Parallel typesetting for critical editions: the eledpar package*

Peter Wilson
Herries Press†
Maïeul Rouquette‡

Abstract

The eledmac package, which is based on the PLAIN TEX set of EDMAC macros, has been used for some time for typesetting critical editions. The eledpar package is an extension to eledmac which enables texts and their critical apparatus to be typeset in parallel, either in two columns or on pairs of facing pages.

Note that before September 2012, eledpar was called ledpar. The changes from ledmac/ledpar to eledmac/eledpar is explained in ledmac documentation.

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 email list in:
`https://lists.berlios.de/pipermail/ledmac-users/`

Contents

*This file (`eledpar.dtx`) has version number v1.4, last revised 2013/07/11.

†`herries dot press at earthlink dot net`
‡`maieul at maieul dot net`

List of Figures

1 Introduction

The EDMAC macros [LW90] for typesetting critical editions of texts have been available for use with TeX for some years. Since EDMAC became available there had been a small but constant demand for a version of EDMAC that could be used with LaTeX. The eledmac package was introduced in 2003 in an attempt to satisfy that request.

Some critical editions contain texts in more than one form, such as a set of verses in one language and their translations in another. In such cases there is a desire to be able to typeset the two texts, together with any critical apparatus, in parallel. The eledpar package is an extension to eledmac that enables two texts and their apparatus to be set in parallel, either in two columns or on pairs of facing pages.

The package has to try and coerce TEX into paths it was not designed for. Use of the package, therefore, may produce some surprising results.

This manual contains a general description of how to use eledpar starting in section 2; the complete source code for the package, with extensive documentation (in sections 8 through 25); and an Index to the source code. As eledpar is an adjunct to eledmac I assume that you have read the eledmac manual. Also eledpar requires eledmac to be used, preferably at least version 0.10 (2011/08/22). You do not need to read the source code for this package in order to use it but doing so may help to answer any questions you might have. On a first reading, I suggest that you should skip anything after the general documentation in sections 2 until 8, unless you are particularly interested in the innards of eledpar.

2 The eledpar package

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

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

The eledpar package lets you typeset two *numbered* texts in parallel. This can be done either as setting the 'Leftside' and 'Rightside' texts in two columns or on facing pages. In the paired pages case footnotes are placed at the bottom of the page on which they are called out — that is, footnotes belonging to the left are set at the foot of a left (even numbered) page, and those for right texts are at the bottom of the relevant right (odd numbered) page. However, in the columnar case, all footnotes are set at the bottom left of the page on which they are called out — they are not set below the relevant column. The line numbering schemes need not be the same for the two texts.

2.1 General

eledmac essentially puts each chunk of numbered text (the text within a `\pstart` ... `\pend`) into a box and then following the `\pend` extracts the text line by line from the box to number and print it. More precisely, the text is first put into the the box as though it was being typeset as normal onto a page and any notes are stored without being typeset. Then each typeset line is extracted from the box and any notes for that line are recalled. The line, with any notes, is then output for printing, possibly with a line number attached. Effectively, all the text is typeset and then afterwards all the notes are typeset.

eledpar similarly puts the left and right chunks into boxes but can't immediately output the text after a `\pend` — it has to wait until after both the left and right texts have been collected before it can start processing. This means that several boxes are required and possibly TeX has to store a lot of text in its memory; both the number of potential boxes and memory are limited. If TeX's memory is overfilled the recourse is to reduce the amount of text stored before printing.

`\maxchunks` It is possible to have multiple chunks in the left and right texts before printing them. The macro `\maxchunks{`⟨*num*⟩`}` specifies the maximum number of chunks within the left or right texts. This is initially set as:

```
\maxchunks{5120}
```

meaning that there can be up to 5120 chunks in the left text and up to 5120 chunks in the right text, requiring a total of 10240 boxes. If you need more chunks then you can increase `\maxchunks`. The `\maxchunks` must be called in the preamble.

TeX has a limited number of boxes; if you get an error message along the lines of 'no room for a new box', then load the package etex, which needs pdflatex or xelatex. If you `\maxchunks` is too little you can get a eledmac error message along the lines: 'Too many `\pstart` without printing. Some text will be lost.' then you will have to either increase `\maxchunks` or use the parallel printing commands (`\Columns` or `\Pages`) more frequently.

When typesetting verse using `\syntax`, each line is treated as a chunk, so be warned that if you are setting parallel verses you might have to increase `\maxchunks` much more than it appears at first sight.

In general, eledmac is a TeX resource hog, and eledpar only makes things worse in this respect.

3 Parallel columns

`pairs` Numbered text that is to be set in columns must be within a `pairs` environment. Within the environment the text for the lefthand and righthand columns is placed within the `Leftside` and `Rightside` environments, respectively; these are described in more detail below in section 5.

`\Columns` The command `\Columns` typesets the texts in the previous pair of `Leftside` and `Rightside` environments. The general scheme for parallel columns looks like this:

```
\begin{pairs}
\begin{Leftside} ... \end{Leftside}
\begin{Rightside} ... \end{Rightside}
\Columns
\begin{Leftside} ... \end{Leftside}
...
\Columns
\end{pairs}
```

There is no required pagebreak before or after the columns.

`\Lcolwidth` `\Rcolwidth` The lengths `\Lcolwidth` and `\Rcolwidth` are the widths of the left and right columns, respectively. By default, these are:

```
\setlength{\Lcolwidth}{0.45\textwidth}
\setlength{\Rcolwidth}{0.45\textwidth}
```

They may be adjusted if one text tends to be 'bulkier' than the other.

`\columnrulewidth` `\columnseparator` The macro `\columnseparator` is called between each left/right pair of lines. By default it inserts a vertical rule of width `\columnrulewidth`. As this is initially defined to be 0pt the rule is invisible. For a visible rule between the columns you could try:

```
\setlength{\columnrulewidth}{0.4pt}
```

You can also modify `\columnseparator` if you want more control. When you use `\stanza`, the visible rule may shift when a verse has a hanging indent. To prevent shifting, use `\setstanzaindents` outside the `Leftside` or `Rightside` environment.

4 Facing pages

`pages` Numbered text that is to be set on facing pages must be within a `pages` environment. Within the environment the text for the lefthand and righthand pages is placed within the `Leftside` and `Rightside` environments, respectively.

`\Pages` The command `\Pages` typesets the texts in the previous pair of `Leftside` and `Rightside` environments. The general scheme for parallel pages looks like this:

```
\begin{pages}
\begin{Leftside} ... \end{Leftside}
\begin{Rightside} ... \end{Rightside}
\Pages
```

```
\begin{Leftside} ... \end{Leftside}
...
\Pages
\end{pages}
```

The `Leftside` text is set on lefthand (even numbered) pages and the `Rightside` text is set on righthand (odd numbered) pages. Each `\Pages` command starts a new even numbered page. After parallel typesetting is finished, a new page is started.

`\Lcolwidth` `\Rcolwidth` Within the `pages` environment the lengths `\Lcolwidth` and `\Rcolwidth` are the widths of the left and right pages, respectively. By default, these are set to the normal textwidth for the document, but can be changed within the environment if necessary.

`\goalfraction` When doing parallel pages eledpar has to guess where TeX is going to put pagebreaks and hopefully get there first in order to put the pair of texts on their proper pages. When it thinks that the fraction `\goalfraction` of a page has been filled, it finishes that page and starts on the other side's text. The definition is:
`\newcommand*{\goalfraction}{0.9}`
If you think you can get more on a page, increase this. On the other hand, if some left text overflows onto an odd numbered page or some right text onto an even page, try reducing it, for instance by:
`\renewcommand*{\goalfraction}{0.8}`

5 Left and right texts

Parallel texts are divided into Leftside and Rightside. The form of the contents of these two are independent of whether they will be set in columns or pages.

`Leftside` `Rightside` The left text is put within the `Leftside` environment and the right text likewise in the `Rightside` environment. The number of `Leftside` and `Rightside` environments must be the same.

Within these environments you can designate the line numbering scheme(s) to be used. The eledmac package originally used counters for specifying the numbering scheme; now both eledmac[1] and the eledpar package use macros instead. Following `\firstlinenum{`⟨*num*⟩`}` the first line number will be ⟨*num*⟩, and following `\linenumincrement{`⟨*num*⟩`}` only every ⟨*num*⟩th line will have a printed number. Using these macros inside the `Leftside` and `Rightside` environments gives you independent control over the left and right numbering schemes. The `\firstsublinenum` and `\sublinenumincrement` macros correspondingly set the numbering scheme for sublines.

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

`\pstart` `\pend` In a serial (non-parallel) mode, each numbered paragraph, or chunk, is contained between the `\pstart` and `\pend` macros, and the paragraph is output when the `\pend` macro occurs. The situation is somewhat different with parallel typesetting as the left text (contained within `\pstart` and `\pend` groups within the

[1] when used with ledpatch v0.2 or greater.

`Leftside` environment) has to be set in parallel with the right text (contained within its own `\pstart` and `\pend` groups within the corresponding `Rightside` environment) the `\pend` macros cannot immediately initiate any typesetting — this has to be controlled by the `\Columns` or `\Pages` macros. Several chunks may be specified within a `Leftside` or `Rightside` environment. A multi-chunk text then looks like:

```
\begin{...side}
  % \beginnumbering
  \pstart first chunk \pend
  \pstart  second chunk \pend
  ...
  \pstart  last chunk \pend
  % \endnumbering
\end{...side}
```

Numbering, via `\beginnumbering` and `\endnumbering`, may extend across several `Leftside` or `Rightside` environments. Remember, though, that the Left/Right sides are effectively independent of each other.

Generally speaking, controls like `\firstlinenum` or `\linenummargin` apply to sequential and left texts. To effect right texts only they have to be within a `Rightside` environment.

If you are using the babel package with different languages (via, say, `\selectlanguage`) for the left and right texts it is particularly important to select the appropriate language within the `Leftside` and `Rightside` environments. The initial language selected for the right text is the babel package's default. Also, it is the *last* `\selectlanguage` in a side that controls the language used in any notes for that side when they get printed. If you are using multilingual notes then it is probably safest to explicitly specify the language(s) for each note rather than relying on the language selection for the side. The right side language is also applied to the right side line numbers.

Corresponding left and right sides must have the same number of paragraph chunks — if there are four on the left there must be four on the right, even if some are empty. The start of each pair of left and right chunks are aligned horizontally on the page. The ends may come at different positions — if one chunk is shorter than the other then blank lines are output on the shorter side until the end of the longer chunk is reached.

However, sometime if the left pstarts are much greater than right pstarts, or *vice-versa*, you can decide to shift the pstarts on the left and right side. That means the start of pstarts are not aligned horizontally on the page, the shift is offset at the end of each double pages. To enable this function, load eledpar with the option `shiftedpstarts`.

6 Numbering text lines and paragraphs

`\beginnumbering` `\endnumbering`

Each section of numbered text must be preceded by `\beginnumbering` and fol-

lowed by `\endnumbering`, like:
`\beginnumbering`
⟨*text*⟩
`\endnumbering`
These have to be separately specified within `Leftside` and `Rightside` environments.

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. Separate auxiliary files are maintained for right hand texts and these are named ⟨*jobname*⟩`.nnR`, using the 'R' to distinguish them from the left hand and serial (non-parallel) texts.

`\memorydump` The command `\memorydump` effectively performs an `\endumbering` immediately followed by a `\beginnumbering` while not restarting the numbering sequence. This has the effect of clearing TeX's memory of previous texts and any associated notes, allowing longer apparent streams of parallel texts. The command should be applied to both left and right texts, and after making sure that all previous notes have been output. For example, along the lines of:

```
\begin{Leftside}
  \beginnumbering
  ...
\end{Leftside}
\begin{Rightside}
  \beginnumbering
  ...
\end{Rightside}
\Pages
\begin{Leftside}
  \memorydump
  ...
\end{Leftside}
\begin{Rightside}
  \memorydump
  ...
```

`\Rlineflag` The value of `\Rlineflag` is appended to the line numbers of the right texts. Its default definition is:
`\newcommand*{\Rlineflag}{R}`
This may be useful for parallel columns but for parallel pages it might be more appropriate to redefine it as:
`\renewcommand*{\Rlineflag}{}`.

`\printlinesR` `\ledsavedprintlines` The `\printlines` macro is ordinarily used to print the line number references for critical footnotes. For footnotes from right side texts a special version is supplied, called `\printlinesR`, which incorporates `\Rlineflag`. (The macro `\ledsavedprintlines` is a copy of the original `\printlines`, just in case ...).

As provided, the package makes no use of `\printlinesR` but you may find it useful. For example, if you only use the B footnote series in righthand texts then you may wish to flag any line numbers in those footnotes with the value of `\Rlineflag`. You could do this by putting the following code in your preamble:

```
\let\oldBfootfmt\Bfootfmt
\renewcommand{\Bfootfmt}[3]{%
  \let\printlines\printlinesR
  \oldBfootfmt{#1}{#2}{#3}}
```

`\numberpstarttrue` `\numberpstartfalse` `\thepstartL` `\thepstartR`

It's possible to insert a number at every `\pstart` command. You must use the `\numberpstarttrue` command to have it. You can stop the numerotation with `\numberpstartfalse`. You can redefine the commands `\thepstartL` and `\thepstartR` to change style. The numbering restarts on each `\beginnumbering`

7 Verse

If you are typesetting verse with eledmac you can use the `\stanza` construct, and you can also use this in right or left parallel texts. In this case each verse line is a chunk which has two implications. (1) you can unexpectedly exceed the `\maxchunks` limit or the overall limit on the number of boxes, and (2) left and right verse lines are matched, which may not be desirable if one side requires more print lines for verse lines than the other does.

