                                  \vskip-\parskip%
                              \else%
                                  \vskip\topsep%
                              \fi%
                              \global\leftskip=\leftmargin%
                              \global\rightskip=\leftmargin%
    }
    \renewcommand{\endquote}{\par%
                              \global\leftskip=0pt%
                              \global\rightskip=0pt%
                              \leavevmode%
                              \skipnumbering%
                              \ifautopar%
                                  \vskip-\parskip%
                              \else%
                                  \vskip\topsep%
                              \fi%
                              }
    \fi
}
```

`\ledsectnotoc` The `\ledsectnotoc` only disables the `\addcontentsline` macro.

```
\newcommand{\ledsectnotoc}{\let\addcontentsline\@gobblethree}
```

`\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
    \ifl@dpairing
      \global\pst@rtedLfalse
    \else
      \ifx\insertlines@list\empty\else
```

```
        \global\noteschanged@true
      \fi
      \ifx\line@list\empty\else
        \global\noteschanged@true
      \fi
    \fi
    \ifnoteschanged@
      \led@mess@NotesChanged
    \fi
  \else
    \led@err@NumberingNotStarted
  \fi
  \autoparfalse\ifl@dpairing\else\endgroup\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.[22]

```
\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.

```
\newcommand*{\resumenumbering}{%
  \ifnumbering
     \global\pst@rtedLtrue
     \global\advance\section@num \@ne
     \led@mess@SectionContinued{\the\section@num}%
     \line@list@stuff{\jobname.\extensionchars\the\section@num}%
     \l@dend@stuff
  \else
    \led@err@NumberingShouldHaveStarted
    \endnumbering
    \beginnumbering
  \fi}

```

20 Line counting

20.1 Choosing the system of lineation

Sometimes you want line numbers that start at 1 at the top of each page; sometimes you want line numbers that start at 1 at each `\pstart`; other times you want line numbers that start at 1 at the start of each section and increase regardless of page breaks. eledmac 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

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

line references within each section. Here we will define internal codes for these systems and the macros you use to select them.

`\ifbypstart@` `\bypstart@true` `\bypstart@false` `\ifbypage@` `\bypage@true` `\bypage@false`

The `\ifbypage@` and `\ifbypstart@` flag specifie the current lineation system:

- line-of-page: `bypstart@ = false` and `bypage@ = true`.
- line-of-pstart: `bypstart@ = true` and `bypage@ = false`.

eledmac will use the line-of-section system unless instructed otherwise.

```
\newif\ifbypage@
\newif\ifbypstart@
```

`\lineation`

`\lineation{`⟨*word*⟩`}` is the macro you use to select the lineation system. Its argument is a string: either `page` or `section` or `pstart`.

```
\newcommand*{\lineation}[1]{{%
  \ifnumbering
    \led@err@LineationInNumbered
  \else
    \def\@tempa{#1}\def\@tempb{page}%
    \ifx\@tempa\@tempb
        \global\bypage@true
        \global\bypstart@false
        \pstartinfootnote[][false]
    \else
       \def\@tempb{pstart}%
       \ifx\@tempa\@tempb
           \global\bypage@false
           \global\bypstart@true
           \pstartinfootnote
       \else
          \def\@tempb{section}
          \ifx\@tempa\@tempb
           \global\bypage@false
           \global\bypstart@false
           \pstartinfootnote[][false]
          \else
            \led@warn@BadLineation
        \fi
       \fi
    \fi
  \fi}}
```

`\linenummargin` `\line@margin` `\l@dgetline@margin`

You call `\linenummargin{`⟨*word*⟩`}` 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 the count `\line@margin`: 0 for left, 1 for right, 2 for outer, and 3 for inner.

```
\newcount\line@margin
\newcommand*{\linenummargin}[1]{{%
  \l@dgetline@margin{#1}%
  \ifnum\@l@dtempcntb>\m@ne
    \global\line@margin=\@l@dtempcntb
  \fi}}
\newcommand*{\l@dgetline@margin}[1]{%
  \def\@tempa{#1}\def\@tempb{left}%
  \ifx\@tempa\@tempb
    \@l@dtempcntb \z@
  \else
     \def\@tempb{right}%
     \ifx\@tempa\@tempb
       \@l@dtempcntb \@ne
     \else
       \def\@tempb{outer}%
       \ifx\@tempa\@tempb
         \@l@dtempcntb \tw@
       \else
         \def\@tempb{inner}%
         \ifx\@tempa\@tempb
           \@l@dtempcntb \thr@@
         \else
           \led@warn@BadLinenummargin
           \@l@dtempcntb \m@ne
         \fi
       \fi
     \fi
  \fi}

```

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

The following counters tell eledmac 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.

```
\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.

```
\newcounter{firstsublinenum}
```

```
  \setcounter{firstsublinenum}{5}
\newcounter{sublinenumincrement}
  \setcounter{sublinenumincrement}{5}

```

`\firstlinenum` `\linenumincrement` `\firstsublinenum` `\sublinenumincrement`

These macros can be used to set the corresponding counters.

```
\newcommand*{\firstlinenum}[1]{\setcounter{firstlinenum}{#1}}
\newcommand*{\linenumincrement}[1]{\setcounter{linenumincrement}{#1}}
\newcommand*{\firstsublinenum}[1]{\setcounter{firstsublinenum}{#1}}
\newcommand*{\sublinenumincrement}[1]{\setcounter{sublinenumincrement}{#1}}

```

`\lockdisp` `\lock@disp` `\l@dgetlock@disp`

When line locking is being used, the `\lockdisp{`⟨*word*⟩`}` 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
\newcommand{\lockdisp}[1]{{%
  \l@dgetlock@disp{#1}%
  \ifnum\@l@dtempcntb>\m@ne
    \global\lock@disp=\@l@dtempcntb
  \else
    \led@warn@BadLockdisp
  \fi}}
\newcommand*{\l@dgetlock@disp}[1]{
  \def\@tempa{#1}\def\@tempb{first}%
  \ifx\@tempa\@tempb
    \@l@dtempcntb \z@
  \else
     \def\@tempb{last}%
     \ifx\@tempa\@tempb
       \@l@dtempcntb \@ne
     \else
       \def\@tempb{all}%
       \ifx\@tempa\@tempb
         \@l@dtempcntb \tw@
       \else
         \@l@dtempcntb \m@ne
       \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
\newcommand{\sublockdisp}[1]{{%
  \l@dgetlock@disp{#1}%
  \ifnum\@l@dtempcntb>\m@ne
    \global\sublock@disp=\@l@dtempcntb
  \else
    \led@warn@BadSublockdisp
  \fi}}

```

\linenumberstyle
\linenumrep
\linenumr@p
\sublinenumberstyle
\sublinenumrep
\sublinenumr@p

We provide a mechanism for using different representations of the line numbers, not just the normal arabic.

NOTE: In v0.7 `\linenumrep` and `\sublinenumrep` replaced the internal `\linenumr@p` and `\sublinenumr@p`.

`\linenumberstyle` and `\sublinenumberstyle` are user level macros for setting the number represention (`\linenumrep` and `\sublinenumrep`) for line and sub-line numbers.

```
\newcommand*{\linenumberstyle}[1]{%
  \def\linenumrep##1{\@nameuse{@#1}{##1}}}
\newcommand*{\sublinenumberstyle}[1]{%
  \def\sublinenumrep##1{\@nameuse{@#1}{##1}}}
```

Initialise the number styles to arabic.

```
\linenumberstyle{arabic}
  \let\linenumr@p\linenumrep
\sublinenumberstyle{arabic}
  \let\sublinenumr@p\sublinenumrep

```

\leftlinenum
\rightlinenum
\linenumsep
\numlabfont
\ledlinenum

`\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.

`\ledlinenum` typesets the line (and subline) number.

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

```
\newlength{\linenumsep}
  \setlength{\linenumsep}{1pc}
\newcommand*{\numlabfont}{\normalfont\scriptsize}
\newcommand*{\ledlinenum}{%
  \numlabfont\linenumrep{\line@num}%
  \ifsublines@
```

```
    \ifnum\subline@num>0\relax
      \unskip\fullstop\sublinenumrep{\subline@num}%
    \fi
  \fi}
\newcommand*{\leftlinenum}{%
  \ledlinenum
  \kern\linenumsep}
\newcommand*{\rightlinenum}{%
  \kern\linenumsep
  \ledlinenum}

```

20.2 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 eledmac this macro doesn't do anything beyond initializing an empty list macro, but in future versions it may do more.

```
\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 eledmac it is no different from `\list@create`.

```
\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}}

```

20.3 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 LaTeX 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 count `\line@num` 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`.

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

\subline@num The count `\subline@num` 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.

```
\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.

```
448 \newif\ifsublines@
```

`\absline@num` The count `\absline@num` 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.

```
449 \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 counts `\@lock` and `\sub@lock` 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.

```
450 \newcount\@lock
451 \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 `\edtext`. 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. The font specifier is a set of four codes for font encoding, family, series, and shape, separated by `/` characters. 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 normal roman font would have a line list entry like this:
  `23|35|0|24|3|0|OT1/cmr/m/n`.

  There is one item in this list for every lemma marked by `\edtext`, even if there are several notes to that lemma, or no notes at all. `\edtext` 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 eledmac what action it's supposed to take at each of these lines. One action, the page-start action, is generated behind the scenes by eledmac 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. eledmac 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 eledmac 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 counts 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 the 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 LaTeX, 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 LaTeX 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@
```

\resetprevline@ Inside the apparatus, at each note, the line number is memorized in a macro called `\prevlineX`, where X is the letter of the current series. This macro is called when using `\numberonlyfirstinline`. This maco must be reset at the same time as the line number. The `\resetprevline@` does this resetting for every series.

\resetprevline@

```
\newcommand*{\resetprevline@}{%
    \def\do##1{\global\csundef{prevline##1}}%
    \dolistloop{\@series}%
}
```

20.4 Reading the line-list file

\read@linelist `\read@linelist{`⟨*file*⟩`}` 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.

```
\newread\@inputcheck
\newcommand*{\read@linelist}[1]{%
  \list@clearing@reg
```

When the file is 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 LaTeX context. We ignore carriage returns, since if we're in horizontal mode they can get interpreted as spaces to be printed.

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.

If the file is not there, we print an informative message.

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

```
  \get@linelistfile{#1}%
  \endgroup
```

When the reading 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=\m@ne
  \ifx\actionlines@list\empty
      \gdef\next@actionline{1000000}%
  \else
      \gl@p\actionlines@list\to\next@actionline
      \gl@p\actions@list\to\next@action
  \fi}

```

`\list@clearing@reg` Clears the lists for `\read@linelist`

```
\newcommand*{\list@clearing@reg}{%
  \list@clear{\line@list}%
  \list@clear{\insertlines@list}%
  \list@clear{\actionlines@list}%
  \list@clear{\actions@list}}
```

`\get@linelistfile` eledmac can take advantage of the LaTeX 'safe file input' macros to get the line-list file.

```
\newcommand*{\get@linelistfile}[1]{%
  \InputIfFileExists{#1}{%
    \global\noteschanged@false
    \begingroup
      \catcode`\[=1 \catcode`\]=2
      \makeatletter \catcode`\^^M=9}{%
    \led@warn@NoLineFile{#1}%
    \global\noteschanged@true
    \begingroup}%
}

```

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. 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 LaTeX 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. 11 above).

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

In order to get the `\setlinenum` to work Peter Wilson had to slip in some new code at the start of the macro, to get the timing of the actions correct. The problem was that his original naive implementation of `\setlinenum` had a unfortunate tendency to change the number of the last line of the *preceding* paragraph. The new code is sort of based on the page number handling and `\setline` It seems that a lot of fiddling with the line number internals is required.

In November 2004 in order to accurately determine page numbers Peter Wilson added these to the macro. It is now:
`\@l{`⟨*page counter number*⟩`}{`⟨*printed page number*⟩`}`
I don't (yet) use the printed number (i.e., the `\thepage`) but it may come in handy later. The macro `\fix@page` checks if a new page has started.

```
\newcommand{\@l}[2]{%
  \fix@page{#1}%
  \@l@reg}
\newcommand*{\@l@reg}{%
  \ifx\l@dchset@num\relax \else
    \advance\absline@num \@ne
    \set@line@action
    \let\l@dchset@num=\relax
    \advance\absline@num \m@ne
    \advance\line@num \m@ne
  \fi
```

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

```
  \advance\absline@num \@ne
         \ifx\next@page@num\relax \else
              \page@action
              \let\next@page@num=\relax
         \fi
         \ifx\sub@change\relax \else
             \ifnum\sub@change>\z@
                \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 \tw@
           \or \or
              \@lock \z@
        \fi
        \ifcase\sub@lock
           \or
              \sub@lock \tw@
           \or \or
              \sub@lock \z@
        \fi
```

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

```
        \ifsublines@
             \ifnum\sub@lock<\tw@
               \advance\subline@num \@ne
             \fi
        \else
             \ifnum\@lock<\tw@
               \advance\line@num \@ne \subline@num \z@
             \fi
        \fi}

```

\@page

`\@page{`⟨*num*⟩`}` 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 count.

```
\newcommand*{\@page}[1]{%
  \ifbypage@
    \line@num \z@ \subline@num \z@
  \fi
  \page@num=#1\relax
```

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.

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

```

\last@page@num
\fix@page

`\fix@page` basically replaces `\@page`. It determines whether or not a new page has been started, based on the page values held by `\@l`.

```
\newcount\last@page@num
  \last@page@num=-10000
\newcommand*{\fix@page}[1]{%
  \ifnum #1=\last@page@num
  \else
    \ifbypage@
      \line@num=\z@ \subline@num=\z@
    \fi
    \page@num=#1\relax
    \last@page@num=#1\relax
    \def\next@page@num{#1}%
  \fi}

```

\@pend \@pendR \@lopL \@lopR These don't do anything at this point, but will have been added to the auxiliary file(s) if the eledpar package has been used. They are just here to stop eledmac from moaning if the eledpar is used for one run and then not for the following one.

```
\newcommand*{\@pend}[1]{}
\newcommand*{\@pendR}[1]{}
\newcommand*{\@lopL}[1]{}
\newcommand*{\@lopR}[1]{}

```

\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.

```
\newcommand*{\sub@on}{\ifsublines@
    \let\sub@change=\relax
  \else
    \def\sub@change{1}%
  \fi}
\newcommand*{\sub@off}{\ifsublines@
    \def\sub@change{-1}%
  \else
    \let\sub@change=\relax
  \fi}

```

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

```
\newcommand*{\@adv}[1]{\ifsublines@
      \advance\subline@num by #1\relax
      \ifnum\subline@num<\z@
        \led@warn@BadAdvancelineSubline
         \subline@num \z@
      \fi
  \else
      \advance\line@num by #1\relax
      \ifnum\line@num<\z@
        \led@warn@BadAdvancelineLine
         \line@num \z@
      \fi
  \fi
  \set@line@action}

```

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

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

```

\l@d@set
\l@dchset@num

The `\l@d@set{`⟨*num*⟩`}` macro sets the line number for the next `\pstart...` to the value specified as its argument. This is used to implement `\setlinenum`.

`\l@dchset@num` is a flag to the `\@l` macro. If it is not `\relax` then a linenumber change is to be done.

```
\newcommand*{\l@d@set}[1]{%
  \line@num=#1\relax
  \advance\line@num \@ne
  \def\l@dchset@num{#1}}
\let\l@dchset@num\relax

```

\page@action

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

```
\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.

```
\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.

```
\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
\do@lockonL

`\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.

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

```

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

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

```
\newcommand*{\do@lockoffL}{%
  \xright@appenditem{\the\absline@num}\to\actionlines@list
  \ifsublines@
    \xright@appenditem{-1006}\to\actions@list
    \ifnum\sub@lock=\tw@
      \sub@lock \thr@@
    \else
      \sub@lock \z@
    \fi
  \else
    \xright@appenditem{-1004}\to\actions@list
    \ifnum\@lock=\tw@
      \@lock \thr@@
    \else
```

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

```

\n@num \n@num@reg This macro implements the \skipnumbering command. It uses a new action code, namely 1007.

```
\newcommand*{\n@num}{\n@num@reg}
\newcommand*{\n@num@reg}{%
  \xright@appenditem{\the\absline@num}\to\actionlines@list
  \xright@appenditem{-1007}\to\actions@list}

```

\@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 \edtext, which computes it and writes it to the line-list file, will be stored in the count \insert@count.

```
    \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 \edtext within the first argument of another instance of \edtext.)

\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.

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

\@ref@reg The first thing \@ref (i.e. \@ref@reg) itself does is to add the specified number of items to the \insertlines@list list.

```
\newcommand*{\@ref}[2]{%
  \@ref@reg{#1}{#2}}
\newcommand*{\@ref@reg}[2]{%
  \global\insert@count=#1\relax
  \loop\ifnum\insert@count>\z@
    \xright@appenditem{\the\absline@num}\to\insertlines@list
    \global\advance\insert@count \m@ne
  \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}

```

20.6 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 eledmac 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{⟨file⟩}` 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.

```
\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\relax
  \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.

```
     \closeout\linenum@out
     \openout\linenum@out=#1\relax
  \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, and its page number.

```
\newcommand*{\new@line}{\write\linenum@out{\string\@l[\the\c@page][\thepage]}}
```

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

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

\page@start Originally the commentary was: `\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.

However, in October 2004 Alexej Krukov discovered that when processing long paragraphs that included Russian, Greek and Latin texts eledmac would go into an infinite loop, emitting thousands of blank pages. This was caused by being unable to find an appropriate place in the output routine. A different algorithm is now used for getting page numbers.

```
\newcommand*{\page@start}{}

```

`\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.

```
\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{`⟨*num*⟩`}` in running text to advance the current visible line-number by a specified value, positive or negative.

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

`\setline` You can use `\setline{`⟨*num*⟩`}` in running text (i.e., within `\pstart...\pend`) to set the current visible line-number to a specified positive value.

```
\newcommand*{\setline}[1]{%
  \ifnum#1<\z@
    \led@warn@BadSetline
  \else
    \write\linenum@out{\string\@set[#1]}%
  \fi}

```

`\setlinenum` You can use `\setlinenum{`⟨*num*⟩`}` before a `\pstart` to set the visible line-number to a specified positive value. It writes a `\l@d@set` command to the line-list file.

```
\newcommand*{\setlinenum}[1]{%
  \ifnum#1<\z@
    \led@warn@BadSetlinenum
  \else
    \write\linenum@out{\string\l@d@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.

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

```

`\ifl@dskipnumber` `\l@dskipnumbertrue` `\l@dskipnumberfalse` `\skipnumbering` `\skipnumbering@reg` In numbered text `\skipnumbering` will suspend the numbering for that particular line.

```
\newif\ifl@dskipnumber
  \l@dskipnumberfalse
\newcommand*{\skipnumbering}{\skipnumbering@reg}
\newcommand*{\skipnumbering@reg}{%
  \write\linenum@out{\string\n@num}%
  \advanceline{-1}}

```

21 Marking text for notes

The `\edtext` (or `\critext`) 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.

For convenience, I will use `\*text` when I do not need to distinguish between `\edtext` and `\critext`. The `\*text` macros take two arguments, the only difference between `\edtext` and `\critext` is how the second argument is delineated.

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

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

Similarly `\edtext` requires the same two arguments but you use it by saying:

```
\edtext{#1}{#2}
```

- `#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. With `\critext` 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 `\critext` are used.) Braces around `#2` are optional with `\critext` and required for `\edtext`.

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. 80). 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 `\l@d@nums` that *it* saw, and then performs a simple conditional test to see whether to print a number or a '‖'.

21.1 `\edtext` and `\critext` themselves

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.

`\end@lemmas` 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 `\critext`'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 `\critext` 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.

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

`\dummy@text` We now need to define a number of macros that allow us to weed out nested instances of `\critext`, 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 `\critext` macros create nested `\@ref` entries in the line-list file.

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

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

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

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

We're going to need another macro that takes one argument and ignores it entirely. This is supplied by the LaTeX `\@gobble{`⟨*arg*⟩`}`.

`\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.[23] 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

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

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 `\protect` in front of it in your file.)

We also need to eliminate all eledmac macros like `\edlabel` and `\setline` that write things to auxiliary files: that writing should be done only once. And we make `\critext` 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 eledmac 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 `\critext` 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 `\critext`. 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 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.)

```
\newcommand*{\no@expands}{%
  \let\select@@lemmafont=0%
  \let\startsub=\relax  \let\endsub=\relax
  \let\startlock=\relax \let\endlock=\relax
  \let\edlabel=\@gobble
  \let\setline=\@gobble \let\advanceline=\@gobble
  \let\critext=\dummy@text
  \let\edtext=\dummy@edtext
  \l@dtabnoexpands
  \morenoexpands}
\let\morenoexpands=\relax

```

`\@tag` Now, we define an empty `\@tag` command. It will be redefine by `\edtext`: its value is the first args. It will be used by the `\Xfootnote` commands.

```
\newcommand{\@tag}{}
%  \end{macrocode}
% \end{macro}
% \begin{macro}{\critext}
% Now we begin \cs{critext} itself. The definition requires a \verb"/" after
% the arguments: this eliminates the possibility of problems about
% knowing where \verb"#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 \verb"#2" is
% a 'delimited parameter'. Since \cs{critext} is always used in running
% text, it seems more appropriate to pay attention to spaces than to
% skip them.
%
% When executed, \cs{critext} first ensures that we're in
% horizontal mode.
%    \begin{macrocode}
\long\def\critext#1#2/{\leavevmode
```

`\@tag` Our normal lemma is just argument `#1`; but that argument could have further invocations of `\critext` within it. We get a copy of the lemma without any `\critext` macros within it by temporarily redefining `\critext` 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 `\critext` 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
    \global\renewcommand{\@tag}{\no@expands #1}%%
```

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

```
    \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.

```
   \@ifundefined{xpg@main@language}{%if not polyglossia
       \flag@start}%
```

```
      {\if@RTL\flag@end\else\flag@start\fi% With polyglossia, you must track whether the
      }
  \endgroup
  \showlemma{#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
  \@ifundefined{xpg@main@language}{%if not polyglossia
      \flag@end}%
      {\if@RTL\flag@start\else\flag@end\fi% With polyglossia, you must track whether the
      }
  }
```

\edtext

```
\newcommand{\edtext}[2]{\leavevmode
  \begingroup
    \global\renewcommand{\@tag}{\no@expands #1}%%
    \set@line
    \global\insert@count=0
    \ignorespaces #2\relax
    \@ifundefined{xpg@main@language}{%if not polyglossia
      \flag@start}%
      {\if@RTL\flag@end\else\flag@start\fi% With polyglossia, you must track whether the
      }%
  \endgroup
  \showlemma{#1}%
  \ifx\end@lemmas\empty \else
    \gl@p\end@lemmas\to\x@lemma
    \x@lemma
    \global\let\x@lemma=\relax
  \fi
  \@ifundefined{xpg@main@language}{%if not polyglossia
      \flag@end}%
      {\if@RTL\flag@start\else\flag@end\fi% With polyglossia, you must track whether the
      }%
  }

```

\ifnumberline The \ifnumberline option can be set to FALSE to disable line numbering.

```
\newif\ifnumberline
\numberlinetrue
```

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

One instance of `\critext` may generate several notes, or it may generate none—it's legitimate for argument `#2` to `\critext` 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.

```
\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\l@d@nums{000|000|000|000|000|000|\edfont@info}%
  \else
    \gl@p\line@list\to\@tempb
    \xdef\l@d@nums{\@tempb|\edfont@info}%
    \global\let\@tempb=\undefined
  \fi}

```

\edfont@info The macro `\edfont@info` returns coded information about the current font.

```
\newcommand*{\edfont@info}{\f@encoding/\f@family/\f@series/\f@shape}

```

21.2 Substitute lemma

\lemma The `\lemma{`⟨*text*⟩`}` macro allows you to change the lemma that's passed on to the notes.

```
\newcommand*{\lemma}[1]{\global\renewcommand{\@tag}{\no@expands #1}}
```

21.3 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 `\l@d@nums` (see p. 55): 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.

```
\newcommand*{\linenum}[1]{%
  \xdef\@tempa{#1|||||||\noexpand\\\l@d@nums}%
  \global\let\l@d@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 `\l@d@nums`, and then calls itself to process the next number in the `\linenum` argument, if there are more numbers in `\l@d@nums` to process.

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

\l@d@add

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

```
\newcommand{\l@d@add}[1]{\xdef\l@d@nums{\l@d@nums#1}}

```

22 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.

22.1 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
\numberpstarttrue
\numberpstartfalse
\labelpstarttrue
\labelpstartfalse
thepstart

\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.

You can use the command \numberpstarttrue to insert a number on every \pstart. To stop the numbering, you must use \numberpstartfalse. To reset the numebering of \pstarts, insert

```
\setcounter{pstart}{0}
```

```

\newcounter{pstart}
\renewcommand{\thepstart}{{\bfseries\@arabic\c@pstart}. }
\newif\ifnumberpstart
\numberpstartfalse
\newif\iflabelpstart
\labelpstartfalse
\newcommand*{\pstart}{
\if@nobreak
\let\@oldnobreak\@nobreaktrue
\else
\let\@oldnobreak\@nobreakfalse
\fi
\@nobreaktrue
\ifnumbering \else
    \led@err@PstartNotNumbered
    \beginnumbering
  \fi
  \ifnumberedpar@
    \led@err@PstartInPstart
    \pend
  \fi
  \list@clear{\inserts@list}%
  \global\let\next@insert=\empty
  \begingroup\normal@pars
  \global\setbox\raw@text=\vbox\bgroup\ifautopar\else\ifnumberpstart\ifinstanza\else\ifsidepstartnum\
  \numberedpar@true
  \iflabelpstart\protected@edef\@currentlabel
       {\p@pstart\thepstart}\fi
  }
```

\pend

\pend must be used to end a numbered paragraph.

```
\newcommand*{\pend}{\ifnumbering \else
    \led@err@PendNotNumbered
```

```
  \fi
  \ifnumberedpar@ \else
    \led@err@PendNoPstart
  \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.

