

The Stellar Command Module
for
Integrating Astronomy and Art
User Guide
Angelo Fraietta
University of New South Wales

© 2018 — 2019 Angelo Fraietta

All rights reserved

First edition: 6 June 2019

Short contents

Short contents	· iii
Contents	· v
List of Figures	· ix
List of Tables	· xi
List of typeset examples	· xii
Preface	· xiii
Introduction	· xv
Terminology	· xvii
1 Starting off	· 1
2 Laying out the page	· 3
3 Poetry	· 11
4 Boxes, verbatims and files	· 27
5 Cross referencing	· 49
6 Back matter	· 53
7 Miscellaneous	· 73
8 For package users	· 97
9 An example book design	· 101
10 An example thesis design	· 109
A Packages and macros	· 131

B	Showcases	· 135
C	Snippets	· 153
D	Pictures	· 161
E	LaTeX and TeX	· 181
F	The terrors of errors	· 199
G	Comments	· 223
	Notes	· 225
	Bibliography	· 227

Contents

Short contents	iii
Contents	v
List of Figures	ix
List of Tables	xi
List of typeset examples	xii
Preface	xiii
Introduction	xv
Terminology	xvii
Units of measurement	xviii
1 Starting off	1
2 Laying out the page	3
2.1 Side footnotes	4
2.1.1 Bottom aligned side footnotes 4, 2.1.2 Setting the layout for <code>\sidefootnote</code> 5, 2.1.3 Styling <code>\sidefootnote</code> 5, 2.1.4 Side footnote example 6	
2.2 Endnotes	7
2.2.1 Changing the appearance 9, 2.2.2 Epigraphs on book or part pages 9	
3 Poetry	11
3.1 Classy verse	13
3.1.1 Indented lines 17, 3.1.2 Numbering 18	
3.2 Titles	18
3.2.1 Main Poem Title layout parameters 19, 3.2.2 Detailed Poem Title layout parameters 20	
3.3 Examples	21
<i>A Limerick 21, Love's lost 21, Fleas 22, In the beginning 22, Mathematics 23, The Young Lady of Ryde 24, Clementine 24, Mouse's Tale 26</i>	

4	Boxes, verbatims and files	27
4.1	Boxes	28
4.2	Long comments	32
4.3	Verbatims	33
	4.3.1 Boxed verbatims 35, 4.3.2 New verbatims 37, 4.3.3 Example: the <code>lcode</code> environment 38	
4.4	Files	40
	4.4.1 Writing to a file 41, 4.4.2 Reading from a file 41, 4.4.3 Example: end-notes 42, 4.4.4 Example: end floats 42, 4.4.5 Example: questions and answers 45	
4.5	Answers	48
5	Cross referencing	49
5.1	Labels and references	49
5.2	Reference by name	50
6	Back matter	53
6.1	Bibliography	53
	6.1.1 BibTex 55	
6.2	Index	56
	6.2.1 Printing an index 56, 6.2.2 Preparing an index 57, 6.2.3 MakeIndex 60, 6.2.4 Controlling MakeIndex output 63, 6.2.5 Indexing and the natbib package 66	
6.3	Glossaries	66
	6.3.1 Controlling the glossary 67	
7	Miscellaneous	73
	<i>In which we talk of many things, but not shoes or ships or sealing wax, nor cabbages and kings.</i>	
7.1	Draft documents	73
7.2	Change marks	73
7.3	Trim marks	74
7.4	Sheet numbering	76
7.5	Gatherings or signatures	76
7.6	Time	77
7.7	Page breaks before lists	77
7.8	Changing counters	77
7.9	New new and provide commands	78
7.10	Changing macros	79
7.11	String arguments	80
7.12	Odd/even page checking	81
7.13	Moving to another page	81
7.14	Number formatting	82
	7.14.1 Numeric numbers 83, 7.14.2 Named numbers 84, 7.14.3 Fractions 86	
7.15	An array data structure	87
7.16	Checking the processor	88

	7.16.1 Checking for pdfLaTeX 88, 7.16.2 Checking for etex 88, 7.16.3 Checking for XeTeX 88, 7.16.4 Checking for LuaTeX 89	
7.17	Leading	89
7.18	Minor space adjustment	89
7.19	Adding a period	89
7.20	Words and phrases	90
7.21	Symbols	91
7.22	Two simple macros	91
7.23	Vertical centering	91
7.24	For package writers	92
	7.24.1 Emulating packages 92, 7.24.2 Inserting code before and after a file, package or class 92	
7.25	Heading hooks	94
7.26	Documenting LaTeX commands	96
8	For package users	97
8.1	Class/package name clash	97
8.2	Support for bidirectional typesetting	98
9	An example book design	101
9.1	Introduction	101
9.2	Design requirements	101
9.3	Specifying the page and typeblock	102
9.4	Specifying the sectional titling styles	104
	9.4.1 The chapter style 104, 9.4.2 Lower level divisions 104	
9.5	Specifying the pagestyle	105
9.6	Captions and the ToC	107
9.7	Preamble or package?	107
10	An example thesis design	109
10.1	Example US thesis typographic requirements	109
	10.1.1 General 109, 10.1.2 Preliminary matter 110, 10.1.3 Table of contents 111, 10.1.4 Lists 112, 10.1.5 Main text 112, 10.1.6 Backmatter 113	
10.2	Code	113
	10.2.1 Initialisation 114, 10.2.2 Page layout 114, 10.2.3 Page styles 116, 10.2.4 The ToC and friends 117, 10.2.5 Chapter styling 118, 10.2.6 Section, etc., styling 118, 10.2.7 Captions 119, 10.2.8 The bibliography 119, 10.2.9 End notes 119, 10.2.10 Preliminary headings 120, 10.2.11 Components of the title and approval pages 121, 10.2.12 The title and approval pages 121, 10.2.13 The last bits 125	
10.3	Usage	125
10.4	Comments	127
A	Packages and macros	131
A.1	Packages	131
A.2	Macros	132

B	Showcases	135
B.1	Chapter styles	135
	<i>B.1.1 Chappell 148, B.1.2 Demo, Demo2 and demo3 149, B.1.3 Pedersen 149, B.1.4 Southall 150, B.1.5 Veelo 151</i>	
C	Snippets	153
	Snippet C.1 (Mirroring the output)	153
	Snippet C.2 (Remove pagenumber if only one page)	154
	Snippet C.3 (A kind of draft note)	154
	Snippet C.4 (Adding indentation to footnotes)	155
	Snippet C.5 (Background image and trimmarks)	155
	Snippet C.6 (Autoadjusted number widths in the ToC)	155
	Snippet C.7 (Using class tools to make a chapter ToC)	157
	Snippet C.8 (An appendix ToC)	159
D	Pictures	161
D.1	Basic principles	161
D.2	Picture objects	163
	<i>D.2.1 Text 163, D.2.2 Boxes 164, D.2.3 Lines 169, D.2.4 Arrows 171, D.2.5 Circles 171</i>	
D.3	Repetitions	174
D.4	Bezier curves	177
E	LaTeX and TeX	181
E.1	The TeX process	182
E.2	LaTeX files	183
E.3	Syntax	184
E.4	(La)TeX commands	185
E.5	Calculation	188
	<i>E.5.1 Numbers 188, E.5.2 Lengths 189</i>	
E.6	Programming	193
F	The terrors of errors	199
F.1	TeX messages	200
	<i>F.1.1 TeX capacity exceeded 208</i>	
F.2	LaTeX errors	210
F.3	LaTeX warnings	214
F.4	Class errors	217
F.5	Class warnings	220
G	Comments	223
G.1	Algorithms	223
	<i>G.1.1 Autoadjusting \marginparwidth 223</i>	
Notes		225
	Chapter 2 Laying out the page	225
Bibliography		227

List of Figures

2.1	Example endnote listing	8
6.1	Raw indexing: (left) index commands in the source text; (right) <code>idx</code> file entries	60
6.2	Processed index: (left) alphabeticized <code>ind</code> file; (right) <code>typeset</code> index	61
10.1	Example Archibald Smythe University title page	122
10.2	Example Archibald Smythe University approval page	123
B.1	The default chapterstyle	135
B.2	The section chapterstyle	136
B.3	The hangnum chapterstyle	136
B.4	The companion chapterstyle	137
B.5	The article chapterstyle	137
B.6	The bianchi chapterstyle	138
B.7	The bringhurst chapterstyle	138
B.8	The brotherton chapterstyle	139
B.9	The chappell chapterstyle	139
B.10	The crosshead chapterstyle	140
B.11	The culver chapterstyle	140
B.12	The dash chapterstyle	140
B.13	The demo2 chapterstyle	141
B.14	The dowing chapterstyle	141
B.15	The ell chapterstyle	142
B.16	The ger chapterstyle	142
B.17	The komalike chapterstyle	143
B.18	The lyhne chapterstyle. This style requires the <code>graphicx</code> package	143
B.19	The madsen chapterstyle. This style requires the <code>graphicx</code> package	144
B.20	The ntglke chapterstyle	145
B.21	The southall chapterstyle	145
B.22	The tandh chapterstyle	146
B.23	The thatcher chapterstyle	146
B.24	The veelo chapterstyle. This style requires the <code>graphicx</code> package	147
B.25	The verville chapterstyle	147
B.26	The wilsondob chapterstyle	148
D.1	Some points in the cartesian coordinate system	162

D.2	Specification of a line or arrow	171
D.3	Sloping lines and arrows	172
D.4	Some measuring scales	175
D.5	Two Bezier curves	179

List of Tables

1	Traditional font size designations	xviii
6.1	MakeIndex configuration file input parameters	61
6.2	MakeIndex configuration file output parameters	64
7.1	Defined words and phrases	90
E.1	Some internal macros for numbers	189

List of typeset examples

3.1	Phantom text in verse	16
3.2	Verse with regular quote marks	17
3.3	Verse with hanging left quote marks	17
5.1	Named references should be to titled elements	51
5.2	Current title	52
7.1	TeX's minimum number in words (English style)	84
7.2	TeX's maximum number in words (American style)	85
7.3	Varieties of fractions in text	86
7.4	Super- and subscripts in text	87
D.1	Picture: text	164
D.2	Picture: text in boxes	165
D.3	Picture: positioning text	166
D.4	Picture: dashed box	166
D.5	Picture: framing	167
D.6	Picture: stacking	168
D.7	Picture: saved boxes	169
D.8	Picture: circles	172
D.9	Picture: ovals	173
D.10	Picture: text in oval	174
D.11	Picture: repetitions	177

Preface

From personal experience and also from lurking on the `comp.text.tex` newsgroup the major problems with using LaTeX are related to document design. Some years ago most questions on CTT were answered by someone providing a piece of code that solved a particular problem, and again and again. More recently these questions are answered along the lines of ‘Use the `———` package’, and again and again.

I have used many of the more common of these packages but my filing system is not always well ordered and I tend to mislay the various user manuals, even for the packages I have written. The memoir class is an attempt to integrate some of the more design-related packages with the LaTeX book class. I chose the book class as the report class is virtually identical to book, except that book does not have an `abstract` environment while report does; however it is easy to fake an `abstract` if it is needed. With a little bit of tweaking, book class documents can be made to look just like article class documents, and the memoir class is designed with tweaking very much in mind.

The memoir class effectively incorporates the facilities that are usually accessed by using external packages. In most cases the class code is new code reimplementing package functionalities. The exceptions tend to be where I have cut and pasted code from some of my packages. I could not have written the memoir class without the excellent work presented by the implementors of LaTeX and its many packages.

Apart from packages that I happen to have written I have gained many ideas from the other packages listed in the Bibliography. One way or another their authors have all contributed, albeit unknowingly. The participants in the `comp.text.tex` newsgroup have also provided valuable input, partly by questioning how to do something in LaTeX, and partly by providing answers. It is a friendly and educational forum.

PETER WILSON
Seattle, WA
June 2001

Introduction

Terminology

Like all professions and trades, typographers and printers have their specialised vocabulary.

First there is the question of pages, leaves and sheets. The trimmed sheets of paper that make up a book are called *leaves*, and I will call the untrimmed sheets the *stock* material. A leaf has two sides, and a *page* is one side of a leaf. If you think of a book being opened flat, then you can see two leaves. The front of the righthand leaf, is called the *recto* page of that leaf, and the side of the lefthand leaf that you see is called the *verso* page of that leaf. So, a leaf has a recto and a verso page. Recto pages are the odd-numbered pages and verso pages are even-numbered.

Then there is the question of folios. The typographical term for the number of a page is *folio*. This is not to be confused with the same term as used in ‘Shakespeare’s First Folio’ where the reference is to the height and width of the book, nor to its use in the phrase ‘*folio* signature’ where the term refers to the number of times a printed sheet is folded. Not every page in a book has a printed folio, and there may be pages that do not have a folio at all. Pages with folios, whether printed or not, form the *pagination* of the book. Pages that are not counted in the pagination have no folios.

I have not been able to find what I think is a good definition for ‘type’ as it seems to be used in different contexts with different meanings. It appears to be a kind of generic word; for instance there are type designers, type cutters, type setters, type foundries,... For my purposes I propose that *type* is one or more printable characters (or variations or extensions to this idea). Printers use the term *sort* to refer to one piece of lead type.

A *typeface* is a set of one or more fonts, in one or more sizes, designed as a stylistic whole.

A *font* is a set of characters. In the days of metal type and hot lead a font meant a complete alphabet and auxiliary characters in a given size. More recently it is taken to mean a complete set of characters regardless of size. A font of roman type normally consists of CAPITAL LETTERS, SMALL CAPITALS, lowercase letters, numbers, punctuation marks, ligatures (such as ‘fi’ and ‘ffi’), and a few special symbols like &.

A *font family* is a set of fonts designed to work harmoniously together, such as a pair of roman and italic fonts.

The size of a font is expressed in points (72.27 points equals 1 inch equals 25.4 millimeters). The size is a rough indication of the height of the tallest character, but different fonts with the same size may have very different actual heights. Traditionally font sizes were referred to by names (see Table 1) but nowadays just the number of points is used.

The typographers’ and printers’ term for the vertical space between the lines of normal text is *leading*, which is also usually expressed in points and is usually larger than the font

Table 1: Traditional font size designations

Points	Name	Points	Name
3	Excelsior	11	Small Pica
3½	Brilliant	12	Pica
4	Diamond	14	English
5	Pearl	18	Great Primer
5½	Agate	24	Double (or Two Line) Pica
6	Nonpareil	28	Double (or Two Line) English
6½	Mignonette	36	Double (or Two Line) Great Primer
7	Minion	48	French Canon (or Four Line Pica)
8	Brevier	60	Five Line Pica
9	Bourgeois	72	Six line Pica
10	Long Primer	96	Eight Line Pica

size. A convention for describing the font and leading is to give the font size and leading separated by a slash; for instance 10/12 for a 10pt font set with a 12pt leading, or 12/14 for a 12pt font set with a 14pt leading.

The normal length of a line of text is often called the *measure* and is normally specified in terms of picas where 1 pica equals 12 points (1pc = 12pt).

Documents may be described as being typeset with a particular font with a particular size and a particular leading on a particular measure; this is normally given in a shorthand form. A 10pt font with 11pt leading on a 20pc measure is described as 10/11 × 20, and 14/16 × 22 describes a 14pt font with 16pt leading set on a 22pc measure.

UNITS OF MEASUREMENT

Typographers and printers use a mixed system of units, some of which we met above. The fundamental unit is the point; Table ?? lists the most common units employed.

One

Starting off

Two

Laying out the page

You may run into problems if the `\sidebar` command comes near a pagebreak, or if the sidebar text gets typeset alongside main text that has non-uniform line spacing (like around a `\section`). Further, the contents of sidebars may not be typeset if they are too near to the end of the document.

`\sidebarwidth \sidebarhsep \sidebarvsep`

The `\text` of a `\sidebar` is typeset in a column of width `\sidebarwidth` and there is a horizontal gap of `\sidebarhsep` between the main text and the sidebar. The length `\sidebarvsep` is the vertical gap between sidebars that fall on the same page; it also has a role in controlling the start of sidebars with respect to the top of the page.

`\sidebartopsep`
`\setsidebarheight{<height>}`

The length `\sidebartopsep` controls the vertical position of the top of a sidebar. The default is 0pt which aligns it with the top of the typeblock. The command `\setsidebarheight` sets the height of sidebars to `<height>`, without making any allowance for `\sidebartopsep`. The `<length>` argument should be equivalent to an integral number of lines. For example:

```
\setsidebarheight{15\onelineskip}
```

The default is the `\textheight`.

Perhaps you would like sidebars to start two lines below the top of the typeblock but still end at the bottom of the typeblock? If so, and you are using the `calc` package [TJ05], then the following will do the job:

```
\setlength{\sidebartopskip}{2\onelineskip}
\setsidebarheight{\textheight-\sidebartopskip}
```

The alignment of the text in a sidebar with the main text may not be particularly good and you may wish to do some experimentation (possibly through a combination of `\sidebarvsep` and `\setsidebarheight`) to improve matters.

Although you can set the parameters for your sidebars individually it is more efficient to use the `\setsidebars` command; it *must* be used if you change the font and/or the height.

`\setsidebars{<hsep>}{<width>}{<vsep>}{<topsep>}{}{<height>}`

The `\setsidebars` command can be used to set the sidebar parameters. `\sidebarhsep` is set to `<hsep>`, `\sidebarwidth` is set to `<width>`, `\sidebarvsep` is set

to $\langle vsep \rangle$, `\sidebartopsep` is set to $\langle topsep \rangle$, `\sidebarfont` is set to $\langle font \rangle$, and finally `\setsidebarheight` is used to set the height to $\langle height \rangle$. The default is:

```
\setsidebars{\marginparsep}{\marginparwidth}{\onelineskip}%
             {0pt}{\normalsize\normalfont}{\textheight}
```

Any, or all, of the arguments can be a `*`, in which case the parameter corresponding to that argument is unchanged. Repeating the above example of changing the topskip and the height, assuming that the other defaults are satisfactory except that the width should be 3cm and an italic font should be used:

```
\setsidebars*{3cm}{*}{2\onelineskip}{\itshape}%
             {\textheight-\sidebartopsep}
```

Changing the marginpar parameters, for example with `\setmarginnotes`, will not affect the sidebar parameters.

Note that `\checkandfixthelayout` neither checks nor fixes any of the sidebar parameters. This means, for instance, that if you change the `\textheight` from its default value and you want sidebars to have the same height then after changing the `\textheight` you have to call `\checkandfixthelayout` and then call `\setsidebars` with the (new) `\textheight`. For instance:

```
...
\settypeblocksize{40\baselineskip}{5in}{*}
...
\checkandfixthelayout
\setsidebars{...}{...}{...}{...}{...}{\textheight}
```

Unfortunately if a sidebar is on a double column page that either includes a double column float or starts a new chapter then the top of the sidebar comes below the float or the chapter title. I have been unable to eliminate this ‘feature’.

2.1 SIDE FOOTNOTES

Besides three already mentioned macros for writing in the margin (`\marginpar`, `\sidepar`, and `\sidebar`) memoir also provide a functionality to add side footnotes. Actually two ways: one is to internally make `\footnote` use `\marginpar` to write in the margin, the other is to collect all side footnotes bottom up in the margin.

<pre>\footnotesatfoot \footnotesinmargin</pre>
--

`\footnotesatfoot` (the default) causes `\footnote` to place its text at the bottom of the page. By issuing `footnotesinmargin \footnote` (and friends like `\footnotetext`) will internally use `\marginpar` to write the footnote to the page.

2.1.1 Bottom aligned side footnotes

Bottom aligned footnotes works just like regular footnotes, just with a separate macro `\sidefootnote{ $\langle text \rangle$ }`, and here the side footnotes are placed at the bottom of the specified margin (more or like as if one had taken the footnotes from the bottom of the page and moved it to the margin instead). All the major functionality is the same as for the normal `\footnote` command.¹

¹`\sidefootnote` does not make sense inside minipages...

```
\sidefootnote[⟨num⟩]{⟨text⟩}
\sidefootnotemark[⟨num⟩]
\sidefootnotetext[⟨num⟩]{⟨text⟩}
```

By default the regular footnotes and the side footnotes use different counters. If one would like them to use the same counter, issue the following in the preamble:

```
\letcountercounter{sidefootnote}{footnote}
```

2.1.2 Setting the layout for \sidefootnote

There are several possibilities to change the appearance of the \sidefootnote:

Specifying the margin in which the side footnote should go, is done by

```
\sidefootmargin{⟨keyword⟩}
```

where *⟨keyword⟩* can be *left*, *right*, *outer*, and *inner*, and their meaning is explained in Figure ?? . The default is *outer*.

```
\sidefoothsep
\sidefootwidth
\sidefootvsep
```

\sidefoothsep is a length controlling the separation from the text to the side footnote column, default \marginparsep. \sidefootwidth is length controlling the width of the side footnote column, default \marginparwidth, and \sidefootvsep is the vertical distance between two side footnotes, default \onelineskip.

```
\sidefootadjust
\setsidefootheight{⟨height⟩}
\sidefootfont
```

\sidefootadjust is a length which specifies the placement of the side footnote column in relation to the bottom of the text block, the default is 0pt. \setsidefootheight sets the maximal height of the side footnote column, default textwidth. Lastly \sidefootfont holds the general font setting for the side footnote,² default \normalfont\footnotesize.

The macro

```
\setsidefeet{⟨hsep⟩}{⟨width⟩}{⟨vsep⟩}{⟨adj⟩}{⟨font⟩}{⟨height⟩}
```

sets the specifications all six settings above in one go.. An “*” means ‘use the current value’. So memoir internally use the following default

```
\setsidefeet{\marginparsep}{\marginparwidth}%
{\onelineskip}{0pt}%
{\normalfont\footnotesize}{\textheight}%
```

It is recommended to use this macro along with the other macros in the preamble to specify document layout.

2.1.3 Styling \sidefootnote

```
\sidefootmarkstyle{⟨code⟩}
```

controls how the side footnote counter is typeset in the side footnote. The default is

²There is a similar macro to control the font of the text alone.

```
\sidefootmarkstyle{\textsuperscript{#1}}
```

The mark is typeset in a box of width `\sidefootmarkwidth`. If this is negative, the mark is outdented into the margin, if zero the mark is flush left, and when positive the mark is indented. The mark is followed by the text of the footnote. Second and later lines of the text are offset by the length `\sidefootmarksep` from the end of the box. The first line of a paragraph within a footnote is indented by `\sidefootparindent`. The default values for these lengths are:

```
\setlength{\sidefootmarkwidth}{0em}
\setlength{\sidefootmarksep}{0em}
\setlength{\sidefootparindent}{1em}
```

Caveat: It is natural to specify a length as `\sidefootparindent` as a \LaTeX length, but it has a down side. If, as we do here, set the value to `1em`, then since the size of the `em` unit changes with the current font size, one will actually end up with an indent corresponding to the font size being used when the

```
\setlength{\sidefootparindent}{1em}
```

was issued, not when it has used (where the font size most often will be `\footnotesize`).

At this point we consider this to be a *feature* not an error. One way to get pass this problem is the following

```
\begingroup% keep font change local
\sidefoottextfont
\global\setlength{\sidefootparindent}{1em}
\endgroup
```

Then it will store the value of `em` corresponding to the font being used.

`\sidefoottextfont`

holds the font being used by the side footnote, default `\normalfont\footnotesize`.

`\sidefootform`

is used to specify the raggedness of the text. Default

```
\newcommand*{\sidefootform}{\rightskip=\z@ \@plus 2em}
```

which is much like `\raggedright` but allows some hyphenation. One might consider using

```
\usepackage{ragged2e}
\newcommand*{\sidefootform}{\RaggedRight}
```

Which does something similar.

2.1.4 Side footnote example

In the margin you will find the result of the following code:

```
Testing\sidefootnote{This is test} bottom aligned
footnotes.\sidefootnote{This is another side
footnote, spanning several lines.
```

```
And several paragraphs}\sidefootnote{And number three}
```


Testing¹ bottom aligned footnotes.^{2,3}

2.2 ENDNOTES

Reimplemented, December 2010³

Endnotes are often used instead of footnotes so as not to interrupt the flow of the main text. Although endnotes are normally put at the end of the document, they may instead be put at the end of each chapter.

The endnotes package already uses the command `\endnote` for an endnote, so the class uses `\pagenote` for an endnote so as not to clash if you prefer to use the package.

```
\makepagenote
\pagenote[⟨id⟩]{⟨text⟩}
\printpagenotes \printpagenotes*
```

The general principle is that notes are written out to a file which is then input at the place where the notes are to be printed. The note file has an `ent` extension, like the table of contents file has a `toc` extension.

You have to put `\makepagenote` in your preamble if you want endnotes. This will open the `ent` note file which is called `\jobname.ent`.

In the body of the text use `\pagenote` to create an endnote, just as you would use `\footnote` to create a footnote. In the books that I have checked there are two common methods of identifying an endnote:

1. Like a footnote, put a number in the text at the location of the note and use the same number to identify the note when it finally gets printed.
2. Put no mark in the text, but when it is finally printed use a few words from the text to identify the origin of the note. The page number is often used as well with this method.

The `⟨text⟩` argument of `\pagenote` is the contents of the note and if the optional `⟨id⟩` argument is not used the result is similar to having used `\footnote` — a number in the main text and the corresponding number in the endnotes listing (as in 1 above). For the second reference style (2 above) use the optional `⟨id⟩` argument for the ‘few words’, and no mark will be put into the main text but `⟨id⟩` will be used as the identification in the listing.

For one set of endnotes covering the whole document put `\printpagenotes` where you want them printed, typically before any bibliography or index. The `\printpagenotes` macro inputs the `ent` endnote file for printing and then closes it to any further notes.

For notes at the end of each chapter put `\printpagenotes*`, which inputs the `ent` file for printing then empties it ready for more notes, at the end of each chapter.

The simple use is like this:

```
\documentclass[...]{memoir}
...
\makepagenote
```

³The former implementation had some difficulties handling certain types of input. A few of the macros used to format the output are no longer supported/used in the new implementation.

¹This is test

²This is another side footnote, spanning several lines.

And several paragraphs

³And number three

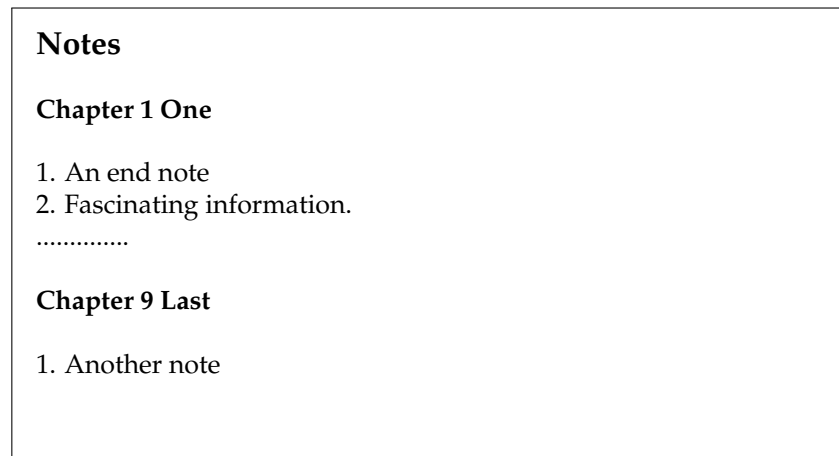


Figure 2.1: Example endnote listing

```
...
\begin{document}
\chapter{One}
... \pagenote{An end note.} ...
... \pagenote{Fascinating information.}
...
\chapter{Last}% chapter 9
... \pagenote{Another note.}% 30th note
...
...
\printpagenotes
...
\end{document}
```

This will result in an endnote listing looking like Figure 2.1.

For notes at the end of each chapter:

```
\documentclass[...] {memoir}
...
\makepagenote
...
\begin{document}
\chapter{One}
... \pagenote{An end note.} ...
...
\printpagenotes*
\chapter{Last}
... \pagenote{Another note.} ...
...
\printpagenotes*
```

```
%%% no more chapters
...
\end{document}
```

```
\continuousnotenums
\notepageref
```

The `pagenote` counter is used for the notes. By default the endnotes are numbered per chapter. If you want the numbering to be continuous throughout the document use the `\continuousnotenums` declaration. Normally the information on which page a note was created is discarded but will be made available to notes in the endnote listing following the `\notepageref` declaration. Both `\continuousnotenums` and `\notepageref` can only be used in the preamble.

```
\notesname
\notedivision
```

When `\printpagenotes` (or `\printpagenotes*`) is called the first thing it does is call the macro `\notedivision`. By default this is defined as:

```
\newcommand*{\notedivision}{\chapter{\notesname}}
```

with

```
\newcommand*{\notesname}{Notes}
```

In other words, it will print out a heading for the notes that will be read from the `ent` file. `\print...` then closes the `ent` file for writing and after this `\inputs` it to get and process the notes.

2.2.1 Changing the appearance

If the something is like a figure with a numbered caption and the numbering depends on the chapter numbering, then the numbers have to be hand set (unless you define a special chapter command for the purpose). For example:

```
... end previous chapter.
\cleartoevenpage[\thispagestyle{empty}] % a skipped page to be empty
\thispagestyle{plain}
\addtocounter{chapter}{1} % increment the chapter number
\setcounter{figure}{0}    % initialise figure counter
\begin{figure}
...
\caption{Pre chapter figure}
\end{figure}

\addtocounter{chapter}{-1} % decrement the chapter number
\chapter{Next chapter}    % increments chapter & resets figure numbers
\addtocounter{figure}{1}  % to account for pre-chapter figure
```

2.2.2 Epigraphs on book or part pages

If you wish to put an epigraphs on `\book` or `\part` pages you have to do a little more work than in other cases. This is because these division commands do some page flipping before and after typesetting the title.

One method is to put the epigraph into the page header as for epigraphs before `\chapter` titles. By suitable adjustments the epigraph can be placed anywhere on the page, independently of whatever else is on the page. A similar scheme may be used for epigraphs on other kinds of pages. The essential trick is to make sure that the *epigraph* pagestyle is used for the page. For an epigraphed bibliography or index, the macros `\prebibhook` or `\preindexhook` can be appropriately modified to do this.

The other method is to subvert the `\beforepartskip` command for epigraphs before the title, or the `\afterpartskip` command for epigraphs after the title (or the equivalents for `\book` pages).

For example:

```
\let\oldbeforepartskip\beforepartskip % save definition
\renewcommand*{\beforepartskip}{%
  \epigraph{...}{...}% an epigraph
  \vfil}
\part{An epigraphed part}
...
\renewcommand*{\beforepartskip}{%
  \epigraph{...}{...}% another epigraph
  \vfil}
\part{A different epigraphed part}
...
\let\beforepartskip\oldbeforepartskip % restore definition
\part{An unepigraphed part}
...
```

Three

Poetry

The typesetting of a poem should ideally be dependent on the particular poem. Individual problems do not usually admit of a general solution, so this manual and code should be used more as a guide towards some solutions rather than providing a ready made solution for any particular piece of verse.

The doggerel used as illustrative material has been taken from [Wil??].

Note that for the examples in this section I have made no attempt to do other than use the minimal (La)TeX capabilities; in particular I have made no attempt to do any special page breaking so some stanzas may cross onto the next page — most undesirable for publication.

The standard LaTeX classes provide the `verse` environment which is defined as a particular kind of list. Within the environment you use `\\` to end a line, and a blank line will end a stanza. For example, here is the text of a single stanza poem:

```
\newcommand{\garden}{  
I used to love my garden \\  
But now my love is dead \\  
For I found a bachelor's button \\  
In black-eyed Susan's bed.  
}
```

When this is typeset as a normal LaTeX paragraph (with no paragraph indentation), i.e.,

```
\noident\garden
```

it looks like:

```
I used to love my garden  
But now my love is dead  
For I found a bachelor's button  
In black-eyed Susan's bed.
```

Typesetting it within the `verse` environment produces:

```
I used to love my garden  
But now my love is dead  
For I found a bachelor's button  
In black-eyed Susan's bed.
```

The stanza could also be typeset within the `alltt` environment, defined in the standard `alltt` package [Bra97], using a normal font and no `\\` line endings.

3. POETRY

```
\begin{alltt}\normalfont
I used to love my garden
But now my love is dead
For I found a bachelor's button
In black-eyed Susan's bed.
\end{alltt}
```

which produces:

```
I used to love my garden
But now my love is dead
For I found a bachelor's button
In black-eyed Susan's bed.
```

The `alltt` environment is like the `verbatim` environment except that you can use LaTeX macros inside it. In the `verse` environment long lines will be wrapped and indented but in the `alltt` environment there is no indentation.

Some stanzas have certain lines indented, often alternate ones. To typeset stanzas like this you have to add your own spacing. For instance:

```
\begin{verse}
There was an old party of Lyme \\
Who married three wives at one time. \\
\hspace{2em} When asked: 'Why the third?' \\
\hspace{2em} He replied: 'One's absurd, \\
And bigamy, sir, is a crime.'
\end{verse}
```

is typeset as:

```
There was an old party of Lyme
Who married three wives at one time.
    When asked: 'Why the third?'
    He replied: 'One's absurd,
And bigamy, sir, is a crime.'
```

Using the `alltt` environment you can put in the spacing via ordinary spaces. That is, this:

```
\begin{alltt}\normalfont
There was an old party of Lyme
Who married three wives at one time.
    When asked: 'Why the third?'
    He replied: 'One's absurd,
And bigamy, sir, is a crime.'
\end{alltt}
```

is typeset as

```
There was an old party of Lyme
```

Who married three wives at one time.
 When asked: ‘Why the third?’
 He replied: ‘One’s absurd,
 And bigamy, sir, is a crime.’

More exotically you could use the TeX `\parshape` command¹:

```
\parshape = 5 0pt \linewidth 0pt \linewidth
           2em \linewidth 2em \linewidth 0pt \linewidth
\noindent There was an old party of Lyme \\
Who married three wives at one time. \\
When asked: ‘Why the third?’ \\
He replied: ‘One’s absurd, \\
And bigamy, sir, is a crime.’ \par
```

which will be typeset as:

There was an old party of Lyme
 Who married three wives at one time.
 When asked: ‘Why the third?’
 He replied: ‘One’s absurd,
 And bigamy, sir, is a crime.’

This is about as much assistance as standard (La)TeX provides, except to note that in the `verse` environment the `\\`* version of `\\` will prevent a following page break. You can also make judicious use of the `\needspace` macro to keep things together.

Some books of poetry, and especially anthologies, have two or more indexes, one, say for the poem titles and another for the first lines, and maybe even a third for the poets’ names. If you are not using memoir then the index [Jon95] and multind [Lon91] packages provide support for multiple indexes in one document.

3.1 CLASSY VERSE

The code provided by the memoir class is meant to help with some aspects of typesetting poetry but does not, and cannot, provide a comprehensive solution to all the requirements that will arise.

The main aspects of typesetting poetry that differ from typesetting plain text are:

- Poems are usually visually centered on the page.
- Some lines are indented, and often there is a pattern to the indentation.
- When a line is too wide for the page it is broken and the remaining portion indented with respect to the original start of the line.

These are the ones that the class attempts to deal with.

<pre>\begin{verse}[<length>] ... \end{verse} \versewidth</pre>
--

¹See the *TeXbook* for how to use this.

The `verse` environment provided by the class is an extension of the usual LaTeX environment. The environment takes one optional parameter, which is a length; for example `\begin{verse}[4em]`. You may have noticed that the earlier verse examples are all near the left margin, whereas verses usually look better if they are typeset about the center of the page. The length parameter, if given, should be about the length of an average line, and then the entire contents will be typeset with the mid point of the length centered horizontally on the page.

The length `\versewidth` is provided as a convenience. It may be used, for example, to calculate the length of a line of text for use as the optional argument to the `verse` environment:

```
\settowidth{\versewidth}{This is the average line,}  
\begin{verse}[\versewidth]
```

`\vleftmargin`

In the basic LaTeX `verse` environment the body of the verse is indented from the left of the typeblock by an amount `\leftmargini`, as is the text in many other environments based on the basic LaTeX `list` environment. For the class's version of `verse` the default indent is set by the length `\vleftmargin` (which is initially set to `\leftmargini`). For poems with particularly long lines it could perhaps be advantageous to eliminate any general indentation by:

```
\setlength{\vleftmargin}{0em}
```

If necessary the poem could even be moved into the left margin by giving `\vleftmargin` a negative length value, such as `-1.5em`.

`\stanzaskip`

The vertical space between stanzas is the length `\stanzaskip`. It can be changed by the usual methods.

`\vin`
`\vgap`
`\vindent`

The command `\vin` is shorthand for `\hspace*{\vgap}` for use at the start of an indented line of verse. The length `\vgap` (initially `1.5em`) can be changed by `\setlength` or `\addtolength`. When a verse line is too long to fit within the typeblock it is wrapped to the next line with an initial indent given by the value of the length `vindent`. Its initial value is twice the default value of `\vgap`.

`\\[<length>]`
`*[<length>]`
`\\![<length>]`

Each line in the `verse` environment, except possibly for the last line in a stanza, must be ended by `\\`, which comes in several variants. In each variant the optional `<length>` is the vertical space to be left before the next line. The `*` form prohibits a page break after the line. The `\\!` form is to be used only for the last line in a stanza when the lines are being numbered; this is because the line numbers are incremented by the `\\` macro. It would normally be followed by a blank line.


```
\verselinebreak[⟨length⟩]
\\>[⟨length⟩]
```

Using `\verselinebreak` will cause later text in the line to be typeset indented on the following line. If the optional `⟨length⟩` is not given the indentation is twice `\vgap`, otherwise it is `⟨length⟩`. The broken line will count as a single line as far as the `altverse` and `patverse` environments are concerned. The macro `\\>` is shorthand for `\verselinebreak`, and unlike other members of the `\\` family the optional `⟨length⟩` is the indentation of the following partial line, not a vertical skip. Also, the `\\>` macro does not increment any line number.

```
\vinphantom{⟨text⟩}
```

Verse lines are sometimes indented according to the space taken by the text on the previous line. The macro `\vinphantom` can be used at the start of a line to give an indentation as though the line started with `⟨text⟩`. For example here are a few lines from the portion of *Fridthjof's Saga* where Fridthjof and Ingeborg part:

Source for example 3.1

```
\settowidth{\versewidth}{Nay, nay, I leave thee not,
                        thou goest too}
\begin{verse}[\versewidth]
\ldots \\*
His judgement rendered, he dissolved the Thing. \\*
\flagverse{Ingeborg} And your decision? \\*
\flagverse{Fridthjof} \vinphantom{And your decision?}
                        Have I ought to choose? \\*
Is not mine honour bound by his decree? \\*
And that I will redeem through Angantyr \\*
His paltry gold doth hide in Nastrand's flood. \\*
Today will I depart. \\*
\flagverse{Ingeborg} \vinphantom{Today will I depart.}
                        And Ingeborg leave? \\*
\flagverse{Fridthjof} Nay, nay, I leave thee not,
                        thou goest too. \\*
\flagverse{Ingeborg} Impossible! \\*
\flagverse{Fridthjof} \vinphantom{Impossible!}
                        O! hear me, ere thou answerest.
\end{verse}
```

Use of `\vinphantom` is not restricted to the start of verse lines — it may be used anywhere in text to leave some blank space. For example, compare the two lines below, which are produced by this code:

```
\noindent Come away with me and be my love --- Impossible. \\
Come away with me \vinphantom{and be my love} --- Impossible.
Come away with me and be my love — Impossible.
Come away with me — Impossible.
```

```
\vleftofline{⟨text⟩}
```

Typeset example 3.1: Phantom text in verse

	...
Ingeborg	His judgement rendered, he dissolved the Thing.
Fridthjof	And your decision?
	Have I ought to choose?
	Is not mine honour bound by his decree?
	And that I will redeem through Angantyr
	His paltry gold doth hide in Nastrand's flood.
	Today will I depart.
Ingeborg	And Ingeborg leave?
Fridthjof	Nay, nay, I leave thee not, thou goest too.
Ingeborg	Impossible!
Fridthjof	O! hear me, ere thou answerest.

A verse line may start with something, for example open quote marks, where it is desirable that it is ignored as far as the alignment of the rest of the line is concerned² — a sort of ‘hanging left punctuation’. When it is put at the start of a line in the `verse` environment the `<text>` is typeset but ignored as far as horizontal indentation is concerned. Compare the two examples.