`astanza`

eledpar provides an `astanza` environment which you can use instead of `\stanza` (simply replace `\stanza` by `\begin{astanza}` and add `\end{astanza}` after the ending `\&`). Within the `astanza` environment each verse line is treated as a paragraph, so there must be no blank lines in the environment otherwise there will be some extraneous vertical spacing.

If you get an error message along the lines of 'Missing number, treated as zero `\sza@0@`' it is because you have forgotten to use `\setstanzaindents` to set the stanza indents.

`\skipnumbering`

The command `\skipnumbering` when inserted in a line of parallel text causes the numbering of that particular line to be skipped. This can useful if you are putting some kind of marker (even if it is only a blank line) between stanzas. Remember, parallel texts must be numbered and this provides a way to slip in an 'unnumbered' line.

The `astanza` environment forms a chunk but you may want to have more than one stanza within the chunk. Here are a couple of ways of doing that with a blank line between each internal stanza, and with each stanza numbered. First some preliminary definitions:

```
\newcommand*{\stanzanum}[2][\stanzaindentbase]{%
  \hskip -#1\llap{\textbf{#2}}\hskip #1\ignorespaces}
\newcommand{\interstanza}{\par\mbox{}\skipnumbering}
```

And now for two stanzas in one. In this first example the line numbering repeats for each stanza.

```
\setstanzaindents{1,0,1,0,1,0,1,0,1,0,1}
\begin{pairs}
\begin{Leftside}
  \firstlinenum{2}
  \linenumincrement{1}
  \beginnumbering
  \begin{astanza}
    \stanzanum{1} First in first stanza &
                  Second in first stanza &
                  Second in first stanza &
                  Third in first stanza &
                  Fourth in first stanza &
    \interstanza
    \setline{2}\stanzanum{2} First in second stanza &
                  Second in second stanza &
                  Second in second stanza &
                  Third in second stanza &
                  Fourth in second stanza \&
  \end{astanza}
  ...
```

And here is a slightly different way of doing the same thing, but with the line numbering being continuous.

```
\setstanzaindents{1,0,1,0,1,0,0,1,0,1,0,1}
\begin{pairs}
\begin{Leftside}
  \firstlinenum{2}
  \linenumincrement{1}
  \beginnumbering
  \begin{astanza}
    \stanzanum{1} First in first stanza &
                  Second in first stanza &
                  Second in first stanza &
                  Third in first stanza &
                  Fourth in first stanza &
    \strut &
    \stanzanum{2}\advanceline{-1} First in second stanza &
                  Second in second stanza &
                  Second in second stanza &
                  Third in second stanza &
                  Fourth in second stanza \&
  \end{astanza}
  ...
```

`\hangingsymbol` Like in eledmac, you could redefine the command `\hangingsymbol` to insert a

character in each hanged line. If you use it, you must run LaTeXtwo time. Example for the french typographie

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

8 Implementation overview

TeX is designed to process a single stream of text, which may include footnotes, tables, and so on. It just keeps converting its input into a stream typeset pages. It was not designed for typesetting two texts in parallel, where it has to alternate from one to the other. Further, TeX essentially processes its input one paragraph at a time — it is very difficult to get at the 'internals' of a paragraph such as the individual lines in case you want to number them or put some mark at the start or end of the lines.

eledmac solves the problem of line numbering by putting the paragraph in typeset form into a box, and then extracting the lines one by one from the box for TeX to put them onto the page with the appropriate page breaks. Most of the eledmac code is concerned with handling this box and its contents.

eledpar's solution to the problem of parallel texts is to put the two texts into separate boxes, and then appropriately extract the pairs of lines from the boxes. This involves duplicating much of the original box code for an extra right text box. The other, smaller, part of the code is concerned with coordinating the line extractions from the boxes.

The package code is presented in roughly in the same order as in eledmac.

9 Preliminaries

Announce the name and version of the package, which is targetted for LaTeX2e. The package also requires the eledmac package.

```
⟨*code⟩
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{eledpar}[2013/07/11 v1.4 eledmac extension for parallel texts]

```

With the option 'shiftedpstarts' a long pstart one the left side (or in the right side) don't make a blank on the corresponding pstart, but the blank is put on the bottom of the page. Consequently, the pstarts on the parallel pages are shifted, but the shifted stop at every end of pages. The `\shiftedverses` is kept for backward compatibility.

`\ifshiftedpstarts`

```
\newif\ifshiftedpstarts
\let\shiftedversestrue\shiftedpstartstrue
\let\shiftedversesfalse\shiftedpstartsfalse
\DeclareOption{shiftedverses}{\shiftedpstartstrue}
\DeclareOption{shiftedpstarts}{\shiftedpstartstrue}
\ProcessOptions
```

As noted above, much of the code is a duplication of the original eledmac code to handle the extra box(es) for the right hand side text, and sometimes for the left hand side as well. In order to distinguish I use 'R' or 'L' in the names of macros for the right and left code. The specifics of 'L' and 'R' are normally

hidden from the user by letting the `Leftside` and `Rightside` environments set things up appropriately.

`\ifl@dpairing` `\ifl@dpaging` `\ifledRcol`

`\ifl@dpairing` is set TRUE if we are processing parallel texts and `\ifl@dpaging` is also set TRUE if we are doing parallel pages. `\ifledRcol` is set TRUE if we are doing the right hand text. `\ifl@dpairing` is defined in eledmac.

```
\l@dpairingfalse
\newif\ifl@dpaging
  \l@dpagingfalse
  \ledRcolfalse
```

`\Lcolwidth` `\Rcolwidth`

The widths of the left and right parallel columns (or pages).

```
\newdimen\Lcolwidth
  \Lcolwidth=0.45\textwidth
\newdimen\Rcolwidth
  \Rcolwidth=0.45\textwidth

```

9.1 Messages

All the error and warning messages are collected here as macros.

`\led@err@TooManyPstarts`

```
\newcommand*{\led@err@TooManyPstarts}{%
  \eledmac@error{Too many \string\pstart\space without printing.
                 Some text will be lost}{\@ehc}}
```

`d@err@BadLeftRightPstarts`

```
\newcommand*{\led@err@BadLeftRightPstarts}[2]{%
  \eledmac@error{The numbers of left (#1) and right (#2)
                 \string\pstart s do not match}{\@ehc}}
```

`\led@err@LeftOnRightPage` `\led@err@RightOnLeftPage`

```
\newcommand*{\led@err@LeftOnRightPage}{%
  \eledmac@error{The left page has ended on a right page}{\@ehc}}
\newcommand*{\led@err@RightOnLeftPage}{%
  \eledmac@error{The right page has ended on a left page}{\@ehc}}
```

10 Sectioning commands

`\section@numR`

This is the right side equivalent of `\section@num`.

Each section will read and write an associated 'line-list file', containing information used to do the numbering. Normally the file will be called ⟨*jobname*⟩`.nn`, where `nn` is the section number. However, for right side texts the file is called ⟨*jobname*⟩`.nnR`. The `\extensionchars` applies to the right side files just as it does to the normal files.

```
\newcount\section@numR
  \section@numR=\z@
```

\ifpst@rtedL
\ifpst@rtedR

\ifpst@rtedL is set FALSE at the start of left side numbering, and similarly for \ifpst@rtedR. \ifpst@rtedL is defined in eledmac.

```
  \pst@rtedLfalse
\newif\ifpst@rtedR
  \pst@rtedRfalse

```

\beginnumbering

For parallel processing the original \beginnumbering is extended to zero \l@dnumpstartsL — the number of chunks to be processed. It also sets \ifpst@rtedL to FALSE.

```
\providecommand*{\beginnumbering}{%
  \ifnumbering
    \led@err@NumberingStarted
    \endnumbering
  \fi
  \global\l@dnumpstartsL \z@
  \global\pst@rtedLfalse
  \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}
```

\beginnumberingR

This is the right text equivalent of \beginnumbering, and begins a section of numbered text.

```
\newcommand*{\beginnumberingR}{%
  \ifnumberingR
    \led@err@NumberingStarted
    \endnumberingR
  \fi
  \global\l@dnumpstartsR \z@
  \global\pst@rtedRfalse
  \global\numberingRtrue
  \global\advance\section@numR \@ne
  \global\absline@numR \z@
  \global\line@numR \z@
  \global\@lockR \z@
  \global\sub@lockR \z@
  \global\sublines@false
  \global\let\next@page@numR\relax
  \global\let\sub@change\relax
  \message{Section \the\section@numR R }%
  \line@list@stuffR{\jobname.\extensionchars\the\section@numR R}%
  \l@dend@stuff
  \setcounter{pstartR}{1}
  \begingroup
  \initnumbering@sectcmd
  \initnumbering@sectcountR
}
```

```
73
```

`\endnumbering` This is the left text version of the regular `\endnumbering` and must follow the last text for a left text numbered section. It sets `\ifpst@rtedL` to FALSE. It is fully defined in eledmac.

`\endnumberingR` This is the right text equivalent of `\endnumbering` and must follow the last text for a right text numbered section.

```
74 \def\endnumberingR{%
75   \ifnumberingR
76     \global\numberingRfalse
77     \normal@pars
78     \ifl@dpairing
79       \global\pst@rtedRfalse
80     \else
81       \ifx\insertlines@listR\empty\else
82         \global\noteschanged@true
83       \fi
84       \ifx\line@listR\empty\else
85         \global\noteschanged@true
86       \fi
87     \fi
88     \ifnoteschanged@
89       \led@mess@NotesChanged
90     \fi
91   \else
92     \led@err@NumberingNotStarted
93   \fi\endgroup}
94
```

`\initnumbering@sectcountR` We don't want the numbering of the right-side section commands to be continuous with the numbering of the left side, we switch the LaTeX counter in `\numberingR`.

```
95 \newcounter{chapterR}
96 \newcounter{sectionR}
97 \newcounter{subsectionR}
98 \newcounter{subsubsectionR}
99 \newcommand{\initnumbering@sectcountR}{
100 \let\c@chapter\c@chapterR
101 \let\c@section\c@sectionR
102 \let\c@subsection\c@subsectionR
103 \let\c@subsubsection\c@subsubsectionR
104 }
```

`\pausenumberingR` These are the right text equivalents of `\pausenumbering` and `\resumenumbering`.
`\resumenumberingR`

```
105 \newcommand*{\pausenumberingR}{%
106   \endnumberingR\global\numberingRtrue}
107 \newcommand*{\resumenumberingR}{%
108   \ifnumberingR
109     \global\pst@rtedRtrue
```

```
    \global\advance\section@numR \@ne
    \led@mess@SectionContinued{\the\section@numR R}%
    \line@list@stuffR{\jobname.\extensionchars\the\section@numR R}%
    \l@dend@stuff
  \else
    \led@err@numberingShouldHaveStarted
    \endnumberingR
    \beginnumberingR
  \fi}

```

\memorydumpL
\memorydumpR

\memorydump is a shorthand for \pausenumbering\resumenumbering. This will clear the memorised stuff for the previous chunks while keeping the numbering going.

```
\newcommand*{\memorydumpL}{%
  \endnumbering
  \numberingtrue
  \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}
\newcommand*{\memorydumpR}{%
  \endnumberingR
  \numberingRtrue
  \global\pst@rtedRtrue
  \global\advance\section@numR \@ne
     \led@mess@SectionContinued{\the\section@numR R}%
  \line@list@stuffR{\jobname.\extensionchars\the\section@numR R}%
  \l@dend@stuff}

```

11 Line counting

11.1 Choosing the system of lineation

M 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. eledpar lets you choose different schemes for the left and right texts.

\ifbypstart@R
\bypstart@Rtrue
\bypstart@Rfalse
\ifbypage@R
\bypage@Rtrue
\bypage@Rfalse

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

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

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

```
\newif\ifbypage@R
\newif\ifbypstart@R
  \bypage@Rfalse
  \bypstart@Rfalse
```

`\lineationR` `\lineationR{`⟨*word*⟩`}` is the macro used to select the lineation system for right texts. Its argument is a string: either `page`, `pstart` or `section`.

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

`\linenummargin` `\line@marginR` You call `\linenummargin{`⟨*word*⟩`}` to specify which margin you want your right text's 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. 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.

For right texts the selection is recorded in the count `\line@marginR`, otherwise in the count `\line@margin`: 0 for left, 1 for right, 2 for outer, and 3 for inner.

```
\newcount\line@marginR
\renewcommand*{\linenummargin}[1]{{%
  \l@dgetline@margin{#1}%
  \ifnum\@l@dtempcntb>\m@ne
    \ifledRcol
      \global\line@marginR=\@l@dtempcntb
    \else
      \global\line@margin=\@l@dtempcntb
```

```
    \fi
  \fi}}
```

By default put right text numbers at the right.

```
\line@marginR=\@ne
```

`\c@firstlinenumR` `\c@linenumincrementR`

The following counters tell eledmac which right text 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{firstlinenumR}
  \setcounter{firstlinenumR}{5}
\newcounter{linenumincrementR}
  \setcounter{linenumincrementR}{5}
```

`\c@firstsublinenumR` `\c@sublinenumincrementR`

The following parameters are just like `firstlinenumR` and `linenumincrementR`, but for sub-line numbers. `sublinenumincrementR` must be at least 1.