```
  \l@dzeropenalties
  \endgraf\global\num@lines=\prevgraf\egroup
  \global\par@line=0
```

We check if lineation is by pstart: in this case, we reset line number, but only in the second line of the pstart, to prevent some trouble. We can't reset line number at the beginning of `\pstart` `\setline` is parsed at the end of previous `\pend`, and so, we must do it at the end of first line of pstart.

```
  \csnumdef{pstartline}{0}
  \loop\ifvbox\raw@text
    \csnumdef{pstartline}{\pstartline+1}%
    \do@line
    \ifbypstart@%
     \ifnumequal{\pstartline}{1}{\setline{1}\resetprevline@}{}%
    \fi
  \repeat
```

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

```
  \flush@notes
  \endgroup
  \ignorespaces
  \ifnumberpstart
   \pstartnumtrue
   \fi
  \@oldnobreak
  \addtocounter{pstart}{1}}

```

`\l@dzeropenalties` A macro to zero penalties for `\pend`.

```
\newcommand*{\l@dzeropenalties}{%
  \brokenpenalty \z@ \clubpenalty \z@
  \displaywidowpenalty \z@ \interlinepenalty \z@ \predisplaypenalty \z@
  \postdisplaypenalty \z@ \widowpenalty \z@}

```

`\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.

```
\newif\ifautopar
\autoparfalse
\newcommand*{\autopar}{
  \ifledRcol
    \ifnumberingR \else
    \led@err@AutoparNotNumbered
    \beginnumberingR
    \fi
    \else
    \ifnumbering \else
    \led@err@AutoparNotNumbered
    \beginnumbering
    \fi
  \fi
  \autopartrue
  \everypar={\setbox0=\lastbox
    \endgraf \vskip-\parskip
    \pstart \noindent \kern\wd0 \ifnumberpstart\ifinstanza\else\thepstart\fi\fi
    \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 a footnotes, for example.

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

```

22.2 Processing one line

\do@line
\l@dunhbox@line

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

```
\newcommand*{\l@dunhbox@line}[1]{\unhbox #1}
\newcommand*{\do@line}{%
 {\vbadness=10000
  \splittopskip=\z@
  \do@linehook
\l@demptyd@ta
  \global\setbox\one@line=\vsplit\raw@text to\baselineskip}%
 \unvbox\one@line \global\setbox\one@line=\lastbox
 \getline@num
 \ifnum\@lock>\@ne
   \inserthangingsymboltrue
 \else
   \inserthangingsymbolfalse
 \fi
 \affixline@num
 \affixpstart@num
 \hb@xt@ \linewidth{\do@insidelinehook\inserthangingsymbol\l@dld@ta\add@inserts\affixside
   \l@dlsn@te
   {\ledllfill\hb@xt@ \wd\one@line{\new@line\l@dunhbox@line{\one@line}}\ledrlfill\l@drd@t
   \l@drsn@te
 }}}%
```

\do@linehook
\do@insidelinehook

Two hooks into `\do@line`. The first is called at the beginning of `\do@line`, the second is called in the line box. The second can, for example, have a `\markboth` command inside, the first can't.

```
\newcommand*{\do@linehook}{}
\newcommand*{\do@insidelinehook}{}
```

\l@demptyd@ta
\l@dld@ta
\l@drd@ta
\l@dcsnotetext

Nulls the `\...d@ta`, which may later hold line numbers. Similarly for `\l@dcsnotetext` for the texts of the sidenotes.

```
\newcommand*{\l@demptyd@ta}{%
 \gdef\l@dld@ta{}%
 \gdef\l@drd@ta{}%
 \gdef\l@dcsnotetext{}}

```

\l@dlsn@te
\l@drsn@te

Zero width boxes of the left and right side notes, together with their kerns.

```
\newcommand{\l@dlsn@te}{%
 \hb@xt@ \z@{\hss\box\l@dlp@rbox\kern\ledlsnotesep}}
\newcommand{\l@drsn@te}{%
  \hb@xt@ \z@{\kern\ledrsnotesep\box\l@drp@rbox\hss}}

```

\ledllfill
\ledrlfill

These macros are called at the left (`\ledllfill`) and the right (`\ledllfill`) of each numbered line. The initial definitions correspond to the original code for `\do@line`.

```
\newcommand*{\ledllfill}{\hfil}
\newcommand*{\ledrlfill}{}

```

22.3 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.

```
\newcommand*{\getline@num}{%
    \global\advance\absline@num \@ne
  \do@actions
  \do@ballast
  \ifnumberline
  \ifsublines@
     \ifnum\sub@lock<\tw@
       \global\advance\subline@num \@ne
     \fi
  \else
     \ifnum\@lock<\tw@
       \global\advance\line@num \@ne
       \global\subline@num \z@
     \fi
  \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 count `\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. 90).

\ballast@count
\c@ballast

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

```
\newcount\ballast@count
\newcounter{ballast}
  \setcounter{ballast}{0}
```

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.

```
\newcommand*{\do@ballast}{\global\ballast@count \z@
  \begingroup
    \advance\absline@num \@ne
    \ifnum\next@actionline=\absline@num
      \ifnum\next@action>-1001\relax
        \global\advance\ballast@count by -\c@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`.

```
\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=\z@ \global\subline@num=\z@
        \resetprevline@
      \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
         \ifsublines@
            \global\subline@num=\@l@dtempcnta
         \else
            \global\line@num=\@l@dtempcnta
         \fi
```

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

```
      \else
         \@l@dtempcnta=-\next@action
         \advance\@l@dtempcnta by -1000
         \do@actions@fixedcode
      \fi
    \fi
```

Now we get information about the next action off the list, and then set `\do@actions@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}

```

`\do@actions@fixedcode` This macro handles the fixed codes for `\do@actions`. It is one big case statement.

```
\newcommand*{\do@actions@fixedcode}{%
  \ifcase\@l@dtempcnta
  \or%                   % 1001
    \global\sublines@true
  \or%                   % 1002
    \global\sublines@false
  \or%                   % 1003
      \global\@lock=\@ne
  \or%                   % 1004
    \ifnum\@lock=\tw@
      \global\@lock=\thr@@
    \else
      \global\@lock=\z@
    \fi
  \or%                   % 1005
     \global\sub@lock=\@ne
  \or%                   % 1006
    \ifnum\sub@lock=\tw@
      \global\sub@lock=\thr@@
    \else
      \global\sub@lock=\z@
    \fi
  \or%                   % 1007
    \l@dskipnumbertrue
  \else
    \led@warn@BadAction
  \fi}


```

22.4 Line number printing

\affixline@num

`\affixline@num` originally took a single argument, a series of commands for printing the line just split off by `\do@line`; it put that line back on the vertical list, and added a line number if necessary. It now just puts a left line number into `\l@dld@ta` or a right line number into `\l@drd@ta` if required.

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.

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

```
\newcommand*{\affixline@num}{%
```

No number is attached if `\ifl@dskipnumber` is TRUE (and then it is set to its normal FALSE value). No number is attached if `\ifnumberline` is FALSE (the normal value is TRUE).

```
\ifnumberline
\ifl@dskipnumber
  \global\l@dskipnumberfalse
\else
  \ifsublines@
    \@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.

```
    \ch@cksub@l@ck
```

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

```
  \else
```

```
    \@l@dtempcntb=\line@num
```

Check on the `\linenumberlist` If it's `\empty` use the standard algorithm.

```
    \ifx\linenumberlist\empty
      \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
    \else
```

The `\linenumberlist` wasn't `\empty`, so here's Wayne's numbering mechanism. This takes place in TeX's mouth.

```
      \@l@dtempcnta=\line@num
      \edef\rem@inder{,\linenumberlist,\number\line@num,}%
        \edef\sc@n@list{\def\noexpand\sc@n@list
        ####1,\number\@l@dtempcnta,####2|{\def\noexpand\rem@inder{####2}}}%
        \sc@n@list\expandafter\sc@n@list\rem@inder|%
          \ifx\rem@inder\empty\advance\@l@dtempcnta\@ne\fi
    \fi
```

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

```
    \ch@ck@l@ck
  \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.

For LaTeX we have to consider two column documents as well. In this case I think we need to put the numbers at the outside of the column — the left of the first column and the right of the second. Do the twocolumn stuff before going on with the original code.

`\l@dld@ta` `\l@drd@ta` A left line number is stored in `\l@dld@ta` and a right one in `\l@drd@ta`.

```
  \if@twocolumn
    \if@firstcolumn
      \gdef\l@dld@ta{\llap{{\leftlinenum}}}%
    \else
      \gdef\l@drd@ta{\rlap{{\rightlinenum}}}%
    \fi
```

```
\else
```

Continuing the original code ...

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

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

```
    \ifodd\@l@dtempcntb
      \gdef\l@drd@ta{\rlap{{\rightlinenum}}}%
    \else
      \gdef\l@dld@ta{\llap{{\leftlinenum}}}%
    \fi
\fi
 \else
```

As 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.

```
  \f@x@l@cks
\fi
\fi
}

```

\ch@cksub@l@ck
\ch@ck@l@ck
\f@x@l@cks

These macros handle line number locking for \affixline@num. \ch@cksub@l@ck checks subline locking. If it fails, then we disable the line-number display by setting the counters to arbitrary but unequal values.

```
\newcommand*{\ch@cksub@l@ck}{%
    \ifcase\sub@lock
      \or
        \ifnum\sublock@disp=\@ne
           \@l@dtempcntb=\z@ \@l@dtempcnta=\@ne
        \fi
      \or
        \ifnum\sublock@disp=\tw@ \else
           \@l@dtempcntb=\z@ \@l@dtempcnta=\@ne
        \fi
      \or
        \ifnum\sublock@disp=\z@
           \@l@dtempcntb=\z@ \@l@dtempcnta=\@ne
        \fi
    \fi}
```

Similarly for line numbers.

```
\newcommand*{\ch@ck@l@ck}{%
    \ifcase\@lock
```

```
        \or
          \ifnum\lock@disp=\@ne
             \@l@dtempcntb=\z@ \@l@dtempcnta=\@ne
          \fi
        \or
          \ifnum\lock@disp=\tw@ \else
             \@l@dtempcntb=\z@ \@l@dtempcnta=\@ne
          \fi
        \or
          \ifnum\lock@disp=\z@
             \@l@dtempcntb=\z@ \@l@dtempcnta=\@ne
          \fi
      \fi}
```

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

```
\newcommand*{\f@x@l@cks}{%
  \ifcase\@lock
  \or
    \global\@lock=\tw@
  \or \or
    \global\@lock=\z@
  \fi
  \ifcase\sub@lock
  \or
    \global\sub@lock=\tw@
  \or \or
    \global\sub@lock=\z@
  \fi}

```

`\pageparbreak` Because of TeX's asynchronous page breaking mechanism we can never be sure juust where it will make a break and, naturally, it has already decided exactly how it will typeset any remainder of a paragraph that crosses the break. This is disconcerting when trying to number lines by the page or put line numbers in different margins. This macro tries to force an invisible paragraph break and a page break.

```
\newcommand{\pageparbreak}{\pend\newpage\pstart\noindent}

```

22.5 Pstart number printing in side

In side, the printing of pstart number is running like the printing of line number. There is only some differences:

`\affixpstart@num` `\pstartnum`

- The pstarts counter is upgrade in the `\pend` command. Consequently, the `\affixpstart@num` command has not to upgrade it, unlike the `\affixline@num` which upgrades the lines counter.

- To print the pstart number only at the beginning of a pstart, and not in every line, a boolean test is made. The `\pstartnum` boolean is set to TRUE at every `\pend`. It's tried in the `\leftpstartnum` and `\rightstartnum` commands. After the try, it is set to FALSE.

`\leftpstartnum`
`\rightstartnum`
`\ifsidepstartnum`

```

\newif\ifsidepstartnum
\newcommand*{\affixpstart@num}{%
    \ifsidepstartnum
        \if@twocolumn
            \if@firstcolumn
                  \gdef\l@dld@ta{\llap{{\leftpstartnum}}}%
            \else
                  \gdef\l@drd@ta{\rlap{{\rightpstartnum}}}%
            \fi
        \else
             \@l@dtempcntb=\line@margin
            \ifnum\@l@dtempcntb>\@ne
                  \advance\@l@dtempcntb \page@num
            \fi
            \ifodd\@l@dtempcntb
                  \gdef\l@drd@ta{\rlap{{\rightpstartnum}}}%
            \else
                  \gdef\l@dld@ta{\llap{{\leftpstartnum}}}%
            \fi
        \fi
    \fi

}
%

\newif\ifpstartnum
\pstartnumtrue
\newcommand*{\leftpstartnum}{
    \ifpstartnum\thepstart
    \kern\linenumsep\fi
    \global\pstartnumfalse
}
\newcommand*{\rightpstartnum}{
    \ifpstartnum
    \kern\linenumsep
    \thepstart
    \fi
    \global\pstartnumfalse
}
```

22.6 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`.

```
\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}

```

22.7 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 count `\@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. 81). Finally, the penalty is checked to see that it doesn't go below −10000.

```
\newcommand*{\add@penalties}{\@l@dtempcnta=\ballast@count
  \ifnum\num@lines>\@ne
    \global\advance\par@line \@ne
    \ifnum\par@line=\@ne
      \advance\@l@dtempcnta \clubpenalty
    \fi
    \@l@dtempcntb=\par@line \advance\@l@dtempcntb \@ne
    \ifnum\@l@dtempcntb=\num@lines
      \advance\@l@dtempcnta \widowpenalty
    \fi
    \ifnum\par@line<\num@lines
      \advance\@l@dtempcnta \interlinepenalty
    \fi
  \fi
    \ifnum\@l@dtempcnta=\z@
      \relax
    \else
      \ifnum\@l@dtempcnta>-10000
        \penalty\@l@dtempcnta
      \else
        \penalty -10000
      \fi
    \fi}

```

22.8 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.

```
\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}

```

23 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 the footnotes, not just one; and there are options to reformat footnotes into paragraphs or into multiple columns.

23.1 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.

\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}

```

23.2 Outer-level footnote commands

`\footnoteoptions@`

The `\footnoteoption@[`⟨*side*⟩`]{`⟨*options*⟩`}{`⟨*value*⟩`}` change the value of on options of Xfootnote, to switch between true and false.

```
\newcommandx*{\footnoteoptions@}[3][1=L,usedefault]{%
   \def\do##1{%
      \ifstrequal{#1}{L}{% In Leftside
         \xright@appenditem{\global\noexpand\settoggle{##1@}{#3}}\to\inserts@list% Switch
         \global\advance\insert@count \@ne% Increment the left insert counter.
      }%
      {%
          \xright@appenditem{\global\noexpand\settoggle{##1@}{#3}}\to\inserts@listR% Swit
          \global\advance\insert@countR \@ne% Increment the right insert counter insert.
      }%
   }%
   \notblank{#2}{\docsvlist{#2}}{}% Parsing all options
}
```

`\footnotelang@lua`

`\footnotelang@lua` is called to remember the information about the language of a lemma when LuaLaTeX is used.

```
\newcommandx*{\footnotelang@lua}[1][1=L,usedefault]{%
   \ifstrequal{#1}{L}{%
   \xright@appenditem{{\csxdef{footnote@luatextextdir}{\the\luatextextdir}}}\to\inserts@l
    \global\advance\insert@count \@ne%
    \xright@appenditem{{\csxdef{footnote@luatexpardir}{\the\luatexpardir}}}\to\inserts@li
    \global\advance\insert@count \@ne%
   }%
   {%
   \xright@appenditem{{\csxdef{footnote@luatextextdir}{\the\luatextextdir}}}\to\inserts@l
    \global\advance\insert@countR \@ne%
    \xright@appenditem{{\csxdef{footnote@luatexpardir}{\the\luatexpardir}}}\to\inserts@li
    \global\advance\insert@countR \@ne%
   }%
}
```

`\footnotelang@poly`

`\footnotelang@poly` is called to remember the information about the language of a lemma when Polyglossia is used.

```
\newcommandx*{\footnotelang@poly}[1][1=L,usedefault]{%
   \ifstrequal{#1}{L}{%
    \if@RTL%
        \xright@appenditem{{\csxdef{footnote@dir}{@RTLtrue}}}\to\inserts@list%Know the la
        \global\advance\insert@count \@ne%
    \else
         \xright@appenditem{{\csxdef{footnote@dir}{@RTLfalse}}}\to\inserts@list%Know the 
         \global\advance\insert@count \@ne%
    \fi%
    \xright@appenditem{{\csxdef{footnote@lang}{\csexpandonce{languagename}}}}\to\inserts@
    \global\advance\insert@count \@ne%
   }%
```

```
  {%
   \if@RTL
       \xright@appenditem{{\csxdef{footnote@dir}{@RTLtrue}}}\to\inserts@listR%Know the language of l
       \global\advance\insert@countR \@ne%
   \else
        \xright@appenditem{{\csxdef{footnote@dir}{@RTLfalse}}}\to\inserts@listR%Know the language of
        \global\advance\insert@countR \@ne%
   \fi
   \xright@appenditem{{\csxdef{footnote@lang}{\csexpandonce{languagename}}}}\to\inserts@listR%Know t
   \global\advance\insert@countR \@ne%
  }%
}
```

23.3 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 the 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 the 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.

```
\notbool{parapparatus@}{\newcommand*}{\newcommand}{\normalvfootnote}[2]{%
  \insert\csname #1footins\endcsname\bgroup
  \csuse{bhookXnote@#1}
  \csuse{Xnotefontsize@#1}
  \footsplitskips
  \spaceskip=\z@skip \xspaceskip=\z@skip
  \csname #1footfmt\endcsname #2[#1]\egroup}
```

`\footsplitskips` Some setup code that is common for a variety of the footnotes.

```
\newcommand*{\footsplitskips}{%
  \interlinepenalty=\interfootnotelinepenalty
  \floatingpenalty=\@MM
  \splittopskip=\ht\strutbox \splitmaxdepth=\dp\strutbox
  \leftskip=\z@skip \rightskip=\z@skip}

```

\mpnormalvfootnote And a somewhat different version for minipages.

```
\notbool{parapparatus@}{\newcommand*}{\newcommand}{\mpnormalvfootnote}[2]{%
  \global\setbox\@nameuse{mp#1footins}\vbox{%
    \unvbox\@nameuse{mp#1footins}
    \csuse{bhookXnote@#1}
    \csuse{Xnotefontsize@#1}
    \hsize\columnwidth
    \@parboxrestore
    \color@begingroup
    \csname #1footfmt\endcsname #2[#1]\color@endgroup}}

```

\ledsetnormalparstuff \normalfootfmt `\normalfootfmt` is a 'normal' macro to take the footnote line and page number information (see p. 55), 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 tricky material in the main text to get the lines numbered automatically (as set up by `\autopar`, for example).

```
\newcommand*{\ledsetnormalparstuff}{%
  \ifluatex%
     \luatextextdir\footnote@luatextextdir%
   \luatexpardir\footnote@luatexpardir%
  \fi%
  \csuse{\csuse{footnote@dir}}%
  \normal@pars%
  \noindent \parfillskip \z@ \@plus 1fil}

\notbool{parapparatus@}{\newcommandx*}{\newcommandx}{\normalfootfmt}[4][4=Z]{% 4th arg is
  \ledsetnormalparstuff%
  \hangindent=\csuse{Xhangindent@#4}
  \strut{\printlinefootnote{#1}{#4}}%
  {\select@lemmafont#1|#2}%
  \iftoggle{nosep@}{\hskip\csuse{inplaceoflemmaseparator@#4}}{\ifcsempty{lemmaseparator@#4
    {\hskip\csuse{inplaceoflemmaseparator@#4}}%
    {\nobreak\hskip\csuse{beforelemmaseparator@#4}\csuse{lemmaseparator@#4}\hskip\csuse{a
  }}%
  #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 the normal font and 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 the footnotes.

With polyglossia, each critical note has a `\footnote@lang` which shows the language of the lemma, and which can be used to switch the bracket from right to left.

```
\def\endashchar{\textnormal{--}}
\newcommand*{\fullstop}{\textnormal{.}}
\newcommand*{\rbracket}{\textnormal{%
  \csuse{text\csuse{footnote@lang}}{%
      \ifluatex%
      \ifdefstring{\footnote@luatextextdir}{TRT}{\thinspace[}{\thinspace]}%
         \else%
      \thinspace]%
      \fi}%
  }%
}

```

`\printpstart` The `\printpstart` macro prints the pstart number for a note.

```
\newcommand{\printpstart}[0]{%
   \ifl@dpairing%
       \ifledRcol%
           \thepstartR%
       \else%
           \thepstartL%
       \fi%
   \else%
       \thepstart%
   \fi%
}
```

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 `\l@d@nums`, in the form described on page 55: 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.

The original `EDMAC` code used several counters at this point, saying:

> 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.

LaTeX tends to use a lot of counters and packages should try and minimise the number of new ones they create. In line with this Peter Wilson have reverted to traditional booleans.

`\ifl@d@pnum` `\ifl@d@ssub` `\ifl@d@elin` `\ifl@d@esl` `\ifl@d@dash`

```
\newif\ifl@d@pnum
  \l@d@pnumfalse
\newif\ifl@d@ssub
  \l@d@ssubfalse
\newif\ifl@d@elin
  \l@d@elinfalse
\newif\ifl@d@esl
  \l@d@eslfalse
\newif\ifl@d@dash
  \l@d@dashfalse
```

`\l@dparsefootspec` `\l@dp@rsefootspec` `\l@dparsedstartpage` `\l@dparsedstartline` `\l@dparsedstartsub` `\l@dparsedendpage` `\l@dparsedendline` `\l@dparsedendsub`

`\l@dparsefootspec{`⟨*spec*⟩`}{`⟨*lemma*⟩`}{`⟨*text*⟩`}` parses a footnote specification. ⟨*lemma*⟩ and ⟨*text*⟩ are the lemma and text respectively. ⟨*spec*⟩ is the line and page number and lemma font specifier in `\l@d@nums` style format. The real work is done by `\l@dp@rsefootspec` which defines macros holding the numeric values.

```
\newcommand*{\l@dparsefootspec}[3]{\l@dp@rsefootspec#1|}
\def\l@dp@rsefootspec#1|#2|#3|#4|#5|#6|#7|{%
  \gdef\l@dparsedstartpage{#1}%
  \gdef\l@dparsedstartline{#2}%
  \gdef\l@dparsedstartsub{#3}%
  \gdef\l@dparsedendpage{#4}%
  \gdef\l@dparsedendline{#5}%
  \gdef\l@dparsedendsub{#6}%
}
```

Initialise the several number value macros.

```
\def\l@dparsedstartpage{0}%
\def\l@dparsedstartline{0}%
\def\l@dparsedstartsub{0}%
\def\l@dparsedendpage{0}%
\def\l@dparsedendline{0}%
\def\l@dparsedendsub{0}%

```

`\setprintlines` 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.

Just a reminder of the arguments:

```
\printlines    #1      |  #2  |    #3    |     #4    |  #5  |  #6     |   #7
\printlines start-page | line | subline | end-page | line | subline | font
```

The macro `\setprintlines` does the work of deciding what numbers should be printed. Its arguments are the same as the first 6 of `\printlines`.

```
\newcommand*{\setprintlines}[6]{%
  \l@d@pnumfalse \l@d@dashfalse
  \ifbypage@
     \ifnum#4=#1 \else
        \l@d@pnumtrue
        \l@d@dashtrue
     \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.

```
  \ifl@d@pnum \l@d@elintrue \else \l@d@elinfalse \fi
  \ifnum#2=#5 \else
      \l@d@elintrue
      \l@d@dashtrue
  \fi
```

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

```
  \l@d@ssubfalse
  \ifnum#3=0 \else
      \l@d@ssubtrue
  \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.

```
  \l@d@eslfalse
  \ifnum#6=0 \else
      \ifnum#6=#3
         \ifl@d@elin \l@d@esltrue \else \l@d@eslfalse \fi
      \else
         \l@d@esltrue
         \l@d@dashtrue
      \fi
  \fi}
```

`\printlines` Now we're ready to print it all. If the lineation is by pstart, we print the pstart.

```
\def\printlines#1|#2|#3|#4|#5|#6|#7|{\begingroup
  \setprintlines{#1}{#2}{#3}{#4}{#5}{#6}%
```

One 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).

```
  \ifl@d@pnum #1\fullstop\fi
  \linenumrep{#2}

  \ifl@d@ssub \fullstop \sublinenumrep{#3}\fi
  \ifl@d@dash \endashchar\fi
  \ifl@d@pnum #4\fullstop\fi
  \ifl@d@elin \linenumrep{#5}\fi
  \ifl@d@esl \ifl@d@elin \fullstop\fi \sublinenumrep{#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.

But if the skip `\preXnotes@` is greater than 0 pt, it's used instead of `\skip\footins` for the first printed series.

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 eledmac 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`.

```
\newcommand*{\normalfootstart}[1]{%
    \ifdimequal{0pt}{\preXnotes@}{}%
      {%
      \iftoggle{preXnotes@}{%
            \togglefalse{preXnotes@}\skip\csname #1footins\endcsname=\csuse{preXnotes@}}%
          {}%
      }%
  \vskip\skip\csname #1footins\endcsname%
  \leftskip0pt \rightskip0pt
  \csname #1footnoterule\endcsname\noindent\leavevmode}
```

`\normalfootnoterule`

`\norrmalfootnoterule` 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.

```
\newcommand*{\normalfootgroup}[1]{{\csuse{Xnotefontsize@#1}\noindent\csuse{txtbeforeXnotes

```

`\mpnormalfootgroup`

A somewhat different version for minipages.

```
\newcommand*{\mpnormalfootgroup}[1]{{
  \vskip\skip\@nameuse{mp#1footins}
  \normalcolor
  \@nameuse{#1footnoterule}
 {\csuse{Xnotefontsize@#1}\noindent\csuse{txtbeforeXnotes@#1}}
 \unvbox\csname mp#1footins\endcsname}}

```

23.4 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 eledmac code.)

```
\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.