Source for example 3.2

```
\noindent ‘‘No, this is what was spoken by the prophet Joel:
\begin{verse}
‘\,‘\,‘‘In the last days,’’ God says, \\\
‘‘I will pour out my spirit on all people. \\\
Your sons and daughters will prophesy, \\\
\ldots \\\
And everyone who calls \ldots ’\,‘
\end{verse}
```

Source for example 3.3

```
\noindent ‘‘No, this is what was spoken by the prophet Joel:
\begin{verse}
\leftline{‘\,‘\,‘}In the last days,’’ God says, \\\
\leftline{‘}I will pour out my spirit on all people. \\\
Your sons and daughters will prophesy, \\\
\ldots \\\
```

²Requested by Matthew Ford who also provided the example text.

Typeset example 3.2: Verse with regular quote marks

“No, this is what was spoken by the prophet Joel:
 “ ‘ “In the last days,” God says,
 “I will pour out my spirit on all people.
 Your sons and daughters will prophesy,
 ...
 And everyone who calls ... ” ’ ”

Typeset example 3.3: Verse with hanging left quote marks

“No, this is what was spoken by the prophet Joel:
 “ ‘ “In the last days,” God says,
 “I will pour out my spirit on all people.
 Your sons and daughters will prophesy,
 ...
 And everyone who calls ... ” ’ ”

```
And everyone who calls \ldots ’\,’
\end{verse}
```

3.1.1 Indented lines

Within the verse environment stanzas are normally separated by a blank line in the input.

```
\begin{altverse} ... \end{altverse}
```

Individual stanzas within `verse` may, however, be enclosed in the `altverse` environment. This has the effect of indenting the 2nd, 4th, etc., lines of the stanza by the length `\vgap`.

```
\begin{patverse} ... \end{patverse}
\begin{patverse*} ... \end{patverse*}
\indentpattern{⟨digits⟩}
```

As an alternative to the `altverse` environment, individual stanzas within the `verse` environment may be enclosed in the `patverse` environment. Within this environment the indentation of each line is specified by an indentation pattern, which consists of an array of digits, d_1 to d_n , and the n th line is indented by d_n times `\vgap`. However, the first line is not indented, irrespective of the value of d_1 .

The indentation pattern for a `patverse` or `patverse*` environment is specified via the `\indentpattern` command, where `⟨digits⟩` is a string of digits (e.g., 3213245281).

With the `patverse` environment, if the pattern is shorter than the number of lines in the stanza, the trailing lines will not be indented. However, in the `patverse*` environment the pattern keeps repeating until the end of the stanza.

3.1.2 Numbering

```
\flagverse{<flag>}
\vleftskip
```

Putting `\flagverse` at the start of a line will typeset `<flag>`, for example the stanza number, ending at a distance `\vleftskip` before the line. The default for `\vleftskip` is 3em.

The lines in a poem may be numbered.

```
\linenumberfrequency{<nth>}
\setverselinenums{<first>}{<startat>}
```

The declaration `\linenumberfrequency{<nth>}` will cause every `<nth>` line of succeeding verses to be numbered. For example, `\linenumberfrequency{5}` will number every fifth line. The default is `\linenumberfrequency{0}` which prevents any numbering. The `\setverselinenums` macro can be used to specify that the number of the first line of the following verse shall be `<first>` and the first printed number shall be `<startat>`. For example, perhaps you are quoting part of a numbered poem. The original numbers every tenth line but if your extract starts with line 7, then

```
\linenumberfrequency{10}
\setverselinenums{7}{10}
```

is what you will need.

```
\thepoemline
\linenumberfont{<font-decl>}
```

The poemline counter is used in numbering the lines, so the number representation is `\thepoemline`, which defaults to arabic numerals, and they are typeset using the font specified via `\linenumberfont`; the default is

```
\linenumberfont{\small\rmfamily}
```

for small numbers in the roman font.

```
\verselinenumbersright
\verselinenumbersleft
\vrightskip
```

Following the declaration `\verselinenumbersright`, which is the default, any verse line numbers will be set in the righthand margin. The `\verselinenumbersleft` declaration will set any subsequent line numbers to the left of the lines. The numbers are set at a distance `\vrightskip` (default 1em) into the margin.

3.2 TITLES

The `\PoemTitle` command is provided for typesetting titles of poems.

```
\PoemTitle[<fortoc>][<forhead>]{<title>}
\NumberPoemTitle
\PlainPoemTitle
\thepoem
```

The `\PoemTitle` command takes the same arguments as the `\chapter` command; it typesets the title for a poem and adds it to the ToC. Following the declaration `\NumberPoemTitle` the title is numbered but there is no numbering after the `\PlainPoemTitle` declaration.

```
\poemtoc{<sec>}
```

The kind of entry made in the ToC by `\PoemTitle` is defined by `\poemtoc`. The initial definition is:

```
\newcommand{\poemtoc}{section}
```

for a section-like ToC entry. This can be changed to, say, `chapter` or `subsection` or

```
\poemtitlemark{<forhead>}
\poemtitlestyle
```

The macro `\poemtitlemark` is called with the argument `<forhead>` so that it may be used to set marks for use in a page header via the normal mark process. The `\poemtitlestyle` macro, which by default does nothing, is provided as a hook so that, for example, it can be redefined to specify a particular pagestyle that should be used. For example:

```
\renewcommand*{\poemtitlemark}[1]{\markboth{#1}{#1}}
\renewcommand*{\poemtitlestyle}{%
  \pagestyle{headings}%
  \thispagestyle{empty}}
```

```
\PoemTitle*{<forhead>}{<title>}
\poemtitlestarmark{<forhead>}
\poemtitlestarpstyle
```

The `\PoemTitle*` command produces an unnumbered title that is not added to the ToC. Apart from that it operates in the same manner as the unstarred version. The `\poemtitlestarmark` and `\poemtitlestarpstyle` can be redefined to set marks and pagestyles.

3.2.1 Main Poem Title layout parameters

```
\PoemTitleheadstart
\printPoemTitlenonum
\printPoemTitlenum
\afterPoemTitlenum
\printPoemTitletitle{<title>}
\afterPoemTitle
```

The essence of the code used to typeset a numbered `<title>` from a `\PoemTitle` is:

```
\PoemTitleheadstart
\printPoemTitlenum
\afterPoemTitlenum
\printPoemTitletitle{title}
\afterPoemTitle
```

If the title is unnumbered then `\printPoemTitlenonum` is used instead of the `\printPoemTitlenum` and `\afterPoemTitlenum` pair of macros.

The various elements of this can be modified to change the layout. By default the number is centered above the title, which is also typeset centered, and all in a `\large` font.

The elements are detailed in the next section.

3.2.2 Detailed Poem Title layout parameters

```
\beforePoemTitleskip
\PoemTitlenumfont
\midPoemTitleskip
\PoemTitlefont
\afterPoemTitleskip
```

As defined, `\PoemTitleheadstart` inserts vertical space before a poem title. The default definition is:

```
\newcommand*{\PoemTitleheadstart}{\vspace{\beforePoemTitleskip}}
\newlength{\beforePoemTitleskip}
\setlength{\beforePoemTitleskip}{1\onelineskip}
```

`\printPoemTitlenum` typesets the number for a poem title. The default definition, below, prints the number centered and in a large font.

```
\newcommand*{\printPoemTitlenum}{\PoemTitlenumfont \thepoem}
\newcommand*{\PoemTitlenumfont}{\normalfont\large\centering}
```

The definition of `\printPoemTitlenonum`, which is used when there is no number, is simply

```
\newcommand*{\printPoemTitlenonum}{{}}
```

`\afterPoemTitlenum` is called between setting the number and the title. It ends a paragraph (thus making sure any previous `\centering` is used) and then may add some vertical space. The default definition is:

```
\newcommand*{\afterPoemTitlenum}{\par\nobreak\vskip \midPoemTitleskip}
\newlength{\midPoemTitleskip}
\setlength{\midPoemTitleskip}{0pt}
```

The default definition of `\printPoemTitletitle` is below. It typesets the title centered and in a large font.

```
\newcommand*{\printPoemTitletitle}[1]{\PoemTitlefont #1}
\newcommand*{\PoemTitlefont}{\normalfont\large\centering}
```

The macro `\afterPoemTitle` finishes off the title typesetting. The default definition is:

```
\newcommand*{\afterPoemTitle}{\par\nobreak\vskip \afterPoemTitleskip}
\newlength{\afterPoemTitleskip}
\setlength{\afterPoemTitleskip}{1\onelineskip}
```

3.3 EXAMPLES

Here are some sample verses using the class facilities.

First a Limerick, but titled and centered:

```
\renewcommand{\poemtoc}{subsection}
\PlainPoemTitle
\PoemTitle{A Limerick}
\settowidth{\versewidth}{There was a young man of Quebec}
\begin{verse}[\versewidth]
There was a young man of Quebec \\
Who was frozen in snow to his neck. \\
\vin When asked: 'Are you friz?' \\
\vin He replied: 'Yes, I is, \\
But we don't call this cold in Quebec.'
\end{verse}
```

which gets typeset as below. The `\poemtoc` is redefined to `subsection` so that the `\poemtitle` titles are entered into the ToC as subsections. The titles will not be numbered because of the `\PlainPoemTitle` declaration.

A Limerick

There was a young man of Quebec
Who was frozen in snow to his neck.
When asked: 'Are you friz?'
He replied: 'Yes, I is,
But we don't call this cold in Quebec.'

Next is the Garden verse within the `altverse` environment. Unlike earlier renditions this one is titled and centered.

```
\settowidth{\versewidth}{But now my love is dead}
\PoemTitle{Love's lost}
\begin{verse}[\versewidth]
\begin{altverse}
\garden
\end{altverse}
\end{verse}
```

Note how the alternate lines are automatically indented in the typeset result below.

Love's lost

I used to love my garden
But now my love is dead
For I found a bachelor's button
In black-eyed Susan's bed.

3. POETRY

It is left up to you how you might want to add information about the author of a poem. Here is one example of a macro for this:

```
\newcommand{\attrib}[1]{%
  \nopagebreak{\raggedleft\footnotesize #1\par}}
```

This can be used as in the next bit of doggerel.

```
\PoemTitle{Fleas}
\settowidth{\versewidth}{What a funny thing is a flea}
\begin{verse}[\versewidth]
What a funny thing is a flea. \\
You can't tell a he from a she. \\
But he can. And she can. \\
Whoopee!
\end{verse}
\attrib{Anonymous}
```

Fleas

What a funny thing is a flea.
You can't tell a he from a she.
But he can. And she can.
Whoopee!

Anonymous

The next example demonstrates the automatic line wrapping for overlong lines.

```
\PoemTitle{In the beginning}
\settowidth{\versewidth}{And objects at rest tended to
  remain at rest}
\begin{verse}[\versewidth]
Then God created Newton, \\
And objects at rest tended to remain at rest, \\
And objects in motion tended to remain in motion, \\
And energy was conserved
  and momentum was conserved
  and matter was conserved \\
And God saw that it was conservative.
\end{verse}
\attrib{Possibly from \textit{Analog}, circa 1950}
```

In the beginning

Then God created Newton,
And objects at rest tended to remain at rest,
And objects in motion tended to remain in motion,

Possibly from *Analog*, circa 1950

Samuel Butler (1612–1680)

3. POETRY

```
There was a young lady of Ryde \\
Who ate some apples and died. \\
The apples fermented \\
Inside the lamented \\
And made cider inside her inside.
\end{patverse}
\end{verse}
```

Note that I used the `\needspace` command to ensure that the limerick will not get broken across a page.

The Young Lady of Ryde

```
There was a young lady of Ryde
Who ate some apples and died.
    The apples fermented
    Inside the lamented
And made cider inside her inside.
```

The next example is a song you may have heard of. This uses `\flagverse` for labelling the stanzas, and because the lines are numbered they can be referred to.

```
\settowidth{\versewidth}{In a cavern, in a canyon,}
\PoemTitle{Clementine}
\begin{verse}[\versewidth]
\linenumberfrequency{2}
\begin{altverse}
\flagverse{1.} In a cavern, in a canyon, \\
Excavating for a mine, \\
Lived a miner, forty-niner, \label{vs:49} \\
And his daughter, Clementine. \\!
\end{altverse}

\begin{altverse}
\flagverse{\textsc{chorus}} Oh my darling, Oh my darling, \\
Oh my darling Clementine. \\
Thou art lost and gone forever, \\
Oh my darling Clementine.
\end{altverse}
\linenumberfrequency{0}
\end{verse}
The ‘forty-niner’ in line~\ref{vs:49} of the song
refers to the gold rush of 1849.
```

Clementine

1. In a cavern, in a canyon,

	Excavating for a mine,	2
	Lived a miner, forty-niner,	
	And his daughter, Clementine.	4
CHORUS	Oh my darling, Oh my darling,	
	Oh my darling Clementine.	6
	Thou art lost and gone forever,	
	Oh my darling Clementine.	

The 'forty-niner' in line 3 of the song refers to the gold rush of 1849.

The last example is a much more ambitious use of `\indentpattern`. In this case it is defined as:

```
\indentpattern{0135554322112346898779775545653222345544456688778899}
```

and the result is shown on the next page.

Mouse's Tale

Fury said to
a mouse, That
he met
in the
house,
'Let us
both go
to law:
I will
prosecute
you. —
Come, I'll
take no
denial;
We must
have a
trial:
For
really
this
morning
I've
nothing
to do.'
Said the
mouse to
the cur,
Such a
trial,
dear sir,
With no
jury or
judge,
would be
wasting
our breath.'
'I'll be
judge,
I'll be
jury.'
Said
cunning
old Fury;
'I'll try
the whole
cause
and
condemn
you
to
death.'

Lewis Carrol, *Alice's Adventures in Wonderland*, 1865

Four

Boxes, verbatims and files

The title of this chapter indicates that it deals with three disconnected topics, but there is method in the seeming peculiarity. By the end of the chapter you will be able to write LaTeX code that lets you put things in your document source at one place and have them typeset at a different place, or places. For example, if you are writing a text book that includes questions and answers then you could write a question and answer together yet have the answer typeset at the end of the book.

Writing in one place and printing in another is based on outputting stuff to a file and then inputting it for processing at another place or time. This is just how LaTeX produces the ToC. It is often important when writing to a file that LaTeX does no processing of any macros, which implies that we need to be able to write verbatim. One use of verbatim in LaTeX is to typeset computer code or the like, and to clearly distinguish the code from the main text it is often typeset within a box. Hence the chapter title.

The class extends the kinds of boxes normally provided, extends the default verbatims, and provides a simple means of writing and reading files.

One problem with verbatims is that they can not be used as part of an argument to a command. For example to typeset something in a framed `minipage` the obvious way is to use the `minipage` as the argument to the `\fbox` macro:

```
\fbox{\begin{minipage}{6cm}
  Contents of framed minipage
\end{minipage}}
```

This works perfectly well until the contents includes some verbatim material, whereupon you will get nasty error messages. However this particular conundrum is solvable, even if the solution is not particularly obvious. Here it is.

We can put things into a box, declared via `\newsavebox`, and typeset the contents of the box later via `\usebox`. The most common way of putting things into a save box is by the `\sbox` or `\savebox` macros, but as the material for saving is one of the arguments to these macros this approach fails. But, `lrbox` is an environment form of `\sbox`, so it can handle verbatim material. The code below, after getting a new save box, defines a new `framedminipage` environment which is used just like the standard `minipage`. The `framedminipage` starts an `lrbox` environment and then starts a `minipage` environment, after which comes the contents. At the end it closes the two environments and calls `\fbox` with its argument being the contents of the saved box *which have already been typeset*.

```
\newsavebox{\minibox}
\newenvironment{framedminipage}[1]{%
  \begin{lrbox}{\minibox}\begin{minipage}{#1}}%
```

```
{\end{minipage}\end{lrbox}\fbox{\usebox{\minibox}}}
```

Question 1. Can you think of any improvements to the definition of the `framedminipage` environment?

Question 2. An answer to question 1 is at the end of this chapter. Suggest how it was put there.

4.1 BOXES

LaTeX provides some commands to put a box round some text. The class extends the available kinds of boxes.

```
\begin{framed} text \end{framed}
\begin{shaded} text \end{shaded}
\begin{snugshade} text \end{snugshade}
```

The `framed`, `shaded`, and `snugshade` environments, which were created by Donald Arseneau as part of his `framed` package [Ars07], put their contents into boxes that break across pages. The `framed` environment delineates the box by drawing a rectangular frame. If there is a pagebreak in the middle of the box, frames are drawn on both pages.

The `shaded` environment typesets the box with a shaded or colored background. This requires the use of the `color` package [Car05], which is one of the required LaTeX packages, or the `xcolor` package [Ker07]. The shading color is `shadecolor`, which you have to define before using the environment. For example, to have a light gray background:

```
\definecolor{shadecolor}{gray}{0.9}
```

For complete information on this see the documentation for the `color` or `xcolor` packages, or one of the LaTeX books like the *Graphics Companion* [GM⁺07]. In the `snugshaded` environment the box clings more closely to its contents than it does in the `shaded` environment.

Recommended alternative

Since the class was originally written, much have happened in the `gfx` generating capabilities in LaTeX, especially the popularity of TikZ has provided many more extensive box and graphics generating packages.

As of 2018 one of the most impressive packages for all sorts of boxes is the `tcolorbox` package by Thomas F. Sturm.

Be aware that the boxes we present in this manual are somewhat delicate; they do not work in all circumstances. For example they will not work with the `multicol` package [Mit18], and any floats or footnotes in the boxes will disappear.

```
\FrameRule \FrameSep \FrameHeightAdjust
```

The `framed` environment puts the text into an ‘`\fbox`’ with the settings:

```
\setlength{\FrameRule}{\fboxrule}
\setlength{\FrameSep}{3\fboxsep}
```

The macro `\FrameHeightAdjust` specifies the height of the top of the frame above the baseline at the top of a page; its initial definition is:

```
\providecommand*\FrameHeightAdjust}{0.6em}
```

```
\MakeFramed{<settings>} \endMakeFramed
\FrameCommand \FrameRestore
```

Internally, the environments are specified using the `MakeFramed` environment. The `<setting>` should contain any adjustments to the text width (applied to `\hsize` and using the `\width` of the frame itself) and a ‘restore’ command, which is normally the provided `\FrameRestore` macro. The frame itself is drawn via the `\FrameCommand`, which can be changed to obtain other boxing styles. The default definition equates to an `\fbox` and is:

```
\newcommand*\FrameCommand}{%
  \setlength{\fboxrule}{\FrameRule}\setlength{\fboxsep}{\FrameSep}%
  \fbox}
```

For example, the `framed`, `shaded` and `snugshade` environments are defined as

```
\newenvironment{framed}{% % uses default \FrameCommand
  \MakeFramed{\advance\hsize -\width \FrameRestore}}%
  {\endMakeFramed}
\newenvironment{shaded}{% % redefines \FrameCommand as \colorbox
  \def\FrameCommand{\fboxsep=\FrameSep \colorbox{shadecolor}}%
  \MakeFramed{\FrameRestore}}%
  {\endMakeFramed}
\newenvironment{snugshade}{% A tight version of shaded
  \def\FrameCommand{\colorbox{shadecolor}}%
  \MakeFramed{\FrameRestore\@setminipage}}%
  {\par\unskip\endMakeFramed}
```

If you wanted a narrow, centered, framed environment you could do something like this:

```
\newenvironment{narrowframed}{%
  \MakeFramed{\setlength{\hsize}{22pc}\FrameRestore}}%
  {\endMakeFramed}
```

where 22pc will be the width of the new framed environment.

```
\begin{leftbar} text \end{leftbar}
```

The `leftbar` environment draws a thick vertical line at the left of the text. It is defined as

```
\newenvironment{leftbar}{%
  \def\FrameCommand{\vrule width 3pt \hspace{10pt}}%
  \MakeFramed{\advance\hsize -\width \FrameRestore}}%
  {\endMakeFramed}
```

By changing the `<setting>` for `\MakeFramed` and the definition of `\FrameCommand` you can obtain a variety of framing styles. For instance, to have rounded corners to the frame instead of the normal sharp ones, you can use the `fancybox` package [Zan98] and the following code:

```
\usepackage{fancybox}
\newenvironment{roundedframe}{%
  \def\FrameCommand{%
    \cornersize*{20pt}%
    \setlength{\fboxsep}{5pt}%
    \ovalbox}%
  \MakeFramed{\advance\hsize-\width \FrameRestore}}%
  {\endMakeFramed}
```

A framed environment is normally used to distinguish its contents from the surrounding text. A title for the environment may be useful, and if there was a pagebreak in the middle, a title on the continuation could be desirable. Doing this takes a bit more work than I have shown so far. This first part was inspired by a posting to CTT by Donald Arseneau.¹

```
\newcommand{\FrameTitle}[2]{%
  \fboxrule=\FrameRule \fboxsep=\FrameSep
  \fbox{\vbox{\nobreak \vskip -0.7\FrameSep
    \rlap{\strut#1}\nobreak\nointerlineskip% left justified
    \vskip 0.7\FrameSep
    \hbox{#2}}}}
\newenvironment{framewithtitle}[2][\FrameFirst@Lab\ (cont.)]{%
  \def\FrameFirst@Lab{\textbf{#2}}%
  \def\FrameCont@Lab{\textbf{#1}}%
  \def\FrameCommand##1{%
    \FrameTitle{\FrameFirst@Lab}{##1}}%
  \def\FirstFrameCommand##1{%
    \FrameTitle{\FrameFirst@Lab}{##1}}%
  \def\MidFrameCommand##1{%
    \FrameTitle{\FrameCont@Lab}{##1}}%
  \def\LastFrameCommand##1{%
    \FrameTitle{\FrameCont@Lab}{##1}}%
  \MakeFramed{\advance\hsize-\width \FrameRestore}}%
  {\endMakeFramed}
```

The `framewithtitle` environment, which is the end goal of this exercise, acts like the `framed` environment except that it puts a left-justified title just after the top of the frame box and before the regular contents.

```
\begin{framewithtitle}[<cont-title>]{<title>} text
\end{framewithtitle}
```

The `<title>` is set in a bold font. If the optional `<cont-title>` argument is given then `<cont-title>` is used as the title on any succeeding pages, otherwise the phrase '`<title>` (cont.)' is used for the continuation title.

If you would like the titles centered, replace the line marked 'left justified' in the code for `\FrameTitle` with the line:

¹On 2003/10/24 in the thread *framed.sty w/heading?*. The particulars are no longer applicable as the framing code in question then has since been revised.

```
\rlap{\centerline{\strut#1}}\nobreak\nointerlineskip% centered
```

The code for the `frametitle` environment is not obvious. The difficulty in creating the environment was that the underlying framing code goes through the ‘stuff’ to be framed by first trying to fit it all onto one page (`\FrameCommand`). If it does not fit, then it takes as much as will fit and typesets that using `\FirstFrameCommand`, then tries to typeset the remainder on the next page. If it all fits then it uses `\LastFrameCommand`. If it doesn’t fit, it typesets as much as it can using `\MidFrameCommand`, and then tries to set the remainder on the following page. The process repeats until all has been set.

If you would prefer to have the title at the top outside the frame the above code needs adjusting.

```
\newcommand{\TitleFrame}[2]{%
  \fboxrule=\FrameRule \fboxsep=\FrameSep
  \vbox{\nobreak \vskip -0.7\FrameSep
    \rlap{\strut#1}\nobreak\nointerlineskip% left justified
    \vskip 0.7\FrameSep
    \noindent\fbox{#2}}}%
\newenvironment{titledframe}[2][\FrameFirst@Lab\ (cont.)]{%
  \def\FrameFirst@Lab{\textbf{#2}}}%
  \def\FrameCont@Lab{\textbf{#1}}}%
  \def\FrameCommand##1{%
    \TitleFrame{\FrameFirst@Lab}{##1}}
  \def\FirstFrameCommand##1{%
    \TitleFrame{\FrameFirst@Lab}{##1}}
  \def\MidFrameCommand##1{%
    \TitleFrame{\FrameCont@Lab}{##1}}
  \def\LastFrameCommand##1{%
    \TitleFrame{\FrameCont@Lab}{##1}}
  \MakeFramed{\hsize\textwidth
    \advance\hsize -2\FrameRule
    \advance\hsize -2\FrameSep
    \FrameRestore}}%
{\endMakeFramed}
```

```
\begin{titledframe}[\langle cont-title \rangle]{\langle title \rangle} text \end{titledframe}
```

The `titledframe` environment is identical to `framewithtitle` except that the title is placed just before the frame. Again, if you would like a centered title, replace the line marked ‘left justified’ in `\TitleFrame` by

```
\rlap{\centerline{\strut#1}}\nobreak\nointerlineskip% centered
```

You can adjust the code for the `framewithtitle` and `titledframe` environments to suit your own purposes, especially as they are not part of the class so you would have to type them in yourself anyway if you wanted to use them, using whatever names you felt suitable.

The class provides two further environments in addition to those from the framed package.

```
\begin{qframe} text \end{qframe}  
\begin{qshade} text \end{qshade}
```

When used within, say, a quotation environment, the `framed` and `shaded` environments do not closely box the indented text. The `qframe` and `qshade` environments do provide close boxing.² The difference can be seen in the following quotation.

This is the start of a quotation environment. It forms the basis showing the difference between the `framed` and `qframe` environments.

This is the second paragraph in the quotation environment and in turn it is within the `qframe` environment.

This is the third paragraph in the quotation environment and in turn it is within the `framed` environment.

This is the fourth and final paragraph within the quotation environment and is not within either a `qframe` or `framed` environment.

If you want to put a frame inside an `adjustwidth` environment then you may well find that `qframe` or `qshade` meet your expectations better than `framed` or `shaded`. Of course, it does depend on what your expectations are.

4.2 LONG COMMENTS

The `%` comment character can be used to comment out (part of) a line of TeX code, but this gets tedious if you need to comment out long chunks of code.

```
\begin{comment} text to be skipped over \end{comment}
```

As an extreme form of font changing, although it doesn't actually work that way, anything in a `comment` environment will not appear in the document; effectively, LaTeX throws it all away. This can be useful to temporarily discard chunks of stuff instead of having to mark each line with the `%` comment character.

```
\newcomment{<name>}  
\commentsoff{<name>}  
\commentson{<name>}
```

The class lets you define your own comment environment via the `\newcomment` command which defines a comment environment called `<name>`. In fact the class itself uses `\newcomment{comment}` to define the `comment` environment. A comment environment `<name>` may be switched off so that its contents are not ignored by using the `\commentsoff` declaration. It may be switched on later by the `\commentson` declaration. In either case `<name>` must have been previously declared as a comment environment via `\newcomment`.

Suppose, for example, that you are preparing a draft document for review by some others and you want to include some notes for the reviewers. Also, you want to include some private comments in the source for yourself. You could use the `comment` environment for your private comments and create another environment for the notes to the reviewers. These notes should not appear in the final document. Your source might then look like:

²Donald Arseneau has said that he may put something similar in a later version the `framed` package.

```

\newcomment{review}
\ifdraftdoc\else
  \commentsoff{review}
\fi
...
\begin{comment}
Remember to finagle the wingle!
\end{comment}
...
\begin{review}
\textit{REVIEWERS: Please pay particular attention to this section.}
\end{review}
...

```

Comment environments cannot be nested, nor can they overlap. The environments in the code below will not work in the manner that might be expected:

```

\newcomment{acomment} \newcomment{mycomment}
\begin{comment}
  \begin{acomment} %% comments cannot be nested
  ...
  \end{acomment}
  ...
  \begin{mycomment}
  ...
\end{comment}
...
\end{mycomment} %% comments cannot overlap

```

More encompassing comment environments are available if you use Victor Eijkhout's comment package [Eij99].

4.3 VERBATIMS

Standard LaTeX defines the `\verb` and `\verb*` commands for typesetting short pieces of text verbatim, short because they cannot include a linebreak. For longer verbatim texts the `verbatim` or `verbatim*` environments can be used. The star forms indicate spaces in the verbatim text by outputting a `~` mark for each space. The class extends the standard verbatims in various ways.

If you have to write a lot of `\verb` text, as I have had to do for this book, it gets tedious to keep on typing this sort of thing: `\verb!verbatim text!`. Remember that the character immediately after the `\verb`, or `\verb*`, ends the verbatim processing.

<pre> \MakeShortVerb{\backslash-char} \DeleteShortVerb{\backslash-char} </pre>
--

The `\MakeShortVerb` macro takes a character preceded by a backslash as its argument, say `\!`, and makes that character equivalent to `\verb!`. Using the character a second time will stop the verbatim processing. Doing, for example `\MakeShortVerb{\!}`, lets you then use `!verbatim text!` instead of the longer winded `\verb!verbatim text!`.

You have to pick as the short verb character one that you are unlikely to use; a good choice is often the | bar character as this rarely used in normal text. This choice, though may be unfortunate if you want to have any tabulars with vertical lines, as the bar character is used to specify those. The `\DeleteShortVerb` macro is provided for this contingency; give it the same argument as an earlier `\MakeShortVerb` and it will restore the short verb character to its normal state.

The `\MakeShortVerb` and `\DeleteShortVerb` macros come from the `shortvrb` package which is part of the LaTeX base system, but I have found them so convenient that I added them to the class.

`\setverbatimfont{<font-declaration>}`

The default font for verbatims is the normal sized monospaced font. The declaration `\setverbatimfont` can be used to specify a different font. The class default is

```
\setverbatimfont{\normalfont\ttfamily}
```

To use a smaller version simply say

```
\setverbatimfont{\normalfont\ttfamily\small}
```

A monospaced font is normally chosen as verbatim text is often used to present program code or typewritten text. If you want a more exotic font, try this

```
\setverbatimfont{\fontencoding{T1}\fontfamily{cmss}\selectfont}
```

and your verbatim text will then look like

We are no longer using the boring old typewriter font
for verbatim text. We used the T1 encoding
to make sure that characters that are often ligatures
like “, or ”, or ---, or <, or >, print as expected.
After this we will switch back to the default verbatim font via
`\setverbatimfont{\normalfont\ttfamily}`

In the normal way of things with an OT1 fontencoding, typesetting the ligatures mentioned above in the sans font produces: ligatures like “, or ”, or —, or ¡, or ¿, which is not what happens in the `\verbatim` environment.

`\begin{verbatim} anything \end{verbatim}`
`\begin{verbatim*} anything \end{verbatim*}`

In the `verbatim` environment³ you can write anything you want (except `\end{verbatim}`), and it will be typeset exactly as written. The `verbatim*` environment is similar except, like with `\verb*`, spaces will be indicated with a `~` mark.

`\tabson[<number>]`
`\tabsoff`

The standard `verbatim` environment ignores any TAB characters; with the class’s environment after calling the `\tabson` declaration the environment will handle TAB characters. By default 4 spaces are used to represent a TAB; the optional `<number>` argument to the

³This version of the `verbatim` environment is heavily based on the `verbatim` package [SRR99] but does provide some extensions.

declaration will set the number of spaces for a TAB to be $\langle number \rangle$. Some folk like to use 8 spaces for a TAB, in which case they would need to declare `\tabson[8]`. Unremarkably, the declaration `\tabsoff` switches off TABs. The class default is `\tabsoff`.

```
\wrappingon
\wrappingoff
\verbatimindent
\verbatimbreakchar{ $\langle char \rangle$ }
```

As noted, whatever is written in a `verbatim` environment is output just as written, even if lines are too long to fit on the page. The declaration `\wrappingon` lets the environment break lines so that they do not overflow. The declaration `\wrappingoff` restores the normal behaviour.

The following is an example of how a wrapped verbatim line looks. In the source the contents of the verbatim was written as a single line.

```
This is an example of line wrapping in the verbatim environment. It %
    is a single line in the source and the \wrappingon %
    declaration has been used.
```

The wrapped portion of verbatim lines are indented from the left margin by the length `\verbatimindent`. The value can be changed by the usual length changing commands. The end of each line that has been wrapped is marked with the $\langle char \rangle$ character of the `\verbatimbreakchar` macro. The class default is `\verbatimbreakchar{\char' \%}`, so that lines are marked with %. To put a `'/'` mark at the end of wrapped lines you can do

```
\setverbatimbreak{\char'\/}
```

or similarly if you would like another character. Another possibility is

```
\setverbatimchar{\char'\/\char'\*}
```

which will make `'/'` the end marker.

4.3.1 Boxed verbatims

Verbatim environments are often used to present program code or, as in this book, LaTeX code. For such applications it can be useful to put the code in a box, or to number the code lines, or perhaps both.

```
\begin{fboxverbatim} anything \end{fboxverbatim}
```

The `fboxverbatim` environment typesets its contents verbatim and puts a tightly fitting frame around the result; in a sense it is similar to the `fbox` command.

```
\begin{boxedverbatim} anything \end{boxedverbatim}
\begin{boxedverbatim*} anything \end{boxedverbatim*}
```

The `boxedverbatim` and `boxedverbatim*` environments are like the `verbatim` and `verbatim*` environments except that a box, allowing page breaks, may be put around the verbatim text and the lines of text may be numbered. The particular format of the output can be controlled as described below.

```
\bvbox \bvtopandtail \bvside \nobvbox
\bvboxsep
```

Four styles of boxes are provided and you can extend these. Following the `\bvbox` declaration, a box is drawn round the verbatim text, breaking at page boundaries if necessary; this is the default style. Conversely, no boxes are drawn after the `\nobvbox` declaration. With the `\bvtopandtail` declaration horizontal lines are drawn at the start and end of the verbatim text, and with the `\bvside` declarations, vertical lines are drawn at the left and right of the text. The separation between the lines and the text is given by the length `\bvboxsep`.

The following hooks are provided to set your own boxing style.

```
\bvtoprulehook \bvtopmidhook \bvendrulehook  
\bvleftsidehook \bvrightsidehook
```

The macros `\bvtoprulehook` and `\bvendrulehook` are called at the start and end of the `boxedverbatim` environment, and before and after page breaks. The macros `\bvleftsidehook` and `\bvrightsidehook` are called at the start and end of each verbatim line. The macro `\bvtopmidhook` is called after `\bvtoprulehook` at the start of the environment. It can be used to add some space if `\bvtoprulehook` is empty.

```
\bvperpagetrue \bvperpagefalse  
\bvtopofpage{<text>} \bvendofpage{<text>}
```

The command `\bvperpagetrue` indicates that a box should be visibly broken at a page-break, while there should be no visible break for `\bvperpagefalse`. If the box continues on to another page then it may be advantageous to place some sort of heading before the verbatim continues. Following the declaration `\bvperpagetrue` the `<text>` argument to `\bvtopofpage` will be typeset after any pagebreak. For example you could set:

```
\bvtopofpage{continued}
```

to print ‘continued’ in the normal text font.

By default, the class sets

```
\bvendofpage{\hrule\kern-.4pt}
```

which causes the `\hrule` to be drawn at the end of a page as the visible break (the rule is 0.4pt thick and the kern backs up that amount after the rule, so it effectively takes no vertical space). This is not always suitable. For instance, if there will be a ‘continued’ message at the top of the following page it may seem odd to draw a line at the bottom of the previous page. In this case, setting

```
\bvendofpage{}
```

will eliminate the rule.

As examples of the use of these hooks, here is how some of the boxed verbatim styles are defined.

The default style is `\bvbox`, which puts separate full boxes on each page.

```
\newcommand{\bvbox}{%  
  \bvperpagetrue  
  \renewcommand{\bvtoprulehook}{\hrule \nobreak \vskip-.1pt}%  
  \renewcommand{\bvleftsidehook}{\vrule}%  
  \renewcommand{\bvrightsidehook}{\vrule}%  
  \renewcommand{\bvendrulehook}{\hrule}%  
  \renewcommand{\bvtopmidhook}{\rule{0pt}{2\fbboxsep} \hss}%
```

```
}
```

The `\nobvbox` turns off all boxing, and is defined as

```
\newcommand{\nobvbox}{%
  \bvperpagefalse
  \renewcommand{\bvtoprulehook}{}%
  \renewcommand{\bvleftsidehook}{}%
  \renewcommand{\bvrightsidehook}{}%
  \renewcommand{\bvendrulehook}{}%
  \renewcommand{\bvtopmidhook}{\rule{0pt}{2\fbboxsep} \hss}%
}
```

The definitions of the other styles, `\bvtopandtail` and `\bvslides`, are intermediate between `\bvbox` and `\nobvbox` in the obvious manner.

```
\linenumberfrequency{<nth>}
\resetbvlinenumber
\setbvlinenums{<first>}{<startat>}
\linenumberfont{<font declaration>}
```

The command `\linenumberfrequency` controls the numbering of lines in a `boxedverbatim` — every `<nth>` line will be numbered. If `<nth>` is 0 or less, then no lines are numbered, if `<nth>` is 1 then each line is numbered, and if `<nth>` is `n`, where `n` is 2 or more, then only every `n`th line is numbered. Line numbering is continuous from one instance of the `boxedverbatim` environment to the next. Outside the environment the line numbers can be reset at any time by the command `\resetbvlinenumber`.

The `\setbvlinenums` macro can be used to specify that the number of the first line of the following `boxedverbatim` shall be `<first>` and the first printed number shall be `<startat>`.

The `\linenumberfont` declaration sets `` as the font for the line numbers. The default specification for this is:

```
\linenumberfont{\footnotesize\rmfamily}
```

Line numbers are always set at the left of the lines because there is no telling how long a line might be and it might clash with a line number set at the right.

```
\bvnumbersinside
\bvnumbersoutside
```

Line numbers are typeset inside the box after the declaration `\bvnumbersinside` and are typeset outside the box after the declaration `\bvnumbersoutside`. The default is to print the numbers inside the box.

Verbatim tabbing, but not wrapping, applies to the `boxedverbatim` environment.

Recommended alternative

Again the `tcolorbox` package offers boxes vs verbatim text.

4.3.2 New verbatims

The class implementation of verbatims lets you define your own kind of verbatim environment. Unfortunately this is not quite as simple as saying

```
\newverbatim{myverbatim}{...}{...}
```

as you can for defining normal environments. Instead, the general scheme is

```
\newenvironment{myverbatim}%
{<non-verbatim stuff> \verbatim <more non-verbatim stuff>}%
{\endverbatim}
```

In particular, you cannot use either the `\begin` or `\end` macros inside the definition of the new verbatim environment. For example, the following code will not work

```
\newenvironment{badverbatim}%
{NBSG\begin{verbatim}}{\end{verbatim}}
```

and this won't work either

```
\newenvironment{badverbatim}%
{\begin{env}\verbatim}\endverbatim\end{env}}
```

And, as with the standard `verbatim` environment, you cannot use the new one in the definition of a new command.

For an example of something that does work, this next little piece of typesetting was done in a new verbatim environment I have called `verbexami`, which starts and ends with a horizontal rule, and it shows the definition of `verbexami`.

The `verbexami` environment

```
\newenvironment{verbexami}%
{\par\noindent\hrule The verbexami environment \verbatim}%
{\endverbatim\hrule}
```

And this is a variation on the theme, with the environment again being enclosed by horizontal rules.

Verbexamii

IS THIS FUN?

```
\newenvironment{verbexamii}%
{\vspace{0.5\baselineskip}\hrule \vspace{0.2\baselineskip}
Verbexamii \verbatim \textsc{Is this fun?}}%
{\endverbatim\hrule\vspace{0.3\baselineskip}}
```

As no doubt you agree, these are not memorable examples of the typesetter's art but do indicate that you can define your own verbatim environments and may need to take a bit of care to get something that passes muster.

I will give some more useful examples, but mainly based on environments for writing verbatim files as I think that these provide a broader scope.