```
\newcounter{firstsublinenumR}
  \setcounter{firstsublinenumR}{5}
\newcounter{sublinenumincrementR}
  \setcounter{sublinenumincrementR}{5}
```

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

These are the user's macros for changing (sub) line numbers. They are defined in eledmac v0.7, but just in case I have started by `\provide`ing them.

```
\providecommand*{\firstlinenum}{}
\providecommand*{\linenumincrement}{}
\providecommand*{\firstsublinenum}{}
\providecommand*{\sublinenumincrement}{}
\renewcommand*{\firstlinenum}[1]{%
  \ifledRcol \setcounter{firstlinenumR}{#1}%
  \else      \setcounter{firstlinenum}{#1}%
  \fi}
\renewcommand*{\linenumincrement}[1]{%
  \ifledRcol \setcounter{linenumincrementR}{#1}%
  \else      \setcounter{linenumincrement}{#1}%
  \fi}
\renewcommand*{\firstsublinenum}[1]{%
  \ifledRcol \setcounter{firstsublinenumR}{#1}%
  \else      \setcounter{firstsublinenum}{#1}%
  \fi}
\renewcommand*{\sublinenumincrement}[1]{%
  \ifledRcol \setcounter{sublinenumincrementR}{#1}%
  \else      \setcounter{sublinenumincrement}{#1}%
  \fi}
```

`\Rlineflag` This is appended to the line numbers of right text.

```
\newcommand*{\Rlineflag}{R}
```

`\linenumrepR` `\linenumrepR{`⟨*ctr*⟩`}` typesets the right line number ⟨*ctr*⟩, and similarly `\sublinenumrepR` `\sublinenumrepR` for subline numbers.

```
\newcommand*{\linenumrepR}[1]{\@arabic{#1}}
\newcommand*{\sublinenumrepR}[1]{\@arabic{#1}}
```

`\leftlinenumR` `\leftlinenumR` and `\rightlinenumR` are the macros that are called to print the `\rightlinenumR` right text's marginal line numbers. Much of the code for these is common and is `\l@dlinenumR` maintained in `\l@dlinenumR`.

```
\newcommand*{\leftlinenumR}{%
  \l@dlinenumR
  \kern\linenumsep}
\newcommand*{\rightlinenumR}{%
  \kern\linenumsep
  \l@dlinenumR}
\newcommand*{\l@dlinenumR}{%
  \numlabfont\linenumrepR{\line@numR}\Rlineflag%
  \ifsublines@
    \ifnum\subline@num>\z@
      \unskip\fullstop\sublinenumrepR{\subline@numR}%
    \fi
  \fi}
```

11.2 Line-number counters and lists

We need another set of counters and lists for the right text, corresponding to those in eledmac for regualr or left text.

`\line@numR` The count `\line@numR` stores the line number that's used in the right text's `\subline@numR` marginal line numbering and in notes. The count `\subline@numR` stores a sub-line `\absline@numR` number that qualifies `\line@numR`. The count `\absline@numR` stores the absolute number of lines since the start of the right text section: that is, the number we've actually printed, no matter what numbers we attached to them.

```
\newcount\line@numR
\newcount\subline@numR
\newcount\absline@numR
```

`\line@listR` Now we can define the list macros that will be created from the line-list file. They `\insertlines@listR` are directly analagous to the left text ones. The full list of action codes and their `\actionlines@listR` meanings is given in the eledmac manual.

`\actions@listR` Here are the commands to create these lists:

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

`\linesinpar@listL` `\linesinpar@listR` `\maxlinesinpar@list` In order to synchonise left and right chunks in parallel processing we need to know how many lines are in each left and right text chunk, and the maximum of these for each pair of chunks.

```
\list@create{\linesinpar@listL}
\list@create{\linesinpar@listR}
\list@create{\maxlinesinpar@list}
```

`\page@numR` The right text page number.

```
\newcount\page@numR
```

11.3 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.

```
\renewcommand*{\read@linelist}[1]{%
```

We do do different things depending whether or not we are processing right text

```
  \ifledRcol
    \list@clear{\line@listR}%
    \list@clear{\insertlines@listR}%
    \list@clear{\actionlines@listR}%
    \list@clear{\actions@listR}%
    \list@clear{\linesinpar@listR}%
    \list@clear{\linesonpage@listR}
  \else
    \list@clearing@reg
    \list@clear{\linesinpar@listL}%
    \list@clear{\linesonpage@listL}%
  \fi
```

Make sure that the `\maxlinesinpar@list` is empty (otherwise things will be thrown out of kilter if there is any old stuff still hanging in there).

```
  \list@clear{\maxlinesinpar@list}
```

Now get the file and interpret it.

```
  \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.

```
\ifledRcol
  \global\page@numR=\m@ne
  \ifx\actionlines@listR\empty
    \gdef\next@actionlineR{1000000}%
  \else
    \gl@p\actionlines@listR\to\next@actionlineR
    \gl@p\actions@listR\to\next@actionR
  \fi
\else
  \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
\fi}

```

This version of \read@linelist creates list macros containing data for the entire section, so they could get rather large. The \memorydump macro is available if you run into macro memory limitations.

11.4 Commands within the line-list file

This section defines the commands that can appear within a line-list file, except for \@lab which is in a later section among the cross-referencing commands it is associated with.

The macros with action in their names contain all the code that modifies the action-code list.

\@l@regR
\@l

\@l does everything related to the start of a new line of numbered text. Exactly what it does depends on whether right text is being processed.

```
\newcommand{\@l@regR}{%
  \ifx\l@dchset@num\relax \else
    \advance\absline@numR \@ne
    \set@line@action
    \let\l@dchset@num\relax
    \advance\absline@numR \m@ne
    \advance\line@numR \m@ne%   % do we need this?
  \fi
  \advance\absline@numR \@ne
  \ifx\next@page@numR\relax \else
    \page@action
    \let\next@page@numR\relax
  \fi
  \ifx\sub@change\relax \else
```

```
    \ifnum\sub@change>\z@
      \sublines@true
    \else
      \sublines@false
    \fi
    \sub@action
    \let\sub@change\relax
  \fi
  \ifcase\@lockR
  \or
    \@lockR \tw@
  \or\or
    \@lockR \z@
  \fi
  \ifcase\sub@lockR
  \or
    \sub@lockR \tw@
  \or\or
    \sub@lockR \z@
  \fi
  \ifsublines@
    \ifnum\sub@lockR<\tw@
      \advance\subline@numR \@ne
    \fi
  \else
    \ifnum\@lockR<\tw@
      \advance\line@numR \@ne \subline@numR \z@
    \fi
  \fi}

\renewcommand*{\@l}[2]{%
  \fix@page{#1}%
  \ifledRcol
    \@l@regR
  \else
    \@l@reg
  \fi}
```

\last@page@numR
\fix@page

We have to adjust \fix@page to handle parallel texts.

```
\newcount\last@page@numR
  \last@page@numR=-10000
\renewcommand*{\fix@page}[1]{%
  \ifledRcol
    \ifnum #1=\last@page@numR
    \else
      \ifbypage@R
        \line@numR \z@ \subline@numR \z@
      \fi
      \page@numR=#1\relax
```

```
      \last@page@numR=#1\relax
      \def\next@page@numR{#1}%
    \fi
  \else
    \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
  \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`.

```
\renewcommand*{\@adv}[1]{%
  \ifsublines@
    \ifledRcol
      \advance\subline@numR by #1\relax
      \ifnum\subline@numR<\z@
        \led@warn@BadAdvancelineSubline
        \subline@numR \z@
      \fi
    \else
      \advance\subline@num by #1\relax
      \ifnum\subline@num<\z@
        \led@warn@BadAdvancelineSubline
        \subline@num \z@
      \fi
    \fi
  \else
    \ifledRcol
      \advance\line@numR by #1\relax
      \ifnum\line@numR<\z@
        \led@warn@BadAdvancelineLine
        \line@numR \z@
      \fi
    \else
      \advance\line@num by #1\relax
      \ifnum\line@num<\z@
        \led@warn@BadAdvancelineLine
        \line@num \z@
      \fi
    \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`.

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

```

`\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.

```
\renewcommand*{\l@d@set}[1]{%
  \ifledRcol
    \line@numR=#1\relax
    \advance\line@numR \@ne
    \def\l@dchset@num{#1}
  \else
    \line@num=#1\relax
    \advance\line@num \@ne
    \def\l@dchset@num{#1}
  \fi}
\let\l@dchset@num\relax

```

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

```
\renewcommand*{\page@action}{%
  \ifledRcol
    \xright@appenditem{\the\absline@numR}\to\actionlines@listR
    \xright@appenditem{\next@page@numR}\to\actions@listR
  \else
    \xright@appenditem{\the\absline@num}\to\actionlines@list
    \xright@appenditem{\next@page@num}\to\actions@list
  \fi}
```

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

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

```

`\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.

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

```

`\do@lockon` `\do@lockonR` `\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.

```
\newcount\@lockR
\newcount\sub@lockR

\newcommand*{\do@lockonR}{%
```

```
  \xright@appenditem{\the\absline@numR}\to\actionlines@listR
  \ifsublines@
    \xright@appenditem{-1005}\to\actions@listR
    \ifnum\sub@lockR=\z@
      \sub@lockR \@ne
    \else
      \ifnum\sub@lockR=\thr@@
        \sub@lockR \@ne
      \fi
    \fi
  \else
    \xright@appenditem{-1003}\to\actions@listR
    \ifnum\@lockR=\z@
      \@lockR \@ne
    \else
      \ifnum\@lockR=\thr@@
        \@lockR \@ne
      \fi
    \fi
  \fi}

\renewcommand*{\do@lockon}{%
  \ifx\next\lock@off
    \global\let\lock@off=\skip@lockoff
  \else
    \ifledRcol
      \do@lockonR
    \else
      \do@lockonL
    \fi
  \fi}
```

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

`\do@lockoff` `\do@lockoffR` `\skip@lockoff`

```

\newcommand{\do@lockoffR}{%
  \xright@appenditem{\the\absline@numR}\to\actionlines@listR
  \ifsublines@
    \xright@appenditem{-1006}\to\actions@listR
    \ifnum\sub@lockR=\tw@
      \sub@lockR \thr@@
    \else
      \sub@lockR \z@
    \fi
  \else
    \xright@appenditem{-1004}\to\actions@listR
    \ifnum\@lockR=\tw@
      \@lockR \thr@@
    \else
      \@lockR \z@
```

```
    \fi
  \fi}

\renewcommand*{\do@lockoff}{%
  \ifledRcol
    \do@lockoffR
  \else
    \do@lockoffL
  \fi}
\global\let\lock@off=\do@lockoff

```

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

```
\providecommand*{\n@num}{}
\renewcommand*{\n@num}{%
  \ifledRcol
    \xright@appenditem{\the\absline@numR}\to\actionlines@listR
    \xright@appenditem{-1007}\to\actions@listR
  \else
    \n@num@reg
  \fi}

```

`\@ref` `\insert@countR` `\@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 for right text, here and within `\edtext`, which computes it and writes it to the line-list file, will be stored in the count `\insert@countR`.

```
    \newcount\insert@countR
```

- `#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`.)

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

```
\renewcommand*{\@ref}[2]{%
  \ifledRcol
    \global\insert@countR=#1\relax
    \loop\ifnum\insert@countR>\z@
      \xright@appenditem{\the\absline@numR}\to\insertlines@listR
      \global\advance\insert@countR \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@numR
  \global\endline@num=\line@numR
  \global\endsubline@num=\subline@numR
\endgroup
```

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

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

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

```
#2
\else
```

And when not in right text

```
  \@ref@reg{#1}{#2}%
\fi}
```

`\@pend` `\@pendR` `\@pend{`⟨*num*⟩`}` adds its argument to the `\linesinpar@listL` list, and analagously for `\@pendR`. If needed, it resets line number. We start off with a `\providecommand` just in case an older version of eledmac is being used which does not define these macros.

```
\providecommand*{\@pend}[1]{}
\renewcommand*{\@pend}[1]{%
  \ifbypstart@\global\line@num=0\fi%
  \xright@appenditem{#1}\to\linesinpar@listL}
\providecommand*{\@pendR}[1]{}
\renewcommand*{\@pendR}[1]{%
  \ifbypstart@R\global\line@numR=0\fi
  \xright@appenditem{#1}\to\linesinpar@listR}

```

`\@lopL` `\@lopR` `\@lopL{`⟨*num*⟩`}` adds its argument to the `\linesonpage@listL` list, and analagously for `\@lopR`. We start off with a `\providecommand` just in case an older version of eledmac is being used which does not define these macros.

```
\providecommand*{\@lopL}[1]{}
```

```
\renewcommand*{\@lopL}[1]{%
  \xright@appenditem{#1}\to\linesonpage@listL}
\providecommand*{\@lopR}[1]{}
\renewcommand*{\@lopR}[1]{%
  \xright@appenditem{#1}\to\linesonpage@listR}

```

11.5 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@outR` The file for right texts will be opened on output stream `\linenum@outR`.