\ledfootinsdim Have a constant value for the `\dimen\footins`

```
\newcommand*{\ledfootinsdim}{0.8\vsize} % kept for backward compatibility, should'nt be used anymore.
```

\preXnotes@ \preXnotes If user redefines `\preXnotes@`, via `\preXnotes` to a value greater than 0 pt, this skip will be added before first series notes instead of the notes skip.

```
\newtoggle{preXnotes@}
\toggletrue{preXnotes@}
\newcommand{\preXnotes@}{0pt}
\newcommand*{\preXnotes}[1]{\renewcommand{\preXnotes@}{#1}}
```

The same, but for familiar footnotes.

\preXnotes \preXnotes@

```
\newtoggle{prenotesX@}
\toggletrue{prenotesX@}
\newcommand{\prenotesX@}{0pt}
\newcommand*{\prenotesX}[1]{\renewcommand{\prenotesX@}{#1}}
```

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

```
\newcommand*{\footnormal}[1]{%
  \csgdef{series@display#1}{normal}
  \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=\csuse{maxhXnotes@#1}
  \skip\csname #1footins\endcsname=\csuse{beforeXnotes@#1}
```

Now do the setup for minipage footnotes. We use as much as possible of the normal setup as we can (so the notes will have a similar layout).

```
  \expandafter\let\csname mpv#1footnote\endcsname=\mpnormalvfootnote
  \expandafter\let\csname mp#1footgroup\endcsname=\mpnormalfootgroup
  \count\csname mp#1footins\endcsname=1000
  \dimen\csname mp#1footins\endcsname=\csuse{maxhXnotes@#1}
  \skip\csname mp#1footins\endcsname=\csuse{beforeXnotes@#1}
}

```

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.

23.5 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 the 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.

```
\newcommand*{\footparagraph}[1]{%
  \csgdef{series@display#1}{paragraph}
  \expandafter\newcount\csname prevpage#1@num\endcsname
  \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}
```

And the extra setup for minipages.

```
  \expandafter\let\csname mpv#1footnote\endcsname=\mppara@vfootnote
  \expandafter\let\csname mp#1footgroup\endcsname=\mppara@footgroup
  \count\csname mp#1footins\endcsname=1000
}
```

`\footfudgefiddle` For paragraphed footnotes TEX has to estimate the amount of space required. If it underestimates this then the notes may get too long and run off the bottom of the text block. `\footfudgefiddle` can be increased from its default 64 (say to 70) to increase the estimate.

```
\providecommand{\footfudgefiddle}{64}
```

`\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.

Peter Wilson thinks that `\columnwidth` should be used here for LaTeX, not `\hsize`. I've also included `\footfudgefiddle`.

```
\newcommand*{\para@footsetup}[1]{{\csuse{Xnotefontsize@#1}
  \dimen0=\baselineskip
  \multiply\dimen0 by 1024
  \divide \dimen0 by \columnwidth \multiply\dimen0 by \footfudgefiddle\relax
  \expandafter
  \xdef\csname #1footfudgefactor\endcsname{%
    \expandafter\strip@pt\dimen0 }}}

```

EDMAC defines `\en@number` which does the same as the LaTeX kernel `\strip@pt`, namely strip the characters `pt` from a dimen value. Eledmac use `\strip@pt`.

`\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.

```
\newcommand*{\parafootstart}[1]{%
  \rightskip=0pt \leftskip=0pt \parindent=0pt
    \ifdimequal{0pt}{\preXnotes@}{}%
      {%
      \iftoggle{preXnotes@}{%
           \togglefalse{preXnotes@}\skip\csname #1footins\endcsname=\csuse{preXnotes@}}%
         {}%
      }%
  \vskip\skip\csname #1footins\endcsname%
  \csname #1footnoterule\endcsname\noindent\leavevmode}
```

`\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.[24]

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.[25] So changes from one language to another will not invoke the proper hyphenation rules in such footnotes. Since critical editions often do deal with several languages, especially in a 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.[26] 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.[27] In other words, be very careful not to say `\break`, or `\penalty-10000`,

[24]Michael Downes, 'Line Breaking in `\unhbox`ed Text', *TUGboat* **11** (1990), pp. 605–612.
[25]See *The TeXbook*, p. 455 (editions after January 1990).
[26]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.
[27]'Line Breaking', p. 610.

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 eledmac 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. 98 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.

```
\newcommand*{\para@vfootnote}[2]{%
  \insert\csname #1footins\endcsname
  \bgroup
    \csuse{bhookXnote@#1}
    \csuse{Xnotefontsize@#1}
    \footsplitskips
    \setbox0=\vbox{\hsize=\maxdimen
      \noindent\csname #1footfmt\endcsname#2[#1]}%
    \setbox0=\hbox{\unvxh0[#1]}%
    \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.

```
    \if@RTL\noindent \leavevmode\fi\box0%
    \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`).

`\mppara@vfootnote` This version is for minipages.

```
\newcommand*{\mppara@vfootnote}[2]{%
  \global\setbox\@nameuse{mp#1footins}\vbox{%
    \unvbox\@nameuse{mp#1footins}%
    \csuse{bhookXnote@#1}
    \csuse{Xnotefontsize@#1}
    \footsplitskips
    \setbox0=\vbox{\hsize=\maxdimen
      \noindent\color@begingroup\csname #1footfmt\endcsname #2[#1]\color@endgroup}%
    \setbox0=\hbox{\unvxh0[#1]}%
    \dp0=\z@
    \ht0=\csname #1footfudgefactor\endcsname\wd0
    \box0
    \penalty0
}}

```

`\unvxh` Here is (modified) 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`.

```
\newcommandx*{\unvxh}[2][2=Z]{% 2th is optional for retro-compatibility
  \setbox0=\vbox{\unvbox#1%
    \global\setbox1=\lastbox}%
  \unhbox1
  \unskip           % remove \rightskip,
  \unskip           % remove \parfillskip,
  \unpenalty        % remove \penalty of 10000,
  \hskip\csuse{afternote@#2}}   % but add the glue to go between 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, the third is the text of the footnote, and the fourth is the series (optional, for backward compatibility).

```
\newcommandx*{\parafootfmt}[4][4=Z]{%
  \insertparafootsep{#4}%
  \ledsetnormalparstuff%
  \printlinefootnote{#1}{#4}%
  {\select@lemmafont#1|#2}%
  \iftoggle{nosep@}{\hskip\csuse{inplaceoflemmaseparator@#4}}{\ifcsempty{lemmaseparator@#
    {\hskip\csuse{inplaceoflemmaseparator@#4}}%
    {\nobreak\hskip\csuse{beforelemmaseparator@#4}\csuse{lemmaseparator@#4}\hskip\csuse{a
  }}%
  #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. The `\insertparafootsep` command is used to insert the `\parafootsep@series` between each note in the *same* page.

`\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.

```
\newcommand*{\para@footgroup}[1]{%
  \unvbox\csname #1footins\endcsname
  \makehboxofhboxes
  \setbox0=\hbox{{\csuse{Xnotefontsize@#1}\csuse{txtbeforeXnotes@#1}}\unhbox0 \removehboxes}%
  \csuse{Xnotefontsize@#1}
  \noindent\unhbox0\par}

```

`\mppara@footgroup` The minipage version.

```
\newcommand*{\mppara@footgroup}[1]{{%
  \vskip\skip\@nameuse{mp#1footins}
  \normalcolor
  \@nameuse{#1footnoterule}%
  \unvbox\csname mp#1footins\endcsname
  \makehboxofhboxes
  \setbox0=\hbox{{\csuse{Xnotefontsize@#1}\csuse{txtbeforeXnotes@#1}}\unhbox0 \removehboxes}%
  \csuse{Xnotefontsize@#1}
  \noindent\unhbox0\par}}

```

`\makehboxofhboxes`
`\removehboxes`

```
\newcommand*{\makehboxofhboxes}{\setbox0=\hbox{}%
  \loop
    \unpenalty
    \setbox2=\lastbox
  \ifhbox2
    \setbox0=\hbox{\box2\unhbox0}%
  \repeat}

\newcommand*{\removehboxes}{\setbox0=\lastbox
  \ifhbox0{\removehboxes}\unhbox0 \fi}

```

23.5.1 Insertion of the footnotes separator

The command `\insertparafootsep{`⟨*series*⟩`}` must be called at the beginning of `\parafootftm` (and like commands).

`\prevpage@num`
`\insertparafootsep`

```
\newcommand{\insertparafootsep}[1]{%
   \ifnumequal{\csuse{prevpage#1@num}}{\page@num}%
        {\ifcsdef{prevline#1}% Be sur \prevline#1 exists.
      {\ifnumequal{\csuse{prevline#1}}{\line@num}%
         {\ifcsempty{symplinenum}{\csuse{parafootsep@#1}}{}}%
         {\csuse{parafootsep@#1}}%
      }%
      {\csuse{parafootsep@#1}}%
   }%
   {}%
   \global\csname prevpage#1@num\endcsname=\page@num%
 }
```

23.6 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
\newcommand*{\rigidbalance}[3]{\setbox0=\box#1 \@k=#2 \@h=#3
  \@@line{\splittopskip=\@h \vbadness=\@M \hfilneg
  \valign{##\vfil\cr\dosplits}}}

\newcommand*{\dosplits}{\ifnum\@k>0 \noalign{\hfil}\splitoff
  \global\advance\@k-1\cr\dosplits\fi}

\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 the footnotes typeset in three columns. It is important to call this only after `\hsize` has been set for the document.

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

The additional setup for minipages.

```
  \expandafter\let\csname mpv#1footnote\endcsname=\mpnormalvfootnote
  \expandafter\let\csname mp#1footgroup\endcsname=\mpthreecolfootgroup
  \mpthreecolfootsetup{#1}
}

```

The `\footstart` and `\footnoterule` macros for these notes assume the normal values (p. 98 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 the 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.

```
\newcommand*{\threecolfootsetup}[1]{%
  \count\csname #1footins\endcsname 333
  \multiply\dimen\csname #1footins\endcsname \thr@@}
```

`\mpthreecolfootsetup` The setup for minipages.

```
\newcommand*{\mpthreecolfootsetup}[1]{%
  \count\csname mp#1footins\endcsname 333
  \multiply\dimen\csname mp#1footins\endcsname \thr@@}

```

`\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 a 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).

```
\notbool{parapparatus@}{\newcommand*}{\newcommand}{\threecolvfootnote}[2]{%
  \insert\csname #1footins\endcsname\bgroup
  \csuse{Xnotefontsize@#1}
  \footsplitskips
  \csname #1footfmt\endcsname #2[#1]\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 4) optional (for backward compatibility): the series.

```
\notbool{parapparatus@}{\newcommandx*}{\newcommandx}{\threecolfootfmt}[4][4=Z]{%
  \normal@pars
  \hsize \csuse{hsizethreecol@#4}
  \parindent=0pt
  \tolerance=5000
  \raggedright
  \hangindent=\csuse{Xhangindent@#4}
  \leavevmode
  \strut{\printlinefootnote{#1}{#4}}%
  {\select@lemmafont#1|#2}%
  \iftoggle{nosep@}{\hskip\csuse{inplaceoflemmaseparator@#4}}{\ifcsempty{lemmaseparator@#4
    {\hskip\csuse{inplaceoflemmaseparator@#4}}%
    {\nobreak\hskip\csuse{beforelemmaseparator@#4}\csuse{lemmaseparator@#4}\hskip\csuse{af
  }}%
  #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.

```
\newcommand*{\threecolfootgroup}[1]{{\notefontsetup
  {\csuse{Xnotefontsize@#1}\noindent\csuse{txtbeforeXnotes@#1}}\par
  \splittopskip=\ht\strutbox
  \expandafter
  \rigidbalance\csname #1footins\endcsname \thr@@ \splittopskip}}
```

`\mpthreecolfootgroup` The setup for minipages.

```
\newcommand*{\mpthreecolfootgroup}[1]{{%
  \vskip\skip\@nameuse{mp#1footins}
  \normalcolor
  \@nameuse{#1footnoterule}
  {\csuse{Xnotefontsize@#1}\noindent\csuse{txtbeforeXnotes@#1}}\par
  \splittopskip=\ht\strutbox
  \expandafter
  \rigidbalance\csname mp#1footins\endcsname \thr@@ \splittopskip}}

```

Two columns

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

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

The additional setup for minipages.

```
  \expandafter\let\csname mpv#1footnote\endcsname=\mpnormalvfootnote
  \expandafter\let\csname mp#1footgroup\endcsname=\mptwocolfootgroup
  \mptwocolfootsetup{#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 giving a gap between them of one tenth of the `\hsize`.

```
\newcommand*{\twocolfootsetup}[1]{%
  \count\csname #1footins\endcsname 500
  \multiply\dimen\csname #1footins\endcsname \tw@}
```

```
\notbool{parapparatus@}{\newcommand*}{\newcommand}{\twocolvfootnote}[2]{\insert\csname #1footins\endc
  \csuse{Xnotefontsize@#1}
  \footsplitskips
  \csname #1footfmt\endcsname #2[#1]\egroup}
```

```
\notbool{parapparatus@}{\newcommandx*}{\newcommandx}{\twocolfootfmt}[4][4=Z]{% 4th arg is optional, f
  \normal@pars
  \hsize \csuse{hsizetwocol@#4}
  \parindent=0pt
  \tolerance=5000
  \raggedright
  \hangindent=\csuse{Xhangindent@#4}
  \leavevmode
  \strut{\printlinefootnote{#1}{#4}}%
  {\select@lemmafont#1|#2}%
  \iftoggle{nosep@}{\hskip\csuse{inplaceoflemmaseparator@#4}}{\ifcsempty{lemmaseparator@#4}%
    {\hskip\csuse{inplaceoflemmaseparator@#4}}%
    {\nobreak\hskip\csuse{beforelemmaseparator@#4}\csuse{lemmaseparator@#4}\hskip\csuse{afterlemmasep
  }}%
  #3\strut\par\allowbreak}
```

```
\newcommand*{\twocolfootgroup}[1]{{\csuse{Xnotefontsize@#1}
   {\csuse{Xnotefontsize@#1}\noindent\csuse{txtbeforeXnotes@#1}}\par
  \splittopskip=\ht\strutbox
  \expandafter
  \rigidbalance\csname #1footins\endcsname \tw@ \splittopskip}}

```

\mptwocolfootsetup
\mptwocolfootgroup
The versions for minipages.

```
\newcommand*{\mptwocolfootsetup}[1]{%
  \count\csname mp#1footins\endcsname 500
  \multiply\dimen\csname mp#1footins\endcsname \tw@}
\newcommand*{\mptwocolfootgroup}[1]{{%
  \vskip\skip\@nameuse{mp#1footins}
  \normalcolor
  \@nameuse{#1footnoterule}
   {\csuse{Xnotefontsize@#1}\noindent\csuse{txtbeforeXnotes@#1}}\par
  \splittopskip=\ht\strutbox
  \expandafter
  \rigidbalance\csname mp#1footins\endcsname \tw@ \splittopskip}}

```

24 Familiar footnotes

24.1 Generality

The original EDMAC provided users with five series of critical footnotes (\Afootnote \Bfootnote \Cfootnote \Dfootnote \Efootnote), and LaTeX provides a single numbered footnote. The eledmac package uses the EDMAC mechanism to provide five series of numbered footnotes.

First, though, the footmisc package has an option whereby two or more consecutive \footnotes have their marks separated by commas. This seems such a useful ability that it is provided automatically by eledmac.

\multiplefootnotemarker
\multfootsep
These macros may have been defined by the memoir class, are provided by the footmisc package and perhaps by other footnote packages.

```
\providecommand*{\multiplefootnotemarker}{3sp}
\providecommand*{\multfootsep}{\textsuperscript{\normalfont,}}

```

\m@mmf@prepare
A pair of self-cancelling kerns. This may have been defined in the memoir class.

```
\providecommand*{\m@mmf@prepare}{%
  \kern-\multiplefootnotemarker
  \kern\multiplefootnotemarker\relax}
```

\m@mmf@check
This may have been defined in the memoir class. If it recognises the last kern as \multiplefootnotemarker it typesets \multfootsep.

```
\providecommand*{\m@mmf@check}{%
  \ifdim\lastkern=\multiplefootnotemarker\relax
    \edef\@x@sf{\the\spacefactor}%
    \unkern
    \multfootsep
    \spacefactor\@x@sf\relax
  \fi}

```

We have to modify `\@footnotetext` and `\@footnotemark`. However, if memoir is used the modifications have already been made.

```
\@ifclassloaded{memoir}{}{%
```

`\@footnotetext` Add `\m@mmf@prepare` at the end of `\@footnotetext`.

```
\let\l@dold@footnotetext\@footnotetext
\renewcommand{\@footnotetext}[1]{%
  \l@dold@footnotetext{#1}%
  \m@mmf@prepare}
```

`\@footnotemark` Modify `\@footnotemark` to cater for adjacent `\footnotes`.

```
\renewcommand*{\@footnotemark}{%
  \leavevmode
  \ifhmode
    \edef\@x@sf{\the\spacefactor}%
    \m@mmf@check
    \nobreak
  \fi
  \@makefnmark
  \m@mmf@prepare
  \ifhmode\spacefactor\@x@sf\fi
  \relax}
```

Finished the modifications for the non-memoir case.

```
}

```

`\l@doldold@footnotetext` `\@footnotetext` In order to enable the regular `\footnotes` in numbered text we have to play around with its `\@footnotetext`, using different forms for when in numbered or regular text.

```
\let\l@doldold@footnotetext\@footnotetext
\renewcommand{\@footnotetext}[1]{%
  \ifnumberedpar@
    \edtext{}{\l@dbfnote{#1}}%
  \else
    \l@doldold@footnotetext{#1}%
  \fi}
```

`\l@dbfnote` `\vl@dbfnote` `\l@dbfnote` adds the footnote to the insert list, and `\vl@dbfnote` calls the original `\@footnotetext`.

```
\newcommand{\l@dbfnote}[1]{%
  \ifnumberedpar@
  \gdef\@tag{#1}%
    \xright@appenditem{\noexpand\vl@dbfnote{{\csexpandonce{@tag}}}{\@thefnmark}}%
                         \to\inserts@list
    \global\advance\insert@count \@ne
  \fi\ignorespaces}
\newcommand{\vl@dbfnote}[2]{%
```

```
  \def\@thefnmark{#2}%
  \l@doldold@footnotetext{#1}}
%     \end{macrocode}
% \end{macro}
% \end{macro}
%
%
%
%
%
%
% \subsection{Footnote formats}
%
% Some of the code for the various formats is remarkably similar to that
% in section~\ref{sec:nfootformat}.
%
% The following macros generally set things up for the 'standard' footnote
% format.
%
% \begin{macro}{\prebodyfootmark}
% \begin{macro}{\postbodyfootmark}
% Two convenience macros for use by \cs{...@footnotemark...} macros.
%    \begin{macrocode}
\newcommand*{\prebodyfootmark}{%
  \leavevmode
  \ifhmode
    \edef\@x@sf{\the\spacefactor}%
    \m@mmf@check
    \nobreak
  \fi}
\newcommand{\postbodyfootmark}{%
  \m@mmf@prepare
  \ifhmode\spacefactor\@x@sf\fi\relax}

```

`\normal@footnotemarkX` `\normal@footnotemarkX{`⟨*series*⟩`}` sets up the typesetting of the marker at the point where the footnote is called for.

```
\newcommand*{\normal@footnotemarkX}[1]{%
  \prebodyfootmark
  \@nameuse{bodyfootmark#1}%
  \postbodyfootmark}

```

`\normalbodyfootmarkX` The `\normalbodyfootmarkX{`⟨*series*⟩`}` *really* typesets the in-text marker. The style is the normal superscript.

```
\newcommand*{\normalbodyfootmarkX}[1]{%
  \hbox{\textsuperscript{\normalfont\@nameuse{@thefnmark#1}}}}
```

`\normalvfootnoteX` `\normalvfootnoteX{`⟨*series*⟩`}{`⟨*text*⟩`}` does the `\insert` for the ⟨*series*⟩ and calls the series' `\footfmt...` to format the ⟨*text*⟩.

```
\newcommand*{\normalvfootnoteX}[2]{%
  \insert\@nameuse{footins#1}\bgroup
    \csuse{bhooknoteX@#1}
    \csuse{notefontsizeX@#1}
    \footsplitskips
    \spaceskip=\z@skip \xspaceskip=\z@skip
    \@nameuse{footfmt#1}{#1}{#2}\egroup}

```

`\mpnormalvfootnoteX` The minipage version.

```
\newcommand*{\mpnormalvfootnoteX}[2]{%
  \global\setbox\@nameuse{mpfootins#1}\vbox{%
    \unvbox\@nameuse{mpfootins#1}
    \csuse{bhooknoteX@#1}
    \csuse{notefontsizeX@#1}
    \hsize\columnwidth
    \@parboxrestore
    \color@begingroup
    \@nameuse{footfmt#1}{#1}{#2}\color@endgroup}}

```

`\normalfootfmtX` `\normalfootfmtX{`⟨*series*⟩`}{`⟨*text*⟩`}` typesets the footnote text, prepended by the marker.

```
\newcommand*{\normalfootfmtX}[2]{%
   \protected@edef\@currentlabel{%
       \@nameuse{@thefnmark#1}%
   }%
  \ledsetnormalparstuff
  \hangindent=\csuse{hangindentX@#1}%
  {{\csuse{notenumfontX@#1}\@nameuse{footfootmark#1}}\strut%\enspace
    #2\strut\par}}

```

`\normalfootfootmarkX` `\normalfootfootmarkX{`⟨*series*⟩`}` is called by `\normalfootfmtX` to typeset the footnote marker in the footer before the footnote text.

```
\newcommand*{\normalfootfootmarkX}[1]{%
  \textsuperscript{\@nameuse{@thefnmark#1}}}

```

`\normalfootstartX` `\normalfootstartX{`⟨*series*⟩`}` is the ⟨*series*⟩ footnote starting macro used in the output routine.

```
\newcommand*{\normalfootstartX}[1]{%
    \ifdimequal{0pt}{\prenotesX@}{}%
      {%
      \iftoggle{prenotesX@}{%
           \togglefalse{prenotesX@} \skip\csname footins#1\endcsname=\csuse{prenotesX@}}%
          {}%
      }%
  \vskip\skip\csname footins#1\endcsname%
  \leftskip=\z@
  \rightskip=\z@
  \@nameuse{footnoterule#1}}

```

\normalfootnoteruleX The rule drawn before the footnote series group.

```
\let\normalfootnoteruleX=\footnoterule

```

\normalfootgroupX `\normalfootgroupX{`⟨*series*⟩`}` sends the contents of the ⟨*series*⟩ insert box to the output page without alteration.

```
\newcommand*{\normalfootgroupX}[1]{%
  \unvbox\@nameuse{footins#1}}

```

\mpnormalfootgroupX The minipage version.

```
\newcommand*{\mpnormalfootgroupX}[1]{%
  \vskip\skip\@nameuse{mpfootins#1}
  \normalcolor
  \@nameuse{footnoterule#1}
  \unvbox\@nameuse{mpfootins#1}}

```

\normalbfnoteX

```
\newcommand{\normalbfnoteX}[2]{%
  \ifnumberedpar@
    \protected@csxdef{thisfootnote}{\csuse{thefootnote#1}}%
    \xright@appenditem{\noexpand\vbfnoteX{#1}{#2}{\csexpandonce{thisfootnote}}}%
                      \to\inserts@list
    \global\advance\insert@count \@ne
  \fi\ignorespaces}

```

\vbfnoteX

```
\newcommand{\vbfnoteX}[3]{%
  \@namedef{@thefnmark#1}{#3}%
  \@nameuse{regvfootnote#1}{#1}{#2}}

```

\vnumfootnoteX

```
\newcommand{\vnumfootnoteX}[2]{%
  \ifnumberedpar@
    \edtext{}{\normalbfnoteX{#1}{#2}}%
  \else
    \@nameuse{regvfootnote#1}{#1}{#2}%
  \fi}

```

\footnormalX

\footnormalX{⟨*series*⟩} initialises the settings for the ⟨*series*⟩ footnotes. This should always be called for each series.

```
\newcommand*{\footnormalX}[1]{%
  \csgdef{series@displayX#1}{normalX}
  \expandafter\let\csname footstart#1\endcsname=\normalfootstartX
  \@namedef{@footnotemark#1}{\normal@footnotemarkX{#1}}
  \@namedef{bodyfootmark#1}{\normalbodyfootmarkX{#1}}
  \expandafter\let\csname regvfootnote#1\endcsname=\normalvfootnoteX
  \expandafter\let\csname vfootnote#1\endcsname=\vnumfootnoteX
  \expandafter\let\csname footfmt#1\endcsname=\normalfootfmtX
  \@namedef{footfootmark#1}{\normalfootfootmarkX{#1}}
  \expandafter\let\csname footgroup#1\endcsname=\normalfootgroupX
  \expandafter\let\csname footnoterule#1\endcsname=\normalfootnoteruleX
  \count\csname footins#1\endcsname=1000
  \dimen\csname footins#1\endcsname=\csuse{maxhnotesX@#1}
  \skip\csname footins#1\endcsname=\csuse{beforenotesX@#1}
```

Aditions for minipages.