4.3.3 Example: the `lcode` environment

In this manual all the example LaTeX code has been typeset in the `lcode` environment; this is a verbatim environment defined especially for the purpose. Below I describe the code for defining my `lcode` environment, but first here is a simple definition of a verbatim environment, which I will call `smallverbatim`, that uses the `\small` font instead of the `\normalsize` font.


```

\newenvironment{smallverbatim}%
  {\setverbatimfont{\normalfont\ttfamily\small}}%
  \verbatim}%
{\endverbatim}

```

The verbatim environment is implemented as a kind of `trivlist`, and lists usually have extra vertical space before and after them. For my environment I did not want any extra spacing so I defined the macro `\@zerosecs` to zero the relevant list spacings. I also wanted the code lines to be inset a little, so I defined a new length called `\gparindent` to use as the indentation.

```

\makeatletter
\newcommand{\@zerosecs}{\setlength{\topsep}{\z@}%
                        \setlength{\partopsep}{\z@}%
                        \setlength{\parskip}{\z@}}
\newlength{\gparindent} \setlength{\gparindent}{\parindent}
\setlength{\gparindent}{0.5\parindent}
% Now, the environment itself
\newenvironment{lcode}{\@zerosecs
  \renewcommand{\verbatim@startline}{%
    \verbatim@line{\hskip\gparindent}}
  \small\setlength{\baselineskip}{\onelineskip}\verbatim}%
{\endverbatim
  \vspace{-\baselineskip}}%
\noindent
}
\makeatother

```

Unless you are intimately familiar with the inner workings of the verbatim processing you deserve an explanation of the `lcode` definition.

Extremely roughly, the code for `\verbatim` looks like this:

```

\def\verbatim{%
  \verbatim@font
  % for each line, until \end{verbatim}
  \verbatim@startline
  % collect the characters in \verbatim@line
  \verbatim@processline{\the\verbatim@line\par}
  % repeat for the next line
}

```

The code first calls `\verbatim@font` to set the font to be used. Then, for each line it does the following:

- Calls the macro `\verbatim@startline` to start off the output version of the line.
- Collects all the characters comprising the line as a single token called `\verbatim@line`.
- If the characters are the string `'\end{verbatim}'` it finishes the verbatim environment.

- Otherwise it calls the macro `\verbatim@processline` whose argument is the characters in the line, treated as a paragraph. It then starts all over again with the next line.

I configured the `\verbatim@startline` macro to indent the line of text using a horizontal skip of `\gparindent`. The rest of the initialisation code, before calling `\verbatim` to do the real processing, just sets up the vertical spacing.

4.4 FILES

LaTeX reads and writes various files as it processes a document. Obviously it reads the document source file, or files, and it writes the log file recording what it has done. It also reads and writes the aux file, and may read and write other files like a toc file.

On occasions it can be useful to get LaTeX to read and/or write other files of your own choosing. Unfortunately standard LaTeX does not provide any easy method for doing this. The memoir class tries to rectify this.

`\jobname`

When you run LaTeX on your source file, say `fred.tex`, LaTeX stores the name of this file (fred) in the macro `\jobname`. LaTeX uses this to name the various files that it writes out — the dvi or pdf file, the log file, the aux file, etc.

TeX can read from 16 input streams and can write to 16 output streams. Normally an input stream is allocated for each kind of file that will be read and an output stream for each kind of file that will be written. On the input side, then, at least two streams are allocated, one for the source tex file and one for the aux file. On the output side again at least two streams are allocated, one for the log file and one for the aux file. When toc and other similar files are also part of the LaTeX process you can see that many of the 16 input and output streams may be allocated before you can get to use one yourself.

`\newoutputstream{<stream>}`
`\newinputstream{<stream>}`

The macros `\newoutputstream` and `\newinputstream` respectively create a new output and input stream called `<stream>`, where `<stream>` should be a string of alphabetic characters, like `myout` or `myin`. The `<stream>` names must be unique, you cannot use the same name for two streams even if one is a input stream and the other is an output stream. If all the 16 streams of the given type have already been allocated TeX will issue a message telling you about this, of the form:

```
No room for a new write    % for an output stream
No room for a new read     % for an input stream
```

The two `\new...stream` commands also provide two empty macros called `\atstreamopen<stream>` and `\atstreamclose<stream>`. If these macros already exist then they are left undisturbed. For example if you do:

```
\newcommand{\atstreamopenmyout}{...}
\newoutputstream{myout}
\newinputstream{myin}
```

Then you will find that three new commands have been created like:

```
\newcommand{\atstreamclosemyout}{}
\newcommand{\atstreamopenmyin}{}
\newcommand{\atstreamclosemyin}{}

```

You can use `\renewcommand` to change the definitions of these if you wish.

```
\IfStreamOpen{<stream>}{<true-code>}{<false-code>}
```

The macro `\IfStreamOpen` checks whether or not the `<stream>` stream is open. If it is then the `<true-code>` argument is processed, while when it is not open the `<false-code>` argument is processed.

4.4.1 Writing to a file

One stream may be used for writing to several different files, although not simultaneously.

```
\openoutputfile{<filename>}{<stream>}
\closeoutputstream{<stream>}

```

The command `\openoutputfile` opens the file called `<filename>`, either creating it if it does not exist, or emptying it if it already exists. It then attaches the file to the output stream called `<stream>` so that it can be written to, and then finally calls the macro named `\atstreamopen<stream>`.

The command `\closeoutputstream` firstly calls the macro named `\atstreamclose<stream>` then closes the output stream `<stream>`, and finally detaches and closes the associated file.

```
\addtostream{<stream>}{<text>}
```

The `\addtostream` command writes `<text>` to the output stream `<stream>`, and hence to whatever file is currently attached to the stream. The `<stream>` must be open. Any commands within the `<text>` argument will be processed before being written. To prevent command expansion, precede the command in question with `\protect`.

Writing verbatim text to a file is treated specially as it is likely to be the most common usage.

```
\begin{verbatimoutput}{<file>} anything \end{verbatimoutput}
\begin{writeverbatim}{<stream>} anything \end{writeverbatim}

```

The text within a `verbatimoutput` environment is written verbatim to the `<file>` file. Alternatively, the contents of the `writeverbatim` environment are written verbatim to the `<stream>` stream.

Specifically, `verbatimoutput` opens the specified file, writes to it, and then closes the file. This means that if `verbatimoutput` is used more than once to write to a given file, then only the contents of the last of these outputs is captured in the file. On the other hand, you can use `writeverbatim` several times to write to the file attached to the stream and, providing the stream has not been closed in the meantime, all will be captured.

4.4.2 Reading from a file

One stream may be used for reading from several files, although not simultaneously.

```
\openinputfile{<filename>}{<stream>}
\closeinputstream{<stream>}

```

The command `\openinputfile` opens the file called `<filename>` and attaches it to the input stream called `<stream>` so that it can be read from. Finally it calls the macro named `\atstreamopen<stream>`. It is an error if `<filename>` can not be found.

The command `\closeinputstream` calls the macro named `\atstreamclose<stream>`, closes the output stream `<stream>`, and then detaches and closes the associated file.

`\readstream{<stream>}`

The command `\readstream` reads the entire contents of the file currently associated with the input stream `<stream>`. This provides the same functionality as `\input{<filename>}`.

`\readaline{<stream>}`

The `\readaline` reads what TeX considers to be one line from the file that is currently associated with the input stream `<stream>`.

Multiple lines can be read by calling `\readaline` multiple times. A warning is issued if there are no more lines to be read (i.e., the end of the file has been reached).

Just as for writing, reading files verbatim is treated specially.

`\verbatiminput{<file>} \verbatiminput*{<file>}`
`\boxedverbatiminput{<file>} \boxedverbatiminput*{<file>}`
`\readverbatim{<stream>} \readverbatim*{<stream>}`
`\readboxedverbatim{<stream>} \readboxedverbatim*{<stream>}`

The commands `\verbatiminput` and `\boxedverbatiminput`, and their starred versions, act like the `verbatim` and `boxedverbatim` environments, except that they get their text from the `<file>` file. It is an error if `<file>` cannot be found. Similarly, `\readverbatim` and `\readboxedverbatim` get their text from the file currently attached to the `<stream>` input stream. It is an error if `<stream>` is not open for input.

4.4.3 Example: endnotes

In an earlier version of the manual, this section contained an example as to how one could make endnotes. The example is now irrelevant, since memoir contain something similar to end notes called page notes, see section 2.2 on page 7.

Those interested in the code from the old example, can find it in the manual source (it has just been commented out).

4.4.4 Example: end floats

There are some documents where all figures are required to be grouped in one place, for instance at the end of the document or perhaps at the end of each chapter. Grouping at the end of a document with chapters is harder, so we'll tackle that one.

The basic idea is to write out verbatim each figure environment and then read them all back in at the end. We will use a stream, let's call it `tryout`, and call our file for figures `tryout.fig`.

```
\newoutputstream{tryout}
\openoutputfile{tryout.fig}{tryout}
```

If all were simple, in the document we could then just do

```
\begin{writeverbatim}{tryout}
\begin{figure} ... \end{figure}
\end{writeverbatim}
```

```
...
\closeoutputstream{tryout}
\input{tryout.fig}
```

So, what's the problem?

By default figure captions are numbered per chapter, and are preceded by the chapter number; more precisely, the definition of a figure number is

```
\thechapter.\arabic{figure}
```

If we simply lump all the figures at the end, then they will all be numbered as if they were in the final chapter. For the sake of argument assume that the last chapter is number 10. The *n*th figure will then be numbered 10.*n*. One thing that we can do rather simply is to change the definition of the figure by using another counter, let's call it *pseudo*, instead of the chapter.

```
\newcounter{pseudo}
\renewcommand{\thepseudo}{\arabic{pseudo}}
\renewcommand{\thefigure}{\thepseudo.\arabic{figure}}
```

Now, all we should have to do is arrange that the proper value of *pseudo* is available before each figure is typeset at the end. The code around the `figure` environments might then look like this

```
\addtostream{tryout}{\protect\setcounter{pseudo}{\thechapter}}
\begin{writeverbatim}{tryout}
\begin{figure}...
```

and a part of the file might then look like

```
...
\setcounter{pseudo}{4}
\begin{figure}...
```

The `\protect` before the `\setcounter` command will stop it from expanding before it is written to the file, while the `\thechapter` command *will* be expanded to give the actual number of the current chapter. This looks better as now at least the figure will be numbered 4.*n* instead of 10.*n*.

There is one last snag — figure numbers are reset at the start of each chapter — but if we just dump the figures at the end of the document then although the chapter part of the number will alter appropriately because of the *pseudo* process, the second part of the number will just increase continuously. It looks as though we should write out a change to the chapter counter at the start of each chapter. If we do that, then we should be able to get rid of the *pseudo* counter, which sounds good. But, and this is almost the last but, what if there are chapters after we have read in the figure file? To cater for this the chapter number of the last chapter before the file must be saved, and then restored after the figures have been processed.

Finally, wouldn't it be much better for the user if everything was wrapped up in an environment that handled all the messy stuff?

Here is the final code that I am going to produce which, by the way, is displayed in the `boxedverbatim` environment with line numbers and the following settings, just in case there is a page break in the middle of the box.

```
\nobvbox
\bvperpagetrue
\bvtopofpage{\begin{center}\normalfont%
              (Continued from previous page)\end{center}}
\bvendofpage{}
\resetbvlینumber
\linenumberfrequency{1}
\bvnumbersoutside
\linenumberfont{\footnotesize\rmfamily}
\begin{boxedverbatim}
...

1 \newoutputstream{tryout}
2 \openoutputfile{\jobname.fig}{tryout}
3 \newcounter{pseudo}
4 \renewcommand{\thefigure}{\thepseudo.\arabic{figure}}
5 \newenvironment{writefigure}{%
6   \ifnum\value{chapter}=\value{pseudo}\else
7     \setcounter{pseudo}{\value{chapter}}
8     \addtostream{tryout}{\protect\stepcounter{chapter}}
9     \addtostream{tryout}{\protect\addtocounter{chapter}{-1}}
10    \addtostream{tryout}{%
11      \protect\setcounter{pseudo}{\thechapter}}
12  \fi
13  \addtostream{tryout}{\protect\begin{figure}}
14  \writeverbatim{tryout}}%
15  {\endwriteverbatim\finishwritefigure}
16 \newcommand{\finishwritefigure}{%
17   \addtostream{tryout}{\protect\end{figure}}}
18 \newcommand{\printfigures}{%
19   \closeoutputstream{tryout}%
20   \input{\jobname.fig}%
21 }
```

The above code should be either put in the preamble or in a separate package file.

The first four lines of the code perform the initial setup described earlier. Lines 1 and 2 set up for outputting to a file `\jobname.fig`, which is where the figures will be collected. Lines 3 and 4 create the new counter we need and change the construction of the figure number. The rest of the code defines a new environment `writefigure` which is to be used instead of the `figure` environment. It writes its content out to the `tryout` stream.

In line 6 a check is made to see if the current values of the `chapter` and `pseudo` counters are the same; nothing is done if they are. If they are different, it means that this is the first figure in the chapter and we have to put appropriate information into the figure file. Line 7 sets the `pseudo` counter to the value of the `chapter` counter (if there is another `writefigure` in the chapter it will then skip over the code in lines 7 to 11). The next lines put (where `N` is the number of the current chapter):

```

\stepcounter{chapter}
\addtocounter{chapter}{-1}
\setcounter{pseudo}{N}

```

into the figure file. Stepping the chapter number (by one) resets the following figure number, and then subtracting one from the stepped number returns the chapter number to its original value. Finally the counter `pseudo` is set to the number of the current chapter.

Line 13 puts

```
\begin{figure}
```

into the figure file, and line 14 starts the `writeverbatim` environment.

For the end of the `writefigure` environment (line 15), the `writeverbatim` environment is ended and after that the `\finishwritefigure` macro is called. This is defined in lines 16 and 17, and simply writes

```
\end{figure}
```

out to the figure file. The `\endwriteverbatim`, and any other kind of `\end...verbatim`, command is very sensitive to anything that follows it, and in this case did not like to be immediately followed by an `\addtostream{...}`, but did not mind it being wrapped up in the `\finishwritefigure` macro.

The `\printfigures` macro defined in the last three lines of the code simply closes the output stream and then inputs the figures file.

As an example of how this works, if we have the following source code:

```

\chapter{The fifth chapter}
...
\begin{writefigure}
%% illustration and caption
\end{writefigure}
...
\begin{writefigure}
%% another illustration and caption
\end{writefigure}

```

then the figure file will contain the following (shown verbatim in the `fboxverbatim` environment).

```

\stepcounter{chapter}
\addtocounter{chapter}{-1}
\setcounter{pseudo}{5}
\begin{figure}
%% illustration and caption
\end{figure}
\begin{figure}
%% another illustration and caption
\end{figure}

```

4.4.5 Example: questions and answers

Text books often have questions at the end of a chapter. Sometimes answers are also provided at the end of the book, or in a separate teachers guide. During the draft stages of

such a book it is useful to keep the questions and answers together in the source and paper drafts, only removing or repositioning the answers towards the end of the writing process.

This example provides an outline for meeting these desires. For pedagogical purposes I use a `\label` and `\ref` technique although there are better methods. The example also shows that not everything works as expected — it is a reasonably accurate rendition of the process that I actually went through in designing it.

First we need a counter for the questions and we'll use an environment for questions as these may be of any complexity. The environment takes one argument — a unique key to be used in a `\label`.

```
\newcounter{question} \setcounter{question}{0}
\renewcommand{\thequestion}{\arabic{question}}
\newenvironment{question}[1]%
  {\refstepcounter{question}
   \par\noindent\textbf{Question \thequestion:}\label{#1}}%
  {\par}
```

I have used `\refstepcounter` to increment the counter so that the `\label` will refer to it, and not some external counter.

We will use a file, called `\jobname.ans` to collect the answers and this will be written to by a stream. There is also a convenience macro, `\printanswers`, for the user to call to print the answers.

```
\newoutputstream{ansout}
```

A matching environment for answers is required. The argument to the environment is the key of the question.

In draft mode it is simple, just typeset the answer and no need to bother with any file printing (remember that `\ifdraftdoc` is true for a draft mode document).

```
\ifdraftdoc % when in draft mode
\newenvironment{answer}[1]%
  {\par\noindent\textbf{Answer \ref{#1}:}}%
  {\par}
\newcommand{\printanswers}{}
\else % when not in draft mode
```

In final mode the answer environment must write its contents verbatim to the `ans` file for printing by `\printanswers`. Dealing with these in reverse order, this is the definition of `\printanswer` when not in draft mode.

```
\newcommand{\printanswers}{%
  \closeoutputstream{ansout}
  \input{\jobname.ans}}
```

Now for the tricky bit, the answer environment. First define an environment that makes sure our output stream is open, and which then writes the answer title to the stream.

```
\newenvironment{@nswer}[1]{\@bsphack
  \IfStreamOpen{ansout}{}{%
    \openoutputfile{\jobname.ans}{ansout}%
  }%
```



```
\addtostream{ansout}{\par\noindent\textbf{Answer \ref{#1}:}}%
}{\@esphack}
```

The macros `\@bsphack` and `\@esphack` are LaTeX kernel macros that will gobble extraneous spaces around the environment. In other words, this environment will take no space in the typeset result. The `\IfStreamOpen` macro is used to test whether or not the stream is open, and if it isn't then it opens it. The answer title is then written out to the stream. Now we can define the `answer` environment so that its contents get written out verbatim.

```
\newenvironment{answer}[1]%
  {\@bsphack\@answer{#1}\writeverbatim{ansout}}%
  {\par\endwriteverbatim\end@answer\@esphack}
\fi % end of \ifdraftdoc ... \else ...
```

When I was testing this code I had a surprise as I got nasty error messages from LaTeX the first time around, but it worked fine when I processed the source a second time! The problem lies in the code line

```
\addtostream{ansout}{\par\noindent\textbf{Answer \ref{#1}:}}%
```

The first time around, LaTeX processed the `\ref` command and of course it was undefined. In this case `\ref` gets replaced by the code to print the error message, which involves macros that have `@` in their names, which LaTeX only understands under special circumstances. The second time around `\ref` gets replaced by the question number and all is well. I then remembered that some commands need protecting when they are written out, so I tried (I've wrapped the line to fit)

```
\addtostream{ansout}{\par\noindent
  \protect\makeatletter\textbf{Answer
  \protect\ref{#1}:}\protect\makeatother}%
```

which did work but seemed very clumsy.

I then took another line of attack, and looked at the definition of `\ref` to see if I could come up with something that didn't expand into `@` names. The result of this was

```
\addtostream{ansout}{\par\noindent\textbf{Answer
  \quietref{#1}:}}%
```

In the kernel file `ltxref.dtx` I found the definition of `\ref` and it used a macro `\@setref` (shown below) to do its work. My `\quietref` locally changes the definition of `\@setref` and then calls `\ref`, which will then use the modified `\@setref`.

```
\def\@setref#1#2#3{% % kernel definition
  \ifx#1\relax
    \protect\G@refundefinedtrue
    \nfss@text{\reset@font\bfseries ??}%
    \@latex@warning{Reference '#3' on page \thepage \space
      undefined}%
  \else
    \expandafter#2#1\null
  \fi}
```

```
\DeclareRobustCommand{\quietref}[1]{\begingroup
\def\@setref##1##2##3{%
\ifx##1\relax ??\else
\expandafter##2##1\null
\fi
\ref{#1}\endgroup}
```

Having gone all round the houses, the simplest solution was actually one that I had skipped over

```
\addtostream{ansout}{\par\noindent\textbf{Answer
\protect\ref{#1}:}}%
```

The advantage of using the `\label` and `\ref` mechanism is that a question and its answer need not be adjacent in the source; I think that you have seen some of the disadvantages. Another disadvantage is that it is difficult to use, although not impossible, if you want the answers in a separate document.

The real answer to all the problems is force an answer to come immediately after the question in the source and to use the question counter directly, as in the endnotes example. In the traditional manner, this is left as an exercise for the reader.

4.5 ANSWERS

Question 1. As a convenience, the argument to the environment could be made optional, defaulting, say, to the current line width. If the default width is used the frame will be wider than the line width, so we really ought to make the width argument specify the width of the frame instead of the minipage. This means calculating a reduced width for the minipage based on the values of `\fboxsep` and `\fboxrule`.

```
\newsavebox{\minibox}
\newlength{\minilength}
\newenvironment{framedminipage}[1][\linewidth]{%
\setlength{\minilength}{#1}
\addtolength{\minilength}{-2\fboxsep}
\addtolength{\minilength}{-2\fboxrule}
\begin{lrbox}{\minibox}\begin{minipage}{\minilength}}%
{\end{minipage}\end{lrbox}\fbox{\usebox{\minibox}}}
```

Question 2. There are at least three reasonable answers. In increasing or decreasing order of probability (your choice) they are:

- I took Sherlock Holmes' advice and followed the methods outlined in the chapter;
- I used a package, such as the answer package which is designed for the purpose;
- I just wrote the answers here.

Five

Cross referencing

A significant aspect of LaTeX is that it provides a variety of cross referencing methods, many of which are automatic. An example of an automatic cross reference is the way in which a `\chapter` command automatically adds its title and page number to the ToC, or where a `\caption` adds itself to a ‘List of...’.

Some cross references have to be specifically specified, such as a reference in the text to a particular chapter number, and for these LaTeX provides a general mechanism that does not require you to remember the particular number and more usefully does not require you to remember to change the reference if the chapter number is later changed.

5.1 LABELS AND REFERENCES

The general LaTeX cross reference method uses a pair of macros.

```
\label{<labstr>}
\ref{<labstr>} \pageref{<labstr>}
```

You can put a `\label` command where you want to label some numbered element in case you might want to refer to the number from elsewhere in the document. The `<labstr>` argument is a sequence of letters, digits, and punctuation symbols; upper and lower case letters are different. The `\ref` macro prints the number associated with `<labstr>`. The `\pageref` macro prints the number of the page where the `\label` specifying the `<labstr>` was placed.

The `\label` and `\ref` mechanism is simple to use and works well but on occasions you might be surprised at what `\ref` prints.

LaTeX maintains a current *ref* value which is set by the `\refstepcounter` command. This command is used by the sectioning commands, by `\caption`, by numbered environments like `equation`, by `\item` in an `enumerate` environment, and so on. The `\label` command writes an entry in the aux file consisting of the `<labstr>`, the current *ref* value and the current page number. A `\ref` command picks up the *ref* value for `<labstr>` and prints it. Similarly `\pageref` prints the page number for `<labstr>`.

The critical point is that the `\label` command picks up the value set by the *most recent visible*¹ `\refstepcounter`.

- A `\label` after a `\section` picks up the `\section` number, not the `\chapter` number.
- A `\label` after a `\caption` picks up the caption number.
- A `\label` *before* a `\caption` picks up the surrounding sectional number.

¹Remember, a change within a group, such as an environment, is not visible outside the group.

If you are defining your own macro that sets a counter, the counter value will be invisible to any `\label` unless it is set using `\refstepcounter`.

```
\fref{<labstr>} \figurerefname  
\tref{<labstr>} \tablerefname  
\pref{<labstr>} \pagerefname
```

The class provides these more particular named references to a figure, table or page. For example the default definitions of `\fref` and `\pref` are

```
\newcommand{\fref}[1]{\figurerefname~\ref{#1}}  
\newcommand{\pref}[1]{\pagerefname~\pageref{#1}}
```

and can be used as

```
\ldots footnote parameters are shown in~\fref{fig:fn}  
on~\pref{fig:fn}.
```

which in this document prints as:

```
...footnote parameters are shown in Figure ?? on page ??.
```

```
\Aref{<labstr>} \appendixrefname  
\Bref{<labstr>} \bookrefname  
\Pref{<labstr>} \partrefname  
\Cref{<labstr>} \chapterrefname  
\Sref{<labstr>} \sectionrefname
```

Similarly, specific commands are supplied for referencing sectional divisions; `\Aref` for Appendix , `\Bref` for Book , `\Pref` for Part , `\Cref` for Chapter , and `\Sref` for divisions below Chapter . For example:

```
This is \Sref{sec:lab&ref} in \Cref{chap:xref}.
```

This is §5.1 in Chapter 5.

5.2 REFERENCE BY NAME

In technical works it is common to reference a chapter, say, by its number. In non-technical works such cross references are likely to be rare, and when they are given it is more likely that the chapter title would be used instead of the number, as in:

```
The chapter \textit{\titleref{chap:bringhurst}} describes \ldots
```

The chapter *An example book design* describes ...

There are two packages, `nameref` [Rahtz01] and `titleref` [Ars01a], that let you refer to things by name instead of number.

Name references were added to the class as a consequence of adding a second optional argument to the sectioning commands. I found that this broke the `nameref` package, and hence the `hyperref` package as well, so they had to be fixed. The change also broke Donald Arseneau's `titleref` package, and it turned out that `nameref` also clobbered `titleref`. The class also provides titles, like `\poemtitle`, that are not recognised by either of the packages. From my viewpoint the most efficient thing to do was to enable the class itself to provide name referencing.

```
\titleref{<labstr>}
```

Typeset example 5.1: Named references should be to titled elements

Labels may be applied to:

1. Chapters, sections, etc.
2. Captions
3. Legends
4. Poem titles
5. Items in numbered lists, etc. ...

Item 1 in section *Reference by name* mentions sections while item *Named references should be to titled elements*, on page 51 in the same section, mentions things like items in enumerated lists that should not be referenced by `\titleref`.

The macro `\titleref` is a class addition to the usual set of cross referencing commands. Instead of printing a number it typesets the title associated with the labelled number. This is really only useful if there *is* a title, such as from a `\caption` or `\section` command. For example, look at this code and its result.

Source for example 5.1

```
Labels may be applied to:
\begin{enumerate}
\item Chapters, sections, etc.          \label{sec:nxref:1}
...
\item Items in numbered lists, etc. \ldots \label{sec:nxref:5}
\end{enumerate}
Item \ref{sec:nxref:1} in section \textit{\titleref{sec:nxref}}
mentions sections while item \titleref{sec:nxref:5}, on page
\pageref{sec:nxref:5} in the same section, mentions things like
items in enumerated lists that should not be referenced
by \verb?\titleref?.
```

As the above example shows, you have to be a little careful in using `\titleref`. Generally speaking, `\titleref{<key>}` produces the last named thing before the `\label` that defines the `<key>`.

`\headnameref \tocnameref`

There can be three possibilities for the name of a sectional division; the full name, the name in the ToC, and the name in the page header. As far as `\titleref` is concerned it does not use the fullname, and so the choice simplifies to the ToC or page header. Following the declaration `\headnameref` it uses the page header name. Following the opposite declaration `\tocnameref`, which is the default, it uses the ToC name.

NOTE: Specifically with the memoir class, do not put a `\label` command inside an

Typeset example 5.2: Current title

This sentence in the section titled ‘Current title’ is an example of the use of the command `\currenttitle`.

argument to a `\chapter` or `\section` or `\caption`, etc., command. Most likely it will either be ignored or referencing it will produce incorrect values. This restriction does not apply to the standard classes, but in any case I think it is good practice not to embed any `\label` commands.

`\currenttitle`

If you just want to refer to the current title you can do so with `\currenttitle`. This acts as though there had been a label associated with the title and then `\titleref` had been used to refer to that label. For example:

Source for example 5.2

This sentence in the section titled ‘`\currenttitle`’ is an example of the use of the command `\verb?\currenttitle?`.

`\theTitleReference{<num>}{<text>}`

Both `\titleref` and `\currenttitle` use the `\theTitleReference` to typeset the title. This is called with two arguments — the number, `<num>`, and the text, `<text>`, of the title. The default definition is:

```
\newcommand{\theTitleReference}[2]{#2}
```

so that only the `<text>` argument is printed. You could, for example, change the definition to

```
\renewcommand{\theTitleReference}[2]{#1\space \textit{#2}}
```

to print the number followed by the title in italics. If you do this, only use `\titleref` for numbered titles, as a printed number for an unnumbered title (a) makes no sense, and (b) will in any case be incorrect.

The commands `\titleref`, `\theTitleReference` and `\currenttitle` are direct equivalents of those in the `titleref` package [Ars01a].

`\namerefon \nameref`

The capability for referencing by name has one potentially unfortunate side effect — it causes some arguments, such as that for `\legend`, to be moving arguments and hence any fragile command in the argument will need `\protecting`. However, not every document will require the use of `\titleref` and so the declaration `\nameref` is provided to switch it off (the argument to `\legend` would then not be moving). The declaration `\namerefon`, which is the default, enables name referencing.

Six

Back matter

The back matter consists of reference and supportive elements for the main matter; things like bibliographies, indexes, glossaries, endnotes, and other material. The class provides additional elements and features of such matter that are not in the standard classes.

6.1 BIBLIOGRAPHY

Just as a reminder the bibliography is typeset within the `thebibliography` environment.

```
\bibname
\begin{thebibliography}{\langle exlabel \rangle}
\bibitem ...
\end{thebibliography}
```

The environment takes one required argument, *\langle exlabel \rangle*, which is a piece of text as wide as the widest label in the bibliography. The value of `\bibname` (default ‘Bibliography’) is used as the title.

```
\bibintoc \nobibintoc
```

The declaration `\bibintoc` will cause the `thebibliography` environment to add the title to the ToC, while the declaration `\nobibintoc` ensures that the title is not added to the ToC. The default is `\bibintoc`.

```
\cite[\langle detail \rangle]{\langle labstr-list \rangle}
```

Within the text you call out a bibliographic reference using the `\cite` command, where *\langle labstr-list \rangle* is a comma-separated list of identifiers for the cited works; there must be no spaces in this list. The optional *\langle detail \rangle* argument is for any additional information regarding the citation such as a chapter or page number; this is printed after the main reference.

Various aspects of a bibliography can be changed and at this point it may be helpful to look at some of the internals of the `thebibliography` environment, which is defined like this

```
\newenvironment{thebibliography}[1]{%
  \bibsection
  \begin{bibitemlist}{#1}}{%
  {\end{bibitemlist}}\postbibhook}
```

The bibliographic entries are typeset as a list, the `bibitemlist`.

```
\bibsection
```

The macro `\bibsection` defines the heading for the `thebibliography` environment; that is, everything before the actual list of items. It is effectively defined as

```
\newcommand{\bibsection}{%
  \chapter*{\bibname}
  \bibmark
  \ifnobibintoc\else
    \phantomsection
    \addcontentsline{toc}{chapter}{\bibname}
  \fi
  \prebibhook}
```

If you want to change the heading, redefine `\bibsection`. For example, to have the bibliography as a numbered section instead of an unnumbered chapter, redefine it like

```
\renewcommand{\bibsection}{%
  \section{\bibname}
  \prebibhook}
```

If you use the `natbib` [Dal99a] and/or the `chapterbib` [Ars01b] packages with the `sectionbib` option, then they change `\bibsection` appropriately to typeset the heading as a numbered section.

`\bibmark`

`\bibmark` may be used in pagestyles for page headers in a bibliography. Its default definition is:

```
\newcommand*{\bibmark}{}
```

but could be redefined like, say,

```
\renewcommand*{\bibmark}{\markboth{\bibname}{}}
```

`\prebibhook \postbibhook`

The commands `\prebibhook` and `postbibhook` are called after typesetting the title of the bibliography and after typesetting the list of entries, respectively. By default, they are defined to do nothing. You may wish to use one or other of these to provide some general information about the bibliography. For example:

```
\renewcommand{\prebibhook}{%
CTAN is the Comprehensive \tex\ Archive Network and URLs for the
several CTAN mirrors can be found at \url{http://www.tug.org}.}
```

`\biblistextra`

Just at the end of setting up the `bibitemlist` the `\biblistextra` command is called. By default this does nothing but you may change it to do something useful. For instance, it can be used to change the list parameters so that the entries are typeset flushleft.

```
\renewcommand*{\biblistextra}{%
  \setlength{\leftmargin}{0pt}%
  \setlength{\itemindent}{\labelwidth}%
  \addtolength{\itemindent}{\labelsep}}
```



```
\setbiblabel{<style>}
```

The style of the labels marking the bibliographic entries can be set via `\setbiblabel`. The default definition is

```
\setbiblabel{[#1]\hfill}
```

where `#1` is the citation mark position, which generates flushleft marks enclosed in square brackets. To have marks just followed by a dot

```
\setbiblabel{#1.\hfill}
```

```
\bibitem[<label>]{<labstr>}
\newblock
```

Within the `bibitemlist` environment the entries are introduced by `\bibitem` instead of the more normal `\item` macro. The required `<labstr>` argument is the identifier for the citation and corresponds to a `<labstr>` for `\cite`. The items in the list are normally labelled numerically but this can be overridden by using the optional `<label>` argument. The `\newblock` command can be used at appropriate places in the entry for encouraging a linebreak (this is used by the `openbib` option).

```
\bibitemsep
```

In the listing the vertical space between entries is controlled by the length `\bibitemsep`, which by default is set to the normal `\itemsep` value. The vertical space is `(\bibitemsep + \parsep)`. If you wish to eliminate the space between items do

```
\setlength{\bibitemsep}{-\parsep}
```

6.1.1 BibTeX

Often the BibTeX program [Pat88a] is used to generate the bibliography list from database(s) of bibliographic data. For BibTeX a bibliographic data base is a `bib` file containing information necessary to produce entries in a bibliography. BibTeX extracts the raw data from the files for each citation in the text and formats it for typesetting according to a particular style.

```
\bibliography{<bibfile-list>}
```

The bibliography will be printed at the location of the `\bibliography` command. Its argument is a comma-separated list of BibTeX `bib` files which will be searched by BibTeX to generate the bibliography. Only the file name(s) should be supplied, the extension must not be given.

```
\nocite{<labstr>} \nocite{*}
```

The command `\nocite` causes BibTeX to make an entry in the bibliography but no citation will appear in the text. The special case `\nocite{*}` causes *every* entry in the database to be listed in the bibliography.

```
\bibliographystyle{<style>}
```

Many different BibTeX styles are available and the particular one to be used is specified by calling `\bibliographystyle` before the bibliography itself. The ‘standard’ bibliography `<style>s` follow the general schemes for mathematically oriented papers and are:

- plain** The entry format is similar to one suggested by Mary-Claire van Leunen [Leu92], and entries are sorted alphabetically and labelled with numbers.
- unsrt** The format is the same as **plain** but that entries are ordered by the citation order in the text.
- alpha** The same as **plain** but entries are labelled like ‘Wil89’, formed from the author and publication date.
- abbrv** The same as **plain** except that some elements, like month names, are abbreviated.

There are many other styles available, some of which are used in collaboration with a package, one popular one being Patrick Daly’s `natbib` [Dal99a] package for the kinds of author-year citation styles used in the natural sciences. Another package is `jurabib` [Ber02] for citation styles used in legal documents where the references are often given in footnotes rather than listed at the end of the document.

I assume you know how to generate a bibliography using BibTeX, so this is just a quick reminder. You first run LaTeX on your document, having specified the bibliography style, cited your reference material and listed the relevant BibTeX database(s). You then run BibTeX, and after running LaTeX twice more the bibliography should be complete. After a change to your citations you have to run LaTeX once, BibTeX once, and then LaTeX twice more to get an updated set of references.

The format and potential contents of a BibTeX database file (a `bib` file) are specified in detail in Lamport [Lam94] and the first of the *Companions* [MG⁺04]. Alternatively there is the documentation by Oren Patashnik [Pat88a] who wrote the BibTeX program.

A BibTeX style, specified in a `bst` file, is written using an anonymous stack based language created specifically for this purpose. If you can’t find a BibTeX style that provides what you want you can either use the `makebst` [Dal99b] package which leads you through creating your own style via a question and answer session, or you can directly write your own. If you choose the latter approach then Patashnik’s *Designing BibTeX files* [Pat88b] is essential reading. As he says, it is best to take an existing style and modify it rather than starting from scratch.

6.2 INDEX

It is time to take a closer look at indexing. The class allows multiple indexes and an index may be typeset as a single or a double column.

The general process is to put indexing commands into your source text, and LaTeX will write this raw indexing data to an `idx` file. The raw index data is then processed, not by LaTeX but by yourself if you have plenty of spare time on your hands, or more usually by a separate program, to create a sorted list of indexed items in a second file (usually an `ind` file). This can then be input to LaTeX to print the sorted index data.

6.2.1 Printing an index

<pre>\makeindex[⟨file⟩] \printindex[⟨file⟩]</pre>

In order to make LaTeX collect indexing information the declaration `\makeindex` must be put in the preamble. By default the raw index data is put into the `jobname.idx` file. If the optional `⟨file⟩` argument is given then index data can be output to `file.idx`. Several `\makeindex` declarations can be used provided they each call for a different file.

The `\printindex` command will print an index where by default the indexed items are assumed to be in a file called `jobname.ind`. If the optional *(file)* argument is given then the indexed items are read from the file called `file.ind`.

```
\begin{theindex} entries \end{theindex}
\onecolindex \twocolindex
\indexname
```

The index entries are typeset within the `theindex` environment. By default it is typeset with two columns but following the `\onecolindex` declaration the environment uses a single column. The default two column behaviour is restored after the `\twocolindex` declaration. The index title is given by the current value of `\indexname` (default 'Index').

```
\indexintoc \noindexintoc
```

The declaration `\indexintoc` will cause the `theindex` environment to add the title to the ToC, while the declaration `\noindexintoc` ensures that the title is not added to the ToC. The default is `\indexintoc`.

```
\indexcolsep
\indexrule
```

The length `\indexcolsep` is the width of the gutter between the two index columns. The length `\indexrule`, default value 0pt, is the thickness of a vertical rule separating the two columns.

```
\preindexhook
```

The macro `\preindexhook` is called after the title is typeset and before the index listing starts. By default it does nothing but can be changed. For example

```
\renewcommand{\preindexhook}{\bold{page numbers are used
to indicate the main reference for an entry.}}
```

```
\indexmark
```

`\indexmark` may be used in pagestyles for page headers in an index. Its default definition is:

```
\newcommand*{\indexmark}{{}}
```

but could be redefined like, say,

```
\renewcommand*{\indexmark}{\markboth{\indexname}{\indexname}}
```

```
\ignorenoidxfile
\reportnoidxfile
```

Following the declaration `\ignorenoidxfile`, which is the default, LaTeX will silently pass over attempts to use an `idx` file which has not been declared via `\makeindex`. After the declaration `\reportnoidxfile` LaTeX will whine about any attempts to write to an unopened file.

6.2.2 Preparing an index

It is easy for a computer to provide a list of all the words you have used, and where they were used. This is called a concordance. Preparing an index, though, is not merely a gathering of words but is an intellectual process that involves recognising and naming

concepts, constructing a logical hierarchy of these and providing links between related concepts. No computer can do that for you though it can help with some tasks, such as sorting things into alphabetical order, eliminating duplicates, and so forth.

Several iterations may be required before you have an acceptable index. Generally you pick out the important words or phrases used on the first pass. Part of the skill of indexing is finding appropriate words to describe things that may not be obvious from the text. If there are several ways of describing something they may all be included using a ‘see’ reference to the most obvious of the terms, alternatively you could use ‘see also’ references between the items. Entries should be broken down into subcategories so that any particular item will not have a long string of page numbers and your reader is more likely to quickly find the relevant place. After having got the first index you will most probably have to go back and correct all the sins of omission and commission, and start the cycle again.

I found that indexing this manual was the most difficult part of preparing it. It was easy to index the names of all the macros, environments, and so on as I had commands that would simultaneously print and index these. It was the concepts that was difficult. I inserted `\index` commands as I went along at what seemed to be appropriate places but turned out not to be. I would use slightly different words for the same thing, and what was worse the same word for different things. It took a long time to improve it to its present rather pitiful state.

`\index[file]{stuff}`

The `\index` macro specifies that *stuff* is to appear in an index. By default the raw index data — the *stuff* and the page number — will be output to the `jobname.idx` file, but if the optional *file* argument is given then output will be to the `file.idx` file.

This book has two indexes. The main index uses the default indexing commands, while the second index does not. They are set up like this:

```
% in preamble
\makeindex
\makeindex[lines]
% in body
...\index{main} ... \index[lines]{First line} ...
...
% at the end
\clearpage
% main index
\pagestyle{Index}
\renewcommand{\preindexhook}{%
The first page number is usually, but not always,
the primary reference to the indexed topic.\vskip\onelineskip}
\printindex

% second index
\clearpage
\pagestyle{ruled}
\renewcommand{\preindexhook}{}
\renewcommand{\indexname}{Index of first lines}
```

```
\onecolindex
\printindex[lines]
```

```
\specialindex{<file>}{<counter>}{<stuff>}
```

The `\index` command uses the page number as the reference for the indexed item. In contrast the `\specialindex` command uses the `<counter>` as the reference for the indexed `<stuff>`. It writes `<stuff>` to the `file.idx` file, and also writes the page number (in parentheses) and the value of the `<counter>`. This means that indexing can be with respect to something other than page numbers. However, if the `hyperref` package is used the special index links will be to pages even though they will appear to be with respect to the `<counter>`; for example, if figure numbers are used as the index reference the `hyperref` link will be to the page where the figure caption appears and not to the figure itself.

```
\showindexmarks \hideindexmarks
\indexmarkstyle
```

The declaration `\showindexmarks` causes the argument to practically any `\index` and `\specialindex` to be printed in the margin of the page where the indexing command was issued. The argument is printed using the `\indexmarkstyle` which is initially specified as

```
\indexmarkstyle{\normalfont\footnotesize\ttfamily}
```

For reasons I don't fully understand, spaces in the argument are displayed as though it was typeset using the starred version of `\verb`. The `\hideindexmarks`, which is the default, turns off `\showindexmarks`.