```
\newwrite\linenum@outR

```

`\iffirst@linenum@out@R` `\first@linenum@out@Rtrue` `\first@linenum@out@Rfalse` Once any file is opened on this stream, we keep it open forever, or else switch to another file that we keep open.

```
\newif\iffirst@linenum@out@R
  \first@linenum@out@Rtrue

```

`\line@list@stuffR` This is the right text version of the `\line@list@stuff{`⟨*file*⟩`}` macro. It is called by `\beginnumberingR` and 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@stuffR}[1]{%
  \read@linelist{#1}%
  \iffirst@linenum@out@R
     \immediate\closeout\linenum@outR
     \global\first@linenum@out@Rfalse
     \immediate\openout\linenum@outR=#1
  \else
     \closeout\linenum@outR
     \openout\linenum@outR=#1
  \fi}

```

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

```
\newcommand*{\new@lineR}{%
  \write\linenum@outR{\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.

```
\renewcommand*{\flag@start}{%
  \ifledRcol
    \edef\next{\write\linenum@outR{%
                  \string\@ref[\the\insert@countR][]}%
    \next
```

```
  \else
    \edef\next{\write\linenum@out{%
                  \string\@ref[\the\insert@count][}}%
    \next
  \fi}
\renewcommand*{\flag@end}{%
  \ifledRcol
    \write\linenum@outR{]}%
  \else
    \write\linenum@out{]}%
  \fi}
```

`\startsub` `\endsub` `\startsub` and `\endsub` turn sub-lineation on and off, by writing appropriate instructions to the line-list file.

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

```
\renewcommand*{\advanceline}[1]{%
  \ifledRcol \write\linenum@outR{\string\@adv[#1]}%
  \else      \write\linenum@out{\string\@adv[#1]}%
  \fi}
```

`\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.

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

```
\renewcommand*{\setlinenum}[1]{%
```

```
\ifnum#1<\z@
  \led@warn@BadSetlinenum
\else
  \ifledRcol \write\linenum@outR{\string\l@d@set[#1]}
  \else      \write\linenum@out{\string\l@d@set[#1]} \fi
\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.

```
\renewcommand*{\startlock}{%
  \ifledRcol \write\linenum@outR{\string\lock@on}%
  \else      \write\linenum@out{\string\lock@on}%
  \fi}
\def\endlock{%
  \ifledRcol \write\linenum@outR{\string\lock@off}%
  \else      \write\linenum@out{\string\lock@off}%
  \fi}
```

`\skipnumbering` In numbered text, `\skipnumbering` in a line will suspend the numbering for that particular line. That is, line numbers are unchanged and no line number will be printed.

```
\renewcommand*{\skipnumbering}{%
  \ifledRcol \write\linenum@outR{\string\n@num}%
             \advanceline{-1}%
  \else
    \skipnumbering@reg
  \fi}
```

12 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.

`\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}
```

\critext Now we begin \critext itself.

We slightly modify the original to make accomodation for when right text is being processed.

```
\long\def\critext#1#2/{\leavevmode
  \begingroup
    \renewcommand{\@tag}{\no@expands #1}%
    \set@line
    \ifledRcol \global\insert@countR \z@
    \else      \global\insert@count \z@ \fi
    \ignorespaces #2\relax
    \@ifundefined{xpg@main@language}{%if not polyglossia
       \flag@start}%
       {\if@RTL\flag@end\else\flag@start\fi% be careful on the direction of writing with p
       }%
  \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% be careful on the direction of writing with p
       }
  }
```

\edtext And similarly for \edtext.

```
\renewcommand{\edtext}[2]{\leavevmode
  \begingroup%
    \renewcommand{\@tag}{\no@expands #1}%
    \set@line%
    \ifledRcol \global\insert@countR \z@%
    \else      \global\insert@count \z@ \fi%
    \ignorespaces #2\relax%
    \@ifundefined{xpg@main@language}{%if not polyglossia
       \flag@start}%
       {\if@RTL\flag@end\else\flag@start\fi% be careful on the direction of writing with p
       }%
  \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% be careful on the direction of writing with p
```

```
700         }%
701 }
702
```

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

```
703 \renewcommand*{\set@line}{%
704   \ifledRcol
705     \ifx\line@listR\empty
706       \global\noteschanged@true
707       \xdef\l@d@nums{000|000|000|000|000|000|\edfont@info}%
708     \else
709       \gl@p\line@listR\to\@tempb
710       \xdef\l@d@nums{\@tempb|\edfont@info}%
711       \global\let\@tempb=\undefined
712     \fi
713   \else
714     \ifx\line@list\empty
715       \global\noteschanged@true
716       \xdef\l@d@nums{000|000|000|000|000|000|\edfont@info}%
717     \else
718       \gl@p\line@list\to\@tempb
719       \xdef\l@d@nums{\@tempb|\edfont@info}%
720       \global\let\@tempb=\undefined
721     \fi
722   \fi}
723
```

13 Parallel environments

The initial set up for parallel processing is deceptively simple.

`pairs` `pages` The `pairs` environment is for parallel columns and the `pages` environment for parallel pages.

`chapterinpages`

```
724 \newenvironment{pairs}{%}
725   \l@dpairingtrue
726   \l@dpagingfalse
727 }{%
728   \l@dpairingfalse
729 }
```

The `pages` environment additionally sets the 'column' widths to the `\textwidth` (as known at the time the package is called). In this environment, there are two text in parallel on 2 pages. To prevent chapters starting on a lefthand page, the `\chapter` command is redefined to not clear pages.

```
730 \newenvironment{pages}{%
731  \let\oldchapter\chapter
732  \let\chapter\chapterinpages
```

```
  \l@dpairingtrue
  \l@dpagingtrue
  \setlength{\Lcolwidth}{\textwidth}%
  \setlength{\Rcolwidth}{\textwidth}%
}{%
  \l@dpairingfalse
  \l@dpagingfalse
  \let\chapter\oldchapter
}
\newcommand{\chapterinpages}{\thispagestyle{plain}%
                    \global\@topnum\z@
                    \@afterindentfalse
                    \secdef\@chapter\@schapter}

```

`ifinstanzaL` `ifinstanzaR` These boolean tests are switched by the `\stanza` command, using either the left or right side.

```
\newif\ifinstanzaL
\newif\ifinstanzaR
```

`Leftside` Within the `pairs` and `pages` environments the left and right hand texts are within `Leftside` and `Rightside` environments, respectively. The `Leftside` environment is simple, indicating that right text is not within its purview and using some particular macros.

```
\newenvironment{Leftside}{%
  \ledRcolfalse
  \let\beginnumbering\beginnumbering\setcounter{pstartL}{1}
  \let\pstart\pstartL
  \let\thepstart\thepstartL
  \let\pend\pendL
  \let\memorydump\memorydumpL
  \Leftsidehook
  \let\oldstanza\stanza
  \renewcommand{\stanza}{\oldstanza\global\instanzaLtrue}
}{
    \let\stanza\oldstanza
    \Leftsidehookend}
```

`\Leftsidehook` `\Leftsidehookend` Hooks into the start and end of the `Leftside` and `Rightside` environments. These are initially empty.

`\Rightsidehook` `\Rightsidehookend`

```
\newcommand*{\Leftsidehook}{}
\newcommand*{\Leftsidehookend}{}
\newcommand*{\Rightsidehook}{}
\newcommand*{\Rightsidehookend}{}

```

`Rightside` The `Rightside` environment is only slightly more complicated than the `Leftside`. Apart from indicating that right text is being provided it ensures that the right right text code will be used.

```
\newenvironment{Rightside}{%
  \ledRcoltrue
  \let\beginnumbering\beginnumberingR
  \let\endnumbering\endnumberingR
  \let\pausenumbering\pausenumberingR
  \let\resumenumbering\resumenumberingR
  \let\memorydump\memorydumpR
  \let\thepstart\thepstartR
  \let\pstart\pstartR
  \let\pend\pendR
  \let\lineation\lineationR
  \Rightsidehook
  \let\oldstanza\stanza
  \renewcommand{\stanza}{\oldstanza\global\instanzaRtrue}
}{%
  \ledRcolfalse
  \let\stanza\oldstanza
  \Rightsidehookend
}
```

14 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.

14.1 Boxes, counters, \pstart and \pend

\num@linesR
\one@lineR
\par@lineR

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, \l@dLcolrawbox or \l@dRcolrawbox for right 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(R) 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 or \one@lineR register, and \par@line(R) will be the number of that line within the paragraph.

```
\newcount\num@linesR
\newbox\one@lineR
\newcount\par@lineR
```

\pstartL
\pstartR

changesv1.12012/09/25Add \labelpstarttrue (from eledmac). changesv1.1.12012/10/01Correct \pstartR bug introduced by 1.1. \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 appropriate box. `\pstart` needs to appear at the start of every paragraph that's to be numbered.

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.

We have to have specific left and right `\pstart` when parallel processing; among other things because of potential changes in the linewidth. The `old` counters are used to have the good reset of the pstart counters at the begining of the `\Pages` command.

```
\newcounter{pstartL}
\newcounter{pstartLold}
\renewcommand{\thepstartL}{{\bfseries\@arabic\c@pstartL}. }
\newcounter{pstartR}
\newcounter{pstartRold}
\renewcommand{\thepstartR}{{\bfseries\@arabic\c@pstartR}. }

\newcommand*{\pstartL}{
\if@nobreak
    \let\@oldnobreak\@nobreaktrue
\else
    \let\@oldnobreak\@nobreakfalse
\fi
    \@nobreaktrue
  \ifnumbering \else
    \led@err@PstartNotNumbered
    \beginnumbering
  \fi
  \ifnumberedpar@
    \led@err@PstartInPstart
    \pend
  \fi
```

If this is the first `\pstart` in a numbered section, clear any inserts and set `\ifpst@rtedL` to FALSE. Save the pstartL counter.

```
  \ifpst@rtedL\else
      \setcounter{pstartLold}{\value{pstartL}}%
    \list@clear{\inserts@list}%
    \global\let\next@insert=\empty
     \global\pst@rtedLtrue
  \fi
  \begingroup\normal@pars
```

When parallel processing we check that we haven't exceeded the maximum number of chunks. In any event we grab a box for the forthcoming text.

```
  \global\advance\l@dnumpstartsL \@ne
  \ifnum\l@dnumpstartsL>\l@dc@maxchunks
    \led@err@TooManyPstarts
```

```
    \global\l@dnumpstartsL=\l@dc@maxchunks
  \fi
  \global\setnamebox{l@dLcolrawbox\the\l@dnumpstartsL}=\vbox\bgroup\ifautopar\else\ifnumberpstart\ifs
                      \hsize=\Lcolwidth
  \numberedpar@true
  \iflabelpstart\protected@edef\@currentlabel
       {\p@pstartL\thepstartL}\fi
  }
\newcommand*{\pstartR}{
\if@nobreak
    \let\@oldnobreak\@nobreaktrue
\else
    \let\@oldnobreak\@nobreakfalse
\fi
    \@nobreaktrue
  \ifnumberingR \else
    \led@err@PstartNotNumbered
    \beginnumberingR
  \fi
  \ifnumberedpar@
    \led@err@PstartInPstart
    \pendR
  \fi
  \ifpst@rtedR\else
      \setcounter{pstartRold}{\value{pstartR}}%
    \list@clear{\inserts@listR}%
    \global\let\next@insertR=\empty
    \global\pst@rtedRtrue
  \fi
  \begingroup\normal@pars
  \global\advance\l@dnumpstartsR \@ne
  \ifnum\l@dnumpstartsR>\l@dc@maxchunks
    \led@err@TooManyPstarts
    \global\l@dnumpstartsR=\l@dc@maxchunks
  \fi
  \global\setnamebox{l@dRcolrawbox\the\l@dnumpstartsR}=\vbox\bgroup\ifautopar\else\ifnumberpstart\ifs
                      \hsize=\Rcolwidth
  \numberedpar@true
   \iflabelpstart\protected@edef\@currentlabel
       {\p@pstartR\thepstartR}\fi
  }
```

\pendL \pend must be used to end a numbered paragraph. Again we need a version that knows about left parallel texts.

```
\newcommand*{\pendL}{\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.

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

End the group that was begun in the `\pstart`.

```
\endgroup
\ignorespaces
\@oldnobreak
  \ifnumberpstart
  \addtocounter{pstartL}{1}
  \fi}

```

`\pendR` The version of `\pend` needed for right texts.

```
\newcommand*{\pendR}{\ifnumberingR \else
    \led@err@PendNotNumbered
  \fi
  \ifnumberedpar@ \else
    \led@err@PendNoPstart
  \fi
  \l@dzeropenalties
  \endgraf\global\num@linesR=\prevgraf\egroup
  \global\par@lineR=0
  \endgroup
  \ignorespaces
  \@oldnobreak
    \ifnumberpstart
    \addtocounter{pstartR}{1}
    \fi
}

```

14.2 Processing one line

For parallel texts we have to be able to process left and right lines independently. For sequential text we happily use the original `\do@line`. Otherwise ...