```
  \expandafter\let\csname mpvfootnote#1\endcsname=\mpnormalvfootnoteX
  \expandafter\let\csname mpfootgroup#1\endcsname=\mpnormalfootgroupX
  \count\csname mpfootins#1\endcsname=1000
  \dimen\csname mpfootins#1\endcsname=\csuse{maxhnotesX@#1}
  \skip\csname mpfootins#1\endcsname=\csuse{beforenotesX@#1}
}

```

24.2 Two columns footnotes

The following macros set footnotes in two columns. It is assumed that the length of each footnote is less than the column width.

\foottwocoolX

\foottwocolX{⟨*series*⟩}

```
\newcommand*{\foottwocolX}[1]{%
  \csgdef{series@displayX#1}{twocol}
  \expandafter\let\csname regvfootnote#1\endcsname=\twocolvfootnoteX
  \expandafter\let\csname footfmt#1\endcsname=\twocolfootfmtX
  \expandafter\let\csname footgroup#1\endcsname=\twocolfootgroupX
  \twocolfootsetupX{#1}
  \expandafter\let\csname mpvfootnote#1\endcsname=\mpnormalvfootnoteX
  \expandafter\let\csname mpfootgroup#1\endcsname=\mptwocolfootgroupX
  \mptwocolfootsetupX{#1}}

```

\twocolfootsetupX
\mptwocolfootsetupX

\twocolfootsetupX{⟨*series*⟩}

```
\newcommand*{\twocolfootsetupX}[1]{%
  \count\csname footins#1\endcsname 500
  \multiply\dimen\csname footins#1\endcsname by \tw@}
\newcommand*{\mptwocolfootsetupX}[1]{%
  \count\csname mpfootins#1\endcsname 500
  \multiply\dimen\csname mpfootins#1\endcsname by \tw@}

```

\twocolvfootnoteX

`\twocolvfootnoteX{`⟨*series*⟩`}`

```
\newcommand*{\twocolvfootnoteX}[2]{%
  \insert\csname footins#1\endcsname\bgroup
    \csuse{notefontsizeX@#1}
    \footsplitskips
    \spaceskip=\z@skip \xspaceskip=\z@skip
    \@nameuse{footfmt#1}{#1}{#2}\egroup}

```

\twocolfootfmtX

`\twocolfootfmtX{`⟨*series*⟩`}`

```
\newcommand*{\twocolfootfmtX}[2]{%
   \protected@edef\@currentlabel{%
       \@nameuse{@thefnmark#1}%
   }%
  \normal@pars
  \hangindent=\csuse{hangindentX@#1}%
  \hsize \csuse{hsizetwocolX@#1}
  \parindent=\z@
%%%  \parfillskip=0pt \@plus 1fil
  \tolerance=5000\relax
  \raggedright
  \leavevmode
  {\csuse{notenumfontX@#1}\@nameuse{footfootmark#1}\strut%\enspace
    #2\strut\par}\allowbreak}

```

\twocolfootgroupX
\mptwocolfootgroupX

`\twocolfootgroupX{`⟨*series*⟩`}`

```
\newcommand*{\twocolfootgroupX}[1]{{\csuse{notefontsizeX@#1}
  \splittopskip=\ht\strutbox
  \expandafter
  \rigidbalance\csname footins#1\endcsname \tw@ \splittopskip}}
\newcommand*{\mptwocolfootgroupX}[1]{{%
  \vskip\skip\@nameuse{mpfootins#1}
  \normalcolor
  \@nameuse{footnoterule#1}
  \splittopskip=\ht\strutbox
  \expandafter
  \rigidbalance\csname mpfootins#1\endcsname \tw@ \splittopskip}}

```

24.3 Three columns footnotes

The following macros set footnotes in three columns. It is assumed that the length of each footnote is less than the column width.

\footthreecolX

`\footthreecolX{`⟨*series*⟩`}`

```
\newcommand*{\footthreecolX}[1]{%
  \csgdef{series@displayX#1}{threecol}
  \expandafter\let\csname regvfootnote#1\endcsname=\threecolvfootnoteX
  \expandafter\let\csname footfmt#1\endcsname=\threecolfootfmtX
  \expandafter\let\csname footgroup#1\endcsname=\threecolfootgroupX
  \threecolfootsetupX{#1}
  \expandafter\let\csname mpvfootnote#1\endcsname=\mpnormalvfootnoteX
  \expandafter\let\csname mpfootgroup#1\endcsname=\mpthreecolfootgroupX
  \mpthreecolfootsetupX{#1}}

```

\threecolfootsetupX
\mpthreecolfootsetupX

`\threecolfootsetupX{`⟨*series*⟩`}`

```
\newcommand*{\threecolfootsetupX}[1]{%
  \count\csname footins#1\endcsname 333
  \multiply\dimen\csname footins#1\endcsname by \thr@@}
\newcommand*{\mpthreecolfootsetupX}[1]{%
  \count\csname mpfootins#1\endcsname 333
  \multiply\dimen\csname mpfootins#1\endcsname by \thr@@}

```

\threecolvfootnoteX

`\threecolvfootnoteX{`⟨*series*⟩`}{`⟨*text*⟩`}`

```
\newcommand*{\threecolvfootnoteX}[2]{%
  \insert\csname footins#1\endcsname\bgroup
    \csuse{notefontsizeX@#1}
    \footsplitskips
    \@nameuse{footfmt#1}{#1}{#2}\egroup}

```

\threecolfootfmtX

`\threecolfootfmtX{`⟨*series*⟩`}`

```
\newcommand*{\threecolfootfmtX}[2]{%
   \protected@edef\@currentlabel{%
       \@nameuse{@thefnmark#1}%
   }%
  \hangindent=\csuse{hangindentX@#1}%
  \normal@pars
  \hsize \csuse{hsizethreecolX@#1}
  \parindent=\z@
%%%  \parfillskip=0pt \@plus 1fil
  \tolerance=5000\relax
  \raggedright
  \leavevmode
  {\csuse{notenumfontX@#1}\@nameuse{footfootmark#1}\strut%\enspace
    #2\strut\par}\allowbreak}

```

\threecolfootgroupX
\mpthreecolfootgroupX

`\threecolfootgroupX{`⟨*series*⟩`}`

```
\newcommand*{\threecolfootgroupX}[1]{{\csuse{notefontsizeX@#1}
  \splittopskip=\ht\strutbox
  \expandafter
  \rigidbalance\csname footins#1\endcsname \thr@@ \splittopskip}}
\newcommand*{\mpthreecolfootgroupX}[1]{{%
  \vskip\skip\@nameuse{mpfootins#1}
  \normalcolor
  \@nameuse{footnoterule#1}
  \splittopskip=\ht\strutbox
  \expandafter
  \rigidbalance\csname mpfootins#1\endcsname \thr@@ \splittopskip}}

```

24.4 Paragraphed footnotes

The following macros set footnotes as one paragraph.

`\footparagraphX` \footparagraphX{⟨*series*⟩}

```
\newcommand*{\footparagraphX}[1]{%
  \csgdef{series@displayX#1}{paragraph}
  \expandafter\newcount\csname prevpage#1@num\endcsname
  \expandafter\let\csname footstart#1\endcsname=\parafootstartX
  \expandafter\let\csname regvfootnote#1\endcsname=\para@vfootnoteX
  \expandafter\let\csname footfmt#1\endcsname=\parafootfmtX
  \expandafter\let\csname footgroup#1\endcsname=\para@footgroupX
  \expandafter\let\csname footnoterule#1\endcsname=\normalfootnoteruleX
  \count\csname footins#1\endcsname=1000
  \expandafter\let\csname mpvfootnote#1\endcsname=\mppara@vfootnoteX
  \expandafter\let\csname mpfootgroup#1\endcsname=\mppara@footgroupX
  \count\csname mpfootins#1\endcsname=1000
  \para@footsetupX{#1}}

```

`\para@footsetupX` \para@footsetupX{⟨*series*⟩}

```
\newcommand*{\para@footsetupX}[1]{{\csuse{notefontsizeX@#1}
  \dimen0=\baselineskip
  \multiply\dimen0 by 1024
  \divide\dimen0 by \hsize \multiply\dimen0 by \footfudgefiddle\relax
  \expandafter
  \xdef\csname footfudgefactor#1\endcsname{%
    \expandafter\strip@pt\dimen0 }}}

```

`\parafootstartX` \parafootstartX{⟨*series*⟩}

```
\newcommand*{\parafootstartX}[1]{%
    \ifdimequal{0pt}{\prenotesX@}{}%
      {%
      \iftoggle{prenotesX@}{%
            \togglefalse{prenotesX@}\skip\csname footins#1\endcsname=\csuse{prenotesX@}}%
          {}%
      }%
  \vskip\skip\csname footins#1\endcsname%
  \leftskip=\z@
  \rightskip=\z@
  \parindent=\z@
  \vskip\skip\@nameuse{footins#1}%
  \@nameuse{footnoterule#1}}

```

\para@vfootnoteX
\mppara@vfootnoteX

\para@vfootnoteX{⟨*series*⟩}{⟨*text*⟩}

```
\newcommand*{\para@vfootnoteX}[2]{%
  \insert\csname footins#1\endcsname
  \bgroup
    \csuse{bhooknoteX@#1}
    \csuse{notefontsizeX@#1}
    \footsplitskips
    \setbox0=\vbox{\hsize=\maxdimen
      \noindent\@nameuse{footfmt#1}{#1}{#2}}%
    \setbox0=\hbox{\unvxh0[#1]}%
    \dp0=\z@
    \ht0=\csname footfudgefactor#1\endcsname\wd0
    \box0
    \penalty0
  \egroup}
\newcommand*{\mppara@vfootnoteX}[2]{%
  \global\setbox\@nameuse{mpfootins#1}\vbox{%
    \unvbox\@nameuse{mpfootins#1}
    \csuse{bhooknoteX@#1}
    \csuse{notefontsizeX@#1}
    \footsplitskips
    \setbox0=\vbox{\hsize=\maxdimen
      \noindent\color@begingroup\@nameuse{footfmt#1}{#1}{#2}\color@endgroup}%
    \setbox0=\hbox{\unvxh0[#1]}%
    \dp0=\z@
    \ht0=\csname footfudgefactor#1\endcsname\wd0
    \box0
    \penalty0}}

```

\parafootfmtX

\parafootfmtX{⟨*series*⟩}

```
\newcommand*{\parafootfmtX}[2]{%
   \protected@edef\@currentlabel{%
       \@nameuse{@thefnmark#1}%
   }%
  \insertparafootsep{#1}%
  \ledsetnormalparstuff
  {\csuse{notenumfontX@#1}\csuse{notenumfontX@#1}\@nameuse{footfootmark#1}\strut%\enspace
    #2\penalty-10}}

```

\para@footgroupX
\mppara@footgroupX

\para@footgroupX{⟨*series*⟩}

```
\newcommand*{\para@footgroupX}[1]{%
  \unvbox\csname footins#1\endcsname
  \makehboxofhboxes
  \setbox0=\hbox{\unhbox0 \removehboxes}%
  \csuse{notefontsizeX@#1}
  \noindent\unhbox0\par}
\newcommand*{\mppara@footgroupX}[1]{{%
  \vskip\skip\@nameuse{mpfootins#1}
  \normalcolor
  \@nameuse{footnoterule#1}
  \unvbox\csname mpfootins#1\endcsname
  \makehboxofhboxes
  \setbox0=\hbox{\unhbox0 \removehboxes}%
  \csuse{notefontsizeX@#1}
  \noindent\unhbox0\par}}

```

24.5 Footnotes' output

\doxtrafeeti
\doreinxtrafeeti

We have to add all the new kinds of familiar footnotes to the output routine. These are the class 1 feet.

```
\newcommand*{\doxtrafeeti}{%
  \setbox\@outputbox \vbox{%
    \unvbox\@outputbox
    \def\do##1{\ifvoid\csuse{footins##1}\else\csuse{footstart##1}{##1}\csuse{footgroup##1}
    \dolistloop{\@series}%
  }}

\newcommand{\doreinxtrafeeti}{%
  \def\do##1{\ifvoid\csuse{footins##1}\else\insert\csuse{footins##1}{\unvbox\csuse{footins
  \dolistloop{\@series}%
  }

```

\addfootinsX

Juste for backward compatibility: print a warning message.

```
\newcommand*{\addfootinsX}[1]{%
  \eledmac@warning{AddfootinsX is obsolete in eledmac 1.0. Use newseries instead.}%
  \footnormalX{#1}%
  \g@addto@macro{\doxtrafeeti}{%
    \setbox\@outputbox \vbox{%
      \unvbox\@outputbox
      \ifvoid\@nameuse{footins#1}\else
        \@nameuse{footstart#1}{#1}\@nameuse{footgroup#1}{#1}\fi}}%as
  \g@addto@macro{\doreinxtrafeeti}{%
    \ifvoid\@nameuse{footins#1}\else
      \insert\@nameuse{footins#1}{\unvbox\@nameuse{footins#1}}\fi}%
  \g@addto@macro{\l@dfambeginmini}{%
    \expandafter\expandafter\expandafter\let\expandafter\expandafter
      \csname footnote#1\endcsname \csname mpfootnote#1\endcsname}%
\g@addto@macro{\l@dfamendmini}{%
  \ifvoid\@nameuse{mpfootins#1}\else\@nameuse{mpfootgroup#1}{#1}\fi}%
}
```

25 Generate series

In this section, X means the name of the series (A, B etc.)

\series \series\series creates one more newseries. It's the public command, which just loops on the private command \newseries@.

```
\newcommand{\newseries}[1]{%
  \def\do##1{\newseries@{##1}}%
  \docsvlist{#1}
}
```

\@series The \series@ macro is an etoolbox list, which contains the name of all series.

```
\newcommand{\@series}{}
```

The command \newseries@\series creates a new series of the footnote.

\newseries@

```
\newcommand{\newseries@}[1]{
```

25.0.1 Test if series is still existing

```
  \xifinlist{#1}{\@series}{\eledmac@warning{Series #1 is still existing !}}
  {%
```

25.0.2 Create all commands to memorize display options

```
  \csgdef{Xhangindent@#1}{0pt}%
  \csgdef{hangindentX@#1}{0pt}
  \csgdef{hsizetwocol@#1}{0.45 \hsize}%
  \csgdef{hsizetwocolX@#1}{0.45 \hsize}%
  \csgdef{hsizethreecol@#1}{.3 \hsize}%
  \csgdef{hsizethreecolX@#1}{.3 \hsize}%
  \csgdef{Xnotenumfont@#1}{\notenumfont}%
  \csgdef{Xendnotenumfont@#1}{\notenumfont}%
  \csgdef{notenumfontX@#1}{\notenumfont}%
  \csgdef{Xnotefontsize@#1}{\notefontsetup}%
  \csgdef{notefontsizeX@#1}{\notefontsetup}%
  \csgdef{Xendnotefontsize@#1}{\notefontsetup}%
  \csgdef{bhooknoteX@#1}{}%
  \csgdef{bhookXnote@#1}{}%
  \csgdef{bhookXendnote@#1}{}%
  \csgdef{boxlinenum@#1}{0pt}%
  \csgdef{boxsymlinenum@#1}{0pt}%
  \newtoggle{numberonlyfirstinline@#1}%
  \newtoggle{numberonlyfirstintwolines@#1}%
  \newtoggle{onlypstartinfootnote@#1}%
```

```
    \newtoggle{pstartinfootnote@#1}%
    \csgdef{symlinenum@#1}{\symplinenum}%
    \newtoggle{nonumberinfootnote@#1}%
    \csgdef{beforenumberinfootnote@#1}{0pt}%
    \csgdef{afternumberinfootnote@#1}{0.5em}%
    \newtoggle{nonbreakableafternumber@#1}%
    \csgdef{beforesymlinenum@#1}{\csuse{beforenumberinfootnote@#1}}%
    \csgdef{aftersymlinenum@#1}{\csuse{afternumberinfootnote@#1}}%
    \csgdef{inplaceofnumber@#1}{1em}%
    \global\cslet{lemmaseparator@#1}{\rbracket}%
    \csgdef{beforelemmaseparator@#1}{0em}%
    \csgdef{afterlemmaseparator@#1}{0.5em}%
    \csgdef{inplaceoflemmaseparator@#1}{1em}%
    \csgdef{afternote@#1}{1em plus.4em minus.4em}%
    \csgdef{parafootsep@#1}{\parafootftmsep}%
    \csgdef{beforeXnotes@#1}{1.2em \@plus .6em \@minus .6em}
    \csgdef{beforenotesX@#1}{1.2em \@plus .6em \@minus .6em}
    \csgdef{txtbeforeXnotes@#1}{}
    \csgdef{maxhnotesX@#1}{\ledfootinsdim}%
    \csgdef{maxhXnotes@#1}{\ledfootinsdim}
```

25.0.3 Create inserts, needed to add notes in foot

Concerning inserts, see chapter 15 of the TeXBook by D. Knuth

```

    \expandafter\newinsert\csname mpfootins#1\endcsname
    \expandafter\newinsert\csname footins#1\endcsname
    \expandafter\newinsert\csname #1footins\endcsname
    \expandafter\newinsert\csname mp#1footins\endcsname
```

25.0.4 Create command for critical apparatus, `\Xfootnote`

Note the double # in command: it's because command is made inside another command.

```

    \global\notbool{parapparatus@}{\expandafter\newcommand\expandafter *}{\expandafter\new
        \begingroup%
        \newcommand{\content}{##2}%
        \ifnumberedpar@
            \ifledRcol%
               \ifluatex%
                   \footnotelang@lua[R]%
               \fi%
               \@ifundefined{xpg@main@language}%if polyglossia
                     {}%
                     {\footnotelang@poly[R]}%
                \footnoteoptions@[R]{##1}{true}%
                \xright@appenditem{\noexpand\prepare@edindex@fornote{\l@d@nums}%
\noexpand\csuse{v#1footnote}{#1}%
                {{\l@d@nums}{\csexpandonce{@tag}}{\csexpandonce{content}}}}\to\inserts@lis
                \footnoteoptions@[R]{##1}{false}%
                \global\advance\insert@countR \@ne%
            \else%
                 \ifluatex%
                   \footnotelang@lua%
                 \fi%
               \@ifundefined{xpg@main@language}%if polyglossia
                     {}%
                     {\footnotelang@poly}%
                \footnoteoptions@{##1}{true}%
                \xright@appenditem{\noexpand\prepare@edindex@fornote{\l@d@nums}%
\noexpand\csuse{v#1footnote}{#1}%
                {{\l@d@nums}{\csexpandonce{@tag}}{\csexpandonce{content}}}}\to\inserts@list
                \global\advance\insert@count \@ne%
                \footnoteoptions@{##1}{false}%
            \fi
        \else
            \csuse{v#1footnote}{#1}{{0|0|0|0|0|0|0}{}{##1}}%
        \fi%
        \ignorespaces%
        \endgroup
        }
```

Set standard display and remember the display.

```
    \csgdef{series@display#1}{}
    \footnormal{#1}
```

25.0.5 Create tools for familiar footnotes (\footnoteX)

First, create the \footnoteX command.

```

    \global\expandafter\newcommand\csname footnote#1\endcsname[1]{%
        \begingroup%
            \newcommand{\content}{##1}%
            \stepcounter{footnote#1}%
            \protected@csxdef{@thefnmark#1}{\csuse{thefootnote#1}}%
            \csuse{@footnotemark#1}%
            \csuse{vfootnote#1}{#1}{\csexpandonce{content}}\m@mmf@prepare%
        \endgroup%
    }
```

The counters.

```
    \newcounter{footnote#1}
      \global\expandafter\renewcommand\csname thefootnote#1\endcsname{\arabic{footnote#1}}
%  \end{macrocode}
% Don't forget to initialize series
%    \begin{macrocode}
    \csgdef{series@displayX#1}{}
    \footnormalX{#1}
```

25.0.6 The endnotes

The `\Xendnote` macro functions to write one endnote to the `.end` file. 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.

```

    \global\expandafter\newcommand\csname #1endnote\endcsname[2]{{\newlinechar='40
        \newcommand{\content}{##1}%
           \immediate\write\l@d@end{\expandafter\string\csname #1end\endcsname%
          {\ifnumberedpar@\l@d@nums\fi}%
           {\ifnumberedpar@\csexpandonce{@tag}\fi}{\csexpandonce{content}}{#1}}}\ignorespa
        }
```

`\Xendnote` commands called `\Xend` commands 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.[28]

```

    \global\csletcs{#1end}{@gobblethree}
%\end{macrocode}
% We need to be able to modify \Eledmac's footnote macros and restore their

    \global\csletcs{#1@@footnote}{#1footnote}
% \cs{Stock series in \cs{@series}
%    \begin{macrocode}

    \listxadd{\@series}{#1}
}
}% End of \newseries
```

25.0.7 Init standards series (A,B,C,D,E,Z)

```
\newseries{A,B,C,D,E,Z}
```

25.0.8 Some tools

`\firstseries` `\seriesatbegin{`$\langle s\rangle$`}` changes the order of series, to put the series $\langle s\rangle$ at the beginning of the list. The series can be the result of a command.

```
\newcommand{\seriesatbegin}[1]{
     \edef\series{#1}
     \def\new{}
     \listeadd{\new}{\series}
    \def\do##1{\ifcsstring{series}{##1}{}{\listadd{\new}{##1}}}
    \dolistloop{\@series}
```

[28] Christophe Hebeisen (`christophe.hebeisen@a3.epfl.ch`) emailed on 2003/11/05 to say he had found that `\@gobblethree` was also defined in the amsfonts package.

```
    \xdef\@series{\new}
}
%   \end{macrocode}
% \end{macro}
% \begin{macro}{\seriesatend}
%  And \cs{seriesatend} moves the series to the end of the list.
% \begin{macrocode}
\newcommand{\seriesatend}[1]{
     \edef\series{#1}
     \def\new{}
    \def\do##1{\ifcsstring{series}{##1}{}{\listadd{\new}{##1}}}
    \dolistloop{\@series}
    \listeadd{\new}{\series}
    \xdef\@series{\new}
}
% \end{macrocode}
% \end{macro}
% \subsection{Display}
% \changes{v1.0}{2012/09/15}{New generic commands to customize footnote display.}
% \subsubsection{Options}
% \begin{macro}{\settoggle@series}
% \changes{v1.1}{2012/09/25}{\cs{settoggle@series} switch the global value of the toggle, not only th
% \cs{settoggle@series}\cs{series}{toggle}{value} is a generic command to switch one toggle for one s
%    \begin{macrocode}
\newcommand{\settoggle@series}[3]{%
    \def\do##1{\global\settoggle{#2@##1}{#3}}
    \ifstrempty{#1}{%
              \dolistloop{\@series}%
           }%
        {%
              \docsvlist{#1}%
           }%
}
```

\setcommand@series

`\setcommand@series{`⟨*series*⟩`}{`⟨*command*⟩`}{`⟨*value*⟩`}` is a generic command to change one command for one series.

```
\newcommandx{\setcommand@series}[4][4]{%
    \def\do##1{
        \csgdef{#2@##1}{#3}
    \ifstrequal{#4}{reload}{\csuse{foot\csuse{series@display##1}}{##1}}{}}
    \ifstrempty{#1}{%
              \dolistloop{\@series}%
    }%
    {%
              \docsvlist{#1}%
    }%
}%
```

\newhookcommand@series

`\newhookcommand@series\command names` is a generic command to add new commands for new commands hook, like `\hsizetwocol`.

```
\newcommand{\newhookcommand@series}[1]{%
  \global\expandafter\newcommand\expandafter*\csname #1\endcsname[2][]{\csuse{setcommand@s
}
\newhookcommand@series{Xhangindent}

\newhookcommand@series{hangindentX}

\newhookcommand@series{hsizetwocol}

\newhookcommand@series{hsizethreecol}

\newhookcommand@series{hsizetwocolX}

\newhookcommand@series{hsizethreecolX}

\newhookcommand@series{Xnotenumfont}

\newhookcommand@series{notenumfontX}

\newhookcommand@series{Xendnotenumfont}

\newhookcommand@series{bhooknoteX}

\newhookcommand@series{bhookXnote}

\newhookcommand@series{bhookXendnote}

\newhookcommand@series{Xnotefontsize}

\newhookcommand@series{notefontsizeX}

\newhookcommand@series{Xendnotefontsize}

\newhookcommand@series{boxlinenum}

\newhookcommand@series{boxsymlinenum}

\newhookcommand@series{parafootsep}

\newhookcommand@series{symlinenum}

\newhookcommand@series{beforenumberinfootnote}

\newhookcommand@series{afternumberinfootnote}

\newhookcommand@series{beforesymlinenum}

\newhookcommand@series{aftersymlinenum}

\newhookcommand@series{inplaceofnumber}
```

```

\newhookcommand@series{lemmaseparator}

\newhookcommand@series{beforelemmaseparator}

\newhookcommand@series{afterlemmaseparator}

\newhookcommand@series{inplaceoflemmaseparator}

\newhookcommand@series{afternote}

\newhookcommand@series{txtbeforeXnotes}

```

hookcommand@series@reload

`\newhookcommand@series@reload` does the same thing as `\newhookcommand@series` but the commands created by this macro also reload the series displaying (normal, paragraph, twocol)

```
\newcommand{\newhookcommand@series@reload}[1]{%
  \global\expandafter\newcommand\expandafter*\csname #1\endcsname[2][]{%
  \csuse{setcommand@series}{##1}{#1}{##2}[reload]
  }%
}
\newhookcommand@series@reload{beforeXnotes}

\newhookcommand@series@reload{beforenotesX}

\newhookcommand@series@reload{maxhnotesX}

\newhookcommand@series@reload{maxhXnotes}
% \end{macrocode}
% \end{macro}
% \begin{macro}{\newhooktoggle@series}
%\cs{newhooktoggle@series}\cs{command names} is a generic command to add new commands for new toggle
%    \begin{macrocode}
\newcommand{\newhooktoggle@series}[1]{%
  \global\expandafter\newcommandx\expandafter*\csname #1\endcsname[2][1,2={true},usedefault]{\settogg
}
\newhooktoggle@series{numberonlyfirstinline}
\newhooktoggle@series{numberonlyfirstintwolines}
\newhooktoggle@series{nonumberinfootnote}
\newhooktoggle@series{pstartinfootnote}
\newhooktoggle@series{onlypstartinfootnote}
\newhooktoggle@series{nonbreakableafternumber}
```

25.0.9 Old commands, kept for backward compatibility

The next commands are kept for ascendant compatibily, but should'nt be used anymore.

\notenumfont
\notefontsetup
\ifledplinenum
\symplinenum

```
\newcommand*{\notenumfont}{\normalfont}
\newcommand*{\notefontsetup}{\footnotesize}
\newif\ifledplinenum
  \ledplinenumtrue
\newcommand*{\symplinenum}{}
```

25.0.10 Hooks for a particular footnote

\nonum@ — \nonum@ toggle is used to disable line number printing in a particular footnote.