The standard classes just provide the plain `\index` command with no optional `<file>` argument. In those classes the contents of the `jobname.idx` file is limited to the index entries actually seen in the document. In particular, if you are using `\include` for some parts of the document and `\includeonly` to exclude one or more files, then any `\index` entries in an excluded file will not appear in the `jobname.idx` file. The new implementation of indexing eliminates that potential problem.

```
\item \subitem \subsubitem
```

The `theindex` environment supports three levels of entries. A `\item` command flags a main entry; a subentry of a main entry is indicated by `\subitem` and a subentry of a subentry is flagged by `\subsubitem`. For example a portion of an index might look like:

```
\item bridge, 2,3,7           bridge, 2, 3, 7
\subitem railway, 24          railway, 24
\subsubitem Tay, 37           Tay, 37
```

if the Tay Bridge¹ was mentioned on page 37.

¹A railway (railroad) bridge in Scotland that collapsed in 1879 killing 90 people. The disaster lives for ever in the poem *The Tay Bridge Disaster* by William McGonagall (1830–?), the first verse of which goes:

Beautiful Railway Bridge of the Silv'ry Tay!
 Alas! I am very sorry to say
 That ninety lives have been taken away
 On the last Sabbath day of 1879,
 Which will be remember'd for a very long time.

p. v:	<code>\index{Alf}</code>	<code>\indexentry{Alf}{v}</code>
p. 1:	<code>\index{Alf}</code>	<code>\indexentry{Alf}{1}</code>
p. 2:	<code>\index{Alf}</code>	<code>\indexentry{Alf}{2}</code>
p. 3:	<code>\index{Alf}</code>	<code>\indexentry{Alf}{3}</code>
p. 5:	<code>\index{Alfabet see{Bet}}</code>	<code>\indexentry{Alfabet see{Bet}}{5}</code>
p. 7:	<code>\index{Alf@textit{Alf}}</code> <code>\index{Bet textbf}</code>	<code>\indexentry{Alf@textit{Alf}}{7}</code> <code>\indexentry{Bet textbf}{7}</code>
p. 8:	<code>\index{Alf!Bet!Con}</code>	<code>\indexentry{Alf!Bet!Con}{8}</code>
p. 9:	<code>\index{Alf!Dan}</code>	<code>\indexentry{Alf!Dan}{9}</code>

Figure 6.1: Raw indexing: (left) index commands in the source text; (right) idx file entries

6.2.3 MakeIndex

It is possible, but time consuming and error prone, to create your index by hand from the output of the `\index` commands you have scattered throughout the text. Most use the `MakeIndex` program to do this for them; there is also the `xindy` program [Keh98] but this is much less known.

`\xindyindex`

It turns out that `xindy` cannot handle a memoir hyperindex (which can be obtained with the aid of the `hyperref` package), although `MakeIndex` can do so.² If you are going to use `xindy` to process the raw index data put `\xindyindex` in the preamble, which will prevent hyperindexing.

`MakeIndex` reads an `idx` file containing the raw index data (which may include some commands to `MakeIndex` itself), sorts the data, and then outputs an `ind` file containing the sorted data, perhaps with some LaTeX commands to control the printing. `MakeIndex` was created as a general purpose index processing program and its operation can be controlled by a ‘makeindex configuration file’ (by default this is an `ist` file). Such a file consists of two parts. The first part specifies `MakeIndex` commands that can be included in the `<stuff>` argument to `\index`. The second part controls how the sorted index data is to be output.

I will only describe the most common elements of what you can put in an `ist` file; consult the `MakeIndex` manual [CH88], or the *Companion* [MG⁺04], for all the details.

You can embed commands, in the form of single characters, in the argument to `\index` that guide `MakeIndex` in converting the raw `idx` file into an `ind` file for final typesetting. The complete set of these is given in Table 6.1. They all have defaults and you can modify these via a `MakeIndex` configuration file.

In the simplest case you just use the name of the index item as the argument to the `\index` command. However, spaces are significant as far as `MakeIndex` is concerned. The following three uses of `\index` will result in three different entries in the final index

```
\index{ entry} \index{entry} \index{entry }
```

The ! character

The `level` specifier starts a new minor level, or subitem, with a maximum of two sub-levels. The default `level` specifier is the special character `!`. For example:

²This deficiency in `xindy` was discovered by Frederic Connes, who also provided the `\xindyindex` command.

Table 6.1: MakeIndex configuration file input parameters

Keyword	Default	Description
keyword (<i>s</i>)	"\\indexentry"	The argument to this command is a MakeIndex index entry
arg_open (<i>c</i>)	'{'	Argument start delimiter
arg_close (<i>c</i>)	'}'	Argument end delimiter
range_open (<i>c</i>)	'('	Start of an explicit page range
range_close (<i>c</i>)	')'	End of an explicit page range
level (<i>c</i>)	'!'	Character denoting a new subitem level
actual (<i>c</i>)	'@'	Character denoting that the following text is to appear in the actual index file
encap (<i>c</i>)	' '	Character denoting that the rest of the argument is to be used as an encapsulating command for the page number
quote (<i>c</i>)	'\"'	Character that escapes the following character
escape (<i>c</i>)	'\\'	Symbol with no special meaning unless followed by the quote character, when both characters will be printed. The quote and escape characters must be different.
page_compositor (<i>s</i>)	"-"	Composite number separator

(*s*) of type string, (*c*) of type character

<code>\begin{theindex}</code>	
<code>\item Alf, v, 1-3</code>	Alf, v, 1-3
<code> \subitem Bet</code>	Bet
<code> \subsubitem Con, 8</code>	Con, 8
<code> \subitem Dan, 9</code>	Dan, 9
<code>\item \textit{Alf}, 7</code>	<i>Alf</i> , 7
<code>\item Alfabet, \see{Bet}{5}</code>	Alfabet, <i>see</i> Bet
<code>\indexspace</code>	
<code>\item Bet, \textbf{7}</code>	Bet, 7
<code>\end{theindex}</code>	

Figure 6.2: Processed index: (left) alphabeticized ind file; (right) typeset index

```
\index{item!sub item!sub sub item}
```

The @ character

An indexable item may be represented in two portions, separated by the actual specifier, which by default is the @ character. The portion before the @ is used when MakeIndex sorts the raw index data, and the portion after the @ is used as the entry text. For example:

```
\index{MakeIndex@\textit{MakeIndex}}
```

will result in the final index entry of \textit{MakeIndex} in the alphabetic position accorded to MakeIndex. The same treatment can be applied for subitems:

```
\index{program!MakeIndex@\textit{MakeIndex}!commands}
```

The | character

Anything after the encap specifier, which by default is the | character, is treated as applying to the page number. In general

```
\index{...|str}
```

will produce a page number, say n, in the form

```
\str{n}
```

For example, if you wanted the page number of one particular entry to be in a bold font, say to indicate that this is where the entry is defined, you would do

```
\index{entry|textbf}
```

As a special case, if you want an index item to have a page range put the two characters | (at the end of the argument on the first page, and the character pair |) at the end of the argument on the last page. For example:

```
... \index{range| ( } pages about range \index{range| ) } ...
```

The two arguments must match exactly except for the final (and). You can also do

```
\index{...|(str}
```

which will produce a page range of the form

```
\str{n-m}
```

In this case, if the range is only a single page, the result is simply

```
\str{n}
```

<pre>\see{<text>}{<page>} seename \seealso{<text>}{<page>} alsoname</pre>

The macros \see and \seealso are specifically for use in an \index command after the |. The \see command replaces the page number by the phrase ‘see <text>’, while the \seealso command adds ‘see also <text>’ to the entry. For example, in the source for this manual I have

```
\index{chapter!style|see{chapterstyle}}
\index{figure|seealso{float}}
```


A `\see` or `\seealso` should be used once only for a particular entry. The ‘see’ texts for `\see` and `\seealso` are stored in `\seename` and `\alsoname`, whose default definitions are:

```
\newcommand*\seename{\see}
\newcommand*\alsoname{\see also}
```

The " and \ characters

If, for some reason, you want to index something that includes one of the `!`, `@`, `|` or `"` characters there is the difficulty of persuading MakeIndex to ignore the special meaning. This is solved by the quote specifier, which is normally the `"` character. The character immediately after `"` is treated as non-special. For example, if you needed to index the `@` and `!` characters:

```
\index{"@ (commercial at)}
\index{"! (exclamation)}
```

The leading `"` is stripped off before entries are alphabetized.

The escape specifier is used to strip the special meaning from the quote specifier. This is usually the `\` character. So, to index the double quote character itself:

```
\index{\ " (double quote)}
```

Example of using the special characters

Here is a short example of indexing the special characters. Given an input like this in the document

```
\index{exclamation mark (!)}
\index{vicious|see{circle}}
\index{atsign@\texttt{"@} sign|\textbf{}}
\index{quote!double ("")}{
\index{circle|see{vicious}}
```

then an index could eventually be produced that looks like:

```
@ sign, 30
circle, see vicious
exclamation mark (!), 21
quote
    double ("), 47
vicious, see circle
```

6.2.4 Controlling MakeIndex output

Table 6.2 lists the parameters that control MakeIndex’s output, except for the keywords that control the setting of page numbers. The special characters and strings are not fixed within the MakeIndex program. The program will read an `ist` file in which you can redefine all of MakeIndex’s defaults.

I have used a file called `memman.ist` for configuring MakeIndex for this manual. Here it is:

Table 6.2: MakeIndex configuration file output parameters

Keyword	Default	Description
<code>preamble (s)</code>	<code>"\\begin{theindex}\\n"</code>	Text for the start of the output file
<code>postamble (s)</code>	<code>"\\n\\n\\end{theindex}\\n"</code>	Text at the end of the output file
<code>group_skip (s)</code>	<code>"\\n\\n\\indexspace\\n"</code>	Vertical space before a new letter group
<code>heading_prefix (s)</code>	<code>" "</code>	Prefix for heading for a new letter group
<code>heading_suffix (s)</code>	<code>" "</code>	Suffix for heading for a new letter group
<code>headings_flag (n)</code>	<code>0</code>	A value = 0 inserts nothing between letter groups. A value > 0 includes an uppercase instance of the new symbol, while a value < 0 includes a lowercase instance, all within <code>heading_prefix</code> and <code>heading_suffix</code>
<code>item_0 (s)</code>	<code>"\\n\\item "</code>	Command inserted in front of a level 0 entry
<code>item_1 (s)</code>	<code>"\\n \\subitem "</code>	As above for a level 1 entry
<code>item_2 (s)</code>	<code>"\\n \\subsubitem "</code>	As above for a level 2 entry
<code>item_01 (s)</code>	<code>"\\n \\subitem "</code>	Command inserted in front of a level 1 entry starting at level 0
<code>item_12 (s)</code>	<code>"\\n \\subsubitem "</code>	Command inserted in front of a level 2 entry starting at level 1
<code>item_x1 (s)</code>	<code>"\\n \\subitem "</code>	Command inserted in front of a level 1 entry when the parent level has no page numbers
<code>item_x2 (s)</code>	<code>"\\n \\subitem "</code>	As above for a level 2 entry
<code>delim_0 (s)</code>	<code>", "</code>	Delimiter between level 0 entry and first page number
<code>delim_1 (s)</code>	<code>", "</code>	As above for level 1 entry
<code>delim_2 (s)</code>	<code>", "</code>	As above for level 2 entry
<code>delim_n (s)</code>	<code>", "</code>	Delimiter between page numbers
<code>delim_r (s)</code>	<code>"-"</code>	Designator for a page range
<code>encap_prefix (s)</code>	<code>"\\\""</code>	Prefix in front of a page encapsulator
<code>encap_infix (s)</code>	<code>"{"</code>	Infix for a page encapsulator
<code>encap_suffix (s)</code>	<code>"}"</code>	Suffix for a page encapsulator
<code>page_precedence (s)</code>	<code>"rnaRA"</code>	Page number precedence for sorting. <code>r</code> and <code>R</code> are lower- and uppercase roman; <code>a</code> and <code>A</code> are lower- and uppercase alphabetic; <code>n</code> is numeric
<code>line_max (n)</code>	<code>"72"</code>	Maximum length of an output line
<code>indent_space (s)</code>	<code>"\\t\\t"</code>	Indentation commands for wrapped lines
<code>indent_length (n)</code>	<code>"16"</code>	Indentation length for wrapped lines

(s) of type string, (n) of type number, "\\n" and "\\t" are newline and tab.

```
% MakeIndex style file memman.ist

% @ is a valid character in some entries, use ? instead
actual '?'

% output main entry <entry> as: \item \idxmark{<entry>},
item_0 "\n\item \idxmark{"
delim_0 "}, "
% not forgetting the subitem case
item_x1 "} \n \subitem "

% Wrap and uppercase head letters
headings_flag 1
heading_prefix "\doidxbookmark{"
heading_suffix "}"
```

Many items that I need to index include @ as part of their names, which is one of the special characters. The actual line says that the character ? performs the same function as the default @ (and by implication, @ is not a special character as far as MakeIndex is concerned).

The item_0 line says that a main entry in the generated index starts

```
\item \idxmark{
```

and the delim_0 and item_x1 lines say that the main entry ends

```
}, % or
}

\subitem
```

Thus, main entries will appear in an ind file like

```
\item \idxmark{a main entry}, <list of page numbers>
\item \idxmark{a main entry with no page numbers}
\subitem subitem, <list of page numbers>
```

Read the MakeIndex manual [CH88] for full details on how to get MakeIndex to do what you want.

The \doidxbookmark that is used to wrap around the letter group headers, can be used to not only style the group header, but can also be used to add the headers in the bookmarks list. This can be done using

```
\newcommand{\doidxbookmark}[1]{\def\@tempa{Symbols}\def\@tempb{#1}%
\centering\bfseries \ifx\@tempa\@tempb %
  Alphabetic
\phantomsection%
\pdfbookmark[0]{Alphabetic}{Alphabetic-idx}%
% \label{AlphabeticAlphabeticAlphabetic-idx}%
\else
#1%
\phantomsection%
\pdfbookmark[0]{#1}{#1-idx}%
```

```
% \label{#1#1#1-idx}%  
\fi%  
\vskip\onelineskip\par}}
```

The labels are generally not needed but can be used to add a visual representation of the index bookmarks into the index itself.

6.2.5 Indexing and the natbib package

The natbib package [Dal99a] will make an index of citations if `\citeindextrue` is put in the preamble after the natbib package is called for.

`\citeindexfile`

The name of the file for the citation index is stored in the macro `\citeindexfile`. This is initially defined as:

```
\newcommand{\citeindexfile}{\jobname}
```

That is, the citation entries will be written to the default `idx` file. This may be not what you want so you can change this, for example to:

```
\renewcommand{\citeindexfile}{names}
```

If you do change `\citeindexfile` then you have to put

```
\makeindex[\citeindexfile]
```

before

```
\usepackage[...]{natbib}
```

So, there are effectively two choices, either along the lines of

```
\renewcommand{\citeindexfile}{authors} % write to authors.idx  
\makeindex[\citeindexfile]  
\usepackage{natbib}  
\citeindextrue  
...  
\renewcommand{\indexname}{Index of citations}  
\printindex[\citeindexfile]
```

or along the lines of

```
\usepackage{natbib}  
\citeindextrue  
\makeindex  
...  
\printindex
```

6.3 GLOSSARIES

Unlike indexes, LaTeX provides less than minimal support for glossaries. It provides a `\makeglossary` command for initiating a glossary and a `\glossary` command which puts its argument, plus the page number, into a `glo` file, and that's it. memoir, combined with the MakeIndex program [CH88], enables you to generate and print a glossary in your document. The commands for creating a glossary are similar to those for indexes.

```
\makeglossary[⟨file⟩]
```

You have to put `\makeglossary` in your preamble if you want a glossary. This opens a file called by default `\jobname.glo`. If you use the optional `⟨file⟩` argument the file `file.glo` will be opened. A glossary `glo` file is analagous to an index `idx` file.

```
\printglossary[⟨file⟩]
```

To print a glossary call `\printglossary` which will print the glossary from file `\jobname.gls`, or from `file.gls` if the optional argument is used. A glossary `gls` file is analagous to an index `ind` file.

```
\glossary[⟨file⟩](⟨key⟩){⟨term⟩}{⟨desc⟩}
```

Use the `\glossary` command to add a `⟨term⟩` and its description, `⟨desc⟩`, to a glossary file. By default this will be `\jobname.glo` but if the optional `⟨file⟩` argument is given then the information will be written to `file.glo`. The `(⟨key⟩)` argument is optional. If present then `⟨key⟩` will be added to the file to act as a sort key for the `⟨term⟩`, otherwise `⟨term⟩` will be used as the sort key.

By using the optional `⟨file⟩` arguments you can have several glossaries, subject to TeX's limitations on the number of files that can be open at any one time.

A simple glossary entry might be:

```
\glossary{glossary}{A list of terms and their descriptions.}
```

The glossary facilities are designed so that the `MakelIndex` program can be used to convert the raw glossary data in a `glo` file into the printable glossary in a `gls` file.

```
\begin{theglossary} entry list \end{theglossary}
```

Glossary entries are typeset in a `theglossary` environment. It is assumed that a `gls` file will contain a complete `theglossary` environment, from `\begin{theglossary}` all the way through to `\end{theglossary}`.

```
\glossitem{⟨term⟩}{⟨desc⟩}{⟨ref⟩}{⟨num⟩}
```

A `\glossitem` is a glossary entry within a `theglossary` environment for a `⟨term⟩` with `⟨description⟩`. The `⟨num⟩` argument is the page or section where the corresponding `\glossary` was issued. The `⟨ref⟩` argument, if not empty, might be the section or page number corresponding to the `⟨num⟩` page or section number. The default definition is

```
\newcommand{\glossitem}[4]{#1 #2 #3 #4}
```

which is not very exciting. You may well prefer to use your own definition.

6.3.1 Controlling the glossary

Setting up makeindex

If you just run `MakelIndex` on a `glo` file you will get lots of errors; `MakelIndex` has to be configured to read a `glo` file and generate a useful `gls` file as by default it expects to read an index `idx` file and produce an index `ind` file. A configuration file like an index `ist` file will be needed. There is no recommended extension for such a file but I have come to favour `gst`. The command line for `MakelIndex` to create a sorted glossary from the raw data in a `glo` file, say `fred.glo`, using a configuration file called, say `basic.gst`, is

```
makeindex -s basic.gst -o fred.gls fred.glo
```

For other jobs just change the file names appropriately.

So, what is in a `gst` file? The potential contents were described earlier, but at a minimum you need this:

```
%%% basic.gst basic makindex glossary style file
%%% Output style parameters
preamble "\\begin{theglossary}"
postamble "\\end{theglossary}\\n"
item_0     "\\n\\glossitem"
delim_0    "\\memglonum{"
encap_suffix "}}"
headings_flag 1
heading_prefix "\\doglobookmark{"
heading_suffix "}"
%%% Input style parameters
keyword "\\glossaryentry"
```

The keyword line says that each entry in an input (`glo`) file will be of the form:

```
\glossaryentry{entry text}{number}
```

and by a miracle of coding, this is what memoir will put in a `glo` file for each `\glossary` command.

The preamble and postamble lines tell the program to start and end its output file with `\begin{theglossary}` and `\end{theglossary}`, respectively. The `item_0` tells the program to start each output entry with `\glossitem`. The `delim_0` says that `{\memglonum{` should be put between the end of the entry text and the (page) number. Finally `encap_suffix` requests `}}` to be put after any ‘encapsulated’ (page) number.

A complete listing of the possible entries in a configuration file, also called a style file, for MakeIndex is in Table 6.1 and 6.2 with the exception of the output file page number setting keywords.

The `\doglobookmark` macro can be used to add bookmarks for the letter groups. In the case of this manual we do not write anything, just add the letters to the glossary entry in the bookmark list. In memsty `\doglobookmark` is defined as

```
\newcommand\doglobookmark[1]{%
  \def\@tempa{Symbols}\def\@tempb{#1}%
  \ifx\@tempa\@tempb %
    \phantomsection\pdfbookmark[0]{Alphabetics}{Alphabetics-glo}%
  \else%
    \phantomsection\pdfbookmark[0]{#1}{#1-glo}%
  \fi%
}
```

MakeIndex uses the word ‘Symbols’ to specify the group that does not start with a letter.

Raw input data

<code>\@@wrglom@m{<file>}{<key>}{<term>}{<desc>}{<ref>}{<num>}</code>

The `\glossary` macro writes its arguments to the aux file in the form of arguments to the `\@@wrglom@m` internal macro. In turn this calls a series of other macros that eventually write the data to the `<file> glo` file in the format (where `@` is the actual flag):

```
\glossaryentry{key@{\memgloterm{term}} {\memglodesc{desc}}{\memgloref{ref}}
|memglonumf{num}}
```

which MakeIndex then effectively converts into

```
\glossitem{\memgloterm{term}}{\memglodesc{desc}}{\memgloref{ref}}
{\memglonum{\memglonumf{num}}}
```

```
\memgloterm{<term>}
\memglodesc{<desc>}
\memgloref{<ref>}
\memglonum{<num>}
```

These macros can be redefined to format the various parts of a glossary entry. Their default definitions are simply

```
\newcommand{\memgloterm}[1]{#1}
\newcommand{\memglodesc}[1]{#1}
\newcommand{\memgloref}[1]{#1}
\newcommand{\memglonum}[1]{#1}
```

For example, if you wanted the term in bold, the description in italics, and no numbers:

```
\renewcommand{\memgloterm}[1]{\textbf{#1}}
\renewcommand{\memglodesc}[1]{\textit{#1}}
\renewcommand{\memglonum}[1]{}
```

There are several macros that effect a glossary entry but which must not be directly modified (the `\memglonumf` shown above as part of the `\glossaryentry` is one of these). Each of the following `\changeGLOSS...` macros takes an optional *<file>* argument. The changes to the underlying macro apply only to the glossary of that particular *<file>* (or the `\jobname` file if the argument is not present).

```
\changeGLOSSactual[<file>]{<char>}
\changeGLOSSref[<file>]{<thecounter>}
\changeGLOSSnum[<file>]{<thecounter>}
\changeGLOSSnumformat[<file>]{<format>}
```

`\changeGLOSSactual` sets *<char>* as the actual character for the *<file>* glossary. It is initially `@`. This must match with the actual specified for the `gst` file that will be applied.

`\changeGLOSSref` specifies that *<thecounter>* should be used to generate the *<ref>* for the *<file>* glossary. It is initially nothing.

`\changeGLOSSnum` specifies that *<thecounter>* should be used to generate the *<num>* for the *<file>* glossary. It is initially `\thepage`.

`\changeGLOSSnumformat` specifies that *<format>* should be used to format the *<num>* for the *<file>* glossary. The format of *<format>* is `|form`, where `|` is the encap character specified in the `gst` file, and `form` is a formatting command, taking one argument (the number), without any backslash. For example

```
\changeGLOSSnumformat{|textbf}
```

to get bold numbers. It is initially set as `|memjustarg`, where this is defined as:

```
\newcommand{\memjustarg}[1]{#1}
```

There must be a format defined for the $\langle num \rangle$ otherwise the arguments to `\glossitem` will not be set correctly.

The `\makeglossary` command uses the `\change...` commands to define the initial versions, so only use the `\change...` macros *after* `\makeglossary`. In this document an early version of the glossary was set up by

```
\makeglossary
\changeGLOSSACTUAL{?}
\makeatletter
\changeGLOSSNUM{\@currentlabel}
\makeatother
\changeGLOSSNUM{\thepage}
```

The first call of `\changeGLOSSNUM` makes the number the current numbered chapter, or numbered section, or numbered I didn't like that when I tried it, so the second call resets the number to the page number.

The listing

The final glossary data in the `gls` file is typeset in the `theglossary` environment, which is much like the `theindex` and `thebibliography` environments.

The environment starts off with a chapter-style unnumbered title. There are several macros for specifying what happens after that.

```
\glossaryname
\glossarymark
\glossaryintoc \noglossaryintoc
```

The title for the glossary is `\glossaryname` whose initial definition is

```
\newcommand*{\glossaryname}{Glossary}
```

`\glossarymark`, which by default does nothing, can be redefined to set marks for headers. The glossary title will be added to the ToC if the `\glossaryintoc` declaration is in force, but will not be added to the ToC following the `\noglossaryintoc`.

```
\preglossaryhook
```

The macro `\preglossaryhook` is called after the glossary title has been typeset. By default it does nothing, but you could redefine it to, for example, add some explanatory material before the entries start.

```
\onecolglossary \twocolglossary
\glossarycolsep \glossaryrule
```

The glossary can be typeset in two columns (`\twocolglossary`) but by default (`\onecolglossary`) it is set in one column. When two columns are used, the length `\glossarycolsep` is the distance between the columns and the length `\glossaryrule` is the width (default 0) of a vertical rule between the columns.

```
\begintheglossaryhook
\atendtheglossaryhook
```

The last thing that `\begin{theglossary}` does is call `\begintheglossaryhook`. Similarly, the first thing that is done at the end of the environment is to call

`\atendtheglossaryhook`. By default these macros do nothing but you can redefine them.

For example, if you wanted the glossary in the form of a description list, the following will do that.

```
\renewcommand*\begintheglossaryhook{\begin{description}}
\renewcommand*\atendtheglossaryhook{\end{description}}
\renewcommand{\glossitem}[4]{\item[#1:] #2 #3 #4}
```

The glossary for this document

The following is the code I have used to produce the glossary in this document.

This is the code in the sty file that I used.

```
\makeglossary
\changeGLOSSACTUAL{?}
\changeGLOSSNUM{\thepage}
\changeGLOSSNUMFORMAT{||hyperpage}%% for hyperlinks
\renewcommand*\glossaryname{Command summary}
\renewcommand*\glossarymark{\markboth{\glossaryname}{\glossaryname}}
\renewcommand{\glossitem}[4]{%
  \sbox\@tempboxa{#1 \space #2 #3 \makebox[2em]{#4}}%
  \par\hangindent 2em
  \ifdim\wd\@tempboxa<0.8\linewidth
    #1 \space #2 #3 \dotfill \makebox[2em][r]{#4}\relax
  \else
    #1 \dotfill \makebox[2em][r]{#4}\\
    #2 #3
  \fi}
```

The redefinition of `\glossitem` works as follows (it is similar to code used in the setting of a `\caption`):

1. Put the whole entry into a temporary box.
2. Set up a hanging paragraph with 2em indentation after the first line.
3. Check if the length of the entry is less than 80% of the linewidth.
4. For a short entry set the name, description, and any reference then fill the remainder of the line with dots with the number at the right margin.
5. For a longer entry, set the title and number on a line, separated by a line of dots, then set the description (and reference) on the following lines.

The `gst` file I have used for this document has a few more items than the basic one.

```
%% memman.gst makindex glossary style file for memman and friends
%% Output style parameters
preamble "\\begin{theglossary}"
postamble "\\end{theglossary}\\n"
group_skip "\\n\\glossaryspace\\n"
item_0 "\\n\\glossitem"
delim_0 "{\\memglonum{"
```

```
encap_suffix "}}}"
indent_space "\t"
indent_length 2
%%% Input style parameters
keyword "\\glossaryentry"
actual '??'
page_compositor "."
```

The `group_skip` line asks that `\glossaryspace` be put between the last entry for one letter and the first for the next letter. The `indent_space` and `indent_length` give a smaller indent for continuation lines in the output than the default.

The `actual` entry says that the input file will use `?` instead of the default `@` as the flag for separating a key from the start of the real entry. The `page_compositor` indicates that any compound numbers will be like 1.2.3 instead of the default 1-2-3.

In the document the raw data is collected by the `\glossary` commands in the body of the text. For instance, although I have not actually used the first two:

```
\glossary(cs)%
  {\cs{cs}\marg{name}}%
  {Typesets \texttt{name} as a macro name with preceding backslash,
   e.g., \cs{name}.}%
\glossary(gmarg)%
  {\cs{gmarg}\marg{arg}}%
  {Typesets \texttt{arg} as a required argument, e.g., \marg{arg}.}
\glossary(glossaryname)%
  {\cs{glossaryname}}%
  {Name for a glossary}%
\glossary(memgloterm)%
  {\cs{memgloterm}\marg{term}}%
  {Wrapper round a glossary term.}%
```

Any change to the glossary entries will be reflected in the `glo` produced from that LaTeX run. `MakeIndex` has to be run the `glo` file using the appropriate `gst` configuration file, and then LaTeX run again to get the corrected, sorted and formatted result printed by `\printglossary`.

In particular, for this document, which also includes an index so that can be processed when the glossary is processed.

```
pdflatex memman
makeindex -s memman.gst -o memman.gls memman.glo
makeindex -s memman.ist memman      %%% for the index
makeindex lines                     %%% for the index of first lines
pdflatex memman
```

Seven

Miscellaneous

In which we talk of many things, but not shoes or ships or sealing wax, nor cabbages and kings.

This chapter started with the `\chapterprecis` command to add the above text, which is also added to the ToC.

The class provides a miscellany of minor facilities, which are described here.

7.1 DRAFT DOCUMENTS

The draft option marks any overfull lines with black rectangles, otherwise the appearance is the same as for a final document.

```
\ifdraftdoc
```

The `\ifdraftdoc` macro is provided so that you can add extra things that you might want to happen when processing a draft; for example you might want to have each page header (or footer) include the word ‘DRAFT’. The code to do this can be put into a construct like the following:

```
\ifdraftdoc
% special things for a draft document
\else
% perhaps special things for a non-draft document
\fi
```

7.2 CHANGE MARKS

When preparing a manuscript it normally goes through several iterations. The macros described in this section can be used to identify changes made to a document throughout its lifecycle.

The commands below implement a simplified version of the change marking in the iso class [Wil00b].

```
\changemarks \nochangemarks
```

The change marking macros only work properly when the draft class option is used. Additionally, the marks are only printed when the `\changemarks` declaration is in effect. Using `\nochangemarks` switches off any marking.

```
\added{<change-id>}
\deleted{<change-id>}
\changed{<change-id>}
```

Each of these macros puts a symbol and $\langle change-id \rangle$ into the margin near where the command is given. The `\added` macro indicates that something has been added to the manuscript and uses \oplus as its symbol. The `\deleted` macro is for indicating that something has been deleted and uses the \neq symbol. The macro `\changed` uses the \Leftrightarrow symbol to indicate that something has been changed.

These marking commands should be attached to some word or punctuation mark in the text otherwise extraneous spaces may creep into the typeset document.

7.3 TRIM MARKS

When the class `showtrims` option is used, trim marks can be placed on each page, usually to indicate where the stock should be trimmed to obtain the planned page size.

```
\showtrimsoff \showtrimson
```

If the `showtrims` class option has been used then the `\showtrimsoff` declaration switches off the trim marks; the `\showtrimson` declaration, which is the default, switches on the trim marks. These declarations do nothing if the `showtrims` option has not been used.

Caveat. Most modern \LaTeX editors make use of the `synctex` features in, say, \pdf\LaTeX to enable reverse search from the PDF previewer back to the source code in the editor. That is, click in a certain manner in the PDF previewer and you will be taken to the corresponding (or there about) line in the source code.

Apparently the `synctex` feature does not like the trimmarks the class provide, or the `showlocs` page style. The code line one is referred back to may be off by tens of lines.

Currently, there is no known workarounds for this deficiency.

Trim marks can be placed at each corner of the (trimmed) page, and also at the middle of each side of the page.

```
\trimXmarks
\trimLmarks
\trimFrame
\trimNone
\trimmarkscolor
```

Some predefined trimming styles are provided. After the declaration `\trimXmarks` marks in the shape of a cross are placed at the four corners of the page. The declaration `\trimLmarks` calls for corner marks in the shape of an 'L', in various orientations depending on the particular corner. After `\trimFrame` a frame will be drawn around each page, coinciding with the page boundaries. The declaration `\trimNone` disables all kinds of trim marking. All three plus `\quarkmarks` (described below) is visibly shown on Figure ??.

The macro `\trimmarkscolor` is despite its name a normal (empty) macro. By redefining it one can change the color of the trimmarks, handy for example if the document has a dark background color. To make them blue use

```
\newcommand*{\trimmarkscolor}{\color{blue}}
```

```
\trimmarks
\tmarktl \tmarktr \tmarkbr \tmarkbl
\tmarktm \tmarkmr \tmarkbm \tmarkml
```

The `\trimmarks` macro is responsible for displaying up to 8 marks. The marks are defined as zero-sized pictures which are placed strategically around the borders of the page.

The command `\trimmarks` places the pictures `\tmarktl`, `\tmarktr`, `\tmarkbl`, and `\tmarkbr` at the top left, top right, bottom right and bottom left corners of the page. The pictures `\tmarktm`, `\tmarkmr`, `\tmarkbm`, and `\tmarkml` are placed at the top middle, middle right, bottom middle and middle left of the edges of the page. All these `\mark.` macros should expand to zero-sized pictures.

`\trimmark`

The declaration `\trimXmarks` uses `\trimmark` for the corner crosses. This is defined as

```
\newcommand{\trimmark}{%
  \begin{picture}(0,0)
    \setlength{\unitlength}{1cm}\thicklines
    \put(-2,0){\line(1,0){4}}
    \put(0,-2){\line(0,1){4}}
  \end{picture}}
```

which produces a zero-sized picture of a 4cm cross. Then `\trimXmarks` is defined as:

```
\newcommand*{\trimXmarks}{%
  \let\tmarktl\trimmark
  \let\tmarktr\trimmark
  \let\tmarkbr\trimmark
  \let\tmarkbl\trimmark}
```

As an example, to draw short lines marking the half-height of the page, try this:

```
\renewcommand*{\tmarkml}{%
  \begin{picture}(0,0)%
    \unitlength 1mm
    \thinlines
    \put(-2,0){\line(-1,0){10}}
  \end{picture}}
\renewcommand*{\tmarkmr}{%
  \begin{picture}(0,0)%
    \unitlength 1mm
    \thinlines
    \put(2,0){\line(1,0){10}}
  \end{picture}}
```

Thin horizontal lines of length 10mm will be drawn at the middle left and middle right of the page, starting 2mm outside the page boundary. This is what we do (now) by default for all four middle parts.

`\quarkmarks`
`\registrationColour{<mark>}`

Following the declaration `\quarkmarks` and trim marks will be in the style of Quark Xpress registration marks.¹ The marks will be typeset using `\registrationColour`. The default definition is simply

¹The code for this was donated by William Adams.

```
\newcommand*{\RegistrationColour}[1]{#1}
```

but you can change that to, say, print the marks in particular color. See Figure ??.

7.4 SHEET NUMBERING

One purpose of trim marks is to show a printer where the stock should be trimmed. In this application it can be useful to also note the sheet number on each page, where the sheet number is 1 for the first page and increases by 1 for each page thereafter. The sheet number is independent of the page number.

<pre>sheetsequence \thesheetsequence</pre>
--

The macro `\thesheetsequence` typesets the current sheet sequence number and is analogous to the `\thepage` macro.

<pre>lastsheet lastpage</pre>

The counter `lastsheet` holds the number of sheets processed during the *previous* run of LaTeX. Similarly, the counter `lastpage` holds the number of the last page processed during the previous run. Note that the last page number is not necessarily the same as the last sheet number. For example:

In this document this is sheet 94 of 253 sheets, and page 76 of 235.

The previous sentence was the result of processing the following code

```
\textit{In this document this is  
sheet \thesheetsequence\ of \thelastsheet\ sheets,  
and page \thepage\ of \thelastpage.}
```

You may wish to use the sheet and/or page numbers as part of some trim marks. The following will note the sheet numbers above the page.

```
\newcommand*{\trimseqpage}{%  
  \begin{picture}(0,0)  
    \unitlength 1mm  
    \put(0,2){\makebox(0,0)[b]{Sheet: \thesheetsequence\ of \thelastsheet}}  
  \end{picture}}  
\let\tmarktm\trimseqpage
```

7.5 GATHERINGS OR SIGNATURES

Sometimes publishers request that they be supplied with a total number of pages that meet their planned *gatherings*.² For instance a gathering may consist of 8 leaves, and as there are two pages to a leaf this is equivalent to 16 pages. To meet this particular requirement there must be a total of $8N$ leaves, or equivalently $16N$ pages, where N will be the number of gatherings.

<pre>\leavespergathering{<num>}</pre>

²There was a thread on CTT, *pagenumber mod 4?* about this in 2008.

The command `\leavespergathering` ensures that there will be exactly the right number of pages output to make a complete set of gatherings of $\langle num \rangle$ leaves ($2\langle num \rangle$ pages) each — if necessary blank pages will be output at the end to make up the correct tally. If $\langle num \rangle$ is less than two (the default) then no additional pages will be output.

7.6 TIME

```
\printtime \printtime*
\hmpunct \amname \pmname
```

The `\printtime` command³ prints the time of day when the document is processed using the 24 hour clock while `\printtime*` uses a 12 hour clock. For example, the effect of the next piece of code is shown below.

This document was processed on: \today\ at \printtime\ (\printtime*).

This document was processed on: April 30, 2019 at 15:13 (3:13 pm).

The punctuation between the hours and minutes is `\hmpunct` which defaults to a colon (:). The macros `\amname` and `\pmname` hold the abbreviations for *ante meridiem* and *post meridiem*, respectively; the defaults are ‘am’ and ‘pm’.

According to the *Chicago Manual of Style* [Chi93] there should be no punctuation between the hours and minutes in the 24 hour system. For the 12 hour system it recommends that small caps be used for the divisions of the day (e.g., A.M. and P.M.) and also that the American practice is to use a colon as the separator between hours and minutes whereas the English practice is to use a period (known to the English as a ‘full stop’). I don’t know what the traditions are in other orthographies.

The `\quarkmarks` declaration uses `\printtime`, so be careful if you change it.

Nicola Talbot’s `datetime` package [Tal06] provides a much more comprehensive collection of styles for printing the time; also for dates.

7.7 PAGE BREAKS BEFORE LISTS

A sentence or two may be used to introduce a list (e.g., `itemize`) and it can be annoying if there is a page break between the introductory words and the first item.

```
\noprelistbreak
```

Putting `\noprelistbreak` immediately before the `\begin{itemize}` should prevent a page break. Ideally, there should be no blank lines between the end of the introduction and the start of the list.

7.8 CHANGING COUNTERS

This is effectively a bundling of the `chngcntr` package [Wil01e].

```
\newcounter{<ctr>}[<within>]
\thectr
```

³I based the code on a similar macro in *TeX for the Impatient* [AHK90].

In LaTeX a new counter called, say `ctr`, is created by the `\newcounter` command as `\newcounter{ctr}`. If the optional `<within>` argument is given, the counter `ctr` is reset to zero each time the counter called `within` is changed; the `within` counter must exist before it is used as the optional argument. The command `\thectr` typesets the value of the counter `ctr`. This is automatically defined for you by the `\newcounter` command to typeset arabic numerals.

```
\counterwithin{<ctr>}{<within>}
\counterwithin*{<ctr>}{<within>}
```

The `\counterwithin` macro makes a `<ctr>` that has been initially defined by `\newcounter{ctr}` act as though it had been defined by `\newcounter{ctr}[within]`. It also redefines the `\thectr` command to be `\thewithin.\arabic{ctr}`. The starred version of the command does nothing to the original definition of `\thectr`.

```
\counterwithout{<ctr>}{<within>}
\counterwithout*{<ctr>}{<within>}
```

The `\counterwithout` macro makes the `ctr` counter that has been initially defined by `\newcounter{ctr}[within]` act as though it had been defined by `\newcounter{ctr}`. It also redefines the `\thectr` command to be `\arabic{ctr}`. The starred version of the command does nothing to the original definition of `\thectr`.