`\l@dleftbox` `\l@drightbox` A line of left text will be put in the box `\l@dleftbox`, and analagously for a line of right text.

```
\newbox\l@dleftbox
\newbox\l@drightbox

```

\countLline
\countRline

We need to know the number of lines processed.

```
\newcount\countLline
  \countLline \z@
\newcount\countRline
  \countRline \z@
```

\@donereallinesL
\@donetotallinesL
\@donereallinesR
\@donetotallinesR

We need to know the number of 'real' lines output (i.e., those that have been input by the user), and the total lines output (which includes any blank lines output for synchronisation).

```
\newcount\@donereallinesL
\newcount\@donetotallinesL
\newcount\@donereallinesR
\newcount\@donetotallinesR
```

\do@lineL

The \do@lineL macro is called to do all the processing for a single line of left text.

```
\newcommand*{\do@lineL}{%
  \advance\countLline \@ne
  \ifvbox\namebox{l@dLcolrawbox\the\l@dpscL}%
  {\vbadness=10000
   \splittopskip=\z@
   \do@lineLhook
   \l@demptyd@ta
   \global\setbox\one@line=\vsplit\namebox{l@dLcolrawbox\the\l@dpscL}
                            to\baselineskip}%
  \unvbox\one@line \global\setbox\one@line=\lastbox
  \getline@numL
\ifnum\@lock>\@ne\inserthangingsymboltrue\else\inserthangingsymbolfalse\fi
  \setbox\l@dleftbox
  \hb@xt@ \Lcolwidth{%
      \affixpstart@numL
    \affixline@num
    \l@dld@ta
    \add@inserts
    \affixside@note
    \l@dlsn@te
    {\ledllfill\hb@xt@ \wd\one@line{\do@insidelineLhook\inserthangingsymbolL\new@line\l@dunhbox@line{
     \l@drsn@te
  }}%
  \add@penaltiesL
  \global\advance\@donereallinesL\@ne
  \global\advance\@donetotallinesL\@ne
\else
  \setbox\l@dleftbox \hb@xt@ \Lcolwidth{\hspace*{\Lcolwidth}}%
  \global\advance\@donetotallinesL\@ne
\fi}
```

```
940
941
```

\do@lineLhook
\do@lineRhook
\do@insidelineLhook
\do@insidelineRhook

Hooks, initially empty, into the respective \do@line(L/R) macros.

```
942 \newcommand*{\do@lineLhook}{}
943 \newcommand*{\do@lineRhook}{}
944 \newcommand*{\do@insidelineLhook}{}
945 \newcommand*{\do@insidelineRhook}{}
946
```

\do@lineR

The \do@lineR macro is called to do all the processing for a single line of right text.

```
947 \newcommand*{\do@lineR}{%
948   \advance\countRline \@ne
949   \ifvbox\namebox{l@dRcolrawbox\the\l@dpscR}%
950   {\vbadness=10000
951    \splittopskip=\z@
952    \do@lineRhook
953    \l@demptyd@ta
954    \global\setbox\one@lineR=\vsplit\namebox{l@dRcolrawbox\the\l@dpscR}
955                             to\baselineskip}%
956   \unvbox\one@lineR \global\setbox\one@lineR=\lastbox
957   \getline@numR
958 \ifnum\@lockR>\@ne\inserthangingsymbolRtrue\else\inserthangingsymbolRfalse\fi
959   \setbox\l@drightbox
960   \hb@xt@ \Rcolwidth{%
961       \affixpstart@numR
962     \affixline@numR
963     \l@dld@ta
964     \add@insertsR
965     \affixside@noteR
966     \l@dlsn@te
967     {\correcthangingR\ledllfill\hb@xt@ \wd\one@lineR{\do@insidelineRhook\inserthangingsymb
968      \l@drsn@te
969   }}%
970   \add@penaltiesR
971   \global\advance\@donereallinesR\@ne
972   \global\advance\@donetotallinesR\@ne
973 \else
974   \setbox\l@drightbox \hb@xt@ \Rcolwidth{\hspace*{\Rcolwidth}}
975   \global\advance\@donetotallinesR\@ne
976 \fi}
977
978
```

14.3 Line and page number computation

\getline@numR

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

```
\newcommand*{\getline@numR}{%
  \global\advance\absline@numR \@ne
  \do@actionsR
  \do@ballastR
\ifnumberline
  \ifsublines@
    \ifnum\sub@lockR<\tw@
      \global\advance\subline@numR \@ne
    \fi
  \else
    \ifnum\@lockR<\tw@
      \global\advance\line@numR \@ne
      \global\subline@numR \z@
    \fi
  \fi
\fi
}
\newcommand*{\getline@numL}{%
    \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@ballastR The real work in the line macros above is done in \do@actions, but before we plunge into that, let's get \do@ballastR out of the way.

```
\newcommand*{\do@ballastR}{\global\ballast@count=\z@
  \begingroup
    \advance\absline@numR \@ne
    \ifnum\next@actionlineR=\absline@numR
      \ifnum\next@actionR>-1001
        \global\advance\ballast@count by -\c@ballast
      \fi
    \fi
  \endgroup}
```

\do@actionsR The \do@actionsR macro looks at the list of actions to take at particular right

\do@actions@fixedcodeR

\do@actions@nextR

text absolute line numbers, and does everything that's specified for the current line.

It may call itself recursively and we use tail recursion, via `\do@actions@nextR` for this.

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


\newcommand*{\do@actionsR}{%
  \global\let\do@actions@nextR=\relax
  \@l@dtempcntb=\absline@numR
  \ifnum\@l@dtempcntb<\next@actionlineR\else
    \ifnum\next@actionR>-1001\relax
      \global\page@numR=\next@actionR
      \ifbypage@R
        \global\line@numR \z@  \global\subline@numR \z@
      \fi
    \else
      \ifnum\next@actionR<-4999\relax   %  9/05 added relax here
        \@l@dtempcnta=-\next@actionR
        \advance\@l@dtempcnta by -5001\relax
        \ifsublines@
          \global\subline@numR=\@l@dtempcnta
        \else
```

```
          \global\line@numR=\@l@dtempcnta
        \fi
      \else
        \@l@dtempcnta=-\next@actionR
        \advance\@l@dtempcnta by -1000\relax
        \do@actions@fixedcodeR
      \fi
    \fi
    \ifx\actionlines@listR\empty
      \gdef\next@actionlineR{1000000}%
    \else
      \gl@p\actionlines@listR\to\next@actionlineR
      \gl@p\actions@listR\to\next@actionR
      \global\let\do@actions@nextR=\do@actionsR
    \fi
  \fi
  \do@actions@nextR}

```

14.4 Line number printing

\l@dcalcnum
\ch@cksub@l@ckR
\ch@ck@l@ckR
\f@x@l@cksR
\affixline@numR

\affixline@numR is the right text version of the \affixline@num macro.

```

\providecommand*{\l@dcalcnum}[3]{%
  \ifnum #1 > #2\relax
    \@l@dtempcnta = #1\relax
    \advance\@l@dtempcnta by -#2\relax
    \divide\@l@dtempcnta by #3\relax
    \multiply\@l@dtempcnta by #3\relax
    \advance\@l@dtempcnta by #2\relax
  \else
    \@l@dtempcnta=#2\relax
  \fi}

\newcommand*{\ch@cksub@l@ckR}{%
  \ifcase\sub@lockR
  \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}

\newcommand*{\ch@ck@l@ckR}{%
  \ifcase\@lockR
  \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}

\newcommand*{\f@x@l@cksR}{%
  \ifcase\@lockR
  \or
    \global\@lockR \tw@
  \or \or
    \global\@lockR \z@
  \fi
  \ifcase\sub@lockR
  \or
    \global\sub@lockR \tw@
  \or \or
    \global\sub@lockR \z@
  \fi}


\newcommand*{\affixline@numR}{%
\ifnumberline
\ifl@dskipnumber
  \global\l@dskipnumberfalse
\else
  \ifsublines@
    \@l@dtempcntb=\subline@numR
    \l@dcalcnum{\subline@numR}{\c@firstsublinenumR}{\c@sublinenumincrementR}%
    \ch@cksub@lockR
  \else
    \@l@dtempcntb=\line@numR
    \ifx\linenumberlist\empty
      \l@dcalcnum{\line@numR}{\c@firstlinenumR}{\c@linenumincrementR}%
    \else
      \@l@dtempcnta=\line@numR
      \edef\rem@inder{,\linenumberlist,\number\line@numR,}%
```

```
        \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
     \ch@ck@l@ckR
  \fi
  \ifnum\@l@dtempcnta=\@l@dtempcntb
    \if@twocolumn
      \if@firstcolumn
        \gdef\l@dld@ta{\llap{{\leftlinenumR}}}%
      \else
        \gdef\l@drd@ta{\rlap{{\rightlinenumR}}}%
      \fi
    \else
      \@l@dtempcntb=\line@marginR
      \ifnum\@l@dtempcntb>\@ne
        \advance\@l@dtempcntb by\page@numR
      \fi
      \ifodd\@l@dtempcntb
        \gdef\l@drd@ta{\rlap{{\rightlinenumR}}}%
      \else
        \gdef\l@dld@ta{\llap{{\leftlinenumR}}}%
      \fi
    \fi
  \fi
  \f@x@l@cksR
\fi
\fi}
```

14.5 Pstart number printing in side

The printing of the pstart number is like in eledmac, with two differences :

- Some commands have versions suffixed by R or L.
- The `\affixpstart@num` and `\affixpstart@numR` commands are called in the `\Pages` command. Consequently, the `pstartL` and `pstartR` counters must be reset at the begining of this command.

`\affixpstart@numL`
`\affixpstart@numR`
`\leftpstartnumR`
`\rightpstartnumR`
`\leftpstartnumL`
`\rightpstartnumL`
`\ifpstartnumR`

```

\newcommand*{\affixpstart@numL}{%
\ifsidepstartnum
\if@twocolumn
    \if@firstcolumn
      \gdef\l@dld@ta{\llap{{\leftpstartnumL}}}%
    \else
      \gdef\l@drd@ta{\rlap{{\rightpstartnumL}}}%
    \fi
```

```
    \else
      \@l@dtempcntb=\line@margin
     \ifnum\@l@dtempcntb>\@ne
       \advance\@l@dtempcntb \page@num
     \fi
     \ifodd\@l@dtempcntb
       \gdef\l@drd@ta{\rlap{{\rightpstartnumL}}}%
     \else
       \gdef\l@dld@ta{\llap{{\leftpstartnumL}}}%
     \fi
    \fi
\fi
}
\newcommand*{\affixpstart@numR}{%
\ifsidepstartnum
\if@twocolumn
     \if@firstcolumn
       \gdef\l@dld@ta{\llap{{\leftpstartnumR}}}%
     \else
       \gdef\l@drd@ta{\rlap{{\rightpstartnumR}}}%
     \fi
    \else
      \@l@dtempcntb=\line@marginR
     \ifnum\@l@dtempcntb>\@ne
       \advance\@l@dtempcntb \page@numR
     \fi
     \ifodd\@l@dtempcntb
       \gdef\l@drd@ta{\rlap{{\rightpstartnumR}}}%
     \else
       \gdef\l@dld@ta{\llap{{\leftpstartnumR}}}%
     \fi
    \fi
\fi
}

\newcommand*{\leftpstartnumL}{
\ifpstartnum
\thepstartL
\kern\linenumsep\global\pstartnumfalse\fi
}
\newcommand*{\rightpstartnumL}{
\ifpstartnum\kern\linenumsep
\thepstartL
\global\pstartnumfalse\fi
}
\newif\ifpstartnumR
\pstartnumRtrue
\newcommand*{\leftpstartnumR}{
\ifpstartnumR
\thepstartR
```

```
\kern\linenumsep\global\pstartnumRfalse\fi
}
\newcommand*{\rightpstartnumR}{
\ifpstartnumR\kern\linenumsep
\thepstartR
\global\pstartnumRfalse\fi
}
```

14.6 Add insertions to the vertical list

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

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

\add@insertsR The right text version.

\add@inserts@nextR

```
\newcommand*{\add@insertsR}{%
  \global\let\add@inserts@nextR=\relax
  \ifx\inserts@listR\empty \else
    \ifx\next@insertR\empty
      \ifx\insertlines@listR\empty
        \global\noteschanged@true
        \gdef\next@insertR{100000}%
      \else
        \gl@p\insertlines@listR\to\next@insertR
      \fi
    \fi
    \ifnum\next@insertR=\absline@numR
      \gl@p\inserts@listR\to\@insertR
      \@insertR
      \global\let\@insertR=\undefined
      \global\let\next@insertR=\empty
      \global\let\add@inserts@nextR=\add@insertsR
    \fi
  \fi
  \add@inserts@nextR}
```

14.7 Penalties

\add@penaltiesL \add@penaltiesR `\add@penaltiesL` is the last macro used by `\do@lineL`. 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 the code below, which is a virtual copy of the original `\add@penalties`, `\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`. Finally, the penalty is checked to see that it doesn't go below −10000.