```
\newtoggle{nonum@}
```

\nosep@ — \nonum@ toggle is used to disable the lemma separator in a particular footnote.

```
\newtoggle{nosep@}
```

25.0.11 Alias

\nolemmaseparator — \nolemmaseparator[⟨*series*⟩] is just an alias for \lemmaseparator[⟨*series*⟩]{}.

```
\newcommandx*{\nolemmaseparator}[1][1]{\lemmaseparator[#1]{}}
```

\interparanoteglue
\ipn@skip — The \ipn@skip skip and \interparanoteglue command are kept for backward compatibility, but should not be used anymore.

```
\newskip\ipn@skip
\newcommand*{\interparanoteglue}[1]{%
             {\notefontsetup\global\ipn@skip=#1 \relax}}
\interparanoteglue{1em plus.4em minus.4em}
```

\parafootftmsep — The \parafootftmsep macro is kept for backward compatibility. It is default value of \parafootsep@series.

```
\newcommand{\parafootftmsep}{}
```

25.0.12 Line number printing

\printlinefootnote — The \printlinefootnote macro is called in each \<type>footfmt command. It controls whether the line number is printed or not, according to the previous options. Its first argument is the information about lines, its second is the series of the footnote.

```
\newcommand{\printlinefootnote}[2]{%
   \def\extractline@##1|##2|##3|##4|##5|##6|##7|{##2}%
   \def\extractsubline@##1|##2|##3|##4|##5|##6|##7|{##3}%
   \def\extractendline@##1|##2|##3|##4|##5|##6|##7|{##5}%
   \def\extractendsubline@##1|##2|##3|##4|##5|##6|##7|{##6}%
   \iftoggle{numberonlyfirstintwolines@#2}{%
       \edef\lineinfo@{\extractline@ #1| - \extractsubline@ #1| - \extractendline@ #1| - \
       }%
       {%
       \edef\lineinfo@{\extractline@ #1| - \extractsubline@ #1|}%
       }%
```

```
\iftoggle{nonum@}{%Try if the line number must printed for this specific not (by default, yes)
\hspace{\csuse{inplaceofnumber@#2}}%
}%
{%
    {%
        \iftoggle{nonumberinfootnote@#2}%Try if the line number must printed (by default, yes)
        {%
        \hspace{\csuse{inplaceofnumber@#2}}%
        }%
        {%
           {\iftoggle{numberonlyfirstinline@#2}% If for this series the line number must be printe
              {%
              \ifcsdef{prevline#2}%
                 {%Be sure the \prevline exists.
                 \ifcsequal{prevline#2}{lineinfo@}%Try it
                    {%
                    \ifcsempty{symlinenum@#2}% Try if a symbol is define
                     {%
                         \hspace{\csuse{inplaceofnumber@#2}}%
                     }%
                     {\hspace{\csuse{beforesymlinenum@#2}}\csuse{Xnotenumfont@#2}%
                     \ifdimequal{\csuse{boxsymlinenum@#2}}{0pt}%
                         {\csuse{symlinenum@#2}}%
                         {\hbox to \csuse{boxsymlinenum@#2}{\csuse{symlinenum@#2}\hfill}}%
                     \hspace{\csuse{aftersymlinenum@#2}}}%
                    }%
                    {%
                      \hspace{\csuse{beforenumberinfootnote@#2}}\csuse{Xnotenumfont@#2}%
                     \ifdimequal{\csuse{boxlinenum@#2}}{0pt}{%
                         \iftoggle{pstartinfootnote@#2}{\printpstart}{}%
                         \printlines#1|}%
                         {%
                         \hbox to \csuse{boxlinenum@#2}{%
                           \iftoggle{pstartinfootnote@#2}{\printpstart}{}%
                             \iftoggle{onlypstartinfootnote@#2}{}{\printlines#1|}%
                   \hfill}%
                         }%
                         \iftoggle{nonbreakableafternumber@#2}{\nobreak}{}\hspace{\csuse{afternumb
                     }%
                 }%
                 {%
                 \hspace{\csuse{beforenumberinfootnote@#2}}\csuse{Xnotenumfont@#2}%
                 \ifdimequal{\csuse{boxlinenum@#2}}{0pt}{%
                     \iftoggle{pstartinfootnote@#2}{\printpstart}{}%
                     \iftoggle{onlypstartinfootnote@#2}{}{\printlines#1|}}%
                     {%
                     \hbox to \csuse{boxlinenum@#2}{%
                         \iftoggle{pstartinfootnote@#2}{\printpstart}{}%
                         \iftoggle{onlypstartinfootnote@#2}{}{\printlines#1|}%
                     \hfill}%
```

```
                        }%
                    \iftoggle{nonbreakableafternumber@#2}{\nobreak}{}\hspace{\csuse{aftern
                    }%
              }%
              {%
              \hspace{\csuse{beforenumberinfootnote@#2}}\csuse{Xnotenumfont@#2}%
              \ifdimequal{\csuse{boxlinenum@#2}}{0pt}{%
                     \iftoggle{pstartinfootnote@#2}{\printpstart}{}%
                     \iftoggle{onlypstartinfootnote@#2}{}{\printlines#1|}%
                     }%
                     {%
                     \hbox to \csuse{boxlinenum@#2}{%
                         \iftoggle{pstartinfootnote@#2}{\printpstart}{}%
                         \iftoggle{onlypstartinfootnote@#2}{}{\printlines#1|}%
                         \hfill}%
                     }%
              \iftoggle{nonbreakableafternumber@#2}{\nobreak}{}\hspace{\csuse{afternumber
              }%
              \csxdef{prevline#2}{\lineinfo@}%
                 }%
        }%
     }%
   }%
}
```

26 Output routine

Now we begin the output routine and associated things.

\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}

```

The next portion is probably the trickiest 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
 \ifvoid\topins\else\unvbox\topins\fi
 \dimen@=\dp\@cclv \unvbox\@cclv % open up \box255
 \do@feet
 \ifr@ggedbottom \kern-\dimen@ \vfil \fi}
```

`\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 eTeX limitations: the number of insertion classes is limited to 2^16.

With luck we might only have to change `\@makecol` and `\@reinserts`. The kernel definition of these, 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
}
```

Now we start actually changing things.

\m@m@makecolfloats
\m@m@makecoltext
\m@m@makecolintro

These macros are defined in the memoir class and form part of the definition of \@makecol.

```
\providecommand{\m@m@makecolfloats}{%
  \xdef\@freelist{\@freelist\@midlist}%
  \global \let \@midlist \@empty
  \@combinefloats}
\providecommand{\m@m@makecoltext}{%
  \ifvbox\@kludgeins
    \@makespecialcolbox
  \else
    \setbox\@outputbox \vbox to\@colht {%
      \@texttop
      \dimen@ \dp\@outputbox
      \unvbox\@outputbox
      \vskip -\dimen@
      \@textbottom}%
  \fi}
\providecommand{\m@m@makecolintro}{}

```

\l@d@makecol

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

```
\gdef\l@d@makecol{%
  \l@ddofootinsert
  \m@m@makecolfloats
  \m@m@makecoltext
  \global \maxdepth \@maxdepth}

```

\ifFN@bottom

The \ifFN@bottom macro is defined by the footmisc package. If this package is not loaded, we define it.

```
\AtBeginDocument{\@ifpackageloaded{footmisc}{}{\newif\ifFN@bottom}}
%     \end{macrocode}
% \end{macro}
% \begin{macro}{\l@ddofootinsert}
% This macro essentially holds the initial portion of the kernel
% \cs{@makecol} code.
% \changes{v0.2.1}{2003/09/13}{Renamed \cs{dofootinsert} as \cs{l@ddofootinsert}}
% \changes{v0.7}{2005/02/25}{Deleted \cs{page@start} from \cs{l@ddofootinsert}}
%    \begin{macrocode}
\newcommand*{\l@ddofootinsert}{%
%%%   \page@start
   \ifvoid\footins
     \setbox\@outputbox \box\@cclv
   \else
     \setbox\@outputbox \vbox {%
       \boxmaxdepth \@maxdepth
       \@tempdima\dp\@cclv
       \unvbox \@cclv
       \ifFN@bottom\vfill\fi\vskip \skip\footins%%% If the option bottom of loadmisc package is loade
       \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.

```
  \l@ddoxtrafeet
}

```

`\doxtrafeet` `\doxtrafeet` is the code extending `\@makecol` to cater for the extra eledmac feet. We have two classes of extra footnotes. We order the footnote inserts so that the regular footnotes are first, then class 1 (familiar footnotes) and finally class 2 (critical footnotes).

```
\newcommand*{\l@ddoxtrafeet}{%
  \doxtrafeeti
  \doxtrafeetii}

```

`\doxtrafeetii` `\doxtrafeetii` is the code extending `\@makecol` to cater for the extra critical feet (class 2 feet). NOTE: the code is likely to be 'featurefull'.

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

`\@opxtrafeetii` The extra critical feet to be aded to the output.

```
\newcommand*{\@opxtrafeetii}{%
  \def\do##1{\ifvoid\csuse{##1footins}\else\csuse{##1footstart}{##1}\csuse{##1footgroup}{##1}\fi}
  \dolistloop{\@series}}
```

`\l@ddodoreinxtrafeet` `\l@ddodoreinxtrafeet` is the code for catering for the extra footnotes within `\@reinserts`. The implementation may well have to change. We use the same classes and ordering as in `\l@ddoxtrafeet`.

```
\newcommand*{\l@ddodoreinxtrafeet}{%
  \doreinxtrafeeti
  \doreinxtrafeetii}

```

`\doreinxtrafeetii` `\doreinxtrafeetii` is the code for catering for the class 2 extra critical footnotes within `\@reinserts`. The implementation may well have to change.

```
\newcommand*{\doreinxtrafeetii}{%
  \def\do##1{\ifvoid\csuse{##1footins}\else\insert\csuse{##1ootins}{\unvbox\csuse{##1foot
  \dolistloop{\@series}
}

```

`\l@d@reinserts` And here is the modified version of `\@reinserts`.

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

```

The memoir class does not use the 'standard' versions of `\@makecol` and `\@reinserts`, due to its sidebar insert. We had better add that code if memoir is used. (It can be awkward dealing with `\if` code within `\if` code, so don't use `\ifl@dmemoir` here.)

```
\@ifclassloaded{memoir}{%
```

memoir is loaded so we use memoir's built in hooks.

```
  \g@addto@macro{\m@mdoextrafeet}{\l@ddoxtrafeet}%
  \g@addto@macro{\m@mdodoreinextrafeet}{\l@ddodoreinxtrafeet}%
  }{%
```

memoir has not been loaded, so redefine `@makecol` and `@reinserts`.

```
  \gdef\@makecol{\l@d@makecol}%
  \gdef\@reinserts{\l@d@reinserts}%
}

```

`\addfootins` `\addfootins` is for backward compatibility, but should'nt be used anymore.

```
\newcommand*{\addfootins}[1]{%
   \eledmac@warning{addfootins is deprecated, use newseries instead}
  \footnormal{#1}
  \g@addto@macro{\@opxtrafeetii}{%
    \ifvoid\@nameuse{#1footins}\else
      \@nameuse{#1footstart{#1}}\@nameuse{#1footgroup}{#1}\fi}
  \g@addto@macro{\doreinxtrafeetii}{%
    \ifvoid\@nameuse{#1footins}\else
      \insert\@nameuse{#1footins}{\unvbox\@nameuse{#1footins}}\fi}
  \g@addto@macro{\l@dedbeginmini}{%
    \expandafter\let\csname #1footnote\endcsname = \@nameuse{mp#1footnote}}
  \g@addto@macro{\l@dedendmini}{%
    \ifvoid\@nameuse{mp#1footins}\else\@nameuse{mpfootgroup#1{#1}}\fi}
}
```

It turns out that `\@doclearpage` also needs modifying.

`\if@led@nofoot` `\@led@extranofeet` We have to check if there are any leftover feet. `\@led@extranofeet` is a hook for handling further footnotes.

```
\newif\if@led@nofoot
\newcommand*{\@led@extranofeet}{}

\@ifclassloaded{memoir}{%
```

If the memoir class is loaded we hook into its modified `\@doclearpage`.

`\@mem@extranofeet`

```
\g@addto@macro{\@mem@extranofeet}{%
  \def\do#1{\ifvoid\csuse{#1footins}\else\@mem@nofootfalse\fi%
  \ifvoid\csuse{footins#1}\else\@mem@nofootfalse\fi%
  }
  \dolistloop{\@series}%
  \@led@extranofeet}
}{%
```

As memoir is not loaded we have to do it all here.

`\@led@testifnofoot` `\@doclearpage`

```
\newcommand*{\@led@testifnofoot}{%
  \@led@nofoottrue
  \ifvoid\footins\else\@led@nofootfalse\fi
  \def\do##1{\ifvoid\csuse{##1footins}\else\@led@nofootfalse\fi%
  \ifvoid\csuse{footins##1}\else\@led@nofootfalse\fi}%
  \dolistloop{\@series}
  \@led@extranofeet}

\renewcommand{\@doclearpage}{%
  \@led@testifnofoot
  \if@led@nofoot
    \setbox\@tempboxa\vsplit\@cclv to\z@ \unvbox\@tempboxa
    \setbox\@tempboxa\box\@cclv
    \xdef\@deferlist{\@toplist\@botlist\@deferlist}%
    \global \let \@toplist \@empty
    \global \let \@botlist \@empty
    \global \@colroom \@colht
    \ifx \@currlist\@empty
    \else
       \@latexerr{Float(s) lost}\@ehb
       \global \let \@currlist \@empty
    \fi
    \@makefcolumn\@deferlist
    \@whilesw\if@fcolmade \fi{\@opcol\@makefcolumn\@deferlist}%
    \if@twocolumn
      \if@firstcolumn
        \xdef\@dbldeferlist{\@dbltoplist\@dbldeferlist}%
        \global \let \@dbltoplist \@empty
        \global \@colht \textheight
        \begingroup
           \@dblfloatplacement
           \@makefcolumn\@dbldeferlist
           \@whilesw\if@fcolmade \fi{\@outputpage
                                       \@makefcolumn\@dbldeferlist}%
        \endgroup
      \else
        \vbox{}\clearpage
      \fi
    \fi
  \else
    \setbox\@cclv\vbox{\box\@cclv\vfil}%
    \l@d@makecol\@opcol
    \clearpage
  \fi}
}

```

27 Cross referencing

Peter Wilson have rewritten portions of the code in this section so that the LaTeX `.aux` file is used. This will also handle `\include`d files.

Further, I have renamed some of the original `EDMAC` macros so that they do not clash with the LaTeX label/ref commands (`EDMAC` and LaTeX use very different mechanisms). In particular, the original `EDMAC` `\label` and `\pageref` have been renamed as `\edlabel` and `\edpageref` respectively.

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

`\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}
```

`\zz@@@` A convenience macro to zero two labeling counters in one go.

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

```

\edlabel The `\edlabel` 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.[29]

This version of the original EDMAC `\label` uses `\@bsphack` and `\@esphack` to eliminate extra space problems and also the LaTeX write methods for the `.aux` file.

Jesse Billett[30] found that the original code could be off by several pages. This version, hopefully cures that, and also allows for non-arabic page numbering.

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

Use code from the kernel `\label` command to write the correct page number (it seems possible that the original EDMAC's `\page@num` scheme might also have had problems in this area).

```
  \protected@write\@auxout{}%
    {\string\l@dmake@labels\space\thepage|\label@refs|{#1}}%
  \@esphack}

```

\advancelabel@refs
\labelrefsparseline
\labelrefsparsesubline

In cases where `\edlabel` is the first element in a paragraph, we have a problem with line counts, because line counts change only at the first horizontal box of the paragraph. Hence, we need to test `\edlabel` if it occurs at the start of a paragraph. To do so, we use `\ifvmode`. If the test is true, we must advance by one unit the amount of text we write into the `.aux` file. We do so using `\advancelabel@refs` command.

```
\newcounter{line}%
\newcounter{subline}%
\newcommand{\advancelabel@refs}{%
    \setcounter{line}{\expandafter\labelrefsparseline\label@refs}%
```

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

[30](`jdb43@cam.ac.uk`) via the `ctt` thread 'ledmac cross referencing', 25 August 2003.

```
    \stepcounter{line}%
    \ifsublines@%
        \setcounter{subline}{\expandafter\labelrefsparsesubline\label@refs}%
        \stepcounter{subline}{1}%
        \def\label@refs{\theline|\thesubline}%
    \else%
        \def\label@refs{\theline|0}%
    \fi%
}
\def\labelrefsparseline#1|#2{#1}
\def\labelrefsparsesubline#1|#2{#2}
```

`\l@dmake@labels` The `\l@dmake@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).

The initial use of `\newcommand` is to catch if `\l@dmake@labels` has been previously defined (by a class or package).

```
\newcommand*{\l@dmake@labels}{}
\def\l@dmake@labels#1|#2|#3|#4{%
  \expandafter\ifx\csname the@label#4\endcsname \relax\else
    \led@warn@DuplicateLabel{#4}%
  \fi
  \expandafter\gdef\csname the@label#4\endcsname{#1|#2|#3}%
  \ignorespaces}

```

LaTeX reads the `aux` file at both the beginning and end of the document, so we have to switch off duplicate label checking after the first time the file is read.

```
\AtBeginDocument{%
  \def\l@dmake@labels#1|#2|#3|#4{}%
}

```

`\@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. However, it appears that this is not the right place to grab the page number. That task is now done in the `\edlabel` macro. This version of `\@lab` appends just the current line and sub-line numbers to `\labelref@list`.

```
\newcommand*{\@lab}{\xright@appenditem
  {\linenumrep{\line@num}|%
     \ifsublines@ \sublinenumrep{\subline@num}\else 0\fi}\to\labelref@list}

```

`\edpageref` `\xpageref` If the specified label exists, `\edpageref` 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 `\edlabel` 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`.

```
\newcommand*{\edpageref}[1]{\l@dref@undefined{#1}\l@dgetref@num{1}{#1}}
\newcommand*{\xpageref}[1]{\l@dgetref@num{1}{#1}}

```

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

```
\newcommand*{\lineref}[1]{\l@dref@undefined{#1}\l@dgetref@num{2}{#1}}
\newcommand*{\xlineref}[1]{\l@dgetref@num{2}{#1}}

```

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

```
\newcommand*{\sublineref}[1]{\l@dref@undefined{#1}\l@dgetref@num{3}{#1}}
\newcommand*{\xsublineref}[1]{\l@dgetref@num{3}{#1}}

```

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.

`\l@dref@undefined` The `\l@dref@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 label has been defined.

```
\newcommand*{\l@dref@undefined}[1]{%
  \expandafter\ifx\csname the@label#1\endcsname\relax
    \led@warn@RefUndefined{#1}%
  \fi}

```

`\l@dgetref@num` Next, `\l@dgetref@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 `\l@dlabel@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`.

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

```

\l@dlabel@parse Notice that we slipped another | delimiter into the penultimate line of \l@dgetref@num, to keep the 'switch-number' separate from the reference numbers. This | is used as another parameter delimiter by \l@dlabel@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 \l@dgetref@num.)

```
\newcommand*{\l@dlabel@parse}{}
\def\l@dlabel@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 \critext for one reason or another. Using \xxref in the second argument of \critext lets you set things up at least semi-automatically.

```
\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.

```
\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. 75 and 55), since \xxref makes a call to \linenum in order to do its work.)

28 Endnotes

\l@d@end
\ifl@dend@
\l@dend@true
\l@dend@false

Endnotes of all varieties are saved up in a file, typically named ⟨*jobname*⟩`.end`. `\l@d@end` is the output stream number for this file, and `\ifl@dend@` is a flag that's `true` when the file is open.

```
\newwrite\l@d@end
\newif\ifl@dend@
```

\l@dend@open
\l@dend@close

`\l@dend@open` and `\l@dend@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.

```
\newcommand{\l@dend@open}[1]{\global\l@dend@true\immediate\openout\l@d@end=#1\relax}
\newcommand{\l@dend@close}{\global\l@dend@false\immediate\closeout\l@d@end}

```

\l@dend@stuff

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

```
\newcommand{\l@dend@stuff}{%
 \ifl@dend@\relax\else
   \l@dend@open{\jobname.end}%
 \fi
 \immediate\write\l@d@end{\string\l@d@section{\the\section@num}}

```

\endprint
\@gobblethree
\l@d@section

The `\endprint` here is nearly identical in its functioning to `\normalfootfmt`.

The endnote file also contains `\l@d@section` commands, which supply the section numbers from the main text; standard eledmac 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#4{{\csuse{bhookXendnote@#4}\csuse{Xendnotefontsize@#4}{\csuse{Xendnotenumfont@#4}
      \enspace{\select@lemmafont#1|#2}\enskip#3\par}}
\providecommand*{\@gobblethree}[3]{}

\let\l@d@section=\@gobble

```

\setprintendlines

The `\printendlines` macro is similar to `\printlines` but is for printing endnotes rather than footnotes.

The principal difference between foot- and endnotes is that footnotes are printed on the page where they are specified but endnotes are printed at a different point in the document. We need an indication of the source of an endnote; `\setprintendlines` provides this by always printing the page number. The coding is slightly simpler than `\setprintlines`.

First of all, we print the second page number only if the ending page number is different from the starting page number.

```
\newcommand*{\setprintendlines}[6]{%
  \l@d@pnumfalse \l@d@dashfalse
  \ifnum#4=#1 \else
    \l@d@pnumtrue
    \l@d@dashtrue
  \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.

```
  \ifl@d@pnum \l@d@elintrue \else \l@d@elinfalse \fi
  \ifnum#2=#5 \else
      \l@d@elintrue
      \l@d@dashtrue
  \fi
```

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

```
  \l@d@ssubfalse
  \ifnum#3=0 \else
      \l@d@ssubtrue
  \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.

```
  \l@d@eslfalse
  \ifnum#6=0 \else
      \ifnum#6=#3
         \ifl@d@elin \l@d@esltrue \else \l@d@eslfalse \fi
      \else
         \l@d@esltrue
         \l@d@dashtrue
      \fi
  \fi}
```

`\printendlines` Now we're ready to print it all.

```
\def\printendlines#1|#2|#3|#4|#5|#6|#7|{\begingroup
  \setprintendlines{#1}{#2}{#3}{#4}{#5}{#6}%
```

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).

```
  \printnpnum{#1} \linenumrep{#2}%
  \ifl@d@ssub \fullstop \sublinenumrep{#3}\fi
  \ifl@d@dash \endashchar\fi
  \ifl@d@pnum \printnpnum{#4}\fi
  \ifl@d@elin \linenumrep{#5}\fi
  \ifl@d@esl \ifl@d@elin \fullstop\fi \sublinenumrep{#6}\fi
\endgroup}

```

`\printnpnum` A macro to print a page number in an endnote.

```
\newcommand*{\printnpnum}[1]{p.#1) }

```

`\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.

```
\newcommand*{\doendnotes}[1]{\l@dend@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 log file.

```
\newcommand*{\noendnotes}{\global\let\l@dend@stuff=\relax
                \global\chardef\l@d@end=16 }
```

29 Side notes

Regular \marginpars do not work inside numbered text — they don't produce any note but do put an extra unnumbered blank line into the text.

`\l@dold@xympar` `\@xympar` Changing \@xympar a little at least ensures that \marginpars in numbered text do not disturb the flow.

```
\let\l@dold@xympar\@xympar
\renewcommand{\@xympar}{%
  \ifnumberedpar@
    \led@warn@NoMarginpars
    \@esphack
  \else
    \l@dold@xympar
  \fi}

```

We provide side notes as replacement for \marginpar in numbered text.