Any number of `\counterwithin{ctr}{...}` and `\counterwithout{ctr}{...}` commands can be issued for a given counter `ctr` if you wish to toggle between the two styles. The current value of `ctr` is unaffected by these commands. If you want to change the value use `\setcounter`, and to change the typesetting style use `\renewcommand` on `\thectr`.

Caveat. As of 2018 `\counterwithout` and `\counterwithin` will be provided by the \LaTeX -kernel, thus we only provide it if it does not already exist.

```
\letcountercounter{<counterA>}{<counterB>}
\unletcounter{<counterA>}
```

At times it is handy to ‘let’ one counter act as if it was a different counter. Say you have two constructions, each with their own counter `A` and `B`, now you want them to cooperate, counting in unison. This can be done using the `\letcountercounter`.

`\letcountercounter{<counterA>}{<counterB>}` \lets (make the same) `<counterA>` to `<counterB>`. The original of `<counterA>` is kept, such that you can unlet it later.

`\unletcounter{<counterA>}` restores `<counterA>` to its un\let condition.

This feature can be quite handy. Say for instance you want figures and tables to counter within the same counter (say `table`), then we need each change to the `figure` counter to actually act on the `table` counter. `\letcountercounter{figure}{table}` solves the problem.

7.9 NEW NEW AND PROVIDE COMMANDS

```
\newloglike{<cmd>}{<string>}
\newloglike*{<cmd>}{<string>}
```

The class provides means of creating new math log-like functions. For example you might want to do


```
\newloglike{\Ei}{Ei}
```

if you are using the exponential integral function in your work. The starred version of the command creates a function that takes limits (like the `\max` function).

The LaTeX kernel defines the `\providecommand` macro that acts like `\newcommand` if the designated macro has not been defined previously, otherwise it does nothing. The class adds to that limited repertoire.

```
\provideenvironment{<name>}[<numargs>][<optarg>]{<begindef>}{<enddef>}
\providelength{<cmd>}
\providecounter{<ctr>}[<within>]
\provideloglike{<cmd>}{<string>}
\provideloglike*{<cmd>}{<string>}
```

The macros `\provideenvironment`, `\providelength` and `\providecounter` take the same arguments as their `\new...` counterparts. If the environment, length or counter has not been defined then it is defined accordingly, otherwise the macros do nothing except produce a warning message for information purposes.

The `\provideloglike` commands are for math log-like functions, but they do not produce any warning messages.

7.10 CHANGING MACROS

Commands are provided for extending simple macro definitions.

```
\addtodef{<macro>}{<prepend>}{<append>}
\addtoiargdef{<macro>}{<prepend>}{<append>}
```

The macro `\addtodef` inserts `<prepend>` at the start of the current definition of `<macro>` and puts `<append>` at the end, where `<macro>` is the name of a macro (including the backslash) which takes no arguments. The `\addtoiargdef` macro is similar except that `<macro>` is the name of a macro that takes one argument.

For example the following are two equivalent definitions of `\mymacro`:

```
\newcommand{\mymacro}[1]{#1 is a violinist in spite of being tone deaf}
```

and

```
\newcommand{\mymacro}[1]{#1 is a violinist}
\addtoiargdef{\mymacro}{}{ in spite of being tone deaf}
```

The `\addtoiargdef` (and `\addtodef`) commands can be applied several times to the same `<macro>`. Revising the previous example we could have

```
\newcommand{\mymacro}[1]{#1 is a violinist}
\addtoiargdef{\mymacro}{Although somewhat elderly, }%
{ in spite of being tone deaf}
\addtoiargdef{\mymacro}{}{ and a bagpiper}
```

which is equivalent to

```
\newcommand{\mymacro}[1]{%
  Although somewhat elderly, #1 is a violinist
  in spite of being tone deaf and a bagpiper}
```

The `<prepend>` and `<append>` arguments may include LaTeX code, as shown in this extract from the class code:

```
\addtoargdef{\date}{}{%
  \begingroup
    \renewcommand{\thanks}[1]{%
      \renewcommand{\thanksmark}[1]{%
        \renewcommand{\thanksgap}[1]{%
          \protected@xdef\thedata{#1}
        }
      }
    }
  \endgroup}
```

Note that in the case of `\addtoargdef` an argument can also refer to the original argument of the `<macro>`.

```
\addtodef*{<macro>}{<prepend>}{<append>}
\addtoiargdef*{<macro>}{<prepend>}{<append>}
```

These starred versions are for use when the original `<macro>` was defined via `\newcommand*`. Using the starred versions is like using `\renewcommand*` and the unstarred versions are like having used `\renewcommand`. It is the version (starred or unstarred) of a sequence of `\addto...` commands that counts when determining whether the equivalent `\renew...` is treated as starred or unstarred.

The `\addto...` macros cannot be used to delete any code from `<macro>` nor to add anything except at the start and end. Also, in general, they cannot be used to change the definition of a macro that takes an optional argument, or a starred macro.

```
\patchcommand{<macro>}{<start-code>}{<end-code>}
```

The `\patchcommand` is from the late Michael Downes' `patchcmd` package [Dow00]. It inserts the `<start-code>` at the start of the current definition of the macro `<macro>`, and inserts `<end-code>` at the end of its current definition. The `<macro>` can have zero to nine parameters. If `<macro>` uses `\futurelet` (e.g., it is a starred command or takes an optional argument) only `<start-code>` is useful — `<end-code>` must be empty otherwise things get messed up. If `<macro>` has any delimited arguments then `\patchcommand` cannot be used.

7.11 STRING ARGUMENTS

In the code for the class I have sometimes used macro arguments that consist of a 'string', like the `*` arguments in the page layout macros (e.g., `\settypeblocksize`), or the `\flushleft`, `\center` and `\flushright` strings for the `\makeheadposition` macro.

```
\nametest{<str1>}{<str2>}
\ifsamename
```

The macro `\nametest` takes two strings as the arguments `<str1>` and `<str2>`. It sets `\ifsamename` true if `<str1>` is the same as `<str2>`, otherwise it sets `\ifsamename` false. For the purposes of `\nametest`, a string is a sequence of characters which may include spaces and may include the `\` backslash character; strings are equal if and only if their character sequences are identical.

Typically, I have used it within macros for checking on argument values. For example:

```

\newcommand{\amacro}[1]{%
  \nametest{#1}{green}
  \ifsamename
%    code for green
  \fi
  \nametest{#1}{red}
  \ifsamename
%    code for red
  \fi
  ...
}

```

7.12 ODD/EVEN PAGE CHECKING

It is difficult to check robustly if the current page is odd or even but the class does provide a robust method based on writing out a label and then checking the page reference for the label. This requires at least two LaTeX runs to stabilise. This has been extracted from the original chngpage package (which is no longer available). (The class code and chngpage code is similar but not identical. There is a later package, changepage [Wil08a] which contains code that is identical to the class.)

```

\checkoddpage
\ifoddpage
\strictpagecheck \easypagecheck

```

The macro `\checkoddpage` sets `\ifoddpage` to true if the current page number is odd, otherwise it sets it to false (the page number is even). The robust checking method involves writing and reading labels, which is what is done after the command `\strictpagecheck` is issued; it may take more than one run before everything settles down. The simple method is just to check the current page number which, because of TeX's asynchronous page breaking algorithm, may not correspond to the actual page number where the `\checkoddpage` command was issued. The simple, but faster, page checking method is used after the `\easypagecheck` command is issued.

```

\cplabel

```

When strict page checking is used the labels consist of a number preceded by the value of `\cplabel`, whose default definition is `^_` (e.g., a label may consist of the characters `^_21`). If this might clash with any of your labels, change `\cplabel` with `\renewcommand`, but the definition of `\cplabel` must be constant for any given document.

7.13 MOVING TO ANOTHER PAGE

Standard LaTeX provides the `\newpage`, `\clearpage` and `\cleardoublepage` commands for discontinuing the current page and starting a new one. The following is a bundling of the nextpage package [Wil00c].

```

\needspace{<length>}

```

This macro decides if there is $\langle length \rangle$ space at the bottom of the current page. If there is, it does nothing, otherwise it starts a new page. This is useful if $\langle length \rangle$ amount of material is to be kept together on one page. The `\needspace` macro depends on penalties for deciding what to do which means that the reserved space is an approximation. However, except for the odd occasion, the macro gives adequate results.

```
\Needspace{\langle length \rangle}
\Needspace*{\langle length \rangle}
```

Like `\needspace`, the `\Needspace` macro checks if there is $\langle length \rangle$ space at the bottom of the current page and if there is not it starts a new page. The command should only be used between paragraphs; indeed, the first thing it does is to call `\par`. The `\Needspace` command checks for the actual space left on the page and is more exacting than `\needspace`.

If either `\needspace` or `\Needspace` produce a short page it will be ragged bottom even if `\flushbottom` is in effect. With the starred `\Needspace*` version, short pages will be flush bottom if `\flushbottom` is in effect and will be ragged bottom when `\raggedbottom` is in effect.

Generally speaking, use `\needspace` in preference to `\Needspace` unless it gives a bad break or the pages must be flush bottom.

```
\movetoevenpage[\langle text \rangle]
\cleartoevenpage[\langle text \rangle]
```

The `\movetoevenpage` stops the current page and starts typesetting on the next even numbered page. The `\clear...` version flushes out all floats before going to the next even page. The optional $\langle text \rangle$ is put on the skipped page (if there is one).

```
\movetooddpage[\langle text \rangle]
\cleartooddpage[\langle text \rangle]
```

These macros are similar to the `\...evenpage` ones except that they jump to the next odd numbered page.

A likely example for the optional $\langle text \rangle$ argument is

```
\cleartooddpage[\vspace*{\fill}THIS PAGE LEFT BLANK\vspace*{\fill}]
```

which will put ‘THIS PAGE LEFT BLANK’ in the centre of any potential skipped (empty) even numbered page.

```
\cleartorecto \cleartoverso
```

These are slightly simpler forms⁴ of `\cleartooddpage` and `\cleartoevenpage`. For example, if you wanted the ToC to start on a verso page, like in *The TeXbook* [Knu84], then do this:

```
\cleartoverso
\tableofcontents
```

7.14 NUMBER FORMATTING

Several methods are provided for formatting numbers. Two classes of number representations are catered for. A ‘numeric number’ is typeset using arabic digits and a ‘named number’ is typeset using words.

⁴Perhaps more robust.

The argument to the number formatting macros is a ‘number’, essentially something that resolves to a series of arabic digits. Typical arguments might be:

- Some digits, e.g., `\ordinal{123}` -> 123rd
- A macro expanding to digits, e.g., `\def\temp{3}\ordinal{\temp}` -> 3rd
- The value of a counter, e.g., `\ordinal{\value{page}}` -> 83rd
- The arabic representation of a counter, e.g., `\ordinal{\thepage}` -> 83rd

However, if the representation of a counter is not completely in arabic digits, such as `\thesection` which here prints as 7.14, it will produce odd errors or peculiar results if it is used as the argument. For instance:

`\ordinal{\thesection}` -> .147th

7.14.1 Numeric numbers

```
\cardinal{<number>}
\fcardinal{<number>}
\fnumbersep
```

The macro `\fcardinal` prints its `<number>` argument formatted using `\fnumbersep` between each triple of digits. The default definition of `\fnumbersep` is:

```
\newcommand{\fnumbersep}{,}
```

Here are some examples:

```
\fcardinal{12} -> 12
\fcardinal{1234} -> 1,234
\fcardinal{1234567} -> 1,234,567
\renewcommand*{\fnumbersep}{\:}\fcardinal{12345678} -> 12 345 678
\renewcommand*{\fnumbersep}{, \:}
```

The `\cardinal` macro is like `\fcardinal` except that there is no separation between any of the digits.

```
\ordinal{<number>}
\fordinal{<number>}
\ordscript{<chars>}
```

The `\fordinal` macro typesets its `<number>` argument as a formatted ordinal, using `\fnumbersep` as the separator. The macro `\ordinal` is similar except that there is no separation between any of the digits.

Some examples are:

```
\fordinal{3} -> 3rd
\fordinal{12341} -> 12,341st
\renewcommand{\ordscript}[1]{\textsuperscript{#1}}
\fordinal{1234567} -> 1,234,567th
\ordinal{1234567} -> 1234567th
```

This is the `\ordinal{\value{chapter}}` chapter. -> This is the 7th chapter.

The characters denoting the ordinal (ordination?) are typeset as the argument of `\ordscript`, whose default definition is:

```
\newcommand{\ordscript}[1]{#1}
```

As the above examples show, this can be changed to give a different appearance.

Typeset example 7.1: TeX's minimum number in words (English style)

The minimum number in TeX is minus two billion, one hundred and forty-seven million, four hundred and eighty-three thousand, six hundred and forty-seven (i.e., -2, 147, 483, 647)

```
\nthstring \iststring \iindstring \iiirdstring
```

The ordinal characters are the values of the macros `\nthstring` (default: th) for most ordinals, `\iststring` (default: st) for ordinals ending in 1 like 21st, `\iindstring` (default: nd) for ordinals like 22nd, and `\iiirdstring` (default: rd) for ordinals like 23rd.

7.14.2 Named numbers

```
\numtoname{<number>}  
\numtoName{<number>}  
\NumToName{<number>}
```

The macro `\numtoname` typesets its `<number>` argument using lowercase words. The other two macros are similar, except that `\numtoName` uses uppercase for the initial letter of the first word and `\NumToName` uses uppercase for the initial letters of all the words.

As examples:

`\numtoname{12345}` -> twelve thousand, three hundred and forty-five

`\numtoName{12345}` -> Twelve thousand, three hundred and forty-five

`\NumToName{12345}` -> Twelve Thousand, Three Hundred and Forty-Five

Source for example 7.1

The minimum number in TeX is `\numtoname{-2147483647}`
(i.e., `\fcardinal{-2147483647}`)

```
\ordinaltoname{<number>}  
\ordinaltoName{<number>}  
\OrdinalToName{<number>}
```

These three macros are similar to `\numtoname`, etc., except that they typeset the argument as a wordy ordinal.

For example:

This is the `\ordinaltoname{\value{chapter}}` chapter. -> This is the seventh chapter.

```
\namenumberand \namenumbercomma \tensunitsep
```

By default some punctuation and conjunctions are used in the representation of named numbers. According to both American and English practice, a hyphen should be inserted between a 'tens' name (e.g., fifty) and a following unit name (e.g., two). This is represented

Typeset example 7.2: TeX's maximum number in words (American style)

The maximum number in TeX is two billion one hundred forty-seven million four hundred eighty-three thousand six hundred forty-seven (i.e., 2147483647).

by the value of `\tensunitsep`. English practice, but not American, is to include commas (the value of `\namenumbercomma`) and conjunctions (the value of `\namenumberand`) in strategic positions in the typesetting. These macros are initially defined as:

```
\newcommand*\{\namenumberand}{ and }
\newcommand*\{\namenumbercomma}{, }
\newcommand*\{\tensunitsep}{-}
```

The next example shows how to achieve American typesetting.

Source for example 7.2

```
\renewcommand*\{\namenumberand}{ }
\renewcommand*\{\namenumbercomma}{ }
The maximum number in TeX is \numtoname{2147483647}
(i.e., \cardinal{2147483647}).
```

```
\minusname \lcminusname \ucminusname
```

When a named number is negative, `\minusname` is put before the spelled out number. The definitions of the above three commands are:

```
\newcommand*\{\lcminusname}{minus }
\newcommand*\{\ucminusname}{Minus }
\let\minusname\lcminusname
```

which means that 'minus' is normally all lowercase. To get 'minus' typeset with an initial uppercase letter simply:

```
\let\minusname\ucminusname
```

Only one version of `\namenumberand` is supplied as I consider that it is unlikely that 'and' would ever be typeset as 'And'. If the initial uppercase is required, renew the macro when appropriate.

There is a group of macros that hold the names for the numbers. To typeset named numbers in a language other than English these will have to be changed as appropriate. You can find them in the class and patch code. The actual picking and ordering of the names is done by an internal macro called `\n@me@number`. If the ordering is not appropriate for a particular language, that is the macro to peruse and modify. Be prepared, though, for the default definitions to be changed in a later release in case there is a more efficient way of implementing their functions.

Typeset example 7.3: Varieties of fractions in text

In summary, fractions can be typeset like $3/4$ or $\frac{3}{4}$ or $\frac{3}{4}$ or $\frac{3}{4}$ because several choices are provided.

If you want to express numbers that fall outside TeX's range, Edward Reingold has produced a set of macros that can write out in words any number within the range -10^{66} to 10^{66} , that is, up to a thousand vigintillion [Rei07].

7.14.3 Fractions

When typesetting a simple fraction in text there is usually a choice of two styles, like $3/4$ or $\frac{3}{4}$, which do not necessarily look as though they fit in with their surroundings. These fractions were typeset via:

... like $3/4$ or $\frac{3}{4}$...

$\frac{\text{\texttt{\textbackslash slashfrac}\{<top>\}\{<bottom>\}}{\text{\texttt{\textbackslash slashfracstyle}\{<num>\}}}$

The class provides the `\slashfrac` command which typesets its arguments like $\frac{3}{4}$. Unlike the `\frac` command which can only be used in math mode, the `\slashfrac` command can be used in text and math modes.

The `\slashfrac` macro calls the `\slashfracstyle` command to reduce the size of the numbers in the fraction. You can also use `\slashfracstyle` by itself.

Source for example 7.3

In summary, fractions can be typeset like $3/4$ or $\frac{3}{4}$ or $\frac{3}{4}$ or $\frac{3}{4}$ because several choices are provided.

$\text{\texttt{\textbackslash textsuperscript}\{<super>\}}$ $\text{\texttt{\textbackslash textsubscript}\{<sub>\}}$
--

While on the subject of moving numbers up and down, the kernel provides the `\textsuperscript` macro for typesetting its argument as though it is a superscript. The class also provides the `\textsubscript` macro for typesetting its argument like a subscript.

Source for example 7.4

In normal text you can typeset superscripts like H^+ and subscripts like H_2O without going into math mode.

Typeset example 7.4: Super- and subscripts in text

In normal text you can typeset superscripts like H^+ and subscripts like H_2O without going into math mode.

7.15 AN ARRAY DATA STRUCTURE

The class includes some macros for supporting the `patverse` environment which may be more generally useful.

```
\newarray{<arrayname>}{<low>}{<high>}
```

`\newarray` defines the `<arrayname>` array, where `<arrayname>` is a name like `MyArray`. The lowest and highest array indices are set to `<low>` and `<high>` respectively, where both are integer numbers.

```
\setarrayelement{<arrayname>}{<index>}{<text>}
\getarrayelement{<arrayname>}{<index>}{<result>}
```

The `\setarrayelement` macro sets the `<index>` location in the `<arrayname>` array to be `<text>`. Conversely, `\getarrayelement` sets the parameterless macro `<result>` to the contents of the `<index>` location in the `<arrayname>` array. For example:

```
\setarrayelement{MyArray}{23}{2~{23}$}
\getarrayelement{MyArray}{23}{\result}
```

will result in `\result` being defined as `\def\result{2~{23}$}`.

```
\checkarrayindex{<arrayname>}{<index>}
\ifbounderror
```

`\checkarrayindex` checks if `<arrayname>` is an array and if `<index>` is a valid index for the array. If both conditions hold then `\ifbounderror` is set false, but if either `<arrayname>` is not an array or, if it is, `<index>` is out of range then `\ifbounderror` is set true.

```
\stringtoarray{<arrayname>}{<string>}
\arraytostring{<arrayname>}{<result>}
```

The macro `\stringtoarray` puts each character from `<string>` sequentially into the `<arrayname>` array, starting at index 1. The macro `\arraytostring` assumes that `<arrayname>` is an array of characters, and defines the macro `<result>` to be that sequence of characters. For example:

```
\stringtoarray{MyArray}{Chars}
\arraytostring{MyArray}{\MyString}
```

is equivalent to

```
\def\MyString{Chars}
```

```
\checkifinteger{<num>}
\ifinteger
```

The command `\checkifinteger` checks if $\langle num \rangle$ is an integer (not less than zero). If it is then `\ifinteger` is set `true`, otherwise it is set `false`.

Note. Please note that `\checkifinteger` may only work on simple input.

7.16 CHECKING THE PROCESSOR

7.16.1 Checking for pdfLaTeX

Both LaTeX and pdfLaTeX can be run on the same document. LaTeX produces a `.dvi` file as output, while pdfLaTeX can produce either a `.dvi` or a `.pdf` file. On modern systems pdfLaTeX produces a pdf file by default.

If you want a dvi file output use LaTeX and if you want a pdf file use pdfLaTeX.

```
\ifpdf ... \fi
```

The class provides `\ifpdf` (by autoloading the `ifpdf` package) which is `true` when the document is being processed by pdfLaTeX and `false` otherwise. You can use it like this:

```
\ifpdf
  code for pdfLaTeX only
\else
  code for LaTeX only
\fi
```

If there is no LaTeX specific code then don't write the `\else` part.

7.16.2 Checking for etex

Modern LaTeX distributions use `etex`, which is an extended version of TeX, as the underlying engine. `etex` provides some more primitives than TeX, which may be useful, but not everybody has `etex` available (Though, as of 2018, this is *very* rare).

```
\ifetex
```

`\ifetex` can be used to determine if `etex` is being used as the underlying engine; it is analogous to `\ifpdf` which tests for pdfLaTeX (provided by autoloading the `ifetex` package). For example:

```
\ifetex
  %% code only processible by etex
\else
  \typeout{etex is not available}
\fi
```

7.16.3 Checking for XeTeX

You have been able to use `\ifpdf` to check if pdfLaTeX is being used to process the document.

```
\ifxetex
```

In a similar manner you can use `\ifxetex` to check if the document is being processed by XeTeX (provided by autoloading the `ifxetex` package).

```
\RequireXeTeX
```

The `ifxetex` package also provides `\RequireXeTeX`, which generates an error if XeTeX is not being used to process the document. This can be useful if you make your own class building upon `memoir`.

7.16.4 Checking for LuaTeX

Similarly you can use

```
\ifluatex
```

to check if the doc is being process by LuaTeX.

7.17 LEADING

LaTeX automatically uses different leading for different font sizes.

```
\baselineskip \onelineskip
```

At any point in a document the standard LaTeX `\baselineskip` length contains the current value of the leading⁵. The class provides the length `\onelineskip` which contains the initial leading for the normal font. This value is useful if you are wanting to specify length values in terms of numbers of lines of text.

7.18 MINOR SPACE ADJUSTMENT

The kernel provides the `\,` macro for inserting a thin space in both text and math mode. There are other space adjustment commands, such as `\!` for negative thin space, and `\;` and `\;` for medium and thick spaces, which can only be used in math mode.

```
\thinspace \medspace \: \!
```

On occasions I have found it useful to be able to tweak spaces in text by some fixed amount, just as in math mode. The kernel macro `\thinspace` specifies a thin space, which is $3/18$ em. The class `\medspace` specifies a medium space of $4/18$ em. As mentioned, the kernel macro `\:` inserts a medium space in math mode. The class version can be used in both math and text mode to insert a medium space. Similarly, the class version of `\!` can be used to insert a negative thin space in both text and math mode.

The math thick space is $5/18$ em. To specify this amount of space in text mode you can combine spacing commands as:

```
\:\:\!
```

which will result in an overall space of $5/18$ em (from $(4 + 4 - 3)/18$).

7.19 ADDING A PERIOD

Much earlier, when showing the code for the sectional division styles for this document, I used the macro `\addperiod`.

```
\addperiod{<text>}
```

This puts a period (a full stop) at the end of `<text>`. I used it to add a period at the end of the `\paragraph` and `\subparagraph` titles. When sectional titles, like `\paragraph` are run-in, it is customary to end them with a period (or occasionally a colon).

⁵This statement ignores any attempts to stretch the baseline.

Table 7.1: Defined words and phrases

Macro	Default	Usage
<code>\abstractname</code>	Abstract	title for <code>abstract</code> environment
<code>\alsoname</code>	see also	used by <code>\seealso</code>
<code>\amname</code>	am	used in printing time of day
<code>\appendixname</code>	Appendix	name for an appendix heading
<code>\appendixpagename</code>	Appendices	name for an <code>\appendixpage</code>
<code>\appendixtocname</code>	Appendices	ToC entry announcing appendices
<code>\bibname</code>	Bibliography	title for <code>\thebibliography</code>
<code>\bookname</code>	Book	name for <code>\book</code> heading
<code>\bookrefname</code>	Book	used by <code>\Bref</code>
<code>\chaptername</code>	Chapter	name for <code>\chapter</code> heading
<code>\chapterrefname</code>	Chapter	used by <code>\Cref</code>
<code>\contentsname</code>	Contents	title for <code>\tableofcontents</code>
<code>\figurename</code>	Figure	name for figure <code>\caption</code>
<code>\figurerefname</code>	Figure	used by <code>\fref</code>
<code>\glossaryname</code>	Glossary	title for <code>\theglossary</code>
<code>\indexname</code>	Index	title for <code>\theindex</code>
<code>\lcminusname</code>	minus	used in named number formatting
<code>\listfigurename</code>	List of Figures	title for <code>\listoffigures</code>
<code>\listtablename</code>	List of Tables	title for <code>\listoftables</code>
<code>\minusname</code>	minus	used in named number formatting
<code>\namenumberand</code>	and	used in named number formatting
<code>\namenumbercomma</code>	,	used in named number formatting
<code>\notesname</code>	Notes	title of <code>\notedivision</code>
<code>\pagename</code>	page	for your use
<code>\pagerefname</code>	page	used by <code>\pref</code>
<code>\partname</code>	Part	name for <code>\part</code> heading
<code>\partrefname</code>	Part	used by <code>\Pref</code>
<code>\pmname</code>	pm	used in printing time of day
<code>\sectionrefname</code>	§	used by <code>\Sref</code>
<code>\seename</code>	see	used by <code>\see</code>
<code>\tablename</code>	Table	name for table <code>\caption</code>
<code>\tablerefname</code>	Table	used by <code>\tref</code>
<code>\ucminusname</code>	Minus	used in named number formatting

7.20 WORDS AND PHRASES

The class provides several macros that expand into English words or phrases. To typeset in another language these need to be changed, or an author or publisher may want some changes made to the English versions. Table 7.1 lists the macros, their default values, and where they used.

Most, if not all, of the tabulated definitions are simple — for example

```
\newcommand*{\partname}{Part}
```

```
\newcommand*{\partrefname}{Part~}
```

and so can be also changed simply.

The definitions of the macros for the names of numbers are more complex — for example for the number 11 (eleven)

```
\newcommand*{\nNamexi}{\iflowertonumname e\else E\fi eleven}
```

That is, each definition includes both a lowercase and an uppercase initial letter, so a bit more care has to be taken when changing these. For specifics read the documentation of the class code.

7.21 SYMBOLS

LaTeX lets you typeset an enormous variety of symbols. The class adds nothing to the standard LaTeX capabilities in this respect. If you want to see what symbols are available then get a copy of Scott Pakin's *The Comprehensive LaTeX Symbol List* [Pak01]. You may have to do a little experimentation to get what you want, though.

For example, the `\texttrademark` command produces the trademark symbol™, but the `\textregistered` command produces®. When I wanted to use the registered trademark symbol it needed to be typeset like a superscript instead of on the baseline. The `\textsuperscript` macro typesets its argument like a superscript, so using

```
\textsuperscript{\textregistered}
```

gave the required result®.

7.22 TWO SIMPLE MACROS

There are two trivial macros that can be generally useful.

```
\memjustarg{<text>}
\mengobble{<text>}
```

The `\memjustarg` macro just uses its argument and is defined as:

```
\newcommand*{\memjustarg}[1]{#1}
```

The `\mengobble` macro gobbles down and swallows its argument. Its definition is:

```
\newcommand{\mengobble}[1]{}
```

Do *not* redefine either `\memjustarg` or `\mengobble`; if you do various pieces of code will behave in unexpected ways that you will not like.

7.23 VERTICAL CENTERING

Earlier there was a description of a method for centering text vertically. The `vplace` environment provides a simpler and more general way.

```
\begin{vplace}[<num>] text \end{vplace}
```

The contents of the `vplace` environment are vertically centered. The optional `<num>` argument can be used to specify the ratio of the upper space to the lower space. You can put other text on the page above or below the centered text. The environment may be useful for title pages.

7.24 FOR PACKAGE WRITERS

The facilities described in this section are for anyone to use but I suspect that they may be most useful to package developers.

7.24.1 Emulating packages

```
\EmulatedPackage{<package>}[<date>]  
\EmulatedPackageWithOptions{<optionlist>}{<package>}[<date>]
```

These commands are for package writers; they are based on a conversation with Donald Arseneau on CTT. They fool LaTeX into thinking that the *<package>* has already been loaded so it won't try loading it again. These are probably only useful if your package includes the actual code for *<package>*.

memoir does include code from several packages and uses a similar internal command to ensure that the packages are not loaded following some later `\usepackage` command. The names of the emulated packages are written to the log file. At the time of writing the emulated packages are: abstract, appendix, array, booktabs, ccaption, chngcntr, crop, dcolumn, delarray, enumerate, epigraph, ifmtarg, ifpdf, index, makeidx, moreverb, needspace, newfile, nextpage, pagenote, patchcmd, parskip, setspace, shortvrb, showidx, tabularx, tltreref, tocibind, tocloft, verbatim, and verse. As well as the emulated packages memoir provides functions equivalent to those in the following packages, although the class does not prevent you from using them: fancyhdr, framed, geometry, sidecap, subfigure, and titlesec.

```
\DisemulatePackage{<package>}
```

This command undoes any prior `\EmulatedPackage` or `\EmulatedPackageWithOptions` for the *<package>* package. For example, if you wish to use the index package instead of memoir's emulation then put

```
\DisemulatePackage{index}  
\usepackage{index}
```

in your preamble.

7.24.2 Inserting code before and after a file, package or class

The kernel provides two commands, `\AtBeginDocument` and `\AtEndDocument` which can only be used in the preamble, for inserting code at the start and end of the document environment.

The kernel also provides the macros `\AtEndOfPackage{<code>}` and `\AtEndOfClass{<code>}` for inserting code at the end of the current package or class. More precisely, these macros call the *<code>* after the package or class file has been input via `\InputIfFileExists`.

The class provides a more comprehensive set of macros for code insertions, which should be used before the relevant file is called for.

```
\AtBeginFile{<file>}{<code>}  
\AtEndFile{<file>}{<code>}
```

The `\AtBeginFile` macro inserts *<code>* just before the *<file>* file is `\input` (or `\included`, etc.). Similarly `\AtEndFile` inserts the *<code>* immediately after the *<file>*. The *<file>* argument must be the same as used in the corresponding `\input` command. If *<file>* includes

an extension, for example `fred.def`, then that is taken as the complete name, otherwise if there is no extension, for instance `fred`, then the `.tex` extension is automatically appended making the full name `fred.tex`.

The `\At...File` commands must be issued *before* the corresponding *<file>* is input otherwise nothing will happen.

```
\AtBeginPackage{<pack>}{<code>}
\AtEndPackage{<pack>}{<code>}
\RequireAtEndPackage{<pack>}{<code>}
```

The `\AtBeginPackage` command will insert *<code>* just before the *<pack>* package is used. Similarly `\AtEndPackage` will insert the *<code>* immediately after the *<pack>*. The *<pack>* argument must be the same as used in the corresponding `\usepackage` command, that is, without any extension. The `\At...Package` commands must be issued *before* the corresponding *<pack>* is used otherwise nothing will happen.

The `\RequireAtEndPackage` command will, like `\AtEndPackage`, insert *<code>* at the end of the *<pack>* package if it has not yet been used. If the package has already been used then the *<code>* is called immediately.

```
\AtBeginClass{<class>}{<code>}
\AtEndClass{<class>}{<code>}
\RequireAtEndClass{<class>}{<code>}
```

The `\AtBeginClass` command will insert *<code>* just before the *<class>* class is used. Similarly `\AtEndClass` will insert the *<code>* immediately after the *<class>*. The *<class>* argument must be the same as used in the corresponding `\LoadClass` command, that is, without any extension. The `\At...Class` commands must be issued *before* the corresponding *<class>* is used otherwise nothing will happen.

The `\RequireAtEndClass` command will, like `\AtEndClass`, insert *<code>* at the end of the *<class>* class if it has not yet been used. If the class has already been used then the *<code>* is called immediately.

There is an unfortunate interaction between the kernel's `\AtEndOfPackage` and the class's `\AtEndPackage`, and similarly for the `\AtEndOfClass` and `\AtEndClass`. I discovered this when I tried to automate using the `memhfixc` package if `hyperref` was being used by putting the following into the memoir code

```
\AtEndPackage{hyperref}{\usepackage{memhfixc}}
```

which caused all sorts of problems.

The kernel scheme looks like this:

```
\newcommand{\usepackage}[1]{%
...
\InputIfFileExists{#1}
<AtEndOfPackage code>}
```

The basic mechanism for implementing the class macros is by modifying the kernel's `\InputIfFileExists` macro, which internally uses a form of `\input` to read in the file, so that the inserted *<code>* comes immediately before and after the `\input`, somewhat like:

```
\renewcommand{\InputIfFileExists}[1]{%
```

```
...
<before code> \input{#1} <after code>}
```

If `\AtEndPackage` is applied to a package that has an internal `\AtEndOfPackage` then the result can be sketched as:

```
\newcommand{\usepackage}[1]{%
...
<before code>
\input{#1}
<after code>
<AtEndOfPackage code>
}
```

In other words the body of the package is read in, the `\AtEndPackage` code is called, and then *after* that the `\AtEndOfPackage` code is called.

The `hyperref` package internally uses `\AtEndOfPackage` to read some files and `memhfixc` had to be input after these. A way to automate `memhfixc` after `hyperref` is:

```
\AtEndPackage{hyperref}{%
\AtBeginDocument{\usepackage{memhfixc}}}
```

but this seems more trouble than it's worth especially since Heiko Oberdiek has kindly updated `hyperref` so that versions after 2006/11/15 will automatically load the `memhfixc` package.

7.25 HEADING HOOKS

On 2nd September 2005 I posted two messages to the `comp.text.tex` newsgroup saying that I was creating a new version of `memoir` and that I would consider inserting hooks into the class code that package writers might find useful. I got no requests for any hooks or anything else from package writers. I therefore assume that no package author sees any problems if a `memoir` class document author uses the package.

However, I have provided macros that may be useful for those who want to do things with the contents of section headings, captions, and the like. The macros are called within the relevant heading or caption code, and by default are defined to do nothing.

Hooks for the `\book` and `\book*` commands.

<pre>\membookinfo{<thebook>}{<fortoc>}{<title>} \membookstarinfo{<title>}</pre>

Hooks for the `\part` and `\part*` commands.

<pre>\mempartinfo{<thepart>}{<fortoc>}{<title>} \mempartstarinfo{<title>}</pre>

In many cases a `\mem...info` macro includes an argument related to the heading's number (`<thepart>` for `\mempartinfo`). In certain circumstances, such as a `\chapter` in the `\frontmatter`, there might not be a number even though the normal unstarred version of the command is used. In these cases the number argument (`<thechapter>` in the case of `\memchapinfo`) is left empty.

Hooks for the `\chapter` and `\chapter*` commands. Note that regular chapters and those as appendices are treated differently.


```
\memchapinfo{<thechapter>}{<fortoc>}{<forhead>}{<title>}
\memchapstarinfo{<fortoc>}{<title>}
\memappchapinfo{<thechapter>}{<fortoc>}{<forhead>}{<title>}
\memappchapstarinfo{<fortoc>}{<title>}
```

Hooks for \section, \subsection, etc., and their starred versions. <name> is the type of section (e.g., section, or subsection, or subsubsection or ...

```
\memsecinfo{<name>}{<thename>}{<fortoc>}{<forhead>}{<title>}
\memsecstarinfo{<name>}{<title>}
```

Hooks for appendix-like page headings.

```
\memapppageinfo{<title>}
\memapppagestarinfo{<title>}
\memleadpageinfo{<pstyle>}{<cmdname>}{<title>}
\memleadpagestarinfo{<pstyle>}{<cmdname>}{<title>}
```

Hooks for \poemtitle, \PoemTitle, and their starred versions.

```
\mempoeminfo{<title>}
\mempoemstarinfo{<title>}
\memPoemTitleinfo{<thepoem>}{<fortoc>}{<forhead>}{<title>}
\memPoemTitlestarinfo{<fortoc>}{<title>}
```

Hooks for the several kinds of \caption and \legend commands.

```
\memcaptioninfo{<type>}{<thetype>}{<fortoc>}{<title>}
\memlegendinfo{<title>}
\memnamedlegendinfo{<fortoc>}{<title>}
\membitwonumcaptioninfo{<type>}{<thetype>}{<fortoc1>}{<title1>}
{<name2>}{<fortoc2>}{<title2>}
\membionenumcaptioninfo{<type>}{<thetype>}{<fortoc1>}{<title1>}
{<name2>}{<fortoc2>}{<title2>}
\membicaptioninfo{<type>}{<thetype>}{<fortoc1>}{<title1>}{<name2>}{<title2>}
```

As an example of how one of these macros might be used, just before the start of this section I put

```
\renewcommand{\memsecinfo}[5]{\edef\Margi{#1}\edef\Margii{#2}%
\edef\Margiii{#3}\edef\Margiv{#4}%
\edef\Margv{#5}}
```

and now I'm putting

```
The arguments are: (1) '\Margi', (2) '\Margii', (3) '\Margiii',
(4) '\Margiv', (5) '\Margv'.
```

The arguments are: (1) 'section', (2) '7.25', (3) 'Heading hooks', (4) 'Heading hooks', (5) 'Heading hooks'.

Warning: Be very careful with the fifth argument of this macro when using hyperref, this argument will then contain a hyper link anchor, which may cause problems when used out of context.

7.26 DOCUMENTING LATEX COMMANDS

The class provides a few macros to help you if you want to describe LaTeX commands.

`\bs \cs{<name>} \cmdprint{<cmd>} \cmd{<cmd>}`

The macro `\bs` simply prints the ‘\’ backslash.

The macro `\cs` prints its argument, putting a backslash in front of it. For example `\cs{name}` prints `\name`.

The argument to `\cmdprint` should be the name of a macro, including the backslash. It is then printed as is. For instance `\cmdprint{\amacro}` prints `\amacro`.

The argument to `\cmd` should be the name of a macro, including the backslash. It is then printed, using `\cmdprint`, and also added to the index file with the assumption that `?` will be used as the ‘actual’ character (the default is `@` which is not of much use if you are trying to index macro names that have `@` as part of their names).

`\meta{<arg>} \marg{<arg>} \oarg{<arg>} \parg{<arg>}`

The macro `\meta{<arg>}` prints `<arg>` for an argument to a macro.

The macro `\marg{<arg>}` prints `{<arg>}` for a required argument.

The macro `\oarg{<arg>}` prints `[<arg>]` for an optional argument.

The macro `\parg{<arg>}` prints `(<arg>)` for a parenthesized argument.

Eight

For package users

Many packages work just as well with memoir as with any of the standard classes. In some instances, though, there may be a problem. In other instances the class and package have been designed to work together but you have to be careful about any code that you might write.

8.1 CLASS/PACKAGE NAME CLASH

A typical indication of a problem is an error message saying that a command has already been defined (see page [211](#)).

When the class and a package both use the same name for a macro that they define something has to give. For the sake of an example, assume that memoir has defined a macro called `\foo` and a package `pack` that is used in the document also defines `\foo`. There are several options that you can choose which might resolve the difficulty.

1. Just keep the class definition:

```
\documentclass{...}
\let\classfoo\foo % save the class' definition
\let\foo\relax    % 'undefine' \foo
\usepackage{pack} % defines \foo
\let\foo\classfoo % restore \foo to the class definition
...
\foo...           % the class \foo
```

2. Just keep the package definition:

```
\documentclass{...}
\let\foo\relax    % 'undefine' \foo
\usepackage{pack} % defines \foo
...
\foo...           % the package \foo
```

3. Keep the class definition and rename the package definition:

```
\documentclass{...}
\let\classfoo\foo % save the class' definition
\let\foo\relax    % 'undefine' \foo
\usepackage{pack} % defines \foo
\let\packfoo\foo  % rename pack's \foo
```

```
\let\foo\classfoo % restore \foo to the class definition
...
\foo...           % the class \foo
\packfoo          % the package \foo
```

4. Keep the package definition and rename the class definition:

```
\documentclass{...}
\let\classfoo\foo % save the class' definition
\let\foo\relax    % 'undefine' \foo
\usepackage{pack} % defines \foo
...
\foo...           % the package \foo
\classfoo...      % the class \foo
```

A potential problem with these options can occur after the package is loaded and there are class or package commands that you use that, knowingly or not, call `\foo` expecting to get the class or the package definition, one of which is now not available (except under a different name).