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

This is for a single chunk. However, as we are probably dealing with several chunks at a time, the above is nor really relevant. I think that it is likely with parallel text that there is no real need to add back any penalties; even if there was, they would have to match across the left and right lines. So, I end up with the following.

```
\newcommand*{\add@penaltiesL}{}
\newcommand*{\add@penaltiesR}{}
```

14.8 Printing leftover notes

`\flush@notesR` The `\flush@notesR` macro is called after the entire right text has been sliced up and sent on to the vertical list.

```
\newcommand*{\flush@notesR}{%
  \@xloop
    \ifx\inserts@listR\empty \else
      \gl@p\inserts@listR\to\@insertR
      \@insertR
      \global\let\@insertR=\undefined
  \repeat}
```

15 Footnotes

15.1 Normal footnote formatting

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 **??**: 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.

`\printlinesR` This is the right text version of `\printlines` and takes account of `\Rlineflag`.
`\ledsavedprintlines` Just in case, `\ledsavedprintlines` is a copy of the original `\printlines`.

Just a reminder of the arguments:

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

```
\def\printlinesR#1|#2|#3|#4|#5|#6|#7|{\begingroup
  \setprintlines{#1}{#2}{#3}{#4}{#5}{#6}%
  \ifl@d@pnum #1\fullstop\fi
  \ifledplinenum \linenumr@p{#2}\Rlineflag\else \symplinenum\fi
  \ifl@d@ssub \fullstop \sublinenumr@p{#3}\fi
  \ifl@d@dash \endashchar\fi
  \ifl@d@pnum #4\fullstop\fi
  \ifl@d@elin \linenumr@p{#5}\Rlineflag\fi
  \ifl@d@esl \ifl@d@elin \fullstop\fi \sublinenumr@p{#6}\fi
\endgroup}

\let\ledsavedprintlines\printlines

```

16 Cross referencing

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

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

```

`\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.

```
\renewcommand*{\edlabel}[1]{\@bsphack
  \ifledRcol
    \write\linenum@outR{\string\@lab}%
    \ifx\labelref@listR\empty
      \xdef\label@refs{\zz@@@}%
    \else
```

```
      \gl@p\labelref@listR\to\label@refs
    \fi
    \ifvmode
        \advancelabel@refs
    \fi
    \protected@write\@auxout{}%
      {\string\l@dmake@labelsR\space\thepage|\label@refs|{#1}}%
  \else
    \write\linenum@out{\string\@lab}%
    \ifx\labelref@list\empty
      \xdef\label@refs{\zz@@@}%
    \else
      \gl@p\labelref@list\to\label@refs
    \fi
    \ifvmode
        \advancelabel@refs
    \fi
  \protected@write\@auxout{}%
    {\string\l@dmake@labels\space\thepage|\label@refs|{#1}}%
  \fi
  \@esphack}
```

`\l@dmake@labelsR` This is the right text version of `\l@dmake@labels`, taking account of `\Rlineflag`.

```
\def\l@dmake@labelsR#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\Rlineflag|#3}%
  \ignorespaces}
\AtBeginDocument{%
  \def\l@dmake@labelsR#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.

```
\renewcommand*{\@lab}{%
  \ifledRcol
    \xright@appenditem{\linenumr@p{\line@numR}|%
      \ifsublines@ \sublinenumr@p{\subline@numR}\else 0\fi}%
      \to\labelref@listR
  \else
    \xright@appenditem{\linenumr@p{\line@num}|%
      \ifsublines@ \sublinenumr@p{\subline@num}\else 0\fi}%
      \to\labelref@list
  \fi}
```

17 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.

\sidenote@marginR Specifies which margin sidenotes can be in.

\sidenotemargin

```
\newcount\sidenote@marginR
\renewcommand*{\sidenotemargin}[1]{{%
  \l@dgetsidenote@margin{#1}%
  \ifnum\@l@dtempcntb>\m@ne
    \ifledRcol
      \global\sidenote@marginR=\@l@dtempcntb
    \else
      \global\sidenote@margin=\@l@dtempcntb
    \fi
  \fi}}
\sidenotemargin{right}
\global\sidenote@margin=\@ne
```

\l@dlsnote The 'footnotes' for left, right, and moveable sidenotes. The whole scheme is reminiscent of the critical footnotes code.

\l@drsnote

\l@dcsnote

```
\renewcommand*{\l@dlsnote}[1]{%
  \begingroup%
  \newcommand{\content}{#1}%
  \ifnumberedpar@
    \ifledRcol%
      \xright@appenditem{\noexpand\vl@dlsnote{\csexpandonce{content}}}%
                        \to\inserts@listR
      \global\advance\insert@countR \@ne%
    \else%
      \xright@appenditem{\noexpand\vl@dlsnote{\csexpandonce{content}}}%
                        \to\inserts@list
      \global\advance\insert@count \@ne%
    \fi
  \fi\ignorespaces\endgroup}
\renewcommand*{\l@drsnote}[1]{%
  \begingroup%
  \newcommand{\content}{#1}%
  \ifnumberedpar@
    \ifledRcol%
      \xright@appenditem{\noexpand\vl@drsnote{\csexpandonce{content}}}%
                        \to\inserts@listR
       \global\advance\insert@countR \@ne%
    \else%
      \xright@appenditem{\noexpand\vl@drsnote{\csexpandonce{content}}}%
                        \to\inserts@list
      \global\advance\insert@count \@ne%
    \fi
  \fi\ignorespaces\endgroup}
```

```
\renewcommand*{\l@dcsnote}[1]{%
  \begingroup%
  \newcommand{\content}{#1}%
  \ifnumberedpar@
    \ifledRcol%
      \xright@appenditem{\noexpand\vl@dcsnote{\csexpandonce{content}}}%
                        \to\inserts@listR
       \global\advance\insert@countR \@ne%
    \else%
      \xright@appenditem{\noexpand\vl@dcsnote{\csexpandonce{content}}}%
                        \to\inserts@list
      \global\advance\insert@count \@ne%
    \fi
  \fi\ignorespaces\endgroup}

```

`\affixside@noteR` The right text version of `\affixside@note`.

```
\newcommand*{\affixside@noteR}{%
    \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 \th
  \gdef\@templ@d{}%
  \ifx\@templ@d\l@dcsnotetext \else%
    \if@twocolumn%
      \if@firstcolumn%
        \setl@dlp@rbox{\sidenotecontent@}%
      \else%
        \setl@drp@rbox{\sidenotecontent@}%
      \fi%
    \else%
      \@l@dtempcntb=\sidenote@marginR%
      \ifnum\@l@dtempcntb>\@ne%
        \advance\@l@dtempcntb by\page@num%
      \fi%
      \ifodd\@l@dtempcntb%
        \setl@drp@rbox{\sidenotecontent@t}%
      \else%
        \setl@dlp@rbox{\sidenotecontent@}%
      \fi%
    \fi%
  \fi}
```

18 Familiar footnotes

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

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

\normalbfnoteX

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

19 Verse

Like in eledmac, the insertion of hangingsymbol is base on `\ifinserthangingsymbol`, and, for the right side, on `\ifinserthangingsymbolR`.

\inserthangingsymbolL
\inserthangingsymbolR

```
\newif\ifinserthangingsymbolR
\newcommand{\inserthangingsymbolL}{%
\ifinserthangingsymbol%
```

```
    \ifinstanzaL%
        \hfill\hangingsymbol%
    \fi%
\fi}
\newcommand{\inserthangingsymbolR}{%
\ifinserthangingsymbolR%
    \ifinstanzaR%
        \hfill\hangingsymbol%
    \fi%
\fi}
```

When a verse is hanged, the column separator is shifted. To prevent it, the `\do@lineL` and `\do@lineR` commands call `\correcthangingL` and `\correcthangingR` commands. These commands insert horizontal skip which length is equal to the hang indent.

`\correcthangingL`
`\correcthangingR`

```
\newcommand{\correcthangingL}{%
\ifl@dpaging\else%
    \ifinstanzaL%
        \ifinserthangingsymbol%
            \hskip \@ifundefined{sza@0@}{0}{\expandafter%
                   \noexpand\csname sza@0@\endcsname}\stanzaindentbase%
        \fi%
    \fi%
\fi}

\newcommand{\correcthangingR}{%
\ifl@dpaging\else%
    \ifinstanzaR%
        \ifinserthangingsymbolR%
            \hskip \@ifundefined{sza@0@}{0}{\expandafter%
                   \noexpand\csname sza@0@\endcsname}\stanzaindentbase%
        \fi%
    \fi%
\fi}
```

Before we can define the main stanza macros we need to be able to save and reset the category code for &. To save the current value we use `\next` from the `\loop` macro.

```
  \chardef\next=\catcode`\&
  \catcode`\&=\active

```

`astanza` This is roughly an environmental form of `\stanza`, which treats its stanza-like contents as a single chunk.

```
\newenvironment{astanza}{%
  \startstanzahook
  \catcode`\&\active
```

```
  \global\stanza@count\@ne
  \ifnum\usenamecount{sza@0@}=\z@
    \let\stanza@hang\relax
    \let\endlock\relax
  \else
%%%    \interlinepenalty\@M % this screws things up, but I don't know why
    \rightskip\z@ plus 1fil\relax
  \fi
  \ifnum\usenamecount{szp@0@}=\z@
    \let\sza@penalty\relax
  \fi
  \def&{%
    \endlock\mbox{}%
    \sza@penalty
    \global\advance\stanza@count\@ne
    \@astanza@line}%
  \def\&{%
    \endlock\mbox{}
    \pend
    \endstanzaextra}%
  \pstart
  \@astanza@line
}{}

```

`\@astanza@line` This gets put at the start of each line in the environment. It sets up the paragraph style — each line is treated as a paragraph.

```
\newcommand*{\@astanza@line}{%
  \parindent=\csname sza@\number\stanza@count @\endcsname\stanzaindentbase
  \par
  \stanza@hang%\mbox{}%
  \ignorespaces}

```

Lastly reset the modified category codes.

```
  \catcode`\&=\next

```

20 Naming macros

The LaTeX kernel provides `\@namedef` and `\@namuse` for defining and using macros that may have non-letters in their names. We need something similar here as we are going to need and use some numbered boxes and counters.

`\newnamebox` `\setnamebox` `\unhnamebox` `\unvnamebox` `\namebox` A set of macros for creating and using 'named'boxes; the macros are called after the regular box macros, but including the string 'name'.

```
\providecommand*{\newnamebox}[1]{%
  \expandafter\newbox\csname #1\endcsname}

```

```
\providecommand*{\setnamebox}[1]{%
  \expandafter\setbox\csname #1\endcsname}
\providecommand*{\unhnamebox}[1]{%
  \expandafter\unhbox\csname #1\endcsname}
\providecommand*{\unvnamebox}[1]{%
  \expandafter\unvbox\csname #1\endcsname}
\providecommand*{\namebox}[1]{%
                      \csname #1\endcsname}

```

`\newnamecount` `\usenamecount` Macros for creating and using 'named' counts.

```
\providecommand*{\newnamecount}[1]{%
  \expandafter\newcount\csname #1\endcsname}
\providecommand*{\usenamecount}[1]{%
                        \csname #1\endcsname}

```

21 Counts and boxes for parallel texts

In sequential text, each chunk (that enclosed by `\pstart` ... `\pend`) is put into a box called `\raw@text` and then immediately printed, resulting in the box being emptied and ready for the next chunk. For parallel processing multiple boxes are needed as printing is delayed. We also need extra counters for various things.

`\maxchunks` `\l@dc@maxchunks` The maximum number of chunk pairs before printing has to be called for. The default is 5120 chunk pairs.

```
\newcount\l@dc@maxchunks
\newcommand{\maxchunks}[1]{\l@dc@maxchunks=#1}
  \maxchunks{5120}

```

`\l@dnumpstartsL` `\l@dnumpstartsR` The numbers of left and right chunks. `\l@dnumpstartsL` is defined in eledmac.

```
\newcount\l@dnumpstartsR

```

`\l@pscL` `\l@pscR` A couple of scratch counts for use in left and right texts, respectively.

```
\newcount\l@dpscL
\newcount\l@dpscR

```

`\l@dsetuprawboxes` This macro creates `\maxchunks` pairs of boxes for left and right chunks. The boxes are called `\l@dLcolrawbox1`, `\l@dLcolrawbox2`, etc.

```
\newcommand*{\l@dsetuprawboxes}{%
  \@l@dtempcntb=\l@dc@maxchunks
  \loop\ifnum\@l@dtempcntb>\z@
    \newnamebox{l@dLcolrawbox\the\@l@dtempcntb}
    \newnamebox{l@dRcolrawbox\the\@l@dtempcntb}
    \advance\@l@dtempcntb \m@ne
```

```
  \repeat}
```

`\l@dsetupmaxlinecounts` `\l@dzeromaxlinecounts`

To be able to synchronise left and right texts we need to know the maximum number of text lines there are in each pair of chunks. `\l@dsetupmaxlinecounts` creates `\maxchunks` new counts called `\l@dmaxlinesinpar1`, etc., and `\l@dzeromaxlinecounts` zeroes all of them.

```
\newcommand*{\l@dsetupmaxlinecounts}{%
  \@l@dtempcntb=\l@dc@maxchunks
  \loop\ifnum\@l@dtempcntb>\z@
    \newnamecount{l@dmaxlinesinpar\the\@l@dtempcntb}
    \advance\@l@dtempcntb \m@ne
  \repeat}
\newcommand*{\l@dzeromaxlinecounts}{%
  \begingroup
  \@l@dtempcntb=\l@dc@maxchunks
  \loop\ifnum\@l@dtempcntb>\z@
    \global\usenamecount{l@dmaxlinesinpar\the\@l@dtempcntb}=\z@
    \advance\@l@dtempcntb \m@ne
  \repeat
  \endgroup}
```

Make sure that all these are set up. This has to be done after the user has had an opportunity to change `\maxchunks`.

```
\AtBeginDocument{%
  \l@dsetuprawboxes
  \l@dsetupmaxlinecounts
  \l@dzeromaxlinecounts
  \l@dnumpstartsL=\z@
  \l@dnumpstartsR=\z@
  \l@dpscL=\z@
  \l@dpscR=\z@}
```

22 Fixing babel

With parallel texts there is the possibility that the two sides might use different languages via `babel`. On the other hand, `babel` might not be called at all (even though it might be already built into the format).