`\sidenote@margin` `\sidenotemargin` `\l@dgetsidenote@margin` These are the sidenote equivalents to \line@margin and \linenummargin for specifying which margin. The default is the right margin (opposite to the default for line numbers).

```
\newcount\sidenote@margin
\newcommand*{\sidenotemargin}[1]{{%
  \l@dgetsidenote@margin{#1}%
  \ifnum\@l@dtempcntb>\m@ne
    \global\sidenote@margin=\@l@dtempcntb
  \fi}}
\newcommand*{\l@dgetsidenote@margin}[1]{%
  \def\@tempa{#1}\def\@tempb{left}%
  \ifx\@tempa\@tempb
    \@l@dtempcntb \z@
  \else
    \def\@tempb{right}%
    \ifx\@tempa\@tempb
      \@l@dtempcntb \@ne
    \else
      \def\@tempb{outer}%
      \ifx\@tempa\@tempb
        \@l@dtempcntb \tw@
      \else
        \def\@tempb{inner}%
        \ifx\@tempa\@tempb
          \@l@dtempcntb \thr@@
        \else
          \led@warn@BadSidenotemargin
          \@l@dtempcntb \m@ne
        \fi
      \fi
    \fi
  \fi}
\sidenotemargin{right}

```

`\l@dlp@rbox` `\l@drp@rbox` We need two boxes to store sidenote texts.

```
\newbox\l@dlp@rbox
\newbox\l@drp@rbox

```

`\ledlsnotewidth` `\ledrsnotewidth` `\ledlsnotesep` `\ledrsnotesep` `\ledlsnotefontsetup` `\ledrsnotefontsetup` These specify the width of the left/right boxes (initialised to `\marginparwidth`, their distance from the text (initialised to `\linenumsep`, and the fonts used.

```
\newdimen\ledlsnotewidth \ledlsnotewidth=\marginparwidth
\newdimen\ledrsnotewidth \ledrsnotewidth=\marginparwidth
\newdimen\ledlsnotesep \ledlsnotesep=\linenumsep
\newdimen\ledrsnotesep \ledrsnotesep=\linenumsep
\newcommand*{\ledlsnotefontsetup}{\raggedleft\footnotesize}
\newcommand*{\ledrsnotefontsetup}{\raggedright\footnotesize}

```

`\ledleftnote` `\ledrightnote` `\ledsidenote` `\ledleftnote{`⟨*text*⟩`}` and `\ledrightnote{`⟨*text*⟩`}` are the user commands for left and right sidenotes. `\ledsidenote{`⟨*text*⟩`}` is the command for a moveable sidenote.

```
\newcommand*{\ledleftnote}[1]{\edtext{}{\l@dlsnote{#1}}}
\newcommand*{\ledrightnote}[1]{\edtext{}{\l@drsnote{#1}}}
\newcommand*{\ledsidenote}[1]{\edtext{}{\l@dcsnote{#1}}}


```

`\l@dlsnote` `\l@drsnote` `\l@dcsnote` The 'footnotes' for left, right, and moveable sidenotes. The whole scheme is reminiscent of the critical footnotes code.

```
\newif\ifrightnoteup
  \rightnoteuptrue
\newcommand*{\l@dlsnote}[1]{%
  \begingroup%
  \newcommand{\content}{#1}%
  \ifnumberedpar@
    \xright@appenditem{\noexpand\vl@dlsnote{\csexpandonce{content}}}%
                         \to\inserts@list
    \global\advance\insert@count \@ne
  \fi\ignorespaces\endgroup}
\newcommand*{\l@drsnote}[1]{%
  \begingroup%
  \newcommand{\content}{#1}%
  \ifnumberedpar@
    \xright@appenditem{\noexpand\vl@drsnote{\csexpandonce{content}}}%
                         \to\inserts@list
    \global\advance\insert@count \@ne
  \fi\ignorespaces\endgroup}
\newcommand*{\l@dcsnote}[1]{\begingroup%
  \newcommand{\content}{#1}%
  \ifnumberedpar@
    \xright@appenditem{\noexpand\vl@dcsnote{\csexpandonce{content}}}%
                         \to\inserts@list
    \global\advance\insert@count \@ne
  \fi\ignorespaces\endgroup}

```

`\vl@dlsnote` `\vl@drsnote` `\vl@dcsnote` Put the left/right text into boxes, but just save the moveable text. `\l@dcsnotetext` is a etoolbox list (comma separated)

```
\newcommand*{\vl@dlsnote}[1]{\setl@dlp@rbox{#1}}
\newcommand*{\vl@drsnote}[1]{\setl@drp@rbox{#1}}
\newcommand*{\vl@dcsnote}[1]{\listgadd{\l@dcsnotetext}{#1}}

```

`\setl@dlp@rbox` `\setl@drpr@box` `\setl@dlprbox{`⟨*lednums*⟩`}{`⟨*tag*⟩`}{`⟨*text*⟩`}` puts ⟨*text*⟩ into the `\l@dlp@rbox` box. And similarly for the right side box. It is these boxes that finally get displayed in the margins.

```
\newcommand*{\setl@dlp@rbox}[1]{%
  {\parindent\z@\hsize=\ledlsnotewidth\ledlsnotefontsetup
   \global\setbox\l@dlp@rbox
   \ifleftnoteup
     =\vbox to\z@{\vss #1}%
   \else
     =\vbox to 0.70\baselineskip{\strut#1\vss}%
   \fi}}
\newcommand*{\setl@drp@rbox}[1]{%
  {\parindent\z@\hsize=\ledrsnotewidth\ledrsnotefontsetup
```

```
   \global\setbox\l@drp@rbox
   \ifrightnoteup
     =\vbox to\z@{\vss#1}%
   \else
     =\vbox to0.7\baselineskip{\strut#1\vss}%
   \fi}}
\newif\ifleftnoteup
   \leftnoteuptrue
```

`\sidenotesep` This macro is used to separate sidenotes of the same line.

```
\newcommand{\sidenotesep}{, }
%   \end{macrocode}
% \end{macro}
% \begin{macro}{\affixside@note}
% This macro puts any moveable sidenote text into the left or right sidenote
% box, depending on which margin it is meant to go in. It's a very much
% stripped down version of \cs{affixlin@num}.
%
% Before do it, we concatenate all moveable sidenotes of the line, using \cs{sidenotesep}
% It's the result that we put on the sidenote.
% \changes{v1.4.1}{2012/11/16}{Remove spurious spaces.}
%    \begin{macrocode}
\newcommand*{\affixside@note}{%
    \def\sidenotecontent@{}%
    \numdef{\itemcount@}{0}%
    \def\do##1{%
        \ifnumequal{\itemcount@}{0}%
            {%
            \appto\sidenotecontent@{##1}}% Not print not separator before the 1st note
            {\appto\sidenotecontent@{\sidenotesep ##1}%
            }%
            \numdef{\itemcount@}{\itemcount@+1}%
    }%
    \dolistloop{\l@dcsnotetext}%
    \ifnumgreater{\itemcount@}{1}{\eledmac@warning{\itemcount@\space sidenotes on line \t
```

And now, the main part of the macro

```
  \gdef\@templ@d{}%
  \ifx\@templ@d\l@dcsnotetext \else%
    \if@twocolumn%
      \if@firstcolumn%
        \setl@dlp@rbox{##1}{\sidenotecontent@}%
      \else%
        \setl@drp@rbox{\sidenotecontent@}%
      \fi%
    \else%
      \@l@dtempcntb=\sidenote@margin%
      \ifnum\@l@dtempcntb>\@ne%
        \advance\@l@dtempcntb by\page@num%
      \fi%
      \ifodd\@l@dtempcntb%
        \setl@drp@rbox{\sidenotecontent@}%
      \else%
        \setl@dlp@rbox{\sidenotecontent@}%
      \fi%
    \fi%
\fi}
```

30 Minipages and such

We can put footnotes into minipages. The preparatory code has been set up earlier, all that remains is to ensure that it is available inside a minipage box. This requires some alteration to the kernel code, specifically the `\@iiiminipage` and `\endminipage` macros. We'll arrange this so that additional series can be easily added.

`\l@dfeetbeginmini` `\l@dfeetendmini` These will be the hooks in `\@iiiminpage` and `\endminipage` They can be extended to handle other things if necessary.

```
\newcommand*{\l@dfeetbeginmini}{\l@dedbeginmini\l@dfambeginmini}
\newcommand*{\l@dfeetendmini}{\l@dedendmini\l@dfamendmini}

```

`\l@dedbeginmini` `\l@dedendmini` These handle the initiation and closure of critical footnotes in a minipage environment.

```
\newcommand*{\l@dedbeginmini}{%
  \def\do##1{\csletcs{v##1footnote}{mpv##1footnote}}%
  \dolistloop{\@series}%
  }
\newcommand*{\l@dedendmini}{%
  \ifl@dpairing
    \ifledRcol
      \flush@notesR
    \else
      \flush@notes
    \fi
  \fi
  \def\do##1{\ifvoid\csuse{mp##1footins}\else\csuse{mp##1footgroup}{##1}\fi}%
  \dolistloop{\@series}%
  }

```

`\l@dfambeginmini` `\l@dfamendmini` These handle the initiation and closure of familiar footnotes in a minipage environment.

```
\newcommand*{\l@dfambeginmini}{%
   \def\do##1{\csletcs{vfootnote##1}{mpvfootnote##1}}%
   \dolistloop{\@series}}
\newcommand*{\l@dfamendmini}{%
  \def\do##1{\ifvoid\csuse{mpfootins##1}\else\csuse{mpfootgroup##1}{##1}\fi}%
  \dolistloop{\@series}}
```

\@iiiminipage This is our extended form of the kernel `\@iiiminipage` defined in `ltboxes.dtx`.

```
\def\@iiiminipage#1#2[#3]#4{%
  \leavevmode
  \@pboxswfalse
  \setlength\@tempdima{#4}%
  \def\@mpargs{{#1}{#2}[#3]{#4}}%
  \setbox\@tempboxa\vbox\bgroup
    \color@begingroup
      \hsize\@tempdima
      \textwidth\hsize \columnwidth\hsize
      \@parboxrestore
      \def\@mpfn{mpfootnote}\def\thempfn{\thempfootnote}\c@mpfootnote\z@
      \let\@footnotetext\@mpfootnotetext
```

The next line is our addition to the original.

```
      \l@dfeetbeginmini%                  added
      \let\@listdepth\@mplistdepth \@mplistdepth\z@
      \@minipagerestore
      \@setminipage}

```

\endminipage This is our extended form of the kernel `\endminipage` defined in `ltboxes.dtx`.

```
\def\endminipage{%
  \par
  \unskip
  \ifvoid\@mpfootins\else
    \l@dunboxmpfoot
  \fi
```

The next line is our addition to the original.

```
  \l@dfeetendmini%                 added
  \@minipagefalse
  \color@endgroup
  \egroup
  \expandafter\@iiiparbox\@mpargs{\unvbox\@tempboxa}}

```

\l@dunboxmpfoot

```
\newcommand*{\l@dunboxmpfoot}{%
    \vskip\skip\@mpfootins
    \normalcolor
    \footnoterule
    \unvbox\@mpfootins}

```

ledgroup This environment puts footnotes at the end, even if that happens to be in the middle of a page, or crossing a page boundary. It is a sort of unboxed, fixed width minipage.

```
\newenvironment{ledgroup}{%
  \def\@mpfn{mpfootnote}\def\thempfn{\thempfootnote}\c@mpfootnote\z@
  \let\@footnotetext\@mpfootnotetext
  \l@dfeetbeginmini%
}{%
  \par
  \unskip
  \ifvoid\@mpfootins\else
    \l@dunboxmpfoot
  \fi
  \l@dfeetendmini%
}

```

ledgroupsized `\begin{ledgroupsized}[`⟨*pos*⟩`]{`⟨*width*⟩`}`

This environment puts footnotes at the end, even if that happens to be in the middle of a page, or crossing a page boundary. It is a sort of unboxed, variable ⟨*width*⟩ minipage. The optional ⟨*pos*⟩ controls the sideways position of numbered text.

```
\newenvironment{ledgroupsized}[2][l]{%
```

Set the various text measures.

```
  \hsize #2\relax
%%  \textwidth #2\relax
%%  \columnwidth #2\relax
```

Initialize fills for centering.

```
  \let\ledllfill\hfil
  \let\ledrlfill\hfil
  \def\@tempa{#1}\def\@tempb{l}%
```

Left adjusted numbered lines

```
      \ifx\@tempa\@tempb
    \let\ledllfill\relax
  \else
    \def\@tempb{r}%
    \ifx\@tempa\@tempb
```

Right adjusted numbered lines

```
      \let\ledrlfill\relax
    \fi
  \fi
```

Set up the footnoting.

```
  \def\@mpfn{mpfootnote}\def\thempfn{\thempfootnote}\c@mpfootnote\z@
  \let\@footnotetext\@mpfootnotetext
  \l@dfeetbeginmini%
}{%
  \par
  \unskip
  \ifvoid\@mpfootins\else
```

```
    \l@dunboxmpfoot
  \fi
  \l@dfeetendmini%
}

```

31 Indexing

Here's some code for indexing using page & line numbers.

\pagelinesep \edindexlab \c@labidx In order to get a correct line number we have to use the label/ref mechanism. These macros are for that.

```
\newcommand{\pagelinesep}{-}
\newcommand{\edindexlab}{$&}
\newcounter{labidx}
\setcounter{labidx}{0}

```

\doedindexlabel This macro sets an \edlabel.

```
\newcommand{\doedindexlabel}{\stepcounter{labidx}%
  \edlabel{\edindexlab\thelabidx}}

```

\thepageline This macro makes up the page/line number combo from the label/ref.

```
\newcommand{\thepageline}{%
  \thepage\pagelinesep\lineref{\edindexlab\thelabidx}}
```

\thestartpageline \theendpageline These macros make up the page/line start/end number when the \edindex command is called in critical notes.

```
\newcommand{\thestartpageline}{\l@dparsedstartpage\pagelinesep\l@dparsedstartline}
\newcommand{\theendpageline}{\l@dparsedendpage\pagelinesep\l@dparsedendline}
%     \end{macrocode}
% \end{macro}
% \end{macro}
%
% \begin{macro}{\if@edindex@fornote@true}
% This boolean test is switching at the beginning of each critical note,
% to allow indexing in this note.
%    \begin{macrocode}
\newif\if@edindex@fornote@
```

\prepare@edindex@fornote This macro is called at the beginning of each critical note. It switches some parameters, to allow indexing in this note, with reference to page and line number.

```
\newcommand{\prepare@edindex@fornote}[1]{%
    \l@dp@rsefootspec#1|%
    \@edindex@fornote@true
}
```

\get@index@command This macro is used to analyse if a text to be indexed has a command after a |.

```
\def\get@index@command#1|#2{\gdef\@index@command{#2}\gdef\@index@txt{#1}}
% \end{macro}
%     \end{macrocode}
% \begin{macro}{\ledinnote}
% \begin{macro}{\ledinnotehyperpage}
% These macros are used to specifiy that an index reference points to a note.
%    \begin{macrocode}
\newcommand{\ledinnote}[2]{\csuse{#1}{#2\emph{n}}}
\newcommand{\ledinnotehyperpage}[2]{\csuse{#1}{\hyperpage{#2}\emph{n}}}
```

The memoir class provides more flexible indexing than the standard classes. We need different code if the memoir class is being used.

```
\@ifclassloaded{memoir}{%
```

memoir is being used.

\makeindex \edindex Need to add the definition of \edindex to \makeindex, and initialise \edindex to do nothing. In this case \edindex has an optional argument. We use the hook provided in memoir v1.61.

```
  \g@addto@macro{\makememindexhook}{%
    \def\edindex{\@bsphack%
      \@ifnextchar [{\l@d@index}{\l@d@index[\jobname]}}}
  \newcommand{\edindex}[2][\jobname]{\@bsphack\@esphack}
```

\l@d@index \l@d@index[file] is the first stage of \edindex, handling the idx file. This a virtually a verbatim copy of memoir's \@index, the change being calling \l@dwrindexm@m instead of \@wrindexm@m.

```
  \def\l@d@index[#1]{%
    \@ifundefined{#1@idxfile}%
    {\ifreportnoidxfile
       \led@warn@NoIndexFile{#1}%
     \fi
     \begingroup
     \@sanitize
     \@nowrindex}%
    {\def\@idxfile{#1}%
     \doedindexlabel
     \begingroup
     \@sanitize
     \l@d@wrindexm@m}}
```

\l@d@wrindexm@m \l@d@@wrindexhyp \l@d@wrindexm@m{item} writes the idx file name and the indexed item to the aux file. These are almost verbatim copies of memoir's \@wrindexm@m and \@@wrindexhyp.

```
  \newcommand{\l@d@wrindexm@m}[1]{\l@d@@wrindexhyp#1||\\}
  \def\l@d@@wrindexhyp#1|#2|#3\\{%
    \ifshowindexmark\@showidx{#1}\fi
    \ifx\\#2\\%
```

```
     \if@edindex@fornote@%
        \protected@write\@auxout{}%
         {\string\@@wrindexm@m{\@idxfile}{#1|(ledinnotehyperpage}{\thestartpageline}}%
        \protected@write\@auxout{}%
         {\string\@@wrindexm@m{\@idxfile}{#1|)ledinnotehyperpage}{\theendpageline}}%
     \else%
        \protected@write\@auxout{}%
         {\string\@@wrindexm@m{\@idxfile}{#1|hyperpage}{\thepageline}}%
     \fi%
  \else
    \def\Hy@temp@A{#2}%
    \ifx\Hy@temp@A\HyInd@ParenLeft
      \if@edindex@fornote@%
        \protected@write\@auxout{}%
          {\string\@@wrindexm@m{\@idxfile}{#1|(ledinnotehyperpage{#2}}{\thestartpagelin
        \protected@write\@auxout{}%
          {\string\@@wrindexm@m{\@idxfile}{#1|)ledinnotehyperpage{#2}}{\theendpageline}
      \else%
        \protected@write\@auxout{}%
          {\string\@@wrindexm@m{\@idxfile}{#1|#2hyperpage}{\thepageline}}%
      \fi%
    \else
      \if@edindex@fornote@%
        \protected@write\@auxout{}%
          {\string\@@wrindexm@m{\@idxfile}{#1|(ledinnote{#2}}{\thestartpageline}}%
        \protected@write\@auxout{}%
          {\string\@@wrindexm@m{\@idxfile}{#1|)ledinnote{#2}}{\theendpageline}}%
      \else%
        \protected@write\@auxout{}%
          {\string\@@wrindexm@m{\@idxfile}{#1|#2}{\thepageline}}%
      \fi%
    \fi
  \fi
  \endgroup
  \@esphack}
```

That finishes the memoir-specific code.

```
}{%
```

memoir is not being used, which makes life somewhat simpler.

\makeindex \edindex Need to add the definition of \edindex to \makeindex, and initialise \edindex to do nothing.

```
\pretocmd{\makeindex}{%
  \def\edindex{\@bsphack
  \doedindexlabel
  \begingroup
  \@sanitize
  \@wredindex}}{}{}
\newcommand{\edindex}[1]{\@bsphack\@esphack}
```

`\@wredindex` Write the index information to the idx file.

```
\newcommandx{\@wredindex}[2][1=\jobname,usedefault]{%#1 = the index name, #2 = the text
  \ifl@imakeidx%
    \if@edindex@fornote@%
      \IfSubStr[1]{#2}{|}{\get@index@command#2}{\get@index@command#2|}%
      \expandafter\imki@wrindexentry{#1}{\@index@txt|(ledinnote{\@index@command}}{\thestartpageline
      \expandafter\imki@wrindexentry{#1}{\@index@txt|)ledinnote{\@index@command}}{\theendpageline}%
    \else%
      \imki@wrindexentry{#1}{#2}{\thepageline}%
    \fi%
  \else%
    \if@edindex@fornote@%
      \IfSubStr[1]{#2}{|}{\get@index@command#2}{\get@index@command#2|}%
      \expandafter\protected@write\@indexfile{}%
        {\string\indexentry{\@index@txt|(ledinnote{\@index@command}}{\thestartpageline}
        }%
      \expandafter\protected@write\@indexfile{}%
        {\string\indexentry{\@index@txt|)ledinnote{\@index@command}}{\theendpageline}
       }%
    \else%
      \protected@write\@indexfile{}%
        {\string\indexentry{#2}{\thepageline}
        }%
    \fi%
   \fi%
\endgroup
\@esphack}
```

That finishes the non-memoir index code.

```
}

```

`\l@d@@wrindexhyp` If the hyperref package is not loaded, it doesn't make sense to clutter up the index with hyperreffing things.

```
\AtBeginDocument{\@ifpackageloaded{hyperref}{}{%
  \def\l@d@@wrindexhyp#1||\\{%
    \ifshowindexmark\@showidx{#1}\fi
    \IfSubStr[1]{#1}{|}{\get@index@command#1}{\get@index@command#1|}%
      \if@edindex@fornote@%
         \protected@write\@auxout{}%
            {\string\@@wrindexm@m{\@idxfile}{\@index@txt|(ledinnote{\@index@command}}{\thestartpageli
         \protected@write\@auxout{}%
            {\string\@@wrindexm@m{\@idxfile}{\@index@txt|)ledinnote{\@index@command}}{\theendpageline
      \else%
         \protected@write\@auxout{}%
            {\string\@@wrindexm@m{\@idxfile}{#1}{\thepageline}}%
      \fi%
    \endgroup
    \@esphack}}}
```

32 Macro as environment

The following is borrowed, and renamed, from the amsmath package. See also the CTT thread 'eeq and amstex', 1995/08/31, started by Keith Reckdahl and ended definitively by David M. Jones.

Several of the [math] macros scan their body twice. This means we must collect all text in the body of an environment form before calling the macro.

\@emptytoks This is actually defined in the amsgen package.

```
\newtoks\@emptytoks

```

The rest is from amsmath.

\l@denvbody A token register to contain the body.

```
\newtoks\l@denvbody

```

\addtol@denvbody `\addtol@denvdody{arg}` adds `arg` to the token register `\l@denvbody`.

```
\newcommand{\addtol@denvbody}[1]{%
  \global\l@denvbody\expandafter{\the\l@denvbody#1}}

```

\l@dcollect@body The macro `\l@dcollect@body` starts the scan for the `\end{...}` command of the current environment. It takes a macro name as argument. This macro is supposed to take the whole body of the environment as its argument. For example, given `cenv#1{...}` as a macro that processes `#1`, then the environment form, `\begin{env}` would call `\l@dcollect@body\cenv`.

```
\newcommand{\l@dcollect@body}[1]{%
  \l@denvbody{\expandafter#1\expandafter{\the\l@denvbody}}%
  \edef\processl@denvbody{\the\l@denvbody\noexpand\end{\@currenvir}}%
  \l@denvbody\@emptytoks \def\l@dbegin@stack{b}%
  \begingroup
    \expandafter\let\csname\@currenvir\endcsname\l@dcollect@@body
    \edef\processl@denvbody{\expandafter\noexpand\csname\@currenvir\endcsname}%
    \processl@denvbody}

```

\l@dpush@begins When adding a piece of the current environment's contents to `\l@denvbody`, we scan it to check for additional `\begin` tokens, and add a 'b' to the stack for any that we find.

```
\def\l@dpush@begins#1\begin#2{%
  \ifx\end#2\else b\expandafter\l@dpush@begins\fi}

```

`\l@dcollect@@body` `\l@dcollect@@body` takes two arguments: the first will consist of all text up to the next `\end` command, and the second will be the `\end` command's argument. If therte are any extra `\begin` commands in the body text, a marker is pushed onto a stack by the `l@dpush@begins` function. Empty state for this stack means we have reached the `\end` that matches our original `\begin`. Otherwise we need to include the `\end` and its argument in the material we are adding to the environment body accumulator.

```
\def\l@dcollect@@body#1\end#2{%
  \edef\l@dbegin@stack{\l@dpush@begins#1\begin\end
                        \expandafter\@gobble\l@dbegin@stack}%
  \ifx\@empty\l@dbegin@stack
    \endgroup
    \@checkend{#2}%
    \addtol@denvbody{#1}%
  \else
    \addtol@denvbody{#1\end{#2}}%
  \fi
  \processl@denvbody % A little tricky! Note the grouping
}

```

There was a question on CTT about how to use `\collect@body` for a macro taking an argument. The following is part of that thread.

```
From: Heiko Oberdiek <oberdiek@uni-freiburg.de>
Newsgroups: comp.text.tex
Subject: Re: Using \collect@body with commands that take >1 argument
Date: Fri, 08 Aug 2003 09:03:20 +0200

eed132@psu.edu (Evan) wrote:
> I'm trying to make a new Latex environment that acts like the>
 \colorbox command that is part of the color package. I looked through
> the FAQ and ran across this bit about using the \collect@body command
> that is part of AMSLaTeX:
> http://www.tex.ac.uk/cgi-bin/texfaq2html?label=cmdasenv
>
> It almost works. If I do something like the following:
>   \newcommand{\redbox}[1]{\colorbox{red}{#1}}
>
>   \makeatletter
>   \newenvironment{redbox}{\collect@body \redbox}{}

You will get an error message: Command \redbox already defined.
Thus you must rename either the command \redbox or the environment
name.

>   \begin{coloredbox}{blue}
>     Yadda yadda yadda... this is on a blue background...
>   \end{coloredbox}
> and can't figure out how to make the \collect@body take this.

>   \collect@body \colorbox{red}
>   \collect@body {\colorbox{red}}

The argument of \collect@body has to be one token exactly.

\documentclass{article}
\usepackage{color}
\usepackage{amsmath}

\newcommand{\redbox}[1]{\colorbox{red}{#1}}
\makeatletter
\newenvironment{coloredbox}[1]{%
  \def\next@{\colorbox{#1}}%
  \collect@body\next@
}{}

% ignore spaces at begin and end of environment
\newenvironment{coloredboxII}[1]{%
  \def\next@{\mycoloredbox{#1}}%
  \collect@body\next@
}{}
\newcommand{\mycoloredbox}[2]{%
  \colorbox{#1}{\ignorespaces#2\unskip}%
}

% support of optional color model argument
\newcommand\coloredboxIII\endcsname{}
\def\coloredboxIII#1#{%
  \@coloredboxIII{#1}%
}
\def\@coloredboxIII#1#2{%
  \def\next@{\mycoloredboxIII{#1}{#2}}%
  \collect@body\next@
}
\newcommand{\mycoloredboxIII}[3]{%
  \colorbox#1{#2}{\ignorespaces#3\unskip}%
}

\makeatother

\begin{document}
  Black text before
  \begin{coloredbox}{blue}
   Hello World
  \end{coloredbox}
  Black text after

  Black text before
  \begin{coloredboxII}{blue}
   Hello World
  \end{coloredboxII}
  Black text after

  Black text before
  \begin{coloredboxIII}[rgb]{0,0,1}
   Hello World
  \end{coloredboxIII}
  Black text after

\end{document}

Yours sincerely
  Heiko <oberdiek@uni-freiburg.de>
```

33 Verse

This is principally Wayne Sullivan's code and commentary from EDSTANZA [Sul92].

The macro \hangingsymbol is used to insert a symbol on each hanging of verses. For example, in french typographie the symbol is '['. We obtain it by the next code:

```
\renewcommand{\hangingsymbol}{[\,}
```

The \ifinstanza boolean is used to be sure that we are in a stanza part.