The memoir class has been available since 2001. It seems likely that if older packages clash with memoir then, as eight years have gone by, the authors are unlikely to do anything about it. If a newer package clashes then contact the author of the package.

If all else fails, ask for help on the CTT newsgroup.

8.2 SUPPORT FOR BIDIDIRECTIONAL TYPESETTING

The bidi system [Kha10] provides means of bidirectional typesetting. The class has built in support for bidi but this means that if you are defining your own macros there are some things you need to be aware of.

When dealing with bidirectional texts the left-to-right (LTR) direction is the familiar one and LaTeX is set up for this. When typesetting right-to-left (RTL) bidi interchanges left and right. The support in memoir consists of replacing many, but not all, of the right- and left-specific constructs. The replacement macros are:

<pre>\memRTLleftskip \memRTLrightskip \memRTLvleftskip \memRTLvrightskip \memRTLraggedright \memRTLraggedleft</pre>

In certain places, but not everywhere:

```
\memRTLleftskip is used instead of \leftskip
\memRTLrightskip is used instead of \rightskip
\memRTLvleftskip is used instead of \vleftskip
\memRTLvrightskip is used instead of \vrightskip
\memRTLraggedleft is used instead of \raggedleft
\memRTLraggedright is used instead of \raggedright
```

So, if you are defining any macros that use the `\...skip` or `\ragged...` macros you may have to use the `\memRTL...` version instead. The memoir definitions of these macros are simply:

```
\newcommand*{\memRTLleftskip}{\leftskip}  
\newcommand*{\memRTLrightskip}{\rightskip}  
...  
\newcommand*{\memRTLraggedleft}{\raggedleft}
```

The bidi system redefines them to suit its purposes. If your work will only be set LTR there is no need to use the `\memRTL...` macros in any of your code. If you might ever be producing a bidirectional document then you may have to use the `\memRTL...` versions of the standard commands in your code. To determine where you might have to use them you will have to consult the bidi documentation as not every use of the standard commands needs to be replaced.

9.1 INTRODUCTION

In this chapter I will work through a reasonably complete design exercise. Rather than trying to invent something myself I am taking the design of Bringhurst's *The Elements of Typographic Style* [Bri99] as the basis of the exercise. This is sufficiently different from the normal LaTeX appearance to demonstrate most of the class capabilities, and also it is a design by a leading proponent of good typography.

As much as possible, this chapter is typeset according to the results of the exercise to provide both a coding and a graphic example.

9.2 DESIGN REQUIREMENTS

The *Elements of Typographic Style* is typeset using Minion as the text font and Syntax (a sans font) for the captions.

The trimmed page size is 23 by 13.3cm. The fore-edge is 3.1cm and the top margin is 1.9cm.

As already noted, the font for the main text is Minion, with 12pt leading on a 21pc measure with 42 lines per page. For the purposes of this exercise I will assume that Minion can be replaced by the font used for this manual. The captions to figures and tables are unnamed and unnumbered and typeset in Syntax. The captions give the appearance of being in a smaller font size than the main text, which is often the case. I'll assume that the `\small\sffseries` font will reasonably do for the captions.

The footer is the same width as the typeblock and the folio is placed in the footer at the fore-edge. There are two blank lines between the bottom of the typeblock and the folio.

There is no header in the usually accepted sense of the term but the chapter title is put on recto pages and section titles are on verso pages. The running titles are placed in the fore-edge margin level with the seventh line of the text in the typeblock. The recto headers are typeset raggedright and the verso ones raggedleft.

Bringhurst also uses many marginal notes, their maximum width being about 51pt, and typeset raggedright in a smaller version of the textfont.

Chapter titles are in small caps, lowercase, in a larger font than for the main text, and a rule is placed between the title and the typeblock. The total vertical space used by a chapter title is three text lines. Chapters are not numbered in the text but are in the ToC.

Section titles are again in small caps, lowercase, in the same size as the text font. The titles are numbered, with both the chapter and section number.

A subsection title, which is the lowest subdivision in the book, is in the italic form of the textfont and is typeset as a numbered non-indented paragraph. These are usually multiline as Bringhurst sometimes uses them like an enumerated list, so on occasion there is a subsection title with no following text.

Only chapter titles are put into the ToC, and these are set raggedright with the page numbers immediately after the titles. There is no LoF or LoT.

Note that unlike the normal LaTeX use of fonts, essentially only three sizes of fonts are used — the textfont size, one a bit larger for the chapter titles, and one a bit smaller for marginal notes and captions. Also, bold fonts are not used except on special occasions,

such as when he is comparing font families and uses large bold versions to make the differences easier to see.

9.3 SPECIFYING THE PAGE AND TYPEBLOCK

The first and second things to do are to specify the sizes of the page after trimming and the typeblock. The trimmed size is easy as we have the dimensions.

*Specifying
the page and
typeblock*

```
\settrimmedsize{23cm}{13.3cm}{*}
```

We want 42 lines of text, so that's what we set as the height of the typeblock; however, we have to remember to ask for lines as the optional *(algorithm)* argument when we finally call `\checkandfixthelayout`.

```
\settypeblocksize{42\onelineskip}{21pc}{*}
```

To make life easier, we'll do no trimming of the top of the stock

```
\setlength{\trimtop}{0pt}
```

but will trim the fore-edge. The next set of calculations first sets the value of the `\trimedge` to be the `\stockwidth`; subtracting the trimmed `\paperwidth` then results in `\trimedge` being the amount to trim off the fore-edge.

```
\setlength{\trimedge}{\stockwidth}
```

```
\addtolength{\trimedge}{-\paperwidth}
```

The sizes of the trimmed page and the typeblock have now been specified. The typeblock is now positioned on the page. The sideways positioning is easy as we know the fore-edge margin to be 3.1cm.

```
\setlrmargins{*}{3.1cm}{*}
```

The top margin is specified as 1.9cm, which is very close to four and a half lines of text. Just in case someone might want to use a different font size, I'll specify the top margin so that it is dependent on the font size. The `\footskip` can be specified now as well (it doesn't particularly matter what we do about the header-related lengths as there isn't anything above the typeblock).

```
\setulmargins{4.5\onelineskip}{*}{*}
```

```
\setheadfoot{\onelineskip}{3\onelineskip}
```

```
\setheaderspaces{\onelineskip}{*}{*}
```

Lastly define the dimensions for any marginal notes.

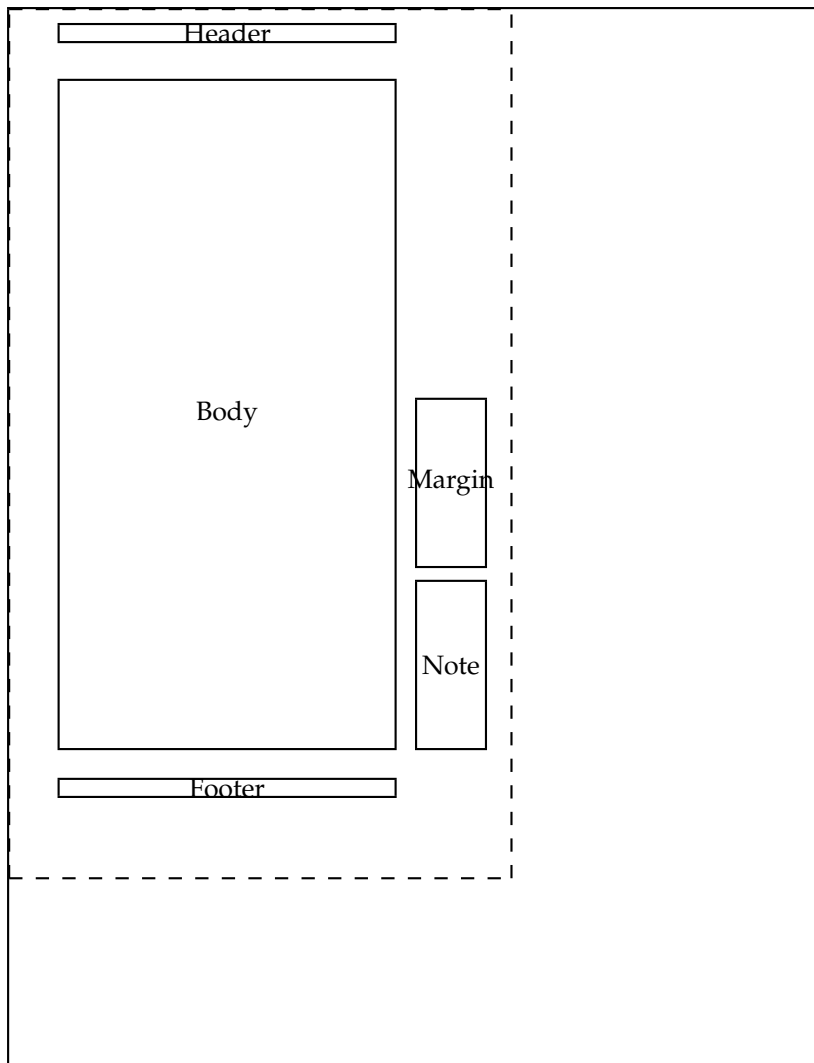
```
\setmarginnotes{17pt}{51pt}{\onelineskip}
```

If this was for real, the page layout would have to be checked and implemented.

```
\checkandfixthelayout[lines]
```

It is possible to implement this layout just for this chapter but I'm not going to tell you either how to do it, or demonstrate it. Except under exceptional circumstances it is not good to make such drastic changes to the page layout in the middle of a document. However, the picture on page 103 illustrates how this layout would look on US letterpaper stock. Looking at the illustration suggests that the layout would look rather odd unless the stock was trimmed down to the page size — another reason for not switching the layout here.

Dashed lines represent the actual page size after trimming the stock.



*An example
book design*

Lengths are to the nearest pt.

<code>\stockheight = 795pt</code>	<code>\stockwidth = 614pt</code>
<code>\pageheight = 654pt</code>	<code>\pagewidth = 378pt</code>
<code>\textheight = 502pt</code>	<code>\textwidth = 252pt</code>
<code>\trimtop = 0pt</code>	<code>\trimedge = 236pt</code>
<code>\uppermargin = 54pt</code>	<code>\spinemargin = 38pt</code>
<code>\headheight = 12pt</code>	<code>\headsep = 30pt</code>
<code>\footskip = 36pt</code>	<code>\marginparsep = 17pt</code>
<code>\marginparpush = 12pt</code>	<code>\columnsep = 10pt</code>
<code>\columnseprule = 0.0pt</code>	

An illustration of Bringhurst's page layout style when printed on US letter paper stock. Also shown are the values used for the page layout parameters for this design.

9.4 SPECIFYING THE SECTIONAL TITLING STYLES

9.4.1 *The chapter style*

Recapping, chapter titles are in small caps, lowercase, in a larger font than for the main text, and a rule is placed between the title and the typeblock. The total vertical space used by a chapter title is three text lines. Chapters are not numbered in the text but are in the ToC. Titles in the ToC are in mixed case.

The definition of the `chapterstyle` is remarkably simple, as shown below.

```
%% Bringham chapter style
\makechapterstyle{bringham}{%
  \renewcommand{\chapterheadstart}{}
  \renewcommand{\printchaptername}{}
  \renewcommand{\chaptername}{}
  \renewcommand{\printchapternum}{}
  \renewcommand{\afterchapternum}{}
  \renewcommand{\printchaptertitle}[1]{%
    \raggedright\Large\scshape\MakeLowercase{##1}}
  \renewcommand{\afterchaptertitle}{}
  \vskip\onelineskip \hrule\vskip\onelineskip
}
```

Most of the specification consists of nulling the majority of the normal LaTeX specification, and modifying just two elements.

The chapter title (via `\printchaptertitle`) is typeset `raggedright` using the `\Large` smallcaps fonts. The `\MakeLowercase` macro is used to ensure that the entire title is lowercase before typesetting it. Titles are input in mixed case.

After the title is typeset the `\afterchaptertitle` macro specifies that one line is skipped, a horizontal rule is drawn and then another line is skipped.

9.4.2 *Lower level divisions*

Section titles are in small caps, lowercase, in the same size as the text font. The titles are numbered, with both the chapter and section number.

The specification is:

```
\setseheadstyle{\raggedright\scshape\MakeLowercase}
\setbeforesecskip{-\onelineskip}
\setaftersecskip{\onelineskip}
```

The macro `\setseheadstyle` lowercases the title and typesets it small caps.

The default skips before and after titles are rubber lengths but this does not bode well if we are trying to line something up with a particular line of text — the presence of section titles may make slight vertical adjustments to the text lines because of the flexible spacing. So, we have to try and have fixed spacings. A single blank line is used before (`\setbeforesecskip`) and after (`\setaftersecskip`) the title text.

A subsection title, which is the lowest subdivision in the book, is in the italic form of the textfont and is typeset as a numbered non-indented paragraph. The code for this is below.

*Specifying
the sectional
titling styles*

```

\setsubseheadstyle{\sethangfrom{\noindent ##1}\raggedright\itshape}
\setbeforesubsecskip{-\onelineskip}
\setaftersubsecskip{\onelineskip}

```

As in the redefinition of the `\section` style, there are fixed spaces before and after the title text. The title is typeset (`\setsubseheadstyle`) raggedright in a normal sized italic font. The macro `\sethangfrom` is used to to redefine the internal `\@hangfrom` macro so that the title and number are typeset as a block paragraph instead of the default hanging paragraph style. Note the use of the double `##` mark for denoting the position of the argument to `\@hangfrom`.

*An example
book design*

9.5 SPECIFYING THE PAGESTYLE

The pagestyle is perhaps the most interesting aspect of the exercise. Instead of the chapter and section titles being put at the top of the pages they are put in the margin starting about seven lines below the top of the typeblock. The folios are put at the bottom of the page aligned with the outside of the typeblock.

As the folios are easy, we'll deal with those first.

```

%% Bringhurst page style
\makepagestyle{bringhurst}
\makeevenfoot{bringhurst}{\thepage}{}{}
\makeoddfoot{bringhurst}{}{}{\thepage}

```

Putting text at a fixed point on a page is typically done by first putting the text into a zero width picture (which as far as LaTeX is concerned takes up zero space) and then placing the picture at the required point on the page. This can be done by hanging it from the header.

We might as well treat the titles so that they will align with any marginal notes, which are `\marginparsep` (17pt) into the margin and `\marginparwidth` (51pt) wide. Earlier in the manual I defined two lengths called `\pwayi` and `\pwayii` which are no longer used. I will use these as scratch lengths in performing some of the necessary calculations.

For the recto page headers the picture will be the *right* part of the header and for the verso pages the picture will be the *left* part of the header, all other parts being empty.

For the picture on the *right* the text must be 17pt to the right of the origin, and some distance below the origin. From some experiments, this distance turns out to be the `\headsep` plus the `\topskip` plus 7.3 lines, which is calculated as follows:

```

\setlength{\pwayi}{\headsep}
\addtolength{\pwayi}{\topskip}
\addtolength{\pwayi}{7.3\onelineskip}

```

There is a nifty internal LaTeX macro called `\strip@pt` which you probably haven't heard about, and I have only recently come across. What it does is strip the 'pt' from a following length, reducing it to a plain real number. Remembering that the default `\unitlength` is 1pt we can do the following, while making sure that the current `\unitlength` is 1pt:

```

\makeatletter
\newcommand{\bringpicr}[1]{%
  \setlength{\unitlength}{1pt}

```

Specifying
the
pagestyle

```

\begin{picture}(0,0)
  \put(\strip@pt\marginparsep, -\strip@pt\pwayi){%
    \begin{minipage}[t]{\marginparwidth}
      \raggedright\itshape #1
    \end{minipage}}
\end{picture}
}
\makeatother

```

The new macro `\bringpicr{<text>}` puts `<text>` into a minipage of width `\marginparwidth`, typeset raggedright in an italic font, and puts the top left of the minipage at the position `(\marginparsep, -\pwayi)` in a zero width picture.

We need a different picture for the *<left>* as the text needs to be typeset raggedleft with the right of the text 17pt from the left of the typeblock. I will use the length `\pwayii` to calculate the sum of `\marginparsep` and `\marginparwidth`. Hence:

```

\makeatletter
\setlength{\pwayii}{\marginparsep}
\addtolength{\pwayii}{\marginparwidth}
\newcommand{\bringpicl}[1]{%
  \setlength{\unitlength}{1pt}
  \begin{picture}(0,0)
    \put(-\strip@pt\pwayii, -\strip@pt\pwayi){%
      \begin{minipage}[t]{\marginparwidth}
        \raggedleft\itshape #1
      \end{minipage}}
    \end{picture}
  }
\makeatother

```

The new macro `\bringpicl{<text>}` puts `<text>` into a minipage of width `\marginparwidth`, typeset raggedleft in an italic font, and puts the top left of the minipage at the position `(-\marginparsep + \marginparwidth, -\pwayi)` in a zero width picture.

Now we can proceed with the remainder of the pagestyle specification. The next bit puts the chapter and section titles into the `\...mark` macros.

```

\makeatletter
\makepsmarks{bringhurst}{%
  \def\chaptermark##1{\markboth{##1}{##1}}
  \def\sectionmark##1{\markright{##1}}
}
\makeatother

```

Finally, specify the evenhead using `\bringpicl` with the section title as its argument, and the oddhead using `\bringpicr` with the chapter title as its argument.

```

\makeevenhead{bringhurst}{\bringpicl{\rightmark}}{}{}
\makeoddhead{bringhurst}{}{}{\bringpicr{\leftmark}}

```

9.6 CAPTIONS AND THE TOC

The captions to figures and tables are set in a small sans font and are neither named nor numbered, and there is no LoF or LoT. Setting the caption titles in the desired font is simple:

```
\captiontitlefont{\small\sffamily}
```

There are two options regarding table and figure captioning: either use the `\legend` command (which produces an anonymous unnumbered title) instead of the `\caption` command, or use the `\caption` command with a modified definition. Just in case the design might change at a later date to required numbered captions, it's probably best to use a modified version of `\caption`. In this case this is simple, just give the `\caption` command the same definition as the `\legend` command.

*An example
book design*

```
\let\caption\legend
```

An aside: I initially used the default caption style (block paragraph) for the diagram on page 103, but this looked unbalanced so now it has the last line centered. As a float environment, like any other environment, forms a group, you can make local changes within the float. I actually did it like this:

```
\begin{figure}
\captiontitlefont{\small\sffamily}
\captionstyle{\centerlastline}
...
\legend{...} \label{...}
\end{figure}
```

For fine typesetting you may wish to change the style of particular captions. The default style for a single line caption works well, but for a caption with two or three lines either the centering or centerlastline style might look better. A very long caption is again probably best done in a block paragraph style.

Only chapter titles are included in the ToC. To specify this we use the `\settocdepth` command.

```
\settocdepth{chapter}
```

The ToC is typeset raggedright with no leaders and the page numbers coming immediately after the chapter title. This is specified via:

```
\renewcommand{\cftchapterfont}{\normalfont}
\renewcommand{\cftchapterpagefont}{\normalfont}
\renewcommand{\cftchapterpresnum}{\bfseries}
\renewcommand{\cftchapterleader}{\bfseries}
\renewcommand{\cftchapterafterpnum}{\cftparfillskip}
```

9.7 PREAMBLE OR PACKAGE?

When making changes to the document style, or just defining a new macro or two, there is the question of where to put the changes — in the preamble of the particular document or into a separate package?

If the same changes/macros are likely to be used in more than one document then I suggest that they be put into a package. If just for the single document then the choice remains open.

I have presented the code in this chapter as though it would be put into the preamble, hence the use of `\makeatletter` and `\makeatother` to surround macros that include the `@` character (see §E.4). The code could just as easily be put into a package called, say, `bringhurst`. That is, by putting all the code, except for the `\makeatletter` and `\makeatother` commands, into a file called `bringhurst.sty`. It is a good idea also to end the code in the file with `\endinput`; LaTeX stops reading the file at that point and will ignore any possible garbage after `\endinput`.

*Preamble or
package?*

You then use the `bringhurst` package just like any other by putting

```
\usepackage{bringhurst}
```

in your document's preamble.

Ten

An example thesis design

Many universities in the United States have strict regulations about the typography of theses. The title and administrative pages are inherently specific to a particular university, but often the design for the body of the thesis clashes with normally accepted typographic practice. This chapter presents fairly typical guidelines for some US universities and code intended to meet them. Let's call the university in question the *Archibald Smythe University*, or ASU for short.

The requirements that are listed below are not from any single university but I have, over the years, seen each one of them. In reality there are likely to be many more nit-picking rules than I have shown.

It amuses me that I have never seen a printed set of requirements that followed the rules laid down therein.

Universities outside the US tend to be more relaxed with the result that theses from these establishments are very often more attractive (certainly less bulky) and more readable. The ASU requirements lead to an exceptionally dull and unattractive appearance.

10.1 EXAMPLE US THESIS TYPOGRAPHIC REQUIREMENTS

10.1.1 General

Paper size The thesis shall be printed on 8.5 by 11 inch plain white paper.

Single-sided The thesis shall be printed single-sided; that is, one side of each sheet of paper will be blank.

Margins Every page of the document shall meet the requirements of a 1.5 inch margin on the left and a 1 inch margin at the top, right, and bottom of the page. Nothing shall appear in any margin.

Fonts The thesis may be set in 10, 11 or 12pt Arial, Century, Garamond, Lucida Bright (10pt only), Tahoma, Times, or Verdana. The same font and size shall be used throughout the thesis.

There shall be no bold type.

Italic type (or underlining) is limited to the names of species, genera, book titles, musical compositions, or foreign words.

Line Spacing All text shall be double-spaced, except material in tables and the optional biographical sketch (which must be single-spaced). You shall single-space individual footnotes and endnotes with a double space between each entry.

10.1.2 Preliminary matter

The preliminary matter consists of the following pages in this order:

1. Title page
2. Approval page
3. Abstract
4. Dedication (optional)
5. Acknowledgements (optional)
6. Table of contents
7. List of tables (if there are any tables)
8. List of figures (if there are any figures)
9. Other lists (e.g., nomenclature, definitions, glossary of terms, etc.)
10. Preface (optional but must be less than ten pages)
11. Under special circumstances further sections may be allowed

The heading for each preliminary page (except the Dedication which shall not have a heading) is centered between the margins, in all capital letters, double-spaced and begin on the first line below the top margin.

The title and approval page are counted as pages one and two, but no page numbers shall appear on them. All subsequent preliminary pages are paginated with lowercase Roman numerals. Starting with 'iii' on the abstract page, place all page numbers at the bottom of the page, centered between the left and right margins and upon the 1 inch bottom margin. Continue numbering consecutively on the subsequent pages up to the first page of the main text.

Title page

1. All text shall be centered between the side margins.
2. Set the title in all capital letters, double-spaced, starting at the top of the page (but below the top margin).
3. On a new line (double-spaced) type 'by' in lowercase letters.
4. On a new line (double-spaced) type your full legal name.
5. At the center of the page type the appropriate description for your degree with the exact wording and line breaks as shown, and single-spaced:

A _____ Presented in Partial Fulfillment
of the Requirements for the Degree

Replace the blanks with the appropriate wording: Thesis and Master of Arts or Dissertation and Doctor of Philosophy.

6. At the bottom of the page type 'ARCHIBALD SMYTHE UNIVERSITY' in all capitals.
7. Type the month and year of the date you will graduate, with the month in title case and no comma between the month and year.
8. The space between your name and the degree description should equal the space between the degree description and the name of the University.

Approval page

1. All text shall be centered between the side margins.
2. Set the title in all capital letters, double-spaced, starting at the top of the page (but below the top margin).
3. On a new line (double-spaced) type 'by' in lowercase letters.
4. On a new line (double-spaced) type your full legal name in title-cased letters.
5. Add two double-spaced lines (four single-spaced lines) and type 'has been aproved' in lowercase
6. Add a double-space.
7. Type the month and year of your oral defense, with the month in title case and no comma between the month and year.
8. At about the center of the page type 'Graduate Supervisory Committee:'
9. A blank (double-spaced) line
10. Type the members' names, without titles, one per line, single-spaced, as follows:
 - a) If you have one chair, type: the chair's name, comma, space 'Chair'
 - b) If two chairs, type: comma, space, 'Co-Chair' after the first two names
 - c) Follow with the other members' names.
11. At the bottom of the page, type 'ACCEPTED BY THE GRADUATE COLLEGE'
12. The space between the date and 'Graduate Supervisory Committe' lines should equal the space between the last member's name and the 'ACCEPTED...' line.

Abstract page

Center the title 'ABSTRACT' at the top of the page. Number the page at the bottom, centered with the Roman numeral 'iii'. If there is a second page, number it similarly with 'iv'.

Dedication and Acknowledgements (optional)

- The dedication and acknowledgements together must not exceed three pages.
- The dedication page is not titled and the text should be centered both vertically and horizontally.
- The heading for the acknowledgements page is 'ACKNOWLEDGEMENTS', centered and at the top of the page.
- Continue the page numbering in lowercase Roman, at the bottom and centered.

10.1.3 Table of contents

1. Type 'TABLE OF CONTENTS' centered at the top of the page.
2. On the next line type the word 'Page' right justified
3. Begin listing any preliminary pages that follow the table of contents (e.g., lists) in ALL CAPS. The title is left justified, the page number is right justified and a dotted line fills the gap between.

4. Double space between entries.
5. Chapter headings and subheadings to three levels shall be listed, with a lower level being indented with respect to a higher level.
6. The wording of headings shall correspond exactly to those in the main body.
7. The page number is centered at the bottom of the page.
8. If the listing continues for more than one page, subsequent pages shall be headed with one line consisting of 'Chapter' left justified and 'Page' right justified.

10.1.4 Lists

For a given kind of list (often figures or tables) called, say, 'things':

1. Type 'LIST OF THINGS' centered at the top of the page.
2. On the next line type 'Thing' left justified and 'Page' right justified.
3. List, double-spaced, the caption or title of the thing left justified and the page number right justified, with a dotted line between them.
4. Use Roman lowercase to number the page(s) at the bottom, centered.
5. If the listing continues for more than one page, subsequent pages shall be headed with one line consisting of 'Thing' left justified and 'Page' right justified.

10.1.5 Main text

Nothing shall appear in the margins.

The top line on a page is the line immediately below the top margin. The top text line is the one following that (i.e. the second line below the margin).

Page numbering

All pages are counted, but the first page of each chapter is not numbered (paginated); other pages are paginated. The first page of the main text is counted as number 1. Numbered pages have the number right justified on the top line.

Headings

Chapter headings shall be centered. On the top line type 'Chapter' followed by the number. On the top text line type the heading in all uppercase. Type the text on the subsequent lines.

Subheadings, consisting of the number and title (not in all caps), shall be centered, with one blank line before and after.

Captions

Table captions, which are left justified, shall be put before the table itself. The first line consists of 'Table' followed by the number; the caption wording commences on the next line.

Captions for figures are similar, except that they shall be put below the figure and 'Table' replaced by 'Figure'.

Tables and figures shall be single-spaced.

Notes

Notes may be placed at the bottom of the page (i.e., footnotes), or grouped in the backmatter (i.e., endnotes) before the reference list.

All notes shall be introduced by a superior number in the text, with the same number used for the text of the note. Notes should be single spaced, with double spacing between them.

10.1.6 Backmatter

The backmatter consists of the following pages, in order (all of which are optional).

1. Notes (if you are using endnotes and grouping them at the end)
2. References (AKA 'Bibliography' or 'Works Cited')
3. Appendices
4. Biographical sketch (optional)

Pagination continues from the main text; but as with chapters, the first pages of any notes, references, or appendices are not numbered. A biographical sketch, if it is included, is the last page and is neither counted nor paginated.

Headings for the backmatter sections shall be in uppercase, centered, and on the top line.

Notes

The section for endnotes should begin on a new, unnumbered page. Subsequent pages should be numbered.

Use 'NOTES', centered and at the top, as the heading for the notes section.

References

Use the reference heading appropriate for your discipline, in uppercase, centered and at the top of the page. Individual references should be single-spaced with the second and later lines of a multiline reference indented with respect to the first line. There should be double-spacing between references.

Appendices

The heading for an appendix consists of the word 'APPENDIX' followed by the uppercase letter signifying its position in the sequence of appendices (e.g., A or B or C or ...). This shall be centered on the top line. The title of the appendix, in uppercase, is centered on the following line. This page is not numbered. Subsequent pages are numbered and the text commences on the top text line of the following page.

Biography

The title for the optional biographical page is 'BIOGRAPHICAL SKETCH', in the usual position. The text shall not exceed the one page.

10.2 CODE

Given the above set of requirements we can produce code that, hopefully, will generate documents that will not fall foul of the inspectorate. For simplicity I'll do the code in

the form of a package called `pwasu.sty`. I will be using some LaTeX kernel commands that you won't normally come across. Some of the macros include `@` as part of their name but this is safe as they are in a package, otherwise they would have to be within a `\makeatletter ... \makeatother` grouping (see §E.4).

10.2.1 Initialisation

First, identify the package and its basic requirements.

```
%%% file pwasu.sty
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{pwasu}[2009/04/18 v0.3 ASU thesis]
```

This is only going to work with memoir, so check if the class is being used, and if not then give an error message and stop reading the package code.

```
%% Only works with the memoir class!!!!!!!!!!!!!!
\@ifclassloaded{memoir}{\let\endpwasu\relax}{\let\endpwasu\endinput
  \PackageError{pwasu}{The pwasu package only works with the memoir class}%
  {\@ehd}}
\endpwasu
```

ASU is very strict about only using a single font in a single size. It is probable that at least one reference will be made to a location on the web. With LaTeX such references are set using the `url` package, which defaults to using a monospaced font for urls. For ASU we have to make sure that they will be made using the body font (and the same applies to any verbatim text, such as computer code listings, for which we can use the `\setverbatimfont` command).

```
\usepackage{url}
\urlstyle{same}
\setverbatimfont{\normalfont\rmfamily}% make verbatims use the body font
```

Ensuring that footnotes use the body font is a simple matter of redefining `\foottextfont`.

```
\renewcommand*{\foottextfont}{\normalfont\normalsize}
```

Noticing that the requirements can involve switching between double and single spacing, some of which will be done internally in the package, give the user a chance to change the default double spacing. The argument to `\setasuspacing` can be either `\OnehalfSpacing` or `\DoubleSpacing`. The result sets `\AsuSpacing` to be one of these commands.

```
% To enable spacing to be changed if necessary by the user
\newcommand*{\setasuspacing}[1]{%
  \let\AsuSpacing#1
  \AsuSpacing}
\setasuspacing{\DoubleSpacing}
```

10.2.2 Page layout

As this is for an American University on letterpaper sized paper we'll assume letterpaper stock, so no cropping will be needed. Setting the side margins is easy:

```
% left, right margins and textwidth
\setlrmarginsandblock{1.5in}{1in}{*}
```

Setting the top and bottom margins requires more thought. LaTeX provides the header and footer areas for page numbers. However, the requirements state that page numbers must be either on the top or bottom line (of the textblock) with the text extending down to the bottom line or up to the top line (of the textblock). I'll organise it according to the layout for the main body. Here, the top of the header is 1 inch below the top of the paper and the bottom of the text is 1 inch above the bottom of the paper (with the footer below that).

```
%% for main body, bottom of text at 1in, footer below
%% top of header at 1in, first text line double spaced
%% below base of header
\newlength{\linespace}
\setlength{\linespace}{\baselineskip} % current equivalent of \onelineskip
\setlength{\headheight}{\onelineskip}
\setlength{\headsep}{\linespace}
\addtolength{\headsep}{-\topskip}
\setlength{\uppermargin}{1in}
\addtolength{\uppermargin}{\headheight}
\addtolength{\uppermargin}{\headsep}
```

And for the bottom margin:

```
%% and for the bottom
\setlength{\lowermargin}{1in}
\setlength{\textheight}{\paperheight}
\addtolength{\textheight}{-\uppermargin}
\addtolength{\textheight}{-\lowermargin}
```

And finally for footnotes:

```
%% footnote settings
\setlength{\footskip}{\onelineskip}
\setlength{\footnotesep}{\onelineskip}
```

The layout on the preliminary pages is different, so I'll need some handy lengths in order to change the layout as appropriate. The user can also make adjustments with these if necessary.

```
%% the fiddle lengths (.ta.. for title/approval page, others for prelims)
\newlength{\toptafiddle} \setlength{\toptafiddle}{2\linespace}
\newlength{\bottafiddle} \setlength{\bottafiddle}{0pt}
\newlength{\topfiddle} \setlength{\topfiddle}{\toptafiddle}
\newlength{\botfiddle} \setlength{\botfiddle}{\onelineskip}
```

That's it for the general layout, except for increasing the paragraph indentation as the line spacing is larger than normal.

```
\setlength{\parindent}{2em}
\checkandfixthelayout[nearest]
```

As the layout is set up, the bottom of the text is one inch above the bottom of the paper. This is fine for the main text and the Title and Approval pages but the text height must be decreased, temporarily, for the other pages in the prelims. Changes to the page layout may be accomplished by the following sneaky procedure. First change from one- to two-column (or vice-versa), which starts a new page, make the changes, then change from two- to one-column (or vice-versa) which starts the same new page again but with the layout changes implemented. The following code implements this for the `\textheight` in a one column document, which is what we are dealing with here.

```
\newcommand*{\addtotextheight}[1]{%
  \twocolumn
  \addtolength{\textheight}{#1}%
  \onecolumn}
```

10.2.3 Page styles

Next I'll tackle the page styles. The style for the main body is simple, with the page number top right, and the *empty* style is the one for chapter pages. The page number for the preliminary pages is centered at the bottom, and the *plain* page style provides that. For the main text define the *asu* page style.

```
%%% pagestyles
%% the main text
\makepagestyle{asu}
\makeevenhead{asu}{\thepage}{}{}
\makeoddhead{asu}{}{}{\thepage}
```

Any 'continuation' pages for the ToC, etc., have a header that consists of a name at the left and the word 'Page' at the right. We need a header for each kind of listing.

```
%% for continuation pages of the ToC, LoF, LoT
\makepagestyle{toc}
\makeevenfoot{toc}{}{\thepage}{}
\makeoddfoot{toc}{}{\thepage}{}
\makeevenhead{toc}{Chapter}{}{Page}
\makeoddhead{toc}{Chapter}{}{Page}
\makepagestyle{lof}
\makeevenfoot{lof}{}{\thepage}{}
\makeoddfoot{lof}{}{\thepage}{}
\makeevenhead{lof}{Figure}{}{Page}
\makeoddhead{lof}{Figure}{}{Page}
\makepagestyle{lot}
\makeevenfoot{lot}{}{\thepage}{}
\makeoddfoot{lot}{}{\thepage}{}
\makeevenhead{lot}{Table}{}{Page}
\makeoddhead{lot}{Table}{}{Page}
```

10.2.4 The ToC and friends

While we're at it, do the code for the LoF and LoT, which is simpler than that needed for the ToC. We have to specify our new pagestyles which can be done by extending the `\listof...` macros. The `\addtodef` macro makes this rather easy.

```
%%% The LoF
\renewcommand{\listfigurename}{LIST OF FIGURES}
\addtodef{\listoffigures}{\clearpage\pagestyle{lof}}{}
```

For the titles, these have to be moved up into the header area, so that they come just below the top margin, and set the initial pagestyle as *plain*. After the title we can insert the relevant column headers.

```
\renewcommand*{\lofheadstart}{\vspace*{-\topfiddle}}
\renewcommand*{\afterloftitle}{\thispagestyle{plain}%
\par\nobreak {\normalfont Figure \hfill Page}\par\nobreak}
```

And the same for the LoT.

```
%%% The LoT
\renewcommand{\listtablename}{LIST OF TABLES}
\addtodef{\listoftables}{\clearpage\pagestyle{lot}}{ }
\renewcommand*{\lotheadstart}{\vspace*{-\topfiddle}}
\renewcommand*{\afterlottitle}{\thispagestyle{plain}%
\par\nobreak {\normalfont Table \hfill Page}\par\nobreak}
```

The ToC is similar but we also have to deal with the entries themselves.

```
%%% Do the ToC
\renewcommand{\contentsname}{TABLE OF CONTENTS}
\addtodef{\tableofcontents}{\clearpage\pagestyle{toc}}{ }
\renewcommand*{\tocheadstart}{\vspace*{-\topfiddle}}
\renewcommand*{\aftertoctitle}{\thispagestyle{plain}%
\par\nobreak \mbox{} \hfill {\normalfont Page}\par\nobreak}
```

And the changes to the entries, all of which are set in the normal font and with dotted leaders, with no extra space between any of the entries.

```
\renewcommand*{\cftchapterfont}{\normalfont}
\renewcommand*{\cftchapterpagefont}{\normalfont}
\renewcommand*{\cftchapterleader}{%
\cftchapterfont\cftdotfill{\cftchapterdotsep}}
\renewcommand*{\cftchapterdotsep}{\cftdotsep}
```

Unlike the typical LaTeX ToC there must be no additional space before chapter entries; also, there should be no additional space inserted by the chapters in the LoF or LoT which just requires a simple redefinition of `\insertchapterspace`.

```
%% no extra space before the entry, or in the LoF/LoT
\setlength{\cftbeforechapterskip}{0pt plus 0pt}
\renewcommand*{\insertchapterspace}{}
```

10.2.5 Chapter styling

Moving on to styling the chapter titles, the first line must be moved up into the header area, and other spacings set to give blank lines. The fonts are just the regular body font. Call the chapterstyle *asu* and make sure that the *empty* pagestyle is used for it.

```
%% chapter style
\makechapterstyle{asu}{%
  \setlength{\beforechapskip}{-\topfiddle}
  \setlength{\midchapskip}{1.0\onelineskip}
  \setlength{\afterchapskip}{1.0\onelineskip}
  \renewcommand*{\chapnamefont}{\normalfont}
  \renewcommand*{\chapnumfont}{\chapnamefont}
  \renewcommand*{\printchapternum}{\centering\chapnumfont \thechapter}
  \renewcommand*{\chaptitelfont}{\normalfont\centering}
  \renewcommand*{\printchapternonum}{\}
  \aliaspagestyle{chapter}{empty}
```

The chapterstyle for any appendices is slightly different as the title, all in uppercase, is on one page by itself and the text starts on the following page. Call this the *asuappendix* chapterstyle.

```
%% chapter style for appendices, text comes on following page
\makechapterstyle{asuappendix}{%
  \setlength{\beforechapskip}{-\topfiddle}
  \setlength{\midchapskip}{1.0\onelineskip}
  \setlength{\afterchapskip}{1.0\onelineskip}
  \renewcommand*{\chapnamefont}{\normalfont}
  \renewcommand*{\chapnumfont}{\chapnamefont}
  \renewcommand*{\printchaptername}{%
    \chapnamefont\MakeUppercase{\@chapapp}}
  \renewcommand*{\printchapternum}{\centering\chapnumfont \thechapter}
  \renewcommand*{\chaptitelfont}{\normalfont\centering}
  \renewcommand*{\printchapternonum}{\}
  \renewcommand*{\afterchaptertitle}{\clearpage}}
```

We have to extend the `\appendix` command to use the new chapter style, and also to ensure that double spacing will be used (certain elements that come before the appendices are single spaced).

```
%%% different chapter style for appendices, (and double spaced)
\addtodef{\appendix}{\chapterstyle{asuappendix}\AsuSpacing}
```

10.2.6 Section, etc., styling

Set up the section headings so that they are centered, use the normal font, and have a blank line before and after.

```
%%% (subsub)section styles
\setseheadstyle{\centering\normalfont}
```



```

\setbeforesecskip{-1\onelineskip plus -1ex minus -.2ex}
\setaftersecskip{1\onelineskip plus .2ex}
\setsubsecheadstyle{\centering\normalfont}
\setbeforesubsecskip{-1\onelineskip plus -1ex minus -.2ex}
\setaftersubsecskip{1\onelineskip plus .2ex}

```

10.2.7 Captions

The captions are set flushleft and raggedright with the name and number on one line and the title on the following line. Fortunately floats are automatically set single spaced, which is what the requirements specify.

```

%% Captions
\captiontitlefont{\normalfont}% title font
\precaption{\raggedright}% for Caption N
\captiondelim{\newline}% newline
\captionstyle{\raggedright}% for title
\setlength{\belowcaptionskip}{\onelineskip}

```

10.2.8 The bibliography

The requirements imply that the title is likely to be 'REFERENCES'. The bibliography is set single spaced but with a blank line between the entries.

```

%% for REFERENCE section
\renewcommand*{\bibname}{REFERENCES}
\setlength{\bibitemsep}{\onelineskip}

```

The second and later lines of any entry are to be indented. We use the `\biblistextra` hook for setting this up.

```

\renewcommand*{\biblistextra}{%
  \setlength{\itemsep}{\bibitemsep}
  \setlength{\labelwidth}{0pt}
  \setlength{\leftmargin}{3em}% hanging indent
  \setlength{\itemindent}{-\leftmargin}}

```

The title for the bibliography is set via the `\bibsection` macro. The heading is unnumbered but is added to the ToC. To get the spacing right the heading, set as a `\chapter*`, which must be called double spaced, and then single spacing is called for after that.

```

\renewcommand*{\bibsection}{%
  \AsuSpacing
  \chapter*{\bibname}\addcontentsline{toc}{chapter}{\bibname}
  \SingleSpacing}

```

10.2.9 End notes

The heading for the Notes section is similar to the bibliography heading.

```

%% endnotes
\renewcommand*{\notesname}{NOTES}
\renewcommand*{\notedivision}{%

```

```
\AsuSpacing
\chapter*{\notesname}
\addcontentsline{toc}{chapter}{\notesname}
\SingleSpacing
```

The rest of the code for endnotes ensures that they are numbered continuously throughout the text, the number is set as a superscript, that there is a blank line between each entry, and that there are no subdivisions within the listing.

```
\continuousnotenums
\renewcommand*{\notenuminnotes}[1]{\textsuperscript{#1}\space}
\renewcommand{\noteinnotes}[1]{#1\\}
\renewcommand*{\pagenotesubhead}[3]{}% no subheads
```

10.2.10 Preliminary headings

There can be any number of sections in the prelims. The titles for these are located in the LaTeX header area. Here's a general macro for setting these.

```
%%% general macro for Abstract, etc., headings
\newcommand*{\pretocitle}[1]{\clearpage\centering
\vspace*{-\topfiddle}#1\par}
%%% Start the ACKNOWLEDGEMENTS
\newcommand{\asuacknowledgements}{\pretocitle{ACKNOWLEDGEMENTS}}
```

The Abstract is the first section after the title and approval pages. At this point we must reduce the `textheight` in order to raise the footer area.

```
%%% Start the ABSTRACT
\newcommand{\asuabstract}{%
\addtotextheight{-\botfiddle}%
\pretocitle{ABSTRACT}}
```

While we are at this, the `textheight` must be reset to its default value just before the first chapter in the main matter. A simple addition to `\mainmatter` handles this.

```
\addtodef{\mainmatter}{\addtotextheight{\botfiddle}{}}
```

The dedication, if any, does not have a heading and the text is centered horizontally and vertically.

```
%% make it easy to center any dedication
\newcommand{\asudedication}[1]{%
{\clearpage\mbox{}}\vfill\centering #1 \par\vfill\clearpage}}
```

There may be sections in the prelims that come after the ToC, and the titles of these are added to the ToC.

```
%% for any headings after the tocleft and before the main body
\newcommand{\prelimtitle}[1]{%
\pretocitle{#1}\addcontentsline{toc}{chapter}{#1}}
```

10.2.11 Components of the title and approval pages

There are several items that are set on the title and approval pages. In order to separate the information from the particular layout, I've defined a macro for defining each item.

```
%%% for the title page and approval page.
% your title
\newcommand{\settitle}[1]{\def\asutitle{#1}}
% you
\newcommand{\setauthor}[1]{\def\asuaauthor{#1}}
% document type (e.g., thesis)
\newcommand{\setdoctype}[1]{\def\asudotype{#1}}
% possible degree
\newcommand{\masters}{\def\asudegree{Master of Arts}}
\newcommand{\doctors}{\def\asudegree{Doctor of Philosophy}}
% defence date
\newcommand{\setdefdate}[1]{\def\asudefdate{#1}}
% graduation date
\newcommand{\setgraddate}[1]{\def\asugraddate{#1}}
% committe chair
\newcommand{\setchair}[1]{\def\asuchair{#1, Chair}}
% committe co-chairs
\newcommand{\setchairs}[2]{%
  \def\asuchair{#1, Co-chair \\\ #2, Co-chair}}
% other members (separated by \s)
\newcommand{\setmembers}[1]{\def\asumembers{#1\par}}
```

Just for fun, create some default settings for these. The successful user will have changed them all!

```
%% Use them like this, and if you don't change them you will
%% get unacceptable title and/or approval pages
\settitle{AN INCREDIBLE PIECE OF WORK OVER WHICH I HAVE STRUGGLED
DAY AND NIGHT FOR FAR TOO LONG AND NOW IT IS OVER}
\setauthor{A. N. Author}
\setdoctype{Polemic}
\masters % going for a Masters degree
%% \doctors % going for a PhD
\setdefdate{April 2018}
\setgraddate{May 2021}
% \setchair{A. Jones} % this one
\setchairs{A. Jones}{B. Doe} % or this one
\setmembers{C. Smith \\\ D. Somebody \\\ Some One Else \\\ Yet Another}
```

10.2.12 The title and approval pages

An example of a title page is shown in Figure 10.1 and an example of the corresponding approval page is in Figure 10.2.