With the normal sequential text each line is initially typeset in the current language environment, and then it is output at which time its attachments are typeset (in the same language environment. In the parallel case lines are typeset in their current language but an attachment might be typeset outside the language environment of its line if the left and right side languages are different. To counter this, we have to make sure that the correct language is used at the proper times.

\ifl@dusedbabel
\l@dusedbabelfalse
\l@dusedbabeltrue

A flag for checking if babel has been used as a package.

```
\newif\ifl@dusedbabel
  \l@dusedbabelfalse
```

\ifl@dsamelang
\l@dsamelangfalse
\l@dsamelangtrue

A flag for checking if the same babel language has been used for both the left and right texts.

```
\newif\ifl@dsamelang
  \l@dsamelangtrue
```

\l@dchecklang

I'm going to use \theledlanguageL and \theledlanguageR to hold the names of the languages used for the left and right texts. This macro sets \ifl@dsamelang TRUE if they are the same, otherwise it sets it FALSE.

```
\newcommand*{\l@dchecklang}{%
  \l@dsamelangfalse
  \edef\@tempa{\theledlanguageL}\edef\@temp{\theledlanguageR}%
  \ifx\@tempa\@tempb
    \l@dsamelangtrue
  \fi}

```

\l@dbbl@set@language

In babel the macro \bbl@set@language{⟨*lang*⟩} does the work when the language ⟨*lang*⟩ is changed via \selectlanguage. Unfortunately for me, if it is given an argument in the form of a control sequence it strips off the \ character rather than expanding the command. I need a version that accepts an argument in the form \lang without it stripping the \.

```
\newcommand*{\l@dbbl@set@language}[1]{%
  \edef\languagename{#1}%
  \select@language{\languagename}%
  \if@filesw
    \protected@write\@auxout{}{\string\select@language{\languagename}}%
    \addtocontents{toc}{\string\select@language{\languagename}}%
    \addtocontents{lof}{\string\select@language{\languagename}}%
    \addtocontents{lot}{\string\select@language{\languagename}}%
  \fi}

```

The rest of the setup has to be postponed until the end of the preamble when we know if babel has been used or not. However, for now assume that it has not been used.

\selectlanguage
\l@duselanguage
\theledlanguageL
\theledlanguageR

\selectlanguage is a babel command. \theledlanguageL and \theledlanguageR are the names of the languages of the left and right texts. \l@duselanguage is similar to \selectlanguage.

```
\providecommand{\selectlanguage}[1]{}
\newcommand*{\l@duselanguage}[1]{}
\gdef\theledlanguageL{}
\gdef\theledlanguageR{}

```

Now do the babel fix or polyglossia, if necessary.

```
\AtBeginDocument{%
 \@ifundefined{xpg@main@language}{%
   \@ifundefined{bbl@main@language}{%
```

Either babel has not been used or it has been used with no specified language.

```
    \l@dusedbabelfalse
    \renewcommand*{\selectlanguage}[1]{}}{%
```

Here we deal with the case where babel has been used. \selectlanguage has to be redefined to use our version of \bbl@set@language and to store the left or right language.

```
    \l@dusedbabeltrue
    \let\l@doldselectlanguage\selectlanguage
    \let\l@doldbbl@set@language\bbl@set@language
    \let\bbl@set@language\l@dbbl@set@language
    \renewcommand{\selectlanguage}[1]{%
      \l@doldselectlanguage{#1}%
      \ifledRcol \gdef\theledlanguageR{#1}%
      \else      \gdef\theledlanguageL{#1}%
      \fi}
```

\l@duselanguage simply calls the original \selectlanguage so that \theledlanguageL and \theledlanguageR are unaltered.

```
    \renewcommand*{\l@duselanguage}[1]{%
      \l@doldselectlanguage{#1}}
```

Lastly, initialise the left and right languages to the current babel one.

```
    \gdef\theledlanguageL{\bbl@main@language}%
    \gdef\theledlanguageR{\bbl@main@language}%
    }%
  }
```

If on Polyglossia

```
 {    \apptocmd{\xpg@set@language}{%
       \ifledRcol \gdef\theledlanguageR{#1}%
       \else      \gdef\theledlanguageL{#1}%
       \fi}%
      \let\l@duselanguage\xpg@set@language
      \gdef\theledlanguageL{\xpg@main@language}%
      \gdef\theledlanguageR{\xpg@main@language}%
% \end{macrocode}
% That's it.
%    \begin{macrocode}
}}
```

23 Parallel columns

\Columns The \Columns command results in the previous Left and Right texts being typeset in matching columns. There should be equal numbers of chunks in the left and

right texts.

```
\newcommand*{\Columns}{%
  \setcounter{pstartL}{\value{pstartLold}}
  \setcounter{pstartR}{\value{pstartRold}}
  \ifnum\l@dnumpstartsL=\l@dnumpstartsR\else
    \led@err@BadLeftRightPstarts{\the\l@dnumpstartsL}{\the\l@dnumpstartsR}%
  \fi
```

Start a group and zero counters, etc.

```
  \begingroup
    \l@dzeropenalties
    \endgraf\global\num@lines=\prevgraf
            \global\num@linesR=\prevgraf
    \global\par@line=\z@
    \global\par@lineR=\z@
    \global\l@dpscL=\z@
    \global\l@dpscR=\z@
```

Check if there are chunks to be processed, and process them two by two (left and right pairs).

```
    \check@pstarts
    \loop\if@pstarts
        \global\pstartnumtrue
        \global\pstartnumRtrue
```

Increment `\l@dpscL` and `\l@dpscR` which here count the numbers of left and right chunks.

```
      \global\advance\l@dpscL \@ne
      \global\advance\l@dpscR \@ne
```

Check if there is text yet to be processed in at least one of the two current chunks, and also whether the left and right languages are the same

```
      \checkraw@text
      \l@dchecklang
{       \loop\ifaraw@text
```

Grab the next pair of left and right text lines and output them, swapping languages if they differ

```
          \ifl@dsamelang
            \do@lineL
            \do@lineR
          \else
            \l@duselanguage{\theledlanguageL}%
            \do@lineL
            \l@duselanguage{\theledlanguageR}%
            \do@lineR
          \fi
          \hb@xt@ \hsize{%
           \hfill \unhbox\l@dleftbox
            \hfill \columnseparator  \hfill
            \unhbox\l@drightbox
```

```
          }%
          \checkraw@text
        \repeat}
```

Having completed a pair of chunks, write the number of lines in each chunk to the respective section files. Increment pstart counters and reset line numbering if it's by pstart.

```
      \@writelinesinparL
      \@writelinesinparR
      \check@pstarts
            \ifbypstart@
          \write\linenum@out{\string\@set[1]}
          \resetprevline@
        \fi
       \ifbypstart@R
          \write\linenum@outR{\string\@set[1]}
          \resetprevline@
        \fi
      \addtocounter{pstartL}{1}
      \addtocounter{pstartR}{1}
    \repeat
```

Having output all chunks, make sure all notes have been output, then zero counts ready for the next set of texts. The boolean tests for stanza are switched to false.

```
    \flush@notes
    \flush@notesR
  \endgroup
  \global\l@dpscL=\z@
  \global\l@dpscR=\z@
  \global\l@dnumpstartsL=\z@
  \global\l@dnumpstartsR=\z@
  \ignorespaces
    \global\instanzaLfalse
    \global\instanzaRfalse}
```

`\columnseparator` `\columnrulewidth`

The separator between line pairs in parallel columns is in the form of a vertical rule extending a little below the baseline and with a height slightly greater than the `\baselineskip`. The width of the rule is `\columnrulewidth` (initially 0pt so the rule is invisible).

```
\newcommand*{\columnseparator}{%
  \smash{\rule[-0.2\baselineskip]{\columnrulewidth}{1.05\baselineskip}}}
\newdimen\columnrulewidth
  \columnrulewidth=\z@
```

`\if@pstarts` `\@pstartstrue` `\@pstartsfalse` `\check@pstarts`

`\check@pstarts` returns `\@pstartstrue` if there are any unprocessed chunks.

```
\newif\if@pstarts
\newcommand*{\check@pstarts}{%
```

```
  \@pstartsfalse
  \ifnum\l@dnumpstartsL>\l@dpscL
    \@pstartstrue
  \else
    \ifnum\l@dnumpstartsR>\l@dpscR
      \@pstartstrue
    \fi
  \fi
}
```

`\ifaraw@text` `\araw@texttrue` `\araw@textfalse` `\checkraw@text`

`\checkraw@text` checks whether the current Left or Right box is void or not. If one or other is not void it sets `\araw@texttrue`, otherwise both are void and it sets `\araw@textfalse`.

```
\newif\ifaraw@text
  \araw@textfalse
\newcommand*{\checkraw@text}{%
  \araw@textfalse
  \ifvbox\namebox{l@dLcolrawbox\the\l@dpscL}
    \araw@texttrue
  \else
    \ifvbox\namebox{l@dRcolrawbox\the\l@dpscR}
      \araw@texttrue
    \fi
  \fi
}
```

`\@writelinesinparL` `\@writelinesinparR`

These write the number of text lines in a chunk to the section files, and then afterwards zero the counter.

```
\newcommand*{\@writelinesinparL}{%
  \edef\next{%
    \write\linenum@out{\string\@pend[\the\@donereallinesL]}}%
  \next
  \global\@donereallinesL \z@}
\newcommand*{\@writelinesinparR}{%
  \edef\next{%
    \write\linenum@outR{\string\@pendR[\the\@donereallinesR]}}%
  \next
  \global\@donereallinesR \z@}
```

24 Parallel pages

This is considerably more complicated than parallel columns.

`\numpagelinesL` `\numpagelinesR` `\l@dminpagelines`

Counts for the number of lines on a left or right page, and the smaller of the number of lines on a pair of facing pages.

```
\newcount\numpagelinesL
\newcount\numpagelinesR
\newcount\l@dminpagelines
```

`\Pages` The `\Pages` command results in the previous Left and Right texts being typeset on matching facing pages. There should be equal numbers of chunks in the left and right texts.

```
\newcommand*{\Pages}{%
  \setcounter{pstartL}{\value{pstartLold}}
  \setcounter{pstartR}{\value{pstartRold}}
  \typeout{}
  \typeout{************************* PAGES *************************}
  \ifnum\l@dnumpstartsL=\l@dnumpstartsR\else
    \led@err@BadLeftRightPstarts{\the\l@dnumpstartsL}{\the\l@dnumpstartsR}%
  \fi
```

Get onto an empty even (left) page, then initialise counters, etc.

```
  \cleartol@devenpage
  \begingroup
    \l@dzeropenalties
    \endgraf\global\num@lines=\prevgraf
            \global\num@linesR=\prevgraf
    \global\par@line=\z@
    \global\par@lineR=\z@
    \global\l@dpscL=\z@
    \global\l@dpscR=\z@
    \writtenlinesLfalse
    \writtenlinesRfalse
```

Check if there are chunks to be processed.

```
    \check@pstarts
    \loop\if@pstarts
```

Loop over the number of chunks, incrementing the chunk counts (`\l@dpscL` and `\l@dpscR` are chunk (box) counts.)

```
      \global\advance\l@dpscL \@ne
      \global\advance\l@dpscR \@ne
```

Calculate the maximum number of real text lines in the chunk pair, storing the result in the relevant `\l@dmaxlinesinpar`.

```
      \getlinesfromparlistL
      \getlinesfromparlistR
      \l@dcalc@maxoftwo{\@cs@linesinparL}{\@cs@linesinparR}%
          {\usenamecount{l@dmaxlinesinpar\the\l@dpscL}}%
      \check@pstarts
    \repeat
```

Zero the counts again, ready for the next bit.

```
    \global\l@dpscL=\z@
    \global\l@dpscR=\z@
```

Get the number of lines on the first pair of pages and store the minumum in `\l@dminpagelines`.

```
    \getlinesfrompagelistL
    \getlinesfrompagelistR
    \l@dcalc@minoftwo{\@cs@linesonpageL}{\@cs@linesonpageR}%
        {\l@dminpagelines}%
```

Now we start processing the left and right chunks (`\l@dpscL` and `\l@dpscR` count the left and right chunks), starting with the first pair.

```
    \check@pstarts
    \if@pstarts
```

Increment the chunk counts to get the first pair.

```
      \global\advance\l@dpscL \@ne
      \global\advance\l@dpscR \@ne
```

We haven't processed any lines from these chunks yet, so zero the respective line counts.

```
      \global\@donereallinesL=\z@
      \global\@donetotallinesL=\z@
      \global\@donereallinesR=\z@
      \global\@donetotallinesR=\z@
```