\hangingsymbol
\ifinstanza

```
\newcommand*{\hangingsymbol}{}
\newif\ifinstanza
\instanzafalse
```

\inserthangingymbol
\ifinserthangingsymbol

The boolean \ifinserthangingsymbol is set to TRUE when \@lock is greater than 1, i.e. when we are not in the first line of a verse. The switch of \ifinserthangingsymbol is made in \do@line before the printing of line but after the line number calculation.

```
\newif\ifinserthangingsymbol
\newcommand{\inserthangingsymbol}{%
\ifinserthangingsymbol%
   \ifinstanza%
      \hfill\hangingsymbol%
   \fi%
\fi%
}
```

\ampersand

Within a stanza the \& macro is going to be usurped. We need an alias in case an & needs to be typeset in a stanza. Define it rather than letting it in case some other package has already defined it.

```
\newcommand*{\ampersand}{\char`\&}

```

`\stanza@count`
`\stanzaindentbase`

Before we can define the main macros we need to save and reset some category codes. To save the current values we use `\next` and `\body` from the `\loop` macro.

```
  \chardef\body=\catcode`\@
  \catcode`\@=11
  \chardef\next=\catcode`\&
  \catcode`\&=\active

```

A count register is allocated for counting lines in a stanza; also allocated is a dimension register which is used to specify the base value for line indentation; all stanza indentations are multiples of this value. The default value of `\stanzaindentbase` is 20pt.

```
  \newcount\stanza@count
  \newlength{\stanzaindentbase}
  \setlength{\stanzaindentbase}{20pt}

```

`\strip@szacnt`
`\setstanzavalues`

The indentations of stanza lines are non-negative integer multiples of the unit called `\stanzaindentbase`. To make it easier for the user to specify these numbers, some list macros are defined. These take numerical values in a list separated by commas and assign the values to special control sequences using `\mathchardef`. Though this does limit the range from 0 to 32767, it should suffice for most applications, including *penalties*, which will be discussed below.

```
\def\strip@szacnt#1,#2|{\def\@tempb{#1}\def\@tempa{#2|}}
\newcommand*{\setstanzavalues}[2]{\def\@tempa{#2,,|}%
       \stanza@count\z@
    \def\next{\expandafter\strip@szacnt\@tempa
          \ifx\@tempb\empty\let\next\relax\else
          \expandafter\mathchardef\csname #1@\number\stanza@count
          @\endcsname\@tempb\relax
          \advance\stanza@count\@ne\fi\next}%
     \next}

```

`\setstanzaindents`
`\setstanzapenalties`
`\managestanza@modulo`

In the original `\setstanzavalues{sza}{...}` had to be called to set the indents, and similarly `\setstanzavalues{szp}{...}` to set the penalties. These two macros are a convenience to give the user one less thing to worry about (misspelling the first argument). Since version 0.13, the `stanzaindentsrepetition` counter can be used when the indentation is repeated every n verses. The `\managestanza@modulo` is a command which modifies the counter stanza@modulo. The command adds 1 to stanza@modulo, but if stanza@modulo is equal to the stanzaindentsrepetition counter, the command restarts it.

```
\newcommand*{\setstanzaindents}[1]{\setstanzavalues{sza}{#1}}
\newcommand*{\setstanzapenalties}[1]{\setstanzavalues{szp}{#1}}

```

```
\newcounter{stanzaindentsrepetition}
\newcount\stanza@modulo

\newcommand*{\managestanza@modulo}[0]{
    \advance\stanza@modulo\@ne
    \ifnum\stanza@modulo>\value{stanzaindentsrepetition}
     \stanza@modulo\@ne
    \fi
}
```

\stanza@line
\stanza@hang
\sza@penalty

Now we arrive at the main works. `\stanza@line` sets the indentation for the line and starts a numbered paragraph—each line is treated as a paragraph. `\stanza@hang` sets the hanging indentation to be used if the stanza line requires more than one print line. If it is known that each stanza line will fit on one print line, it is advisable to set the hanging indentation to zero. `\sza@penalty` places the specified penalty following each stanza line. By default, this facility is turned off so that no penalty is included. However, the user may initiate these penalties to indicate good and bad places in the stanza for page breaking.

```
\def\stanza@line{
    \ifnum\value{stanzaindentsrepetition}=0
        \parindent=\csname sza@\number\stanza@count
               @\endcsname\stanzaindentbase
    \else
        \managestanza@modulo
        \parindent=\csname sza@\number\stanza@modulo
               @\endcsname\stanzaindentbase
    \fi
    \pstart\stanza@hang\ignorespaces}
\xdef\stanza@hang{\noexpand\leavevmode\noexpand\startlock
            \hangindent\expandafter
            \noexpand\csname sza@0@\endcsname\stanzaindentbase
            \hangafter\@ne}
\def\sza@penalty{\count@\csname szp@\number\stanza@count @\endcsname
         \ifnum\count@>\@M\advance\count@-\@M\penalty-\else
         \penalty\fi\count@}
```

\startstanzahook
\endstanzaextra
\stanza

Now we have the components of the `\stanza` macro, which appears at the start of a group of lines. This macro initializes the count and checks to see if hanging indentation and penalties are to be included. Hanging indentation suspends the line count, so that the enumeration is by verse line rather than by print line. If the print line count is desired, invoke `\let\startlock=\relax` and do the same for `\endlock`. Here and above we have used `\xdef` to make the stored macros take up a bit less space, but it also makes them more obscure to the reader. Lines of the stanza are delimited by ampersands &. The last line of the stanza must end with `\&`. For convenience the macro `\endstanzaextra` is incuded. The user may use this to add vertical space or penalties between stanzas.

As a further convenience, the macro `\startstanzahook` is called at the beginning of a stanza. This can be defined to do something useful.

```
\let\startstanzahook\relax
\let\endstanzaextra\relax
\xdef\stanza{\noexpand\instanzatrue\expandafter
        \begingroup\startstanzahook%
        \catcode`\&\active\global\stanza@count\@ne\stanza@modulo\@ne
        \noexpand\ifnum\expandafter\noexpand
        \csname sza@0@\endcsname=\z@\let\noexpand\stanza@hang\relax
        \let\noexpand\endlock\relax\noexpand\else\interlinepenalty
        \@M\rightskip\z@ plus 1fil\relax\noexpand\fi\noexpand\ifnum
        \expandafter\noexpand\csname szp@0@\endcsname=\z@
        \let\noexpand\sza@penalty\relax\noexpand\fi \def\noexpand&{%
        \noexpand\endlock\noexpand\pend\noexpand\sza@penalty\global
        \advance\stanza@count\@ne\noexpand\stanza@line}\def\noexpand
        \&{\noexpand\endlock\noexpand\pend\endgroup\noexpand\instanzafalse\expandafter\en
        \noexpand\stanza@line}

```

`\flagstanza` Use `\flagstanza[len]{text}` at the start of a line to put `text` a distance `len` before the start of the line. The default for `len` is `\stanzaindentbase`.

```
\newcommand*{\flagstanza}[2][\stanzaindentbase]{%
  \hskip -#1\llap{#2}\hskip #1\ignorespaces}

```

The ampersand & is used to mark the end of each stanza line, except the last, which is marked with `\&`. This means that `\halign` may not be used directly within a stanza line. This does not affect macros involving alignments defined outside `\stanza  \&`. Since these macros usurp the control sequence `\&`, the replacement `\ampersand` is defined to be used if this symbol is needed in a stanza. Also we reset the modified category codes and initialize the penalty default.

```
  \catcode`\&=\next
  \catcode`\@=\body
%%  \let\ampersand=\&
  \setstanzavalues{szp}{0}

```

34 Arrays and tables

This is based on the work by Herbert Breger in developing `tabmac.tex`.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This is file tabmac.tex 1.0.
% You find here macros for tabular structures compatible with
% Edmac (authored by Lavagnino/Wujastyk). The use of the macros is
% explained in German language in file tabanlei.dvi. The macros were
% developed for Edmac 2.3, but this file has been adjusted to Edmac 3.16.
%
% ATTENTION: This file uses some Edmac control sequences (like
% \text, \Afootnote etc.) and redefines \morenoexpands. If you yourself
% redefined some Edmac control sequences, be careful: some adjustements
% might be necessary.
% October 1996
%
% My kind thanks to Nora G^?deke for valuable support. Any hints and
% comments are welcome, please contact Herbert Breger,
% Leibniz-Archiv, Waterloostr. 8, D -- 30169 Hannover, Germany
% Tel.: 511 - 1267 327
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

The original `tabmac.tex` file was void of comments or any explanatory text other than the above notice. The algorithm is Breger's. I have made some cosmetic changes to the original code and reimplemented some things so they are more LaTeX-like. All the commentary is mine, as are any mistakes or errors.

`\l@dtabnoexpands` An extended and modified version of the original additional no expansions..

```
\newcommand*{\l@dtabnoexpands}{%
  \let\rtab=0%
  \let\ctab=0%
  \let\ltab=0%
  \let\rtabtext=0%
  \let\ltabtext=0%
  \let\ctabtext=0%
  \let\edbeforetab=0%
  \let\edaftertab=0%
  \let\edatab=0%
  \let\edatabell=0%
  \let\edatleft=0%
  \let\edatright=0%
  \let\edvertline=0%
  \let\edvertdots=0%
  \let\edrowfill=0%
}

```

`\l@dampcount` `\l@dcolcount` `\l@dampcount` is a counter for the & column dividers and `\l@dcolcount` is a counter for the columns. These were `\Undcount` and `\stellencount` respectively.

```
\newcount\l@dampcount
  \l@dampcount=1\relax
\newcount\l@dcolcount
  \l@dcolcount=0\relax

```

`\hilfsbox` `\hilfsskip` `\Hilfsbox` `\hilfscount` Some (temporary) helper items.

```
\newbox\hilfsbox
\newskip\hilfsskip
\newbox\Hilfsbox
```

```
\newcount\hilfscount

```

30 columns should be adequate (compared to the original 60). These are the column widths. (Originally these were German spelled numbers e.g., `\eins`, `\zwei`, etc).

```
\newdimen\dcoli
\newdimen\dcolii
\newdimen\dcoliii
\newdimen\dcoliv
\newdimen\dcolv
\newdimen\dcolvi
\newdimen\dcolvii
\newdimen\dcolviii
\newdimen\dcolix
\newdimen\dcolx
\newdimen\dcolxi
\newdimen\dcolxii
\newdimen\dcolxiii
\newdimen\dcolxiv
\newdimen\dcolxv
\newdimen\dcolxvi
\newdimen\dcolxvii
\newdimen\dcolxviii
\newdimen\dcolxix
\newdimen\dcolxx
\newdimen\dcolxxi
\newdimen\dcolxxii
\newdimen\dcolxxiii
\newdimen\dcolxxiv
\newdimen\dcolxxv
\newdimen\dcolxxvi
\newdimen\dcolxxvii
\newdimen\dcolxxviii
\newdimen\dcolxxix
\newdimen\dcolxxx
\newdimen\dcolerr   % added for error handling

```

`\l@dcolwidth` This is a cunning way of storing the columnwidths indexed by the column number `\l@dcolcount`, like an array. (was `\Dimenzuordnung`)

```
\newcommand{\l@dcolwidth}{\ifcase \the\l@dcolcount \dcoli %???
  \or \dcoli \or \dcolii \or \dcoliii
  \or \dcoliv \or \dcolv \or \dcolvi
  \or \dcolvii \or \dcolviii \or \dcolix \or \dcolx
  \or \dcolxi \or \dcolxii \or \dcolxiii
  \or \dcolxiv \or \dcolxv \or \dcolxvi
  \or \dcolxvii \or \dcolxviii \or \dcolxix \or \dcolxx
  \or \dcolxxi \or \dcolxxii \or \dcolxxiii
  \or \dcolxxiv \or \dcolxxv \or \dcolxxvi
  \or \dcolxxvii \or \dcolxxviii \or \dcolxxix \or \dcolxxx
  \else \dcolerr \fi}

```

\stepl@dcolcount This increments the column counter, and issues an error message if it is too large.

```
\newcommand*{\stepl@dcolcount}{\advance\l@dcolcount\@ne
  \ifnum\l@dcolcount>30\relax
    \led@err@TooManyColumns
  \fi}

```

\l@dsetmaxcolwidth Sets the column width to the maximum value seen so far. (was \dimenzuordnung)

```
\newcommand{\l@dsetmaxcolwidth}{%
  \ifdim\l@dcolwidth < \wd\hilfsbox
    \l@dcolwidth = \wd\hilfsbox
  \else \relax \fi}

```

\EDTEXT \xedtext \CRITEXT \xcritext We need to be able to modify the \edtext and \critext macros and also restore their original definitions.

```
\let\EDTEXT=\edtext
\newcommand{\xedtext}[2]{\EDTEXT{#1}{#2}}
\let\CRITEXT=\critext
\long\def\xcritext #1#2/{\CRITEXT{#1}{#2}/}
```

\EDLABEL \xedlabel We need to be able to modify and restore the \edlabel macro.

```
\let\EDLABEL=\edlabel
\newcommand*{\xedlabel}[1]{\EDLABEL{#1}}
```

\EDINDEX \xedindex \nulledindex Macros supporting modification and restoration of \edindex.

```
\let\EDINDEX=\edindex
\ifl@dmemoir
  \newcommand{\xedindex}{\@bsphack%
      \@ifnextchar [{\l@d@index}{\l@d@index[\jobname]}}
  \newcommand{\nulledindex}[2][\jobname]{\@bsphack\@esphack}
\else
  \newcommand{\xedindex}{\@bsphack%
    \doedindexlabel
    \begingroup
    \@sanitize
    \@wredindex}
  \newcommand{\nulledindex}[1]{\@bsphack\@esphack}
\fi

```

\@line@@num Macro supporting restoration of \linenum.

```
\let\@line@@num=\linenum
```

\l@dgobbledarg \l@dgobbledarg replaces its delineated argument by `\relax` (was `\verschwinden`).

\l@dgobblearg `\l@dgobblearg{`⟨*arg*⟩`}` replaces its argument by `\relax`.

```
\def\l@dgobbledarg #1/{\relax}
\newcommand*{\l@dgobblearg}[1]{\relax}

```

\Relax \NEXT \@hilfs@count

```
\let\Relax=\relax
\let\NEXT=\next
\newcount\@hilfs@count

```

\measuremcell Measure (recursively) the width required for a math cell. (was `\messen`)

```
\def\measuremcell #1&{%
    \ifx #1\\ \ifnum\l@dcolcount=0\let\NEXT\relax%
              \else\l@dcheckcols%
                    \l@dcolcount=0%
                    \let\NEXT\measuremcell%
              \fi%
    \else\setbox\hilfsbox=\hbox{$\displaystyle{#1}$}%
      \stepl@dcolcount%
      \l@dsetmaxcolwidth%
      \let\NEXT\measuremcell%
    \fi\NEXT}

```

\measuretcell Measure (recursively) the width required for a text cell. (was `\messentext`)

```
\def\measuretcell #1&{%
    \ifx #1\\ \ifnum\l@dcolcount=0\let\NEXT\relax%
              \else\l@dcheckcols%
                    \l@dcolcount=0%
                    \let\NEXT\measuretcell%
              \fi%
    \else\setbox\hilfsbox=\hbox{#1}%
      \stepl@dcolcount%
      \l@dsetmaxcolwidth%
      \let\NEXT\measuretcell%
    \fi\NEXT}

```

\measuremrow Measure (recursively) the width required for a math row. (was `\Messen`)

```
\def\measuremrow #1\\{%
   \ifx #1&\let\NEXT\relax%
   \else\measuremcell #1&\\&\\&%
      \let\NEXT\measuremrow%
   \fi\NEXT}
```

\measuretrow Measure (recursively) the width required for a text row. (was `\Messentext`)

```
\def\measuretrow #1\\{%
```

```
  \ifx #1&\let\NEXT\relax%
  \else\measuretcell #1&\\&\\&%
    \let\NEXT\measuretrow%
  \fi\NEXT}

```

\edtabcolsep The length \edtabcolsep controls the distance between columns. (was \abstand)

```
\newskip\edtabcolsep
\global\edtabcolsep=10pt

```

\NEXT
\Next

```
\let\NEXT\relax
\let\Next=\next
```

\variab

```
\newcommand{\variab}{\relax}

```

\l@dcheckcols Check that the number of columns is consistent. (was \tabfehlermeldung)

```
\newcommand*{\l@dcheckcols}{%
  \ifnum\l@dcolcount=1\relax
  \else
    \ifnum\l@dampcount=1\relax
    \else
      \ifnum\l@dcolcount=\l@dampcount\relax
      \else
        \l@d@err@UnequalColumns
      \fi
    \fi
    \l@dampcount=\l@dcolcount
  \fi}

```

\l@dmodforcritext
\l@drestoreforcritext Modify and restore various macros for when \critext is used.

```
\newcommand{\l@dmodforcritext}{%
  \let\critext\relax%
  \def\do##1{\global\csletcs{##1footnote}{l@dgobbledarg}}
  \dolistloop{\@series}%
  \let\edindex\nulledindex%
  \let\linenum\@gobble}
\newcommand{\l@drestoreforcritext}{%
  \def\do##1{\csdef{##1footnote}##1##2/{\csuse{##1@@footnote}{##1}{##2}}}
  \dolistloop{\@series}%
  \let\edindex\xedindex}

```

\l@dmodforedtext
\l@drestoreforedtext Modify and restore various macros for when \edtext is used.

```
\newcommand{\l@dmodforedtext}{%
```

```
  \let\edtext\relax
  \def\do##1{\global\csletcs{##1footnote}{l@dgobblearg}}
  \dolistloop{\@series}%
  \let\edindex\nulledindex
  \let\linenum\@gobble}
\newcommand{\l@drestoreforedtext}{%
  \def\do##1{\csgdef{##1footnote}##1{\csuse{##1@@footnote}{##1}}}
  \dolistloop{\@series}%
  \let\edindex\xedindex}
```

`\l@dnullfills` `\l@drestorefills` Nullify and restore some column fillers, etc.

```
\newcommand{\l@dnullfills}{%
  \def\edlabel##1{}%
  \def\edrowfill##1##2##3{}%
}
\newcommand{\l@drestorefills}{%
  \def\edrowfill##1##2##3{\@EDROWFILL@{##1}{##2}{##3}}%
}

```

The original definition of `\rverteilen` and friends (‘verteilen’ is approximately ‘distribute’) was along the lines:

```
\def\rverteilen #1&{\def\label##1{}%
     \ifx #1! \ifnum\l@dcolcount=0%\removelastskip
                   \let\Next\relax%
              \else\l@dcolcount=0%
                    \let\Next=\rverteilen%
              \fi%
     \else%
         \footnoteverschw%
         \stepl@dcolcount%
         \setbox\hilfsbox=\hbox{$\displaystyle{#1}$}%
         \let\critext=\xcritext\let\Dfootnote=\D@@footnote
         \let\Afootnote=\A@@footnote\let\Bfootnote=\B@@footnote
         \let\Cfootnote=\C@@footnote\let\linenum=\@line@@num%
         \hilfsskip=\Dimenzuordnung%
         \advance\hilfsskip by -\wd\hilfsbox
         \def\label##1{\xlabel{##1}}%
         \hskip\hilfsskip$\displaystyle{#1}$%
         \hskip\edtabcolsep%
         \let\Next=\rverteilen%
     \fi\Next}
```

where the lines

```
         \let\critext=\xcritext\let\Dfootnote=\D@@footnote
         \let\Afootnote=\A@@footnote\let\Bfootnote=\B@@footnote
         \let\Cfootnote=\C@@footnote\let\linenum=\@line@@num%
         \hilfsskip=\Dimenzuordnung%
```

```
          \advance\hilfsskip by -\wd\hilfsbox
          \def\label##1{\xlabel{##1}}%
```

were common across the several `*verteilen*` macros, and also

```
 \def\footnoteverschw{%
   \let\critext\relax
   \let\Afootnote=\verschwinden
   \let\Bfootnote=\verschwinden
   \let\Cfootnote=\verschwinden
   \let\Dfootnote=\verschwinden
   \let\linenum=\@gobble}
```

`\letsforverteilen` Gathers some lets and other code that is common to the `*verteilen*` macros.

```
\newcommand{\letsforverteilen}{%
  \let\critext\xcritext
  \let\edtext\xedtext
  \let\edindex\xedindex
  \def\do##1{\global\csletcs{##1footnote}{##1@@footnote}}
  \dolistloop{\@series}%
  \let\linenum\@line@@num
  \hilfsskip=\l@dcolwidth%
  \advance\hilfsskip by -\wd\hilfsbox
  \def\edlabel##1{\xedlabel{##1}}}

```

`\setmcellright` Typeset (recursively) cells of display math right justified. (was `\rverteilen`)

```
\def\setmcellright #1&{\def\edlabel##1{}%
                   \let\edindex\nulledindex
       \ifx #1\\ \ifnum\l@dcolcount=0%\removelastskip
                    \let\Next\relax%
                \else\l@dcolcount=0%
                     \let\Next=\setmcellright%
                \fi%
       \else%
           \disablel@dtabfeet%
           \stepl@dcolcount%
           \setbox\hilfsbox=\hbox{$\displaystyle{#1}$}%
           \letsforverteilen%
           \hskip\hilfsskip$\displaystyle{#1}$%
           \hskip\edtabcolsep%
           \let\Next=\setmcellright%
       \fi\Next}

```

`\settcellright` Typeset (recursively) cells of text right justified. (was `\rverteilentext`)

```
\def\settcellright #1&{\def\edlabel##1{}%
                        \let\edindex\nulledindex
```

```
      \ifx #1\\ \ifnum\l@dcolcount=0%\removelastskip
                    \let\Next\relax%
                \else\l@dcolcount=0%
                     \let\Next=\settcellright%
                \fi%
      \else%
          \disablel@dtabfeet%
          \stepl@dcolcount%
          \setbox\hilfsbox=\hbox{#1}%
          \letsforverteilen%
          \hskip\hilfsskip#1%
          \hskip\edtabcolsep%
          \let\Next=\settcellright%
      \fi\Next}
```

\setmcellleft Typeset (recursively) cells of display math left justified. (was \lverteilen)

```
\def\setmcellleft #1&{\def\edlabel##1{}%
                       \let\edindex\nulledindex
   \ifx #1\\ \ifnum\l@dcolcount=0 \let\Next\relax%
             \else\l@dcolcount=0%
                  \let\Next=\setmcellleft%
             \fi%
    \else   \disablel@dtabfeet%
             \stepl@dcolcount%
             \setbox\hilfsbox=\hbox{$\displaystyle{#1}$}%
             \letsforverteilen
             $\displaystyle{#1}$\hskip\hilfsskip\hskip\edtabcolsep%
             \let\Next=\setmcellleft%
    \fi\Next}

```

\settcellleft Typeset (recursively) cells of text left justified. (was \lverteilentext)

```
\def\settcellleft #1&{\def\edlabel##1{}%
                       \let\edindex\nulledindex
   \ifx #1\\ \ifnum\l@dcolcount=0 \let\Next\relax%
             \else\l@dcolcount=0%
                  \let\Next=\settcellleft%
             \fi%
    \else   \disablel@dtabfeet%
             \stepl@dcolcount%
             \setbox\hilfsbox=\hbox{#1}%
             \letsforverteilen
             #1\hskip\hilfsskip\hskip\edtabcolsep%
             \let\Next=\settcellleft%
    \fi\Next}
```

\setmcellcenter Typeset (recursively) cells of display math centered. (was \zverteilen)

```
\def\setmcellcenter #1&{\def\edlabel##1{}%
                       \let\edindex\nulledindex
```

```
    \ifx #1\\ \ifnum\l@dcolcount=0\let\Next\relax%
             \else\l@dcolcount=0%
                   \let\Next=\setmcellcenter%
             \fi%
    \else    \disablel@dtabfeet%
             \stepl@dcolcount%
             \setbox\hilfsbox=\hbox{$\displaystyle{#1}$}%
             \letsforverteilen%
             \hskip 0.5\hilfsskip$\displaystyle{#1}$\hskip0.5\hilfsskip%
             \hskip\edtabcolsep%
             \let\Next=\setmcellcenter%
     \fi\Next}

```

\settcellcenter Typeset (recursively) cells of text centered. (new)

```
\def\settcellcenter #1&{\def\edlabel##1{}%
                        \let\edindex\nulledindex
    \ifx #1\\ \ifnum\l@dcolcount=0 \let\Next\relax%
              \else\l@dcolcount=0%
                    \let\Next=\settcellcenter%
              \fi%
     \else   \disablel@dtabfeet%
              \stepl@dcolcount%
              \setbox\hilfsbox=\hbox{#1}%
              \letsforverteilen%
              \hskip 0.5\hilfsskip #1\hskip 0.5\hilfsskip%
              \hskip\edtabcolsep%
              \let\Next=\settcellcenter%
     \fi\Next}

```

\NEXT

```
\let\NEXT=\relax

```

\setmrowright Typeset (recursively) rows of right justified math. (was \rsetzen)

```
\def\setmrowright #1\\{%
    \ifx #1& \let\NEXT\relax
    \else \centerline{\setmcellright #1&\\&\\&}
          \let\NEXT=\setmrowright
    \fi\NEXT}
```

\settrowright Typeset (recursively) rows of right justified text. (was \rsetzentext)

```
\def\settrowright #1\\{%
    \ifx #1& \let\NEXT\relax
    \else \centerline{\settcellright #1&\\&\\&}
          \let\NEXT=\settrowright
    \fi\NEXT}

```

\setmrowleft Typeset (recursively) rows of left justified math. (was `\lsetzen`)

```
\def\setmrowleft #1\\{%
    \ifx #1&\let\NEXT\relax
    \else \centerline{\setmcellleft #1&\\&\\&}
          \let\NEXT=\setmrowleft
    \fi\NEXT}
```

\settrowleft Typeset (recursively) rows of left justified text. (was `\lsetzentext`)

```
\def\settrowleft #1\\{%
    \ifx #1& \let\NEXT\relax
    \else \centerline{\settcellleft #1&\\&\\&}
          \let\NEXT=\settrowleft
    \fi\NEXT}

```

\setmrowcenter Typeset (recursively) rows of centered math. (was `\zsetzen`)

```
\def\setmrowcenter #1\\{%
    \ifx #1& \let\NEXT\relax%
    \else \centerline{\setmcellcenter #1&\\&\\&}
          \let\NEXT=\setmrowcenter
     \fi\NEXT}
```

\settrowcenter Typeset (recursively) rows of centered text. (new)

```
\def\settrowcenter #1\\{%
    \ifx #1& \let\NEXT\relax
    \else \centerline{\settcellcenter #1&\\&\\&}
          \let\NEXT=\settrowcenter
    \fi\NEXT}

```

\nullsetzen (was `\nullsetzen`)

```
\newcommand{\nullsetzen}{%
    \stepl@dcolcount%
    \l@dcolwidth=0pt%
    \ifnum\l@dcolcount=30\let\NEXT\relax%
          \l@dcolcount=0\relax
    \else\let\NEXT\nullsetzen%
    \fi\NEXT}