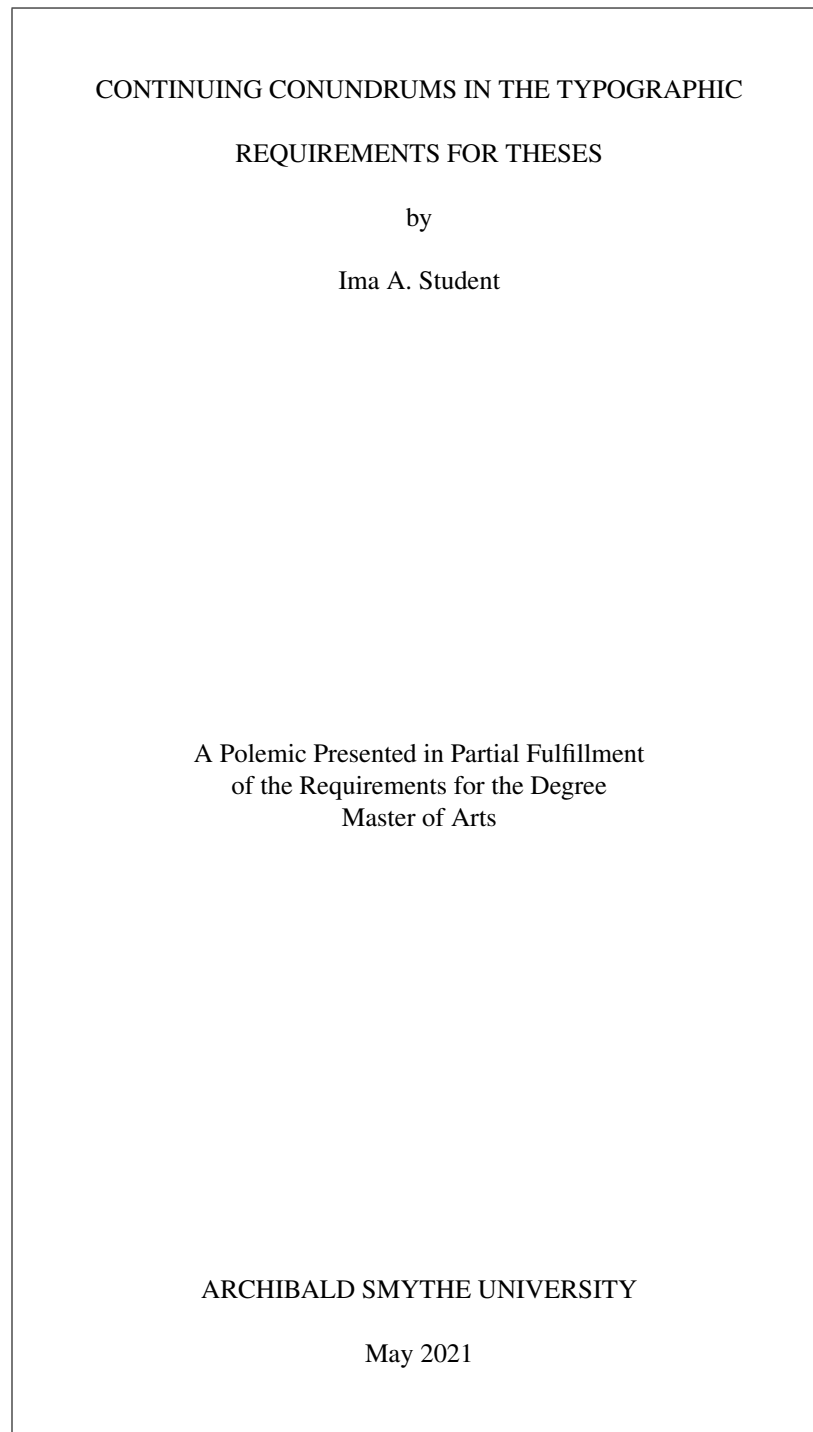


Figure 10.1: Example Archibald Smythe University title page

CONTINUING CONUNDRUMS IN THE TYPOGRAPHIC

REQUIREMENTS FOR THESES

by

Ima A. Student

has been approved

April 2018

Graduate Supervisory Committee:

S. Holmes, Co-Chair

J. Moriarty, Co-Chair

G. E. Challenger

A. Quartermain

J. H. Watson

ACCEPTED BY THE GRADUATE COLLEGE

Figure 10.2: Example Archibald Smythe University approval page

Now we can set up the layouts for the title and approval pages. The information typeset on these pages is obtained from the previous set of commands. Note that the last line on each of these pages has to be set upon the bottom margin. The ASU inspectorate is likely to be very keen on this, perhaps using a ruler to measure the actual distance from the bottom of the page to ensure that it is the magic 1 inch. I have included an `\enlargethispage` by the amount `\bottafiddle` so the user can make any fine adjustment that might be requested. Similarly, the length `\toptafiddle` may be altered to fine tune the position of the title. Hopefully, neither of these tweaks will be needed, but if so, then use `\addtolength` instead of `\setlength` to make an adjustment.

```
%%% typesets the TITLE page
\newcommand{\thetitlepage}{\{
  \clearpage
  \thispagestyle{empty}
  \centering
  \vspace*{-\toptafiddle}
  \asutitle \\\ by \\\ \asuauthor
  \vfill

  {\SingleSpace
  A \asudoctype\ Presented in Partial Fulfillment \\\
  of the Requirements for the Degree \\\
  \asudegree\par}

  \vfill
  ARCHIBALD SMYTHE UNIVERSITY \\\
  \asugraddate
  \par
  \enlargethispage{\bottafiddle}
  \clearpage}}
```

And similiary for the approval page.

```
%%% typesets the APPROVAL page
\newcommand{\approvalpage}{\{
  \thispagestyle{empty}
  \centering
  \vspace*{-\toptafiddle}
  \asutitle \\\ by \\\ \asuauthor \\\[3\onelineskip]
  has been approved \\\
  \asudefdate

  \vfill

  Graduate Supervisory Committee: \\\[-0.5\onelineskip]
  {\SingleSpacing
  \asuchair \\\
  \asumembers}
```

```

\vfill
ACCEPTED BY THE GRADUATE COLLEGE
\par
\enlargethispage{\bottafiddle}
\clearpage}}

```

10.2.13 The last bits

The biographical sketch has a title (which is not added to the ToC), the text is single spaced and there is no page number. It is easiest to provide this as an environment.

```

%% put your biographical text in this environment
%% \begin{biosketch} I'm a person who has accomplished .... \end{biosketch}
\newenvironment{biosketch}{%
  \pretoctitle{BIOGRAPHICAL SKETCH}\thispagestyle{empty}\SingleSpacing}%
{}

```

Make sure that the requisite initial page style and appropriate chapter style is used.

```

%% use the asu chapterstyle and plain pagestyle
\chapterstyle{asu}
\pagestyle{plain}

```

```

%%%%%%%%%%%%%% end of *.sty file
\endinput
%%%%%%%%%%%%%%

```

10.3 USAGE

This is a sketch of how an ASU thesis could be written.

With the wide textblock, 12pt is too small for reading ease, so best not to use 11pt or 10pt.

Times Roman comes with LaTeX, but you are effectively writing a book, not a newspaper column. If you have Garamond or Lucida Bright then seriously consider using one or other of them.¹ Lucida Bright is probably more appropriate if your thesis includes mathematics while Garamond is perhaps more in keeping if your thesis topic falls into the humanities area. If the requirements did not limit your choices then there are other fonts that might better express your work. In any case I suggest that you do not use a sans font (e.g., Arial, Tahoma or Verdana from the ASU list).

```

\documentclass[oneside,12pt]{memoir}
\usepackage{mathptmx} % Times New Roman
\usepackage{pwasu} % the package

```

¹Garamond is a commercial font and, for example, is available along with many other fonts from FontSite (<http://www.fontsite.com>) with LaTeX support from <http://contrapunctus.net/fs500tex>. Lucida Bright, another commercial font, is available from TUG at <http://tug.org/store/lucida> and is supported by several LaTeX packages.

The general sequence in your document, after you have set the data for the TITLE and APPROVAL pages and any other specifics and packages in the preamble, is:

```
% if you can get away without the default \DoubleSpacing, then
%\setasuspending{\OnehalfSpacing}
%% if you use endnotes, then
\makepagenote
\begin{document}
\maxtocdepth{subsection} % put 3 levels into the ToC
\frontmatter
\thetitlepage
\approvalpage
\asuabstract
abstract text
%% if you have any acknowledgements, then
\asuacknowledgements
acknowledgements text
% \asudedication{ text } % if you want a dedication
\tableofcontents
% \listoffigures % if you have any figures
% \listoftables % if you have any tables
%% if you have more prelim sections, then
%% \clearpage
%% \pagestyle{plain}
%% \prelimtitle{title} text % for each section before main text
\mainmatter
\pagestyle{asu}
\chapter{...} % start of your main text
... report on lots of incredible work, now you are on your
own until...

%% if endnotes then
\printpagenotes
%% if a bibliography then
\begin{thebibliography}...\end{thebibliography}
%% if appendices, then
\appendix
\chapter{...}
...
%% if Biographical sketch then
\begin{biosketch} ... \end{biosketch}
\end{document}
```

If you actually try any of the above code and it does not work, then I may have made a typo or two, or maybe you have, or perhaps we both have. In any event, the code is more of a sketch of what might be needed than a prescription of how it must be done.

10.4 COMMENTS

Having read through the requirements you will have realised that whatever committees set them had not advanced beyond the 19th century technology of the typewriter.² When I wrote my thesis some forty years ago it was, of necessity, single sided so that carbon copies could be made by the typist (who would have objected strongly to having to type the hundred and fifty or so pages six times). I must admit, though, that the sixth copy was almost too faint and blurry to be read comfortably even though the typist had used thin paper and kept replacing the carbon paper. In this day of double sided printers and double sided copiers I see no reason except inertia to keep a single sided requirement. Many students, and faculty members, have beaten their heads against the diehards and very rarely have they managed to prevail.

In contrast to the ASU style I have a copy of a doctoral thesis [Sne04] for Vrije Universiteit, Amsterdam. This is a professionally printed 100 page, double sided, glossy paperbound book with an attractive coloured photograph of a sunset on the front and rear covers. The page size is 40.5pc by 57pc with spine and top margins of 5pc and fore-edge and bottom margins of 7pc.³ The textblock, then, is 28.5 by 45pc set with 45 lines of a 10pt Lucida Bright seriffed font. Chapter and section heads are in a sans font, with the chapter heads larger than the section heads. Caption names are a bold sans with the caption text in an italic. Headers on the verso pages are the chapter title with the section title as the recto header. The page numbers are in the footers by the fore-edge margin. Altogether, a much more appealing production than Archibald Smythe University will permit.

²Remington sold their first commercial typewriter in 1873 which even then had the QWERTY keyboard layout. By 1910 typewriter designs were pretty well standardised.

³Professional printers use points and picas for their measurements.

Appendices

A

Packages and macros

The memoir class does not provide for everything that you have seen in the manual. I have used some packages that you are very likely to have in your LaTeX distribution, and have supplemented these with some additional macros, some of which I will show you.

A.1 PACKAGES

The packages that I have used that you are likely to have, and if you do not have them please consider getting them, are:

- `etex` lets you take advantage of eTeX's extended support for counters and such.
Note that from 2015 and onwards, the allocation of extra registers have now been build into the LaTeX kernel. Thus in most cases the `etex` package is no longer necessary. There are how ever extra very special features left in `etex` that *some* users may need. In that case please remember to load `etex` by placing `\RequirePackage{etex}` *before* `\documentclass`!
- `url` [Ars99] is for typesetting URL's without worrying about special characters or line breaking.
- `fixltx2e` [MC00] eliminates some infelicities of the original LaTeX kernel. In particular it maintains the order of floats on a twocolumn page and ensures the correct marking on a twocolumn page.
Note that as of 2015, the functionality of this package has been merged into the L^AT_EX kernel. Loading this package does nothing.
- `alltt` [Bra97] is a basic package which provides a verbatim-like environment but `\`, `{`, and `}` have their usual meanings (i.e., LaTeX commands are not disabled).
- `graphicx` [CR99] is a required package for performing various kinds of graphical functions.
- `color` [Car05] is a required package for using color, or `xcolor` [Ker07] is an enhanced version of color.
- `latexsym` gives access to some extra symbols.
- `amsmath` for when you are doing anything except the simplest kind of maths typesetting.
- `fontenc` for using fonts with anything other than the original OT1 encoding (i.e., for practically any font).
- `pifont` for typesetting Pifonts (i.e., Symbol and Zapf Dingbats)

Apart from the packages that are supplied as part of the memoir distribution, the packages that I used and you most likely do not have are:

- layouts [Wil03a]. I used it for all the layout diagrams. For example, Figure ?? and Figure ?? were drawn simply by:

```
\begin{figure}
\centering
\setlayoutscale{1}
\drawparameterstrue
\drawheading{}
\caption{Displayed sectional headings} \label{fig:displaysehead}
\end{figure}

\begin{figure}
\centering
\setlayoutscale{1}
\drawparameterstrue
\runinheadtrue
\drawheading{}
\caption{Run-in sectional headings} \label{fig:runsehead}
\end{figure}
```

The package also lets you try experimenting with different layout parameters and draw diagrams showing what the results would be in a document.

The version of layouts used for this manual is v2.4 dated 2001/04/30. Earlier versions will fail when attempting to draw some figures (e.g., to draw Figure ??).

- fonttable [Wil09a]. I used this for the font tables (e.g., Table ??). You must have at least version 1.3 dated April 2009 for processing the manual (earlier versions are likely to produce errors in the number formatting area with minor, but odd looking, effect on the printed result).

A.2 MACROS

Originally the preamble of the manual contained many macro definitions, probably more than most documents would because:

- I am having to typeset many LaTeX commands, which require some sort of special processing;
- I have tried to minimize the number of external packages needed to LaTeX this manual satisfactorily, and so have copied various macros from elsewhere;
- I wanted to do some automatic indexing;
- I wanted to set off the syntax specifications and the code examples from the main text.

I have since put the majority of these into a package file called `memsty.sty`. To get the whole glory you will have to read the preamble, and the `memsty` package file but I show a few of the macros below as they may be of more general interest.

`\Ppstyle{<pagestyle>} \pstyle{<pagestyle>}`

The command `\Ppstyle` prints its argument in the font used to indicate pagestyles and the command `\pstyle` prints its pagestyle argument and also makes a pagestyle entry in the index. Its definition is

```
\newcommand*{\pstyle}[1]{\Ppstyle{#1}%
  \index{#1 pages?\Ppstyle{#1} (pagestyle)}}%
  \index{pagestyle!#1?\Ppstyle{#1}}}
```

The first part prints the argument in the text and the second adds two entries to the `idx` file. The fragment `#1 pages` is what the `MakeIndex` program will use for sorting entries, and the fragment following the `?` character is what will be put into the index.

```
\Pcstyle{\chapterstyle} \cstyle{\chapterstyle}
```

The command `\Pcstyle` prints its argument in the font used to indicate chapterstyles and `\cstyle` prints its chapterstyle argument and also makes a chapterstyle entry in the index. Its definition is

```
\newcommand*{\cstyle}[1]{\Pcstyle{#1}%
  \index{#1 chaps?\Pcstyle{#1} (chapterstyle)}}%
  \index{chapterstyle!#1?\Pcstyle{#1}}}
```

which is almost identical to `\pstyle`.

There is both a *companion* chapterstyle and a *companion* pagestyle. The strings used for sorting the index entries for these are `companion chaps` and `companion pages` respectively, so the chapterstyle will come before the pagestyle in the index. The reason for distinguishing between the string used for sorting and the actual entry is partly to distinguish between different kinds of entries for a single name and partly to avoid any formatting commands messing up the sorted order.

```
\begin{syntax} syntax \end{syntax}
```

The `syntax` environment is for specifying command and environment `syntax`. Its definition is

```
\newcommand*{\tightcenter}{%
  \topsep=0.25\onelineskip\trivlist \centering\item\relax}
\def\endtightcenter{\endtrivlist}
\newenvironment{syntax}{\begin{tightcenter}
  \begin{tabular}{|p{0.9\linewidth}|} \hline}%
  {\hline
  \end{tabular}
  \end{tightcenter}}
```

It is implemented in terms of the `tabular` environment, centered within the `typeblock`, which forms a box that will not be broken across a pagebreak. The box frame is just the normal horizontal and vertical lines that you can use with a `tabular`. The width is fixed at 90% of the text width. As it is a `tabular` environment, each line of `syntax` must be ended with `\\`. Note that normal LaTeX processing occurs within the `syntax` environment, so you can effectively put what you like inside it. The `center` environment is defined in terms of a `trivlist` and `\centering`. I wanted to be able to control the space before and after the `\centering` so I defined the `tightcenter` environment which enabled me to do this.

`\begin{lcode} LaTeX code \end{lcode}`

I use the `lcode` environment for showing examples of LaTeX code. It is a special kind of `verbatim` environment where the font size is `\small` but the normal `\baselineskip` is used, and each line is indented.

At the bottom the environment is defined in terms of a `list`, although that is not obvious from the code; for details see the class code [Wil09b]. I wanted the environment to be a tight list and started off by defining two helper items.

```
% \@zrowseps sets list before/after skips to minimum values
\newcommand*{\@zrowseps}{\setlength{\topsep}{\z@}
                        \setlength{\partopsep}{\z@}
                        \setlength{\parskip}{\z@}}
% \gparindent is relative to the \parindent for the body text
\newlength{\gparindent} \setlength{\gparindent}{0.5\parindent}
```

The macro `\@zrowseps` sets the before, after and middle skips in a list to 0pt (`\z@` is shorthand for 0pt). The length `\gparindent` will be the line indentation in the environment.

```
% Now we can do the new lcode verbatim environment.
% This has no extra before/after spacing.
\newenvironment{lcode}{\@zrowseps
  \renewcommand{\verbatim@startline}{%
    {\verbatim@line{\hskip\gparindent}}
  \small\setlength{\baselineskip}{\onelineskip}\verbatim}%
  {\endverbatim
   \vspace{-\baselineskip}\noindent}
```

The fragment `{\hskip\gparindent}` puts `\gparindent` space at the start of each line.

The fragment `\small\setlength{\baselineskip}{\onelineskip}` sets the font size to be `\small`, which has a smaller `\baselineskip` than the normal font, but this is corrected for by changing the local `\baselineskip` to the normal skip, `\onelineskip`. At the end of the environment there is a negative space of one line to compensate for a one line space that LaTeX inserts.

B

Showcases

The memoir memoir class has several features that involve a *style* and it provide several of these styles. This chapter is used to showcase these styles.

B.1 CHAPTER STYLES

For more about defining chapter styles, see section ??, page ??.

Chapter 1

Demonstration of the default chapter style

The above is a demonstration of the *default* chapterstyle. It is one of several styles that come as part of the memoir class.

Figure B.1: The default chapterstyle

2 Demonstration of the section chapter style

The above is a demonstration of the *section* chapterstyle. It is one of several styles that come as part of the memoir class.

Figure B.2: The section chapterstyle

3 Demonstration of the hangnum chapter style

The above is a demonstration of the *hangnum* chapterstyle. It is one of several styles that come as part of the memoir class.

Figure B.3: The hangnum chapterstyle

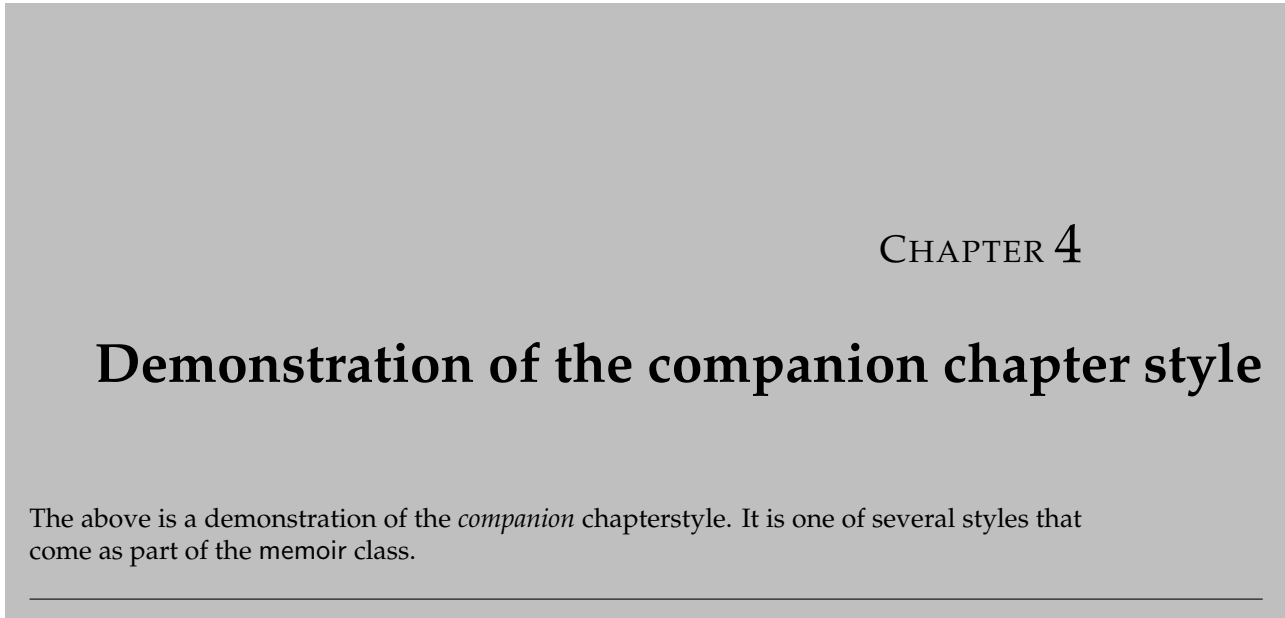


Figure B.4: The companion chapterstyle

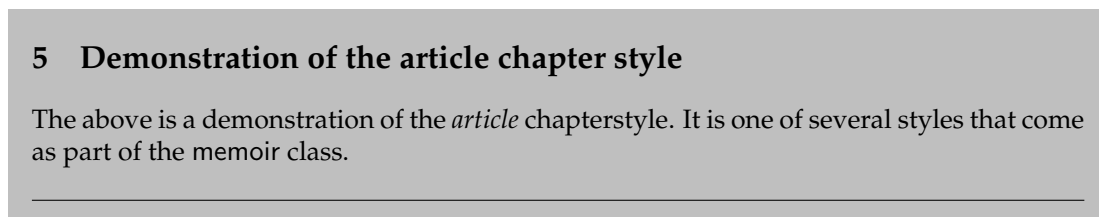


Figure B.5: The article chapterstyle

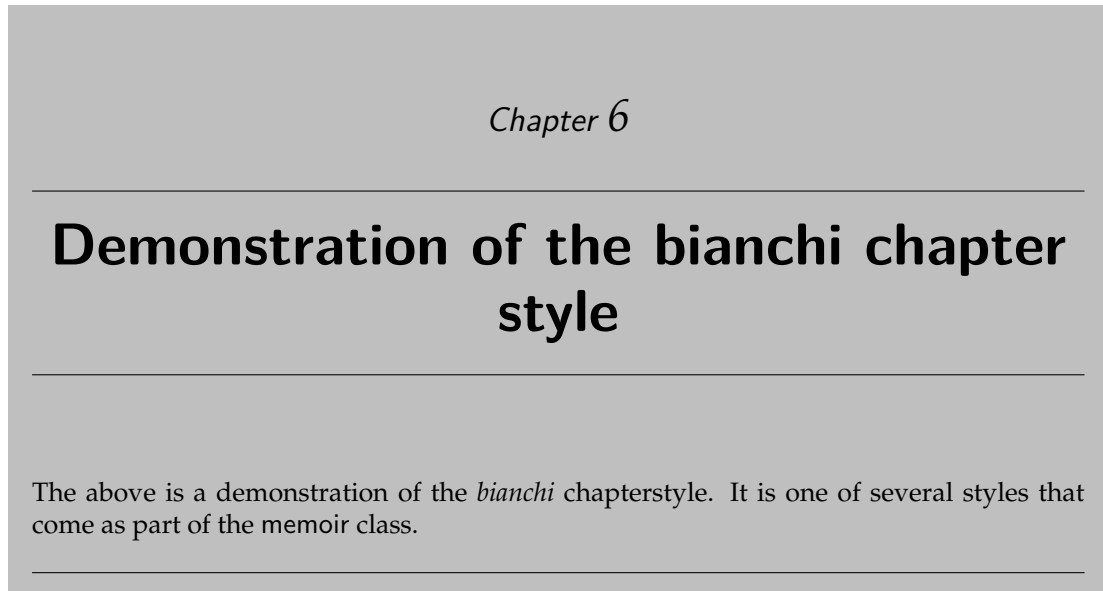


Figure B.6: The bianchi chapterstyle

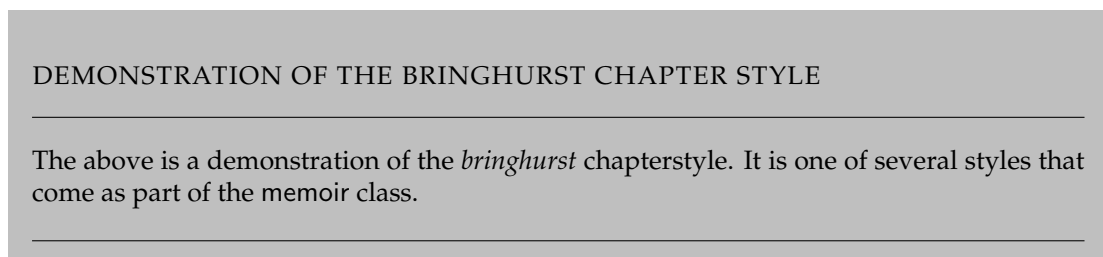


Figure B.7: The bringhurst chapterstyle

Chapter 8

Demonstration of the *brotherton* chapter style

The above is a demonstration of the *brotherton* chapterstyle. It is one of several styles that come as part of the memoir class.

Figure B.8: The *brotherton* chapterstyle

Chapter 9

*Demonstration of the *chappell* chapter style*

The above is a demonstration of the *chappell* chapterstyle. It is one of several styles that come as part of the memoir class.

Figure B.9: The *chappell* chapterstyle

10 Demonstration of the crosshead chapter style

The above is a demonstration of the *crosshead* chapterstyle. It is one of several styles that come as part of the memoir class.

Figure B.10: The crosshead chapterstyle

I Demonstration of the culver chapter style

The above is a demonstration of the *culver* chapterstyle. It is one of several styles that come as part of the memoir class.

Figure B.11: The culver chapterstyle

— 2 —

Demonstration of the dash chapter style

The above is a demonstration of the *dash* chapterstyle. It is one of several styles that come as part of the memoir class.

Figure B.12: The dash chapterstyle

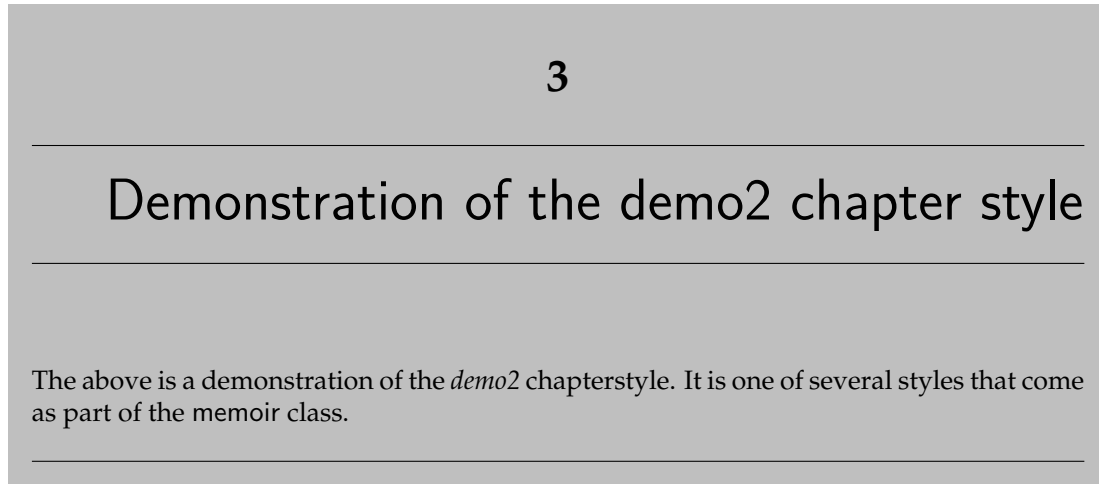


Figure B.13: The demo2 chapterstyle

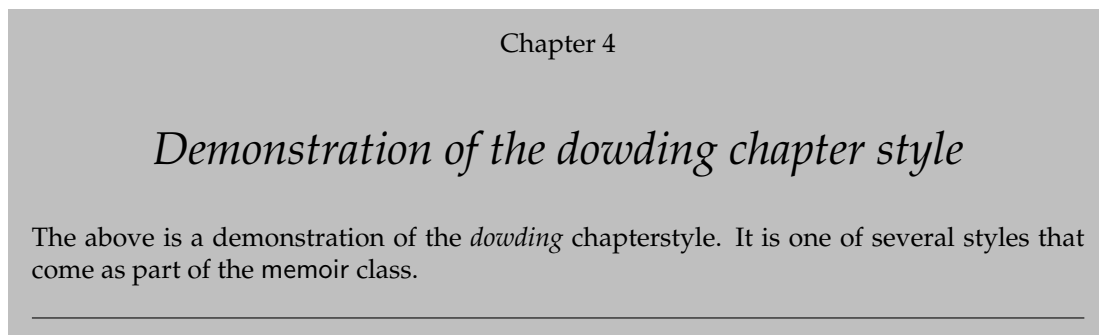


Figure B.14: The dowing chapterstyle

Demonstration of the *ell* chapter style

The above is a demonstration of the *ell* chapterstyle. It is one of several styles that come as part of the memoir class.

Figure B.15: The *ell* chapterstyle

Chapter 6

Demonstration of the *ger* chapter style

The above is a demonstration of the *ger* chapterstyle. It is one of several styles that come as part of the memoir class.

Figure B.16: The *ger* chapterstyle

7 Demonstration of the komalike chapter style

The above is a demonstration of the *komalike* chapterstyle. It is one of several styles that come as part of the memoir class.

Figure B.17: The komalike chapterstyle

CHAPTER 8

Demonstration of the lyhne chapter style

The above is a demonstration of the *lyhne* chapterstyle. It is one of several styles that come as part of the memoir class.

Figure B.18: The lyhne chapterstyle. This style requires the graphicx package

Demonstration of the madsen chapter style

The above is a demonstration of the *madsen* chapterstyle. It is one of several styles that come as part of the memoir class.

Figure B.19: The madsen chapterstyle. This style requires the graphicx package

Chapter 10

Demonstration of the *ntglike* chapter style

The above is a demonstration of the *ntglike* chapterstyle. It is one of several styles that come as part of the memoir class.

Figure B.20: The *ntglike* chapterstyle

1 Demonstration of the *southall* chapter style

The above is a demonstration of the *southall* chapterstyle. It is one of several styles that come as part of the memoir class.

Figure B.21: The *southall* chapterstyle

2 Demonstration of the tandh chapter style

The above is a demonstration of the *tandh* chapterstyle. It is one of several styles that come as part of the memoir class.

Figure B.22: The tandh chapterstyle

CHAPTER 3

DEMONSTRATION OF THE THATCHER CHAPTER STYLE

The above is a demonstration of the *thatcher* chapterstyle. It is one of several styles that come as part of the memoir class.

Figure B.23: The thatcher chapterstyle

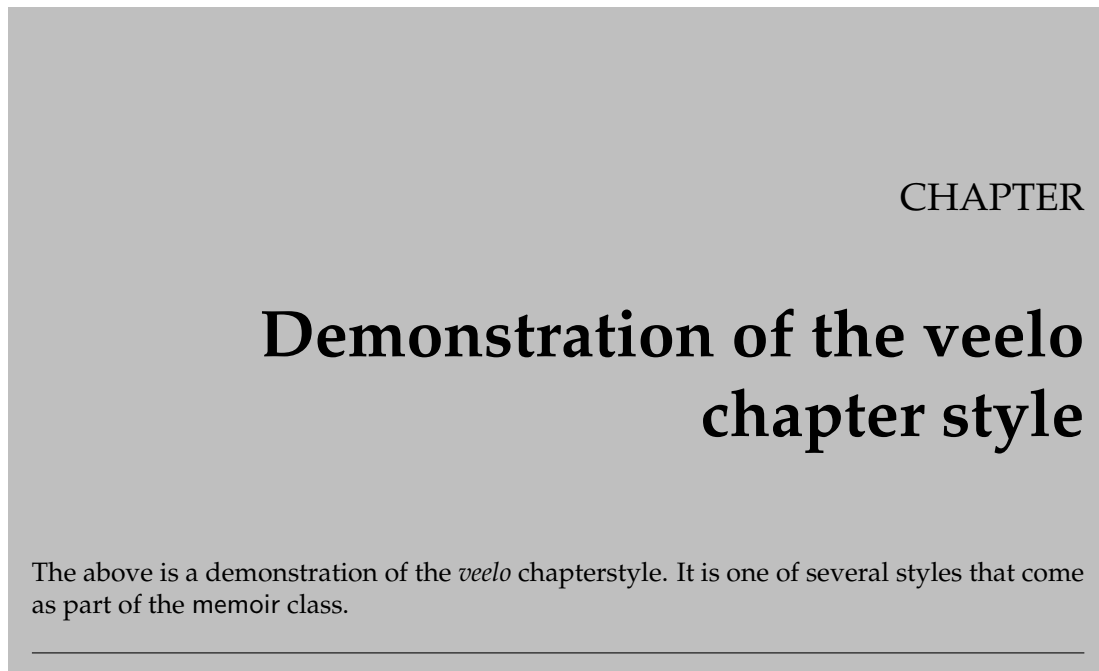


Figure B.24: The veelo chapterstyle. This style requires the graphicx package

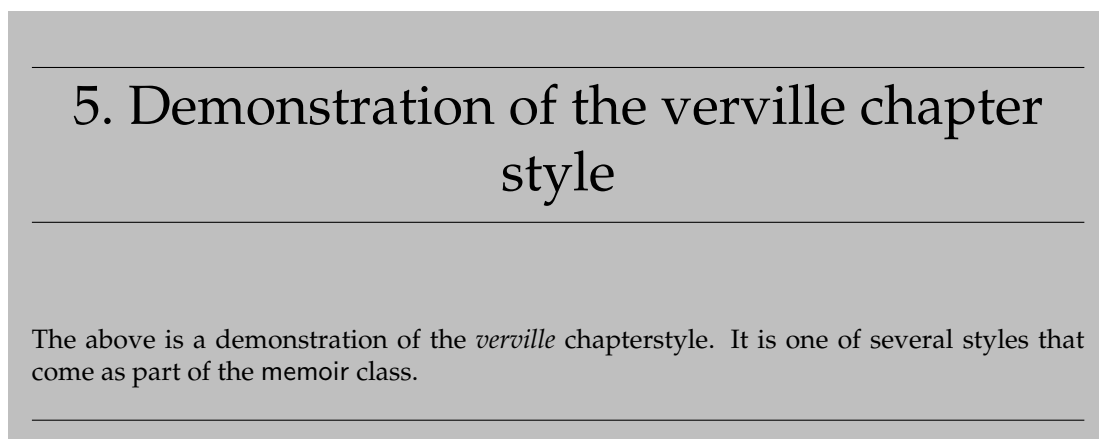


Figure B.25: The verville chapterstyle

6 *Demonstration of the wilsondob chapter style*

The above is a demonstration of the *wilsondob* chapterstyle. It is one of several styles that come as part of the memoir class.