Start a loop over the boxes (chunks).

```
      \checkraw@text
%      \begingroup
{       \loop\ifaraw@text
```

See if there is more that can be done for the left page and set up the left language.

```
          \checkpageL
          \l@duselanguage{\theledlanguageL}%
%%%          \begingroup
{            \loop\ifl@dsamepage

```

Process the next (left) text line, adding it to the page.

```
               \do@lineL
               \advance\numpagelinesL \@ne
               \ifshiftedpstarts
                       \ifdim\ht\l@dleftbox>0pt\hb@xt@ \hsize{\ledstrutL\unhbox\l@dleftbox
               \else
                       \hb@xt@ \hsize{\ledstrutL\unhbox\l@dleftbox}%
               \fi
```

Perhaps we have to move to the next (left) box. Check if we have got all we can onto the page. If not, repeat for the next line.

```

               \get@nextboxL
               \checkpageL
             \repeat
```

That (left) page has been filled. Output the number of real lines on the page — if the page break is because the page has been filled with lines, use the actual number, otherwise the page has been ended early in order to synchronise with the facing page so use an impossibly large number.

```
            \ifl@dpagefull
              \@writelinesonpageL{\the\numpagelinesL}%
            \else
              \@writelinesonpageL{1000}%
            \fi
```

Zero the left page lines count and clear the page to get onto the facing (odd, right) page.

```
            \numpagelinesL \z@
            \clearl@dleftpage }%
```

Now do the same for the right text.

```
          \checkpageR
          \l@duselanguage{\theledlanguageR}%
{            \loop\ifl@dsamepage
              \do@lineR
              \advance\numpagelinesR \@ne
              \ifshiftedpstarts
                      \ifdim\ht\l@drightbox>0pt\hb@xt@ \hsize{\ledstrutR\unhbox\l@drightbox}\fi%
              \else
                      \hb@xt@ \hsize{\ledstrutR\unhbox\l@drightbox}%
              \fi
              \get@nextboxR
              \checkpageR
            \repeat
            \ifl@dpagefull
              \@writelinesonpageR{\the\numpagelinesR}%
            \else
              \@writelinesonpageR{1000}%
            \fi
            \numpagelinesR=\z@
```

The page is full, so move onto the next (left, odd) page and repeat left text processing.

```
            \clearl@drightpage}
```

More to do? If there is we have to get the number of lines for the next pair of pages before starting to output them.

```
          \checkraw@text
          \ifaraw@text
            \getlinesfrompagelistL
            \getlinesfrompagelistR
            \l@dcalc@minoftwo{\@cs@linesonpageL}{\@cs@linesonpageR}%
                             {\l@dminpagelines}%
          \fi
        \repeat}
```

We have now output the text from all the chunks.

```
    \fi
```

Make sure that there are no inserts hanging around.

```
    \flush@notes
    \flush@notesR
  \endgroup
```

Zero counts ready for the next set of left/right text chunks. The boolean tests for stanza are switched to false.

```
  \global\l@dpscL=\z@
  \global\l@dpscR=\z@
  \global\l@dnumpstartsL=\z@
  \global\l@dnumpstartsR=\z@
    \global\instanzaLfalse
    \global\instanzaRfalse
  \ignorespaces}
```

`\ledstrutL` `\ledstrutR` Struts inserted into leftand right text lines.

```
\newcommand*{\ledstrutL}{\strut}
\newcommand*{\ledstrutR}{\strut}
```

`\cleartoevenpage` `\cleartol@devenpage` `\clearl@dleftpage` `\clearl@drightpage` `\cleartoevenpage`, which is defined in the memoir class, is like `\clear(double)page` except that we end up on an even page. `\cleartol@devenpage` is similar except that it first checks to see if it is already on an empty page. `\clearl@dleftpage` and `\clearl@drightpage` get us onto an odd and even page, respectively, checking that we end up on the immediately next page.

```
\providecommand{\cleartoevenpage}[1][\@empty]{%
  \clearpage
  \ifodd\c@page\hbox{}#1\clearpage\fi}
\newcommand*{\cleartol@devenpage}{%
  \ifdim\pagetotal<\topskip%  on an empty page
  \else
    \clearpage
  \fi
  \ifodd\c@page\hbox{}\clearpage\fi}
\newcommand*{\clearl@dleftpage}{%
  \clearpage
  \ifodd\c@page\else
    \led@err@LeftOnRightPage
    \hbox{}%
    \cleardoublepage
  \fi}
\newcommand*{\clearl@drightpage}{%
  \clearpage
  \ifodd\c@page
    \led@err@RightOnLeftPage
    \hbox{}%
```

```
    \cleartoevenpage
  \fi}
```

`\getlinesfromparlistL` `\getlinesfromparlistL` gets the next entry from the `\linesinpar@listL` and
`\@cs@linesinparL` puts it into `\@cs@linesinparL`; if the list is empty, it sets `\@cs@linesinparL` to
`\getlinesfromparlistR` 0. Similarly for `\getlinesfromparlistR`.
`\@cs@linesinparR`

```
\newcommand*{\getlinesfromparlistL}{%
  \ifx\linesinpar@listL\empty
    \gdef\@cs@linesinparL{0}%
  \else
    \gl@p\linesinpar@listL\to\@cs@linesinparL
  \fi}
\newcommand*{\getlinesfromparlistR}{%
  \ifx\linesinpar@listR\empty
    \gdef\@cs@linesinparR{0}%
  \else
    \gl@p\linesinpar@listR\to\@cs@linesinparR
  \fi}
```

`\getlinesfrompagelistL` `\getlinesfrompagelistL` gets the next entry from the `\linesonpage@listL` and
`\@cs@linesonpageL` puts it into `\@cs@linesonpageL`; if the list is empty, it sets `\@cs@linesonpageL`
`\getlinesfrompagelistR` to 1000. Similarly for `\getlinesfrompagelistR`.
`\@cs@linesonpageR`

```
\newcommand*{\getlinesfrompagelistL}{%
  \ifx\linesonpage@listL\empty
    \gdef\@cs@linesonpageL{1000}%
  \else
    \gl@p\linesonpage@listL\to\@cs@linesonpageL
  \fi}
\newcommand*{\getlinesfrompagelistR}{%
  \ifx\linesonpage@listR\empty
    \gdef\@cs@linesonpageR{1000}%
  \else
    \gl@p\linesonpage@listR\to\@cs@linesonpageR
  \fi}
```

`\@writelinesonpageL` These macros output the number of lines on a page to the section file in the form
`\@writelinesonpageR` of `\@lopL` or `\@lopR` macros.

```
\newcommand*{\@writelinesonpageL}[1]{%
  \edef\next{\write\linenum@out{\string\@lopL{#1}}}%
  \next}
\newcommand*{\@writelinesonpageR}[1]{%
  \edef\next{\write\linenum@outR{\string\@lopR{#1}}}%
  \next}
```

`\l@dcalc@maxoftwo` `\l@dcalc@maxoftwo{`⟨*num*⟩`}{`⟨*num*⟩`}{`⟨*count*⟩`}` sets ⟨*count*⟩ to the maximum of
`\l@dcalc@minoftwo` the two ⟨*num*⟩.

Similarly `\l@dcalc@minoftwo{`⟨*num*⟩`}{`⟨*num*⟩`}{`⟨*count*⟩`}` sets ⟨*count*⟩ to the minimum of the two ⟨*num*⟩.

```
\newcommand*{\l@dcalc@maxoftwo}[3]{%
  \ifnum #2>#1\relax
    #3=#2\relax
  \else
    #3=#1\relax
  \fi}
\newcommand*{\l@dcalc@minoftwo}[3]{%
  \ifnum #2<#1\relax
    #3=#2\relax
  \else
    #3=#1\relax
  \fi}
```

`\ifl@dsamepage` `\l@dsamepagetrue` `\l@dsamepagefalse` `\ifl@dpagefull` `\l@dpagefulltrue` `\l@dpagefullfalse` `\checkpageL` `\checkpageR`

`\checkpageL` tests if the space and lines already taken on the page by text and footnotes is less than the constraints. If so, then `\ifl@dpagefull` is set FALSE and `\ifl@dsamepage` is set TRUE. If the page is spatially full then `\ifl@dpagefull` is set TRUE and `\ifl@dsamepage` is set FALSE. If it is not spatially full but the maximum number of lines have been output then both `\ifl@dpagefull` and `\ifl@dsamepage` are set FALSE.

```
\newif\ifl@dsamepage
  \l@dsamepagetrue
\newif\ifl@dpagefull
\newcommand*{\checkpageL}{%
  \l@dpagefulltrue
  \l@dsamepagetrue
  \check@goal
  \ifdim\pagetotal<\ledthegoal
    \ifnum\numpagelinesL<\l@dminpagelines
    \else
      \l@dsamepagefalse
      \l@dpagefullfalse
    \fi
  \else
    \l@dsamepagefalse
    \l@dpagefulltrue
  \fi}
\newcommand*{\checkpageR}{%
  \l@dpagefulltrue
  \l@dsamepagetrue
  \check@goal
  \ifdim\pagetotal<\ledthegoal
    \ifnum\numpagelinesR<\l@dminpagelines
    \else
      \l@dsamepagefalse
      \l@dpagefullfalse
    \fi
```

```
  \else
    \l@dsamepagefalse
    \l@dpagefulltrue
  \fi}
```

`\ledthegoal` `\goalfraction` `\check@goal` `\ledthegoal` is the amount of space allowed to taken by text and footnotes on a page before a forced pagebreak. This can be controlled via `\goalfraction`. `\ledthegoal` is calculated via `\check@goal`.

```
\newdimen\ledthegoal
\ifshiftedpstarts
       \newcommand*{\goalfraction}{0.95}
\else
       \newcommand*{\goalfraction}{0.9}
\fi

\newcommand*{\check@goal}{%
  \ledthegoal=\goalfraction\pagegoal}
```

`\ifwrittenlinesL` `\ifwrittenlinesL` Booleans for whether line data has been written to the section file.

```
\newif\ifwrittenlinesL
\newif\ifwrittenlinesR
```

`\get@nextboxL` `\get@nextboxR` If the current box is not empty (i.e., still contains some lines) nothing is done. Otherwise if and only if a synchronisation point is reached the next box is started.

```
\newcommand*{\get@nextboxL}{%
  \ifvbox\namebox{l@dLcolrawbox\the\l@dpscL}% box is not empty
```

The current box is not empty; do nothing.

```
  \else%                                          box is empty
```

The box is empty; check if enough lines (real and blank) have been output.

```
    \ifnum\usenamecount{l@dmaxlinesinpar\the\l@dpscL}>\@donetotallinesL
    \else
```

Sufficient lines have been output.

```
      \ifwrittenlinesL
      \else
```

Write out the number of lines done, and set the boolean so this is only done once.

```
        \@writelinesinparL
        \writtenlinesLtrue
      \fi
      \ifnum\l@dnumpstartsL>\l@dpscL
```

There are still unprocessed boxes. Recalculate the maximum number of lines needed, and move onto the next box (by incrementing `\l@dpscL`). If needed, restart the line numbering. Increment the pstartL counter.

```
      \writtenlinesLfalse
       \ifbypstart@
        \ifnum\value{pstartL}<\value{pstartLold}
        \else
          \global\line@num=0
          \resetprevline@
        \fi
        \fi
      \addtocounter{pstartL}{1}
      \global\pstartnumtrue
      \l@dcalc@maxoftwo{\the\usenamecount{l@dmaxlinesinpar\the\l@dpscL}}%
                       {\the\@donetotallinesL}%
                       {\usenamecount{l@dmaxlinesinpar\the\l@dpscL}}%
      \global\@donetotallinesL \z@
      \global\advance\l@dpscL \@ne
    \fi
  \fi
\fi}
\newcommand*{\get@nextboxR}{%
  \ifvbox\namebox{l@dRcolrawbox\the\l@dpscR}% box is not empty
  \else%                                      box is empty
    \ifnum\usenamecount{l@dmaxlinesinpar\the\l@dpscR}>\@donetotallinesR
    \else
      \ifwrittenlinesR
      \else
        \@writelinesinparR
        \writtenlinesRtrue
      \fi
      \ifnum\l@dnumpstartsR>\l@dpscR
        \writtenlinesRfalse
        \ifbypstart@R
            \ifnum\value{pstartR}<\value{pstartRold}
            \else
               \global\line@numR=0
               \resetprevline@
            \fi
        \fi
        \addtocounter{pstartR}{1}
        \global\pstartnumRtrue
        \l@dcalc@maxoftwo{\the\usenamecount{l@dmaxlinesinpar\the\l@dpscR}}%
                         {\the\@donetotallinesR}%
                         {\usenamecount{l@dmaxlinesinpar\the\l@dpscR}}%
        \global\@donetotallinesR \z@
        \global\advance\l@dpscR \@ne
      \fi
    \fi
  \fi}
```

25 The End

¡/code¿

References

[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`)

[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. *eledmac A presumptuous attempt to port EDMAC,TABMAC and EDSTANZA to LaTeX*. December 2004. (Available from CTAN in `macros/latex/contrib/eledmac`)

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.

D

E

F

G

L

M

N

O

P

R

S

T

U

V

W

X

Z

Change History