```

\edatleft `\edatleft[`⟨*math*⟩`]{`⟨*symbol*⟩`}{`⟨*len*⟩`}` (combination and generalisation of original `\Seklam` and `\Seklamgl`). Left ⟨*symbol*⟩, 2⟨*len*⟩ high with prepended ⟨*math*⟩ vertically centered.

```
\newcommand{\edatleft}[3][\@empty]{%
  \ifx#1\@empty
    \vbox to 10pt{\vss\hbox{$\left#2\vrule width0pt height #3
                            depth 0pt \right. $\hss}\vfil}
  \else
    \vbox to 4pt{\vss\hbox{$#1\left#2\vrule width0pt height #3
```

```
                 depth 0pt \right. $}\vfil}
  \fi}
```

`\edatright` `\edatright[`⟨*math*⟩`]{`⟨*symbol*⟩`}{`⟨*len*⟩`}` (combination and generalisation of original `\seklam` and `\seklamgl`). Right ⟨*symbol*⟩, 2⟨*len*⟩ high with appended ⟨*math*⟩ vertically centered.

```
\newcommand{\edatright}[3][\@empty]{%
  \ifx#1\@empty
    \vbox to 10pt{\vss\hbox{$\left.\vrule width0pt height #3
                 depth 0pt \right#2 $\hss}\vfil}
  \else
    \vbox to 4pt{\vss\hbox{$\left.\vrule width0pt height #3
                 depth 0pt \right#2 #1 $}\vfil}
  \fi}

```

`\edvertline` `\edvertline{`⟨*len*⟩`}` vertical line ⟨*len*⟩ high. (was `\sestrich`)

```
\newcommand{\edvertline}[1]{\vbox to 8pt{\vss\hbox{\vrule height #1}\vfil}}

```

`\edvertdots` `\edvertdots{`⟨*len*⟩`}` vertical dotted line ⟨*len*⟩ high. (was `\sepunkte`)

```
\newcommand{\edvertdots}[1]{\vbox to 1pt{\vss\vbox to #1%
         {\cleaders\hbox{$\m@th\hbox{.}\vbox to 0.5em{ }$}\vfil}}}

```

I don't know if this is relevant here, and I haven't tried it, but the following appeared on CTT.

```
From: mdw@nsict.org (Mark Wooding)
Newsgroups: comp.text.tex
Subject: Re: Dotted line
Date: 13 Aug 2003 13:51:14 GMT

Alexis Eisenhofer <alexis@eisenhofer.de> wrote:
> Can anyone provide me with the LaTex command for a vertical dotted line?

How dotted? Here's the basic rune.
  \newbox\linedotbox
  \setbox\linedotbox=\vbox{...}
  \leaders\copy\linedotbox\vskip2in

For just dots, this works:
  \setbox\linedotbox=\vbox{\hbox{\normalfont.}\kern2pt}

For dashes, something like
  \setbox\linedotbox=\vbox{\leaders\vrule\vskip2pt\vskip2pt}
is what you want.  (Adjust the `2pt' values to taste.  The first one is
the length of the dashes, the second is the length of the gaps.)

For dots in mid-paragraph, you need to say something like
```

```
  \lower10pt\vbox{\leaders\copy\linedotbox\vskip2in}
which is scungy but works.

-- [mdw]
```

`\edfilldimen` A length. (was `\klamdimen`)

```
\newdimen\edfilldimen
\edfilldimen=0pt

```

`\c@addcolcount` `\theaddcolcount` A counter to hold the number of a column. We use a roman number so that we can grab the column dimension from `\dcol...`.

```
\newcounter{addcolcount}
  \renewcommand{\theaddcolcount}{\roman{addcolcount}}
```

`\l@dtabaddcols` `\l@dtabaddcols{`⟨*startcol*⟩`}{`⟨*endcol*⟩`}` adds the widths of the columns ⟨*startcol*⟩ through ⟨*endcol*⟩ to `\edfilldimen`. It is a LaTeX style reimplementation of the original `\@add@`.

```
\newcommand{\l@dtabaddcols}[2]{%
  \l@dcheckstartend{#1}{#2}%
  \ifl@dstartendok
  \setcounter{addcolcount}{#1}%
  \@whilenum \value{addcolcount}<#2\relax \do
  {\advance\edfilldimen by \the \csname dcol\theaddcolcount\endcsname
   \advance\edfilldimen by \edtabcolsep
   \stepcounter{addcolcount}}%
  \advance\edfilldimen by \the \csname dcol\theaddcolcount\endcsname
  \fi
}

```

`\ifl@dstartendok` `\l@dcheckstartend` `\l@dcheckstartend{`⟨*startcol*⟩`}{`⟨*endcol*⟩`}` checks that the values of ⟨*startcol*⟩ and ⟨*endcol*⟩ are sensible. If they are then `\ifl@dstartendok` is set TRUE, otherwise it is set FALSE.

```
\newif\ifl@dstartendok
\newcommand{\l@dcheckstartend}[2]{%
  \l@dstartendoktrue
  \ifnum #1<\@ne
    \l@dstartendokfalse
    \led@err@LowStartColumn
  \fi
  \ifnum #2>30\relax
    \l@dstartendokfalse
    \led@err@HighEndColumn
  \fi
  \ifnum #1>#2\relax
    \l@dstartendokfalse
    \led@err@ReverseColumns
%%      \eledmac@error{Start column is greater than end column}{\@ehc}%
  \fi
}

```

\edrowfill \@edrowfill@ \@EDROWFILL@ `\edrowfill{`⟨*startcol*⟩`}{`⟨*endcol*⟩`}`fill fills columns ⟨*startcol*⟩ to ⟨*endcol*⟩ inclusive with ⟨*fill*⟩ (e.g. `\hrulefill`, `\upbracefill`). This is a LaTex style reimplementation and generalization of the original `\waklam`, `\Waklam`, `\waklamec`, `\wastricht` and `\wapunktel` macros.

```
\newcommand*{\edrowfill}[3]{%
  \l@dtabaddcols{#1}{#2}%
  \hb@xt@ \the\l@dcolwidth{\hb@xt@ \the\edfilldimen{#3}\hss}}
\let\@edrowfill@=\edrowfill
\def\@EDROWFILL@#1#2#3{\@edrowfill@{#1}{#2}{#3}}

```

\edbeforetab \edaftertab The macro `\edbeforetab{`⟨*text*⟩`}{`⟨*math*⟩`}` puts ⟨*text*⟩ at the left margin before array cell entry ⟨*math*⟩. Conversely, the macro `\edaftertab{`⟨*math*⟩`}{`⟨*text*⟩`}` puts ⟨*text*⟩ at the right margin after array cell entry ⟨*math*⟩. `\edbeforetab` should be in the first column and `\edaftertab` in the last column. The following macros support these.

\leftltab `\leftltab{`⟨*text*⟩`}` for `\edbeforetab` in `\ltab`. (was `\linksltab`)

```
\newcommand{\leftltab}[1]{%
  \hb@xt@\z@{\vbox{\edtabindent%
  \moveleft\Hilfsskip\hbox{\ #1}}\hss}}

```

\leftrtab `\leftrtab{`⟨*text*⟩`}{`⟨*math*⟩`}` for `\edbeforetab` in `\rtab`. (was `\linksrtab`)

```
\newcommand{\leftrtab}[2]{%
  #2\hb@xt@\z@{\vbox{\edtabindent%
    \advance\Hilfsskip by\dcoli%
    \moveleft\Hilfsskip\hbox{\ #1}}\hss}}

```

\leftctab `\leftctab{`⟨*text*⟩`}{`⟨*math*⟩`}` for `\edbeforetab` in `\ctab`. (was `\linksztab`)

```
\newcommand{\leftctab}[2]{%
        \hb@xt@\z@{\vbox{\edtabindent\l@dcolcount=\l@dampcount%
        \advance\Hilfsskip by 0.5\dcoli%
        \setbox\hilfsbox=\hbox{\def\edlabel##1{}%
        \disablel@dtabfeet$\displaystyle{#2}$}%
        \advance\Hilfsskip by -0.5\wd\hilfsbox%
        \moveleft\Hilfsskip\hbox{\ #1}}\hss}%
        #2}

```

\rightctab `\rightctab{`⟨*math*⟩`}{`⟨*text*⟩`}` for `\edaftertab` in `\ctab`. (was `\rechtsztab`)

```
\newcommand{\rightctab}[2]{%
        \setbox\hilfsbox=\hbox{\def\edlabel##1{}%
        \disablel@dtabfeet#2}\l@dampcount=\l@dcolcount%
        #1\hb@xt@\z@{\vbox{\edtabindent\l@dcolcount=\l@dampcount%
        \advance\Hilfsskip by 0.5\l@dcolwidth%
        \advance\Hilfsskip by -\wd\hilfsbox%
        \setbox\hilfsbox=\hbox{\def\edlabel##1{}%
        \disablel@dtabfeet$\displaystyle{#1}$}%
        \advance\Hilfsskip by -0.5\wd\hilfsbox%
        \advance\Hilfsskip by \edtabcolsep%
        \moveright\Hilfsskip\hbox{ #2}}\hss}%
        }

```

`\rightltab` `\rightltab{`⟨*math*⟩`}{`⟨*text*⟩`}` for `\edaftertab` in `\ltab`. (was `\rechtsltab`)

```
\newcommand{\rightltab}[2]{%
        \setbox\hilfsbox=\hbox{\def\edlabel##1{}%
        \disablel@dtabfeet#2}\l@dampcount=\l@dcolcount%
        #1\hb@xt@\z@{\vbox{\edtabindent\l@dcolcount=\l@dampcount%
        \advance\Hilfsskip by\l@dcolwidth%
        \advance\Hilfsskip by-\wd\hilfsbox%
        \setbox\hilfsbox=\hbox{\def\edlabel##1{}%
        \disablel@dtabfeet$\displaystyle{#1}$}%
        \advance\Hilfsskip by-\wd\hilfsbox%
        \advance\Hilfsskip by\edtabcolsep%
        \moveright\Hilfsskip\hbox{ #2}}\hss}%
        }

```

`\rightrtab` `\rightrtab{`⟨*math*⟩`}{`⟨*text*⟩`}` for `\edaftertab` in `\rtab`. (was `\rechtsrtab`)

```
\newcommand{\rightrtab}[2]{%
        \setbox\hilfsbox=\hbox{\def\edlabel##1{}%
        \disablel@dtabfeet#2}%
        #1\hb@xt@\z@{\vbox{\edtabindent%
        \advance\Hilfsskip by-\wd\hilfsbox%
        \advance\Hilfsskip by\edtabcolsep%
        \moveright\Hilfsskip\hbox{ #2}}\hss}%
        }

```

`\rtab` `\edbeforetab` `\edaftertab` `\rtab{`⟨*body*⟩`}` typesets ⟨*body*⟩ as an array with the entries right justified. (was `\rtab`) (Here and elsewhere, `\edbeforetab` and `\edaftertab` were originally `\davor` and `\danach`) The original `\rtab` and friends included a fair bit of common code which I have extracted into macros.

The process is first to measure the ⟨*body*⟩ to get the column widths, and then in a second pass to typeset the body.

```
\newcommand{\rtab}[1]{%
  \l@dnullfills
    \def\edbeforetab##1##2{\leftrtab{##1}{##2}}%
    \def\edaftertab##1##2{\rightrtab{##1}{##2}}%
    \measurembody{#1}%
  \l@drestorefills
    \variab
    \setmrowright #1\\&\\%
    \enablel@dtabfeet}

```

\measurembody

\measurembody{⟨*body*⟩} measures the array ⟨*body*⟩.

```
\newcommand{\measurembody}[1]{%
  \disablel@dtabfeet%
  \l@dcolcount=0%
  \nullsetzen%
  \l@dcolcount=0
  \measuremrow #1\\&\\%
  \global\l@dampcount=1}

```

\rtabtext

\rtabtext{⟨*body*⟩} typesets ⟨*body*⟩ as a tabular with the entries right justified. (was \rtabtext)

```
\newcommand{\rtabtext}[1]{%
 \l@dnullfills
    \measuretbody{#1}%
 \l@drestorefills
    \variab
    \settrowright #1\\&\\%
    \enablel@dtabfeet}

```

\measuretbody

\measuretbody{⟨*body*⟩} measures the tabular ⟨*body*⟩.

```
\newcommand{\measuretbody}[1]{%
  \disablel@dtabfeet%
  \l@dcolcount=0%
  \nullsetzen%
  \l@dcolcount=0
  \measuretrow #1\\&\\%
  \global\l@dampcount=1}

```

\ltab

\edbeforetab

\edaftertab

Array with entries left justified. (was \ltab)

```
\newcommand{\ltab}[1]{%
 \l@dnullfills
    \def\edbeforetab##1##2{\leftltab{##1}{##2}}%
    \def\edaftertab##1##2{\rightltab{##1}{##2}}%
    \measurembody{#1}%
  \l@drestorefills
    \variab
    \setmrowleft #1\\&\\%
    \enablel@dtabfeet}

```

\ltabtext Tabular with entries left justified. (was \ltabtext)

```
\newcommand{\ltabtext}[1]{%
  \l@dnullfills
    \measuretbody{#1}%
  \l@drestorefills
    \variab
    \settrowleft #1\\&\\%
    \enablel@dtabfeet}

```

\ctab \edbeforetab \edaftertab Array with centered entries. (was \ztab)

```
\newcommand{\ctab}[1]{%
  \l@dnullfills
    \def\edbeforetab##1##2{\leftctab{##1}{##2}}%
    \def\edaftertab##1##2{\rightctab{##1}{##2}}%
    \measurembody{#1}%
  \l@drestorefills
    \variab
    \setmrowcenter #1\\&\\%
    \enablel@dtabfeet}

```

\ctabtext Tabular with entries centered. (new)

```
\newcommand{\ctabtext}[1]{%
  \l@dnullfills
    \measuretbody{#1}%
  \l@drestorefills
    \variab
    \settrowcenter #1\\&\\%
    \enablel@dtabfeet}

```

\spreadtext (was \breitertext)

```
\newcommand{\spreadtext}[1]{%\l@dcolcount=\l@dampcount%
   \hb@xt@ \the\l@dcolwidth{\hbox{#1}\hss}}
```

\spreadmath (was \breiter, 'breiter' = 'broadly')

```
\newcommand{\spreadmath}[1]{%
  \hb@xt@ \the\l@dcolwidth{\hbox{$\displaystyle{#1}$}\hss}}

```

I have left the remaining TABMAC alone, apart from changing some names. I'm not yet sure what they do or how they do it. Authors should not use any of these as they are likely to be mutable.

\tabellzwischen (was \tabellzwischen)

```
\def\tabellzwischen #1&{%
     \ifx #1\\ \let\NEXT\relax \l@dcolcount=0
     \else   \stepl@dcolcount%
```

```
              \l@dcolwidth = #1 mm
              \let\NEXT=\tabellzwischen
      \fi \NEXT }

```

`\edatabell` For example `\edatabell 4 & 19 & 8 \\` specifies 3 columns with widths of 4, 19, and 8mm. (was `\atabell`)

```
\def\edatabell #1\\{%
    \tabellzwischen #1&\\&}
```

`\Setzen` (was `\Setzen`, ‘setzen’ = ‘set’)

```
\def\Setzen #1&{%
    \ifx #1\relax \let\NEXT=\relax
    \else \stepl@dcolcount%
          \let\tabelskip=\l@dcolwidth
          \EDTAB #1|
          \let\NEXT=\Setzen
    \fi\NEXT}

```

`\EDATAB` (was `\ATAB`)

```
\def\EDATAB #1\\{%
    \ifx #1\Relax \centerline{\Setzen #1\relax&}
             \let\Next\relax
    \else \centerline{\Setzen #1&\relax&}
          \let\Next=\EDATAB
    \fi\Next}
```

`\edatab` (was `\atab`)

```
\newcommand{\edatab}[1]{%
     \variab%
     \EDATAB #1\\\Relax\\}

```

`\HILFSskip` `\Hilfsskip` More helpers.

```
\newskip\HILFSskip
\newskip\Hilfsskip

```

`\EDTABINDENT` (was `\TABINDENT`)

```
\newcommand{\EDTABINDENT}{%
    \ifnum\l@dcolcount=30\let\NEXT\relax\l@dcolcount=0%
    \else\stepl@dcolcount%
         \advance\Hilfsskip by\l@dcolwidth%
         \ifdim\l@dcolwidth=0pt\advance\hilfscount\@ne
         \else\advance\Hilfsskip by \the\hilfscount\edtabcolsep%
         \hilfscount=1\fi%
         \let\NEXT=\EDTABINDENT%
    \fi\NEXT}%
```

\edtabindent (was \tabindent)

```
\newcommand{\edtabindent}{%
    \l@dcolcount=0\relax
    \Hilfsskip=0pt%
    \hilfscount=1\relax
    \EDTABINDENT%
    \hilfsskip=\hsize%
    \advance\hilfsskip -\Hilfsskip%
    \Hilfsskip=0.5\hilfsskip%
    }%

```

\EDTAB (was \TAB)

```
\def\EDTAB #1|#2|{%
    \setbox\tabhilfbox=\hbox{$\displaystyle{#1}$}%
    \setbox\tabHilfbox=\hbox{$\displaystyle{#2}$}%
    \advance\tabelskip -\wd\tabhilfbox%
    \advance\tabelskip -\wd\tabHilfbox%
    \unhbox\tabhilfbox\hskip\tabelskip%
    \unhbox\tabHilfbox}%

```

\EDTABtext (was \TABtext)

```
\def\EDTABtext #1|#2|{%
    \setbox\tabhilfbox=\hbox{#1}%
    \setbox\tabHilfbox=\hbox{#2}%
    \advance\tabelskip -\wd\tabhilfbox%
    \advance\tabelskip -\wd\tabHilfbox%
    \unhbox\tabhilfbox\hskip\tabelskip%
    \unhbox\tabHilfbox}%
```

\tabhilfbox \tabHilfbox Further helpers.

```
\newbox\tabhilfbox
\newbox\tabHilfbox

```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% That finishes tabmac
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

edarrayl edarrayc edarrayr The ‘environment’ forms for \ltab, \ctab and \rtab.

```
\newenvironment{edarrayl}{\l@dcollect@body\ltab}{}
\newenvironment{edarrayc}{\l@dcollect@body\ctab}{}
\newenvironment{edarrayr}{\l@dcollect@body\rtab}{}

```

edtabularl edtabularc edtabularr The ‘environment’ forms for \ltabtext, \ctabtext and \rtabtext.

```
\newenvironment{edtabularl}{\l@dcollect@body\ltabtext}{}
```

```
\newenvironment{edtabularc}{\l@dcollect@body\ctabtext}{}
\newenvironment{edtabularr}{\l@dcollect@body\rtabtext}{}

```

Here's the code for enabling `\edtext` (instead of `\critext`).

`\usingcritext` `\disablel@dtabfeet` `\enablel@dtabfeet` `\usingedtext`

Declarations for using `\critext{}.../` or using `\edtext{}{}` inside tabulars. The default at this point is for `\edtext`.

```
\newcommand{\usingcritext}{%
  \def\disablel@dtabfeet{\l@dmodforcritext}%
  \def\enablel@dtabfeet{\l@drestoreforcritext}}
\newcommand{\usingedtext}{%
  \def\disablel@dtabfeet{\l@dmodforedtext}%
  \def\enablel@dtabfeet{\l@drestoreforedtext}}

\usingedtext

```

Appendix A Migration from ledmac to eledmac

In eledmac, some changes were made in the code to allow for easy customization. This can cause problems for people who have made their own customizations. The next sections explain how to correct this.

If you created your own series using `\addfootins` and `\addfootinsX`, you should instead use the `\newseries` command (see 4.6 p.22). You must delete your `\Xfootnote` command.

If you customized the `\XXXXXXfmt` command, you should see if commands for display options (4.3 p.17) and options in `\Xfootnote` (4.1 p.15) can't do the same things. If not, you can add a new ticket in Github to request a new function it[31].

If for some reason you don't want to make the modifications to use eledmac new functions, you can continue to use your own `\XXXXXfmt` command, but you must replace:

```
\renewcommand*{XXXXfmt}[3]
```

with

```
\renewcommandx*{XXXXfmt}[4][4=Z]
```

If you dont' do that, you will see a spurious `[X]`, where X is series letter.

If you used a `\protect` command inside a `\footnote` command inside a numbered section, you must change the `\protect` to `\noexpand`. If you don't, the command after the `\protect` won't be displayed.

[31]`https://github.com/maieul/ledmac/issues`

References

[Bre96] Herbert Breger. `TABMAC`. October 1996. (Available from CTAN in `macros/plain/contrib/tabmac`)

[Bur01] John Burt. 'Typesetting critical editions of poetry'. *TUGboat*, **22**, 4, pp 353–361, December 2001. (Code available from CTAN in `macros/latex/contrib/poemscol`)

[Eck03] Matthias Eckermann. *The Parallel-Package*. April 2003. (Available from CTAN in `macros/latex/contrib/parallel`)

[Fai03] Robin Fairbairns. *footmisc — a portmanteau package for customising footnotes in LaTeX*. February 2003. (Available from CTAN in `macros/latex/contrib/footmisc`)

[LW90] John Lavagnino and Dominik Wujastyk. 'An overview of `EDMAC`: a Plain TeX format for critical editions'. *TUGboat*, **11**, 4, pp. 623–643, November 1990. (Code available from CTAN in `macros/plain/contrib/edmac`)

[Lüc03] Uwe Lück. 'ednotes — critical edition typesetting with LaTeX'. *TUGboat*, **24**, 2, pp. 224–236, 2003. (Code available from CTAN in `macros/latex/contrib/ednotes`)

[Sul92] Wayne G. Sullivan. *The file `edstanza.doc`*. June 1992. (Available from CTAN in `macros/plain/contrib/edmac`)

[Wil02] Peter Wilson. *The memoir class for configurable typesetting*. November 2002. (Available from CTAN in `macros/latex/contrib/memoir`)

[Wil04] Peter Wilson and Maïeul Rouquette. *Parallel typesetting for critical editions: the eledpar package*. December 2004. (Available from CTAN in `macros/latex/contrib/ledmmac`)

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

Change History

\ifledRcol: Added \ifledRcol and \ifnumberingR for/from eledpar 44
v0.12.1.
General: Don't number \pstarts of stanza. 1
The numbering of \pstarts restarts on each \beginnumbering. 1
v0.13.
General: New stanzaindentsrepetition counter to repeat stanza indents every n verses. 23
New stanzaindentsrepetition counter: to repeat stanza indents every n verses. 1
\managestanza@modulo: New stanzaindentsrepetition counter to repeat stanza indents every n verses. 158
v0.13.1.
General: \thepstartL and \thepstartR use now \bfseries and not \bf, which is deprecated and makes conflicts with memoir class. 1
v0.14.
General: Tweaked \edlabel to get correct line number if the command is first element of a paragraph. 1
\edlabel: Tweaked \edlabel to get correct line number if the command is first element of a paragraph. 137
v0.15.
General: Line numbering can be reset at each pstart. 48
New management of hangingsymbol insertion, preventing undesirable insertions. 157
Possibilty to print \pstart number in side. 11
\affixline@num: Line numbering can be disabled. 84
\printlines: Line numbering can be reset at each pstart. 97
v0.17.
General: New new management of hangingsymbol insertion, preventing undesirable insertions. 157
v1.0.
General: \lemma can contain commands. 15
Debug in lineation command . 11
New generic commands to customize footnote display. 17
Options nonum and nosep in \Xfootnote. 15
Options of \Xfootnotes. 92
Possibility to have commands in sidenotes. 27
Some compatibility break with eledmac. Change of name: eledmac. 1
\morenoexpands: Change to be compatible with new features 72
v1.0.1.
General: Correction on \numberonlyfirstinline with lineation by pstart or by page. 17
v1.1.
General: Add \labelpstarttrue. 11
Add \numberonlyfirstintwolines 17
Add \pstartinfootnote and \onlypstartinfootnote 17
New hook to add arbitrary code at the beginning of the notes . 19
New options for block of notes. 20
New package option: parapparatus. 1
New tools to change order of series 124
sectioning commands 33
\ledfootinsdim: Deprecated \ledfootinsdim 99
\preXnotes: New skip \preXnotes@ 99
v1.2.
General: Add \ledsectnotoc command. 33
\endquote: Compatibility of \ledchapter with the *memoir* class. 45
\preXnotes: Debug in familiar footnotes (but introduced by v1.1). 99

v1.3.
- `\endquote`: *Quotation* and quote environment inside the numbering sections. 45

v1.4.
- General: Compatibility of `\edtext` (and `\critext`) with the right-to-left direction (with Polyglossia). 74
  - Compatibility with LuaTeX of RTL notes. 40
- `\newseries@`: Remembers the language of the lemma, in order to create a correct direction for the footnote separator. 122
- `\normalfootfmt`: Direction of footnotes with polyglossia. 94
- `\rbracket`: Switch the right bracket to a left bracket when the lemma is RTL (needs polyglossia or LuaTeX). 95

v1.4.1.
- `\endquote`: New option *noquotation*. 45
- `\labelrefsparsesubline`: Fix bug with `\edlabel`. 137

v1.4.2.
- General: Debug with some special classes. 1

v1.4.3.
- General: Add `\nonbreakableafternumber`. 18
  - Spurious space after familiar footnotes. 1

v1.4.4.
- General: Label inside familiar footnotes. 1

v1.4.5.
- General: Bug with komasscript + eledpar + chapter. 1

v1.4.6.
- General: Bug with memoir class introduced by 1.4.5. 1

v1.4.7.
- `\endquote`: Compatibility of sectionning commands with `\autopar`. 45

v1.4.8.
- General: Corrects a bug with parallel texts introduced by 1.1. . . . 1

v1.4.9.
- `\normalbfnoteX`: Allow to redefine `\thefootnoteX` with alph when some packages are loaded. . . 114

v1.5.
- General: Correct indexing when the call is made in critical notes. 150
- `\do@insidelinehook`: Added `\do@insidelinehook` for use in `\do@line` 80
- `\edindex`: Compatibility with imakeidx package, and possibility to use multiple index with `\edindex`. 151
- `\ifFN@bottom`: Use the bottom option of footmisc package. . . . 132