Figure B.26: The wilsondob chapterstyle

The code for some of these styles is given in below. For details of how the other chapter styles are defined, look at the documented class code. This should give you ideas if you want to define your own style.

Note that it is not necessary to define a new chapterstyle if you want to change the chapter headings — you can just change the individual macros without putting them into a style.

B.1.1 Chappell

A style that includes rules is one that I based on the chapter heads in [CB99] and which I have called *chappell* after the first author. The style, which is shown in Figure B.9, can easily form the basis for general heads in non-technical books.

```
\makechapterstyle{chappell}{%
  \setlength{\beforechapskip}{0pt}
  \renewcommand*{\chapnamefont}{\large\centering}
  \renewcommand*{\chapnumfont}{\large}
  \renewcommand*{\printchapternonum}{%
    \vphantom{\printchaptername}%
    \vphantom{\chapnumfont 1}%
    \afterchapternum
    \vskip -\onelineskip}
  \renewcommand*{\chaptitelfont}{\Large\itshape}
  \renewcommand*{\printchaptertitle}[1]{%
    \hrule\vskip\onelineskip \centering\chaptitelfont ##1}}
```

The style centers the chapter number, draws a rule across the page under it, and below that comes the title, again centered. All the fiddling in the `\printchapternonum` macro is to try and ensure that the rule above the title is at the same height whether or not the chapter is numbered (the ToC being an example of an unnumbered heading).

B.1.2 Demo, Demo2 and demo3

I created a *demo* chapterstyle quite a time ago and used it on occasions in earlier editions of this Manual. Here is the original code.

```
\makechapterstyle{demo}{%
  \renewcommand*{\printchaptername}{\centering}
  \renewcommand*{\printchapternum}{\chapnumfont \numtoName{\c@chapter}}
  \renewcommand*{\chapttitlefont}{\normalfont\Huge\sffamily}
  \renewcommand*{\printchaptertitle}[1]{%
    \hrule\vskip\onelineskip \raggedleft \chapttitlefont ##1}
  \renewcommand*{\afterchaptertitle}{%
    {\vskip\onelineskip \hrule\vskip \afterchapskip}
}% end demo
```

This has one serious failing and what I now believe is a poor design decision. The failing is that if you have any appendices that use the *demo* chapterstyle then they are numbered instead of being lettered. The poor design is that the position of the title with respect to the top of the page is not the same for numbered and unnumbered chapters. The *demo2* chapterstyle below fixes both of these at the expense of simplicity (at least for me).

```
\makechapterstyle{demo2}{%
  \renewcommand*{\printchaptername}{\centering}
  \renewcommand*{\printchapternum}{\chapnumfont
    \ifanappendix \thechapter \else \numtoName{\c@chapter}\fi}
  \renewcommand*{\chapttitlefont}{\normalfont\Huge\sffamily}
  \renewcommand*{\printchaptertitle}[1]{%
    \hrule\vskip\onelineskip \raggedleft \chapttitlefont ##1}
  \renewcommand*{\afterchaptertitle}{%
    \vskip\onelineskip \hrule\vskip \afterchapskip}
  \setlength{\beforechapskip}{3\baselineskip}
  \renewcommand*{\printchapternonum}{%
    \vphantom{\chapnumfont One}
    \afterchapternum%
    \vskip\topskip}
  \setlength{\beforechapskip}{2\onelineskip}
}% end{demo2}
```

You may find it instructive to compare the code for the *demo* and *demo2* chapterstyles.

The *demo* chapterstyle is still available in the class for backward compatibility reasons, but I strongly advise against anyone using it.

By chance I inadvertentlyly typed a chapterstyle that was a mixture of the *pedersen* and *demo2* styles. As a result there is now a *demo3* chapterstyle as well. The only difference between the two styles is in the definition of `\chapnumfont` which in *demo3* is:

```
\renewcommand*{\chapnumfont}{\normalfont\HUGE\itshape}
```

B.1.3 Pedersen

I have modified Troels Pedersen's original code to make it a little more efficient and flexible.

```

\newcommand*{\colorchapnum}{}
\newcommand*{\colorchaptitle}{}
\makechapterstyle{pedersen}{%
  \setlength{\beforechapskip}{-20pt}
  \setlength{\afterchapskip}{10pt}
  \renewcommand*{\chapnamefont}{\normalfont\LARGE\itshape}
  \renewcommand*{\chapnumfont}{\normalfont\HUGE\itshape\colorchapnum}
  \renewcommand*{\chaptitlenamefont}{\normalfont\huge\itshape\colorchaptitle}
  \renewcommand*{\afterchapternum}{}
  \renewcommand*{\printchaptername}{}
  \setlength{\midchapskip}{20mm}
  \renewcommand*{\chapternamenum}{}
  \renewcommand*{\printchapternum}{%
    \sidebar{\raisebox{0pt}[0pt][0pt]{\makebox[0pt][l]{%
      \resizebox{!}{\midchapskip}{\chapnumfont\thechapter}}}}
  \renewcommand*{\printchaptertitle}[1]{\chaptitlenamefont ##1}
}

```

The chapter number is scaled up from its normal size and set in a sidebar.

`\colorchapnum \colorchaptitle`

The title is set with `colorchaptitle` and the number with `colorchapnum`, both of which default to doing nothing. Lars Madsen has suggested an attractive red color for these:

```

\usepackage{color}
\definecolor{ared}{rgb}{.647,.129,.149}
\renewcommand*{\colorchapnum}{\color{ared}}
\renewcommand*{\colorchaptitle}{\color{ared}}
\chapterstyle{pedersen}

```

The uncolored version is used for the `chaptersyle` for this chapter; because of setting the number in a sidebar it does not display well anywhere other than as a real chapter head.

B.1.4 Southall

On 2006/01/08 Thomas Dye posted his *southall* chapterstyle on [comp.text.tex](#) and kindly gave me permission to include it here. It is based on the headings in a Cambridge Press book¹ by Aidan Southall. It produces a simple numbered heading with the title set as a block paragraph, and with a horizontal rule underneath. His original code called for lining figures for the number but I have commented out that bit. I also changed the code to eliminate the need for the two new lengths that Thomas used.

```

\makechapterstyle{southall}{%
  \setlength{\afterchapskip}{5\baselineskip}
  \setlength{\beforechapskip}{36pt}
  \setlength{\midchapskip}{\textwidth}
  \addtolength{\midchapskip}{-\beforechapskip}
  \renewcommand*{\chapterheadstart}{\vspace*{2\baselineskip}}
}

```

¹Which I haven't seen

```

\renewcommand*{\chaptitelfont}{\huge\rmfamily\raggedright}
\renewcommand*{\chapnumfont}{\chaptitelfont}
\renewcommand*{\printchaptername}{}
\renewcommand*{\chapternamenum}{}
\renewcommand*{\afterchapternum}{}
\renewcommand*{\printchapternum}{%
  \begin{minipage}[t][\baselineskip][b]{\beforechapskip}
    {\vspace{0pt}\chapnumfont%%\figureversion{lining}
      \thechapter}
  \end{minipage}}
\renewcommand*{\printchaptertitle}[1]{%
  \hfill\begin{minipage}[t]{\midchapskip}
    {\vspace{0pt}\chaptitelfont ##1\par}\end{minipage}}
\renewcommand*{\afterchaptertitle}{}
\par\vspace{\baselineskip}%
\hrulefill \par\nobreak\noindent \vskip\afterchapskip}}

```

The resulting style is shown in Figure B.21.

B.1.5 Veelo

Bastiaan Veelo posted the code for a new chapter style to CTT on 2003/07/22 under the title *[memoir] [contrib] New chapter style*. His code, which I have slightly modified and changed the name to *veelo*, is below. I have also exercised editorial privilege on his comments.

I thought I'd share a new chapter style to be used with the memoir class. The style is tailored for documents that are to be trimmed to a smaller width. When the bound document is bent, black tabs will appear on the fore side at the places where new chapters start as a navigational aid. We are scaling the chapter number, which most DVI viewers will not display accurately.

Bastiaan.

In the style as I modified it, `\beforechapskip` is used as the height of the number and `\midchapskip` is used as the length of the black bar.

```

\newlength{\numberheight}
\newlength{\barlength}
\makechapterstyle{veelo}{%
  \setlength{\afterchapskip}{40pt}
  \renewcommand*{\chapterheadstart}{\vspace*{40pt}}
  \renewcommand*{\afterchapternum}{\par\nobreak\vskip 25pt}
  \renewcommand*{\chapnamefont}{\normalfont\LARGE\flushright}
  \renewcommand*{\chapnumfont}{\normalfont\HUGE}
  \renewcommand*{\chaptitelfont}{\normalfont\HUGE\bfseries\flushright}
  \renewcommand*{\printchaptername}{%
    \chapnamefont\MakeUppercase{\@chapapp}}
  \renewcommand*{\chapternamenum}{}
  \setlength{\beforechapskip}{18mm}
  \setlength{\midchapskip}{\paperwidth}
  \addtolength{\midchapskip}{-\textwidth}
}

```

```

\addtolength{\midchapskip}{-\spinemargin}
\renewcommand*{\printchapternum}{%
  \makebox[0pt][l]{\hspace{.8em}%
    \resizebox{!}{\numberheight}{\chapnumfont \thechapter}%
    \hspace{.8em}%
    \rule{\midchapskip}{\beforechapskip}%
  }}%
\makeoddfoot{plain}{}{}{\thepage}}

```

If you use this style you will also need to use the `graphicx` package [CR99] because of the `\resizebox` macro. The *veelo* style works best for chapters that start on recto pages.

C

Snippets

This chapter is (over time) meant to hold various pieces of code for memoir that we have gathered over the years or others have contributed, and which we think might be useful for others. In some cases they will have been moved from the text to this place, in order to make the manual less cluttered.

If you have some memoir related code you would like to share, feel free to send it to daleif@math.au.dk.

SNIPLET OVERVIEW

Snippet C.1 (Mirroring the output)	153
Snippet C.2 (Remove pagenumber if only one page)	154
Snippet C.3 (A kind of draft note)	154
Snippet C.4 (Adding indentation to footnotes)	155
Snippet C.5 (Background image and trimmarks)	155
Snippet C.6 (Autoadjusted number widths in the ToC)	155
Snippet C.7 (Using class tools to make a chapter ToC)	157
Snippet C.8 (An appendix ToC)	159

SNIPLET C.1 (MIRRORING THE OUTPUT)

The memoir class is not quite compatible with the crop package. This is usually not a problem as we provide our own crop marks. But crop provide one feature that we do not: mirroring of the output. The following snippet was posted on CTT by Heiko Oberdiek (2009/12/05, thread *Memoir and mirrored pdf output*)

```
\usepackage{atbegshi}
\usepackage{graphicx}
\AtBeginShipout{%
  \sbox\AtBeginShipoutBox{%
    \kern-1in\relax
    \reflectbox{%
      \rlap{\kern1in\copy\AtBeginShipoutBox}%
      \kern\stockwidth
    }%
  }%
}
```

SNIPLET C.2 (REMOVE PAGENUMBER IF ONLY ONE PAGE)

Memoir counts all the pages used. You can use this information in various ways. For example, say you are preparing a setup to write small assignments in, these may or may not be just one page. How do we remove the footer automatically if there is only one page?

Easy, place the following in the preamble (compile at least twice):

```
\AtEndDocument{\ifnum\value{lastsheet}=1\thispagestyle{empty}\fi}
```

SNIPLET C.3 (A KIND OF DRAFT NOTE)

Bastiaan Veelo has kindly provided example code for another form of a side note, as follows.

```
%% A new command that allows you to note down ideas or annotations in
%% the margin of the draft. If you are printing on a stock that is wider
%% than the final page width, we will go to some length to utilise the
%% paper that would otherwise be trimmed away, assuming you will not be
%% trimming the draft. These notes will not be printed when we are not
%% in draft mode.
\makeatletter
\ifdraftdoc
  \newlength{\draftnotewidth}
  \newlength{\draftnotesignwidth}
  \newcommand{\draftnote}[1]{\@bsphack%
    {%% do not interfere with settings for other marginal notes
      \strictpagecheck%
      \checkoddpage%
      \setlength{\draftnotewidth}{\foremargin}%
      \addtolength{\draftnotewidth}{\trimedge}%
      \addtolength{\draftnotewidth}{-3\marginparsep}%
      \ifoddpage
        \setlength{\marginparwidth}{\draftnotewidth}%
        \marginpar{\flushleft\textbf{\textit{\HUGE\ !\ }}\small #1}%
      \else
        \settowidth{\draftnotesignwidth}{\textbf{\textit{\HUGE\ !}}}%
        \addtolength{\draftnotewidth}{-\draftnotesignwidth}%
        \marginpar{\raggedleft\makebox[0pt][r]{%% hack around
          \parbox[t]{\draftnotewidth}{%%funny behaviour
            \raggedleft\small\hspace{0pt}\#1%
          }}\textbf{\textit{\HUGE\ !}}}%
        }%
      \fi
    }\@esphack}
```

```

\else
  \newcommand{\draftnote}[1]{\@bsphack\@esphack}
\fi
\makeatother

```

Bastiaan also noted that it provided an example of using the `\foremargin` length. If you want to try it out, either put the code in your preamble, or put it into a package (i.e., `.sty` file) without the `\makeat . . .` commands.

SNIPLET C.4 (ADDING INDENTATION TO FOOTNOTES)

At times a document design calls for a footnote configuration equal to the default but everything indented more to the right. This can be achieved via

```

\newlength\myextrafootnoteindent
\setlength\myextrafootnoteindent{\parindent}
\renewcommand\makefootmarkhook{%
  \addtolength{\leftskip}{\myextrafootnoteindent}}

```

In this example we indent the footnotes to match the overall paragraph indentation. We need to save the current value of `\parindent` since it is reset in the footnotes.

SNIPLET C.5 (BACKGROUND IMAGE AND TRIMMARKS)

This snippet comes from another problem described in CTT. If one use the `eso-pic` package to add a background image, this image ends up on top of the trim marks. To get it *under* the trim marks Rolf Niepraschk suggested the following trick

```

\RequirePackage{atbegshi}\AtBeginShipoutInit
\documentclass[... ,showtrims]{memoir}
...
\usepackage{eso-pic}
...

```

SNIPLET C.6 (AUTOADJUSTED NUMBER WIDTHS IN THE TOC)

When the ToC is typeset the chapter, section etc. number is typeset within a box of a certain fixed width (one width for each sectional type). If this width is too small for the current document, the user have to manually adjust this width.

In this snippet we present a method where we automatically record the widest.

It a later memoir version, we may add similar code to the core.

There are two ways to record the widest entries of the various types, either preprocess the entire ToC or measure it as a part of the ToC typesetting, store it in the `.aux` and reuse it on the next run. We will use the later approach. There is one caveat: The `.aux` file is read at `\begin{document}`, so we need to postpone our adjustments via `\AtBeginDocument`.

The following solution use some ToC related hooks within the class, plus the `etoolbox` and `calc` packages.

First we create some macros to store information within the .aux file, and retrieve it again.

```
\makeatletter
\newcommand\mem@auxrestore[2]{\csgdef{stored@value@#1}{#2}}
\newcommand\memstorevalue[2]{%
  \@bsphack%
  \immediate\write\@mainaux{\string\mem@auxrestore{#1}{#2}}%
  \@esphack}
\newcommand\RetrieveStoredLength[1]{%
  \ifcsdef{stored@value@#1}{\csuse{stored@value@#1}}{Opt}}%
\makeatletter
```

Here `\RetrieveStoredLength` can be used in most `\setlength` cases, at least when the `calc` package is loaded. The argument will be the name of the variable one asked to be stored. If no corresponding value has been found for a given name, 0pt is returned.

Next we need to prepare the hooks. In this case we will show how to take care of `\chapter`, `\section` and `\subsection`. `\chapter` is relatively easy:¹

```
\newlength\tmplen      % scratch length
\newlength\widestchapter % guess, they are zero by default
\renewcommand\chapternumberlinehook[1]{%
  \settowidth\tmplen{\hbox{\cftchapterfont#1}}%
  \ifdimgreater\tmplen\widestchapter{%
    \global\widestchapter=\tmplen}{}}
```

We use an alternative syntax to make the `\widestchapter` global.

Handling `\section` and `\subsection` is slightly more tricky, as they both use `numberline`. Instead we rely on the local value of the magic macro `\cftwhatismynname`.

```
\newlength\widestsection
\newlength\widestsubsection
\renewcommand\numberlinehook[1]{%
  % use a loop handler to loop over a list of possible
  % types. \forcsvlist comes from etoolbox
  \forcsvlist{\ToHookListHandler{#1}}{section,subsection,subsubsection,%
    paragraph,subparagraph,figure,table}}
% the actual handler.
\newcommand\ToHookListHandler[2]{%
  \edef\tmpstr{#2}%
  \ifdefstrequal{\cftwhatismynname}{\tmpstr}{%
    \settowidth\tmplen{\hbox{\csuse{cft\cftwhatismynname font}#1}}%
    \ifcslength{widest#2}{% is this length defined?
      \ifdimgreater\tmplen{\csuse{widest#2}}{%
        \global\csuse{widest#2}=\tmplen}{}}{}}{}}
```

¹In some cases you may want to use `{\@chapapp@head\cftbsnum #1\cftasnum}`

Even though the list mention more macros, we only use those we have added corresponding lengths for.

Next we need to store the values at the end of the document

```
\AtBeginDocument{\AtEndDocument{
  \memstorevalue{widestchapter}{\the\widestchapter}
  \memstorevalue{widestsection}{\the\widestsection}
  \memstorevalue{widestsubsection}{\the\widestsubsection}
}}
```

Here is how to get the standard class setup for a three level TOC. We also add a little extra padding to the boxes. Remember that it may take a few compilations before the ToC settles down.

```
\newlength\cftnumpad      % padding
\setlength\cftnumpad{0.5em}
\AtBeginDocument{
  \cftsetindents{chapter}{0pt}{\%
    \RetrieveStoredLength{widestchapter}+\cftnumpad}
  \cftsetindents{section}{\%
    \cftchapterindent+\cftchapternumwidth}{\%
    \RetrieveStoredLength{widestsection}+\cftnumpad}
  \cftsetindents{subsection}{\%
    \cftsectionindent+\cftsectionnumwidth}{\%
    \RetrieveStoredLength{widestsubsection}+\cftnumpad}
}
```

SNIPLET C.7 (USING CLASS TOOLS TO MAKE A CHAPTER TOC)

By using a few hooks, we will be able to create a simple chapter toc. First a few notes:

- (a) In this class, the TOC data can be reused, thus we can load the TOC data as many times as we would like.
- (b) Data in the TOC is stored as arguments the `\contentsline` macro, say (see also Figure ?? on page ??)

```
\contentsline{chapter}{\chapternumberline {1}Test}{3}
```

where the first argument determines which macro is used to process the data. Each of these macros look at the value of the `tocdepth` counter to know whether to typeset or not.

- (c) Using some hooks we can insert local changes to `tocdepth` in order to only typeset the sections from the current chapter.

The idea is to be able to add hooks at key points in the ToC data, and then use these hooks to enable and disable typesetting.

We will need to add hooks just after a chapter line (like the one above), and we will need to be able to insert hooks just before items that mark the end of a chapter, that is the next `\chapter`, `\part`, `\book`, plus a macro like `\appendixpage` which also write to the ToC.

First we define hooks that add hooks into the TOC. We use a counter to make each start and end hook unique. We add *end markers* above the ToC entries for `\chapter`, `\part` and `\book`.

```
\newcounter{tocmarker}
\renewcommand\mempreadchaptertotochook{\cftinserthook{toc}{end-\thetocmarker}}
\renewcommand\mempreadparttotochook    {\cftinserthook{toc}{end-\thetocmarker}}
\renewcommand\mempreadbooktotochook    {\cftinserthook{toc}{end-\thetocmarker}}
\renewcommand\mempreadappagetotochook{\cftinserthook{toc}{end-\thetocmarker}}
% start marker
\renewcommand\mempostaddchaptertotochook{%
  \stepcounter{tocmarker}\cftinserthook{toc}{start-\thetocmarker}}
\let\normalchangetocdepth\changetocdepth % for later
```

The hooks inserted into the TOC file, does nothing by default. You will notice that the line above will now look like:

```
\cftinsert {end-0}
\contentsline{chapter}{\chapternumberline {1}Test}{3}
\cftinsert {start-1}
...
\cftinsert {end-1}
\contentsline{chapter}{\chapternumberline {2}Test}{5}
```

Thus to get a chapter toc command we need to make sure that (1) all entries are disabled, (2) at start-1 we reenale TOC entries, and (3) at end-1 disable TOC entries again. Here is the rest of the code, explained via comments.

```
\makeatletter
\newcommand\chaptertoc{
  % make changes local, remember counters a global
  \begingroup
  % store current value, to be restored later
  \setcounter{@memmarkcntra}{\value{tocdepth}}
  % when ever \settocdepth is used, it adds the new value to the
  % ToC data. This cause problems when we want to disable all
  % entries. Luckily the data is added via a special macro, we we
  % redefine it, remember we stored the original value earlier.
  \let\changetocdepth@gobble
  % disable all entries (using our copy from above)
  \normalchangetocdepth{-10}
  % enable toc data within our block, we go as far as subsubsection
```

```

\cftinsertcode{start-\thetocmarker}{\normalchangetocdepth{3}}
% when the block is done, disable the remaining
\cftinsertcode{end-\thetocmarker}{\normalchangetocdepth{-10}}
% remove the spacing above the toc title
\let\toheadstart\relax
% remove the toc title itself
\let\printtoctitle\@gobble
% remove space below title
\let\aftertoctitle\relax
% reformat TOC entries:
\setlength{\cftsectionindent}{0pt}
\setlength{\cftsubsectionindent}{\cftsectionnumwidth}
\setlength{\cftsubsubsectionindent}{\cftsubsectionindent}
\addtolength{\cftsubsubsectionindent}{\cftsubsectionnumwidth}
\renewcommand\cftsectionfont{\small}
\renewcommand\cftsectionpagefont{\small}
\renewcommand\cftsubsectionfont{\small}
\renewcommand\cftsubsectionpagefont{\small}
\renewcommand\cftsubsubsectionfont{\small}
\renewcommand\cftsubsubsectionpagefont{\small}
% include the actual ToC data
\tableofcontents*
\endgroup
% restore tocdepth
\setcounter{tocdepth}{\value{@memmarkcntra}}
% to indent or not after the chapter toc
\m@mindentafterchapter
% space between chapter toc and text
\par\bigskip
% handles indentation after the macro
\@afterheading}
\makeatother

```

Note that if the `\chapterprecistoc` or `\chapterprecis` has been used then that data is also added to the ToC data, and we will need to locally disable it in the chapter ToC. This can be done by adding

```
\let\precistoc\@gobble
```

to the `\chaptertoc` definition above, just make sure it is added before calling before `\tableofcontents*`.

SNIPLET C.8 (AN APPENDIX TOC)

Here we assume a structure like

```

\tableofcontents*
\chapter

```

```

\chapter
\chapter
\appendix
\appendixpage
\appendixtableofcontents
\chapter
\chapter
\chapter

```

where the first ToC should just show until (and including) `\appendixpage`, and `\appendixtableofcontents` should only list the appendices.

We also assume that no `\settocdepth`'s have been issued after `\appendixpage`.

We only need a single hook after `\appendixpage`.

```

\renewcommand\mempostaddappagetotohook{\cftinserthook{toc}{BREAK}}
\cftinsertcode{BREAK}{\changetocdepth{-10}}
\let\normalchangetocdepth\changetocdepth % needed for later

```

Then the definition of the actual appendix ToC:

```

\makeatletter
\newcommand\appendixtableofcontents{
  \begingroup
  \let\changetocdepth@gobble
  \normalchangetocdepth{-10}
  \cftinsertcode{BREAK}{\normalchangetocdepth{3}}
  \renewcommand\contentsname{Appendices overview}
  \tableofcontents*
  \endgroup
}
\makeatother

```

D

Pictures

There are many freely available LaTeX introductions on CTAN and other places. One thing that these apparently are not covering is the traditional `picture` environment. It can be very handy in many applications, though for more complex drawings the reader might be better off with `TikZ/pgf` or `PSTricks`. For the benefit of the general reader we here provide a lesson in the standard `picture` environment.

Writers comment: There are many extensions to the stock `picture` environment provided by the LaTeX kernel. We have chosen not to deal with them in this chapter but instead concentrate on what you get as is from the kernel. But there are a few handy packages that the reader might want to explore: `picture` (by Heiko Oberdiek) which extends the `\put` syntax to include arbitrary lengths, like 50mm; `pict2e` which is mentioned in [GM⁺07] but just recently was released; `eepic`. All packages are available from CTAN.

This chapter describes how to draw diagrams and pictures using LaTeX. Pictures are drawn in the `picture` environment. You can draw straight lines, arrows and circles; you can also put text into your pictures.

Most often pictures are drawn within a `figure` environment, but they may also be drawn as part of the normal text.

D.1 BASIC PRINCIPLES

The positions of picture elements are specified in terms of a two-dimensional cartesian coordinate system. A *coordinate* is a number, such as 7, -21 or 1.78. In the cartesian coordinate system, a pair of coordinates (i.e., a pair of numbers) specifies a position relative to the position designated as (0,0). This special position is called the *origin*. The first of the coordinate pair gives the value of the horizontal distance from the origin to the position. A positive coordinate is an offset to the right and a negative number is an offset to the left. The first value of a coordinate pair is called the *x coordinate*. The second value of a coordinate pair is called the *y coordinate* and gives the vertical offset from the origin (positive upwards and negative downwards).

`\unitlength`

To draw a picture we also need to specify the units of measurement. By default, LaTeX takes the printer's point (there are 72.27 points to an inch) as the measurement of length. The value of the unit of length measurement within a `picture` environment is actually given by the value of the `\unitlength` length declaration. This can be changed to any length that you like via the `\setlength` command. For example,

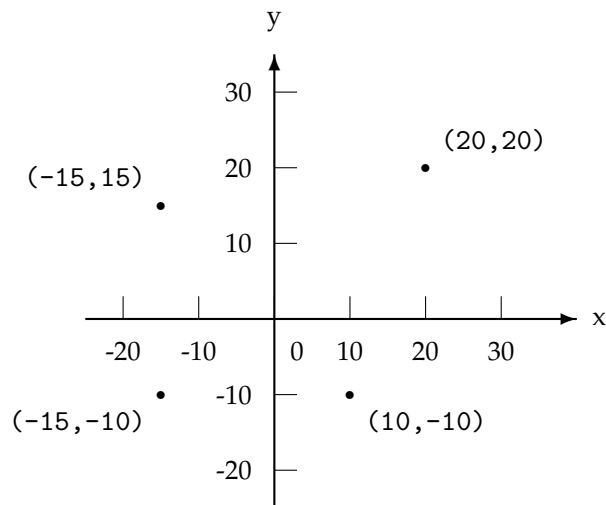


Figure D.1: Some points in the cartesian coordinate system

`\setlength{\unitlength}{2mm}`

will make the value of `\unitlength` to be two millimeters.

Figure D.1 shows the positions of some points and their coordinate values. Coordinate pairs are typed as a pair of numbers, separated by a comma, and enclosed in parentheses.

```
\thinlines
\thicklines
\linethickness{<len>}
```

In general, LaTeX can draw lines of only two thicknesses, thin and thick. The required thickness is specified via either a `\thicklines` or a `\thinlines` declaration, with the latter being the default.

There is another declaration, `\linethickness`, which can be used to change the thickness of horizontal and vertical lines only. It sets the thickness of these lines to `<len>`, but has no effect on any sloping lines.

A `picture` environment has a required size pair argument that specifies the width and height of the picture, in terms of the value of `\unitlength`.

```
\begin{picture}(<width, height>) <contents> \end{picture}
\begin{picture}(<width, height>)(<llx, lly>) <contents> \end{picture}
```

The environment creates a box of size `<width>` by `<height>`, which will not be split across pages. The default position of the origin in this environment is at the lower left hand corner of the box. For example,

```
\begin{picture}(80,160)
```

creates a picture box of width 80 and height 160 whose lower left hand corner is at $(0, 0)$. There is also an optional coordinate pair argument (which comes after the required argu-

ment) that specifies the coordinates of the lower left hand corner of the box if you do not want the default origin.

```
\begin{picture}(80,160)(10,20)
```

specifies a picture box of width 80 and height 160, as before, but with the bottom left hand corner having coordinates of (10,20). Thus, the top right hand corner will have coordinates (90,180). Note that the optional argument is enclosed in parentheses not square brackets as is ordinarily the case. Typically, the optional argument is used when you want to shift everything in the picture. LaTeX uses the required argument to determine the space required for typesetting the result.

You are not limited to drawing within the box, but if you do draw outside the box the result might run into other text on the page, or even run off the page altogether. LaTeX only reserves the space you specify for the picture and does not know if anything protrudes. In particular

```
\begin{picture}(0,0)
```

creates a zero-sized picture which takes no space at all. This can be very useful if you need to position things on the page.

Within the `picture` environment, LaTeX is in a special *picture* mode (which is a restricted form of LR mode). The only commands that can appear in picture mode are `\put`, `\multiput` and `\qbezier` commands, and a few declarations such as the type style and the thickness declarations. By the way, you should only change the value of `\unitlength` outside picture mode otherwise LaTeX will get confused about its measurements.

D.2 PICTURE OBJECTS

In a picture everything is placed and drawn by the `\put` (or its `\multiput` variant) command.

```
\put(<x,y>){<object>}
```

`\put` places `<object>` in the picture with the object's *reference point* at position `(<x,y>)`.

The following sections describe the various picture objects.

D.2.1 Text

Text is the simplest kind of picture object. This is typeset in LR mode and the reference point is at the lower left hand corner of the text.

Source for example [D.1](#)

```
\setlength{\unitlength}{1mm} % measurements in millimeters
\begin{picture}(30,10)       % define size of picture
\put(0,0){\framebox(30,10){}} % draw frame around picture
\put(10,5){Some text}       % place text
\thicklines
\put(10,5){\vector(1,1){0}} % mark reference point
\end{picture}
\setlength{\unitlength}{1pt} % reset measurements to default
```

Typeset example D.1: Picture: text



In the diagram, and those following, the reference point is indicated by an arrow. Also, a box is drawn round the diagram at the same size as the `picture` environment.

D.2.2 Boxes

A box picture object is made with one of the box commands. When used in picture mode, the box commands have a slightly different form than when in normal text. The first argument of a box command is a size pair that specifies the width and height of the box. The last argument is the text to be placed in the box. The reference point of a box is the lower left hand corner.

```
\framebox(<width, height>) [<pos>] {<text>}
\makebox(<width, height>) [<pos>] {<text>}
```

The `\framebox` command draws a framed box of the specified ($\langle width, height \rangle$) dimensions around the text.

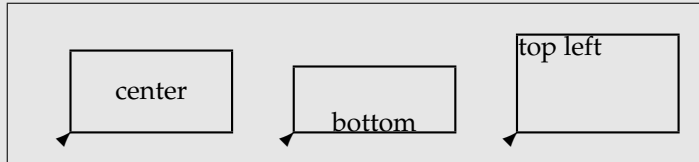
Source for example [D.2](#)

```
\setlength{\unitlength}{1pc}
\begin{picture}(22,5)
\put(0,0){\framebox(22,5){}}           % empty box
\thicklines
\put(2,1){\framebox(5,2.5){center}}    % centered text
\put(2,1){\vector(1,1){0}}             % ref point
\put(9,1){\framebox(5,2)[b]{bottom}}   % bottomed text
\put(9,1){\vector(1,1){0}}             % ref point
\put(16,1){\framebox(5,3)[tl]{top left}} % cornered text
\put(16,1){\vector(1,1){0}}           % ref point
\end{picture}
\setlength{\unitlength}{1pt}
```

The default position of the *text* is centered in the box. However, this can be changed via an optional argument (which is enclosed in square brackets), placed between the coordinate pair and the text argument. This argument consists of either one or two of the following letters.

l (left) Places the contents at the left of the box.

Typeset example D.2: Picture: text in boxes



r (right) Places the contents at the right of the box.

t (top) Places the contents at the top of the box.

b (bottom) Places the contents at the bottom of the box.

These place the text in the corresponding position in the box. In a two-letter argument the order of the letters is immaterial. For example, [tr] and [rt] will both result in the text being placed at the top right hand corner of the box. Unlike the normal `\framebox` command, a `\framebox` in a `picture` environment does not add any extra space around the text.

Corresponding to the `\framebox` there is a `\makebox` command which does not draw a frame around its contents. The `\makebox` command takes the same arguments as the `\framebox`. Particularly interesting is when you specify a zero sized `\makebox`. A `\makebox(0,0){text}` command will make the reference point the center of text. Similarly, the other positioning arguments which will adjust the reference point with respect to the box contents. This can be used for fine-tuning the position of text in a picture.

Source for example [D.3](#)

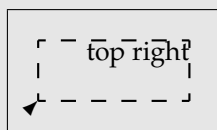
```
\setlength{\unitlength}{1pc}
\begin{picture}(16,2)
\put(0,0){\framebox(16,2){}}
\thicklines
\put(3.5,1){\makebox(0,0){center}}      % ref at text center
\put(3.5,1){\vector(0,-1){0}}
\put(7,1){\makebox(0,0)[b]{bottom}}    % ref at text bottom
\put(7,1){\vector(0,1){0}}
\put(11,1){\makebox(0,0)[tl]{top left}} % ref at text top left
\put(11,1){\vector(1,-1){0}}
\end{picture}
\setlength{\unitlength}{1pt}
```

You can draw a dashed box with the `\dashbox` command.

Typeset example D.3: Picture: positioning text



Typeset example D.4: Picture: dashed box



`\dashbox{<len>}{<width, height>}[<pos>]{<text>}`

The first argument of this command specifies the length of each dash. The following arguments are the same as for the other box commands.

Source for example [D.4](#)

```
\setlength{\unitlength}{4mm}
\begin{picture}(7,4)
\put(0,0){\framebox(7,4){}}
\thicklines
\put(1,1){\dashbox{0.5}(5,2)[tr]{top right}}
\put(1,1){\vector(1,1){0}}
\end{picture}
\setlength{\unitlength}{1pt}
```

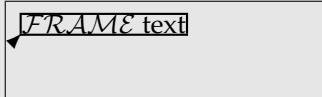
The appearance of the box is best when the width and height of the box are integer multiples of the dash length. In the example the dash length has been set to 0.5 with the width and height set as (5,2); thus the width and height are respectively ten and four times the dash length.

The `\frame` command draws a frame around the contents of the box that exactly fits the contents.

`\frame{<contents>}`

It takes a single required argument which is the contents.

Typeset example D.5: Picture: framing

Source for example [D.5](#)

```
\setlength{\unitlength}{1pc}
\begin{picture}(10,3)
\put(0,0){\framebox(10,3){}}
\thicklines
\put(0.5,2){\frame{$\mathcal{FRAME}$ text}}
\put(0.5,2){\vector(1,1){0}}
\end{picture}
\setlength{\unitlength}{1pt}
```

The `\shortstack` command enables you to stack text vertically. It produces a box with a single column of text. As with the other boxes, the reference point is at the lower left hand corner, although no frame is drawn around the stack. The `\shortstack` command is an ordinary box making command, but it is not often used outside picture mode.

```
\shortstack[⟨pos⟩]{⟨text⟩}
```

Each line of `⟨text⟩`, except for the last, is ended by a `\\` command. The default is to center each text line within the column. However, there is an optional positioning argument. A value of `l` for `⟨pos⟩` will left align the text and a value of `r` will right align the text.

Source for example [D.6](#)

```
\setlength{\unitlength}{1mm}
\begin{picture}(75,25)
\put(0,0){\framebox(75,25){}}
\put(3,3){\shortstack{Default \\ short \\ Stack}}
\put(3,3){\vector(1,1){0}}
\put(23,3){\shortstack[l]{Left\\aligned\\short\\Stack}}
\put(23,3){\vector(1,1){0}}
\put(43,3){\shortstack[r]{Right\\aligned\\short\\Stack}}
\put(43,3){\vector(1,1){0}}
```

Typeset example D.6: Picture: stacking

			Extra
Default	Left	Right	
short	aligned	aligned	spaced
Stack	short	short	
✓ Stack	✓ Stack	✓ Stack	✓ Stack

```
\put(63,3){\shortstack{Extra \\\[4ex] spaced \\\[2ex] Stack}}
\put(63,3){\vector(1,1){0}}
\end{picture}
\setlength{\unitlength}{1pt}
```

The rows in a stack are not evenly spaced. The spacing between two rows can be changed in one of two ways.

1. Add a strut to a row. A strut is a vertical rule with no width.
2. Use the optional argument to the `\\` command. This optional argument is a length value.

```
\\[len]
```

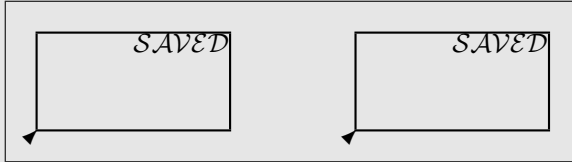
It has the effect of adding additional *len* vertical space between the two lines that the `\\` separates.

```
\newsavebox{<box>}
\savebox{<box>}( <width, height> ) [ <pos> ] {<text>}
\sbox{<box>}{<text>}
\usebox{<box>}
```

Just as in normal text you can save and reuse boxes. The `\savebox` macro in picture mode is a variant of the normal text version, but the other three commands are the same in both picture and paragraph modes, and are described in Chapter 4. In picture mode you have to specify the size of the storage box when saving it, via the *(width, height)* argument to `\savebox`.

A `\savebox` command can be used within a picture to store a picture object. The first argument of `\savebox` is the name of the storage bin to be used. The following arguments are the same as the `\makebox` command. The result is stored, not drawn. When you have saved something it can be drawn in either the same or other pictures via the `\usebox` command. This command takes one argument, which is the name of the storage bin.

Typeset example D.7: Picture: saved boxes

Source for example [D.7](#)

```

\setlength{\unitlength}{1pc}
\begin{picture}(18,5)
\put(0,0){\framebox(18,5){}}
\newsavebox{\Mybox}
\savebox{\Mybox}(6,3)[tr]{\mathcal{SAVED}}
\thicklines
\put(1,1){\frame{\usebox{\Mybox}}}
\put(11,1){\frame{\usebox{\Mybox}}}
\put(1,1){\vector(1,1){0}}
\put(11,1){\vector(1,1){0}}
\end{picture}
\setlength{\unitlength}{1pt}

```

It can take LaTeX a long time to draw something. When a box is saved it actually contains the typeset contents, which then just get printed out when the box is used. It can save processing time if something which appears several times is saved and then used as and where required. On the other hand, a saved box can use up a significant amount of LaTeX's internal storage space. The `\sbox` command with an empty text argument can be used to delete the contents of a bin. For example,

```
\sbox{\Mybox}{}
```

will empty the `\Mybox` box. Note that this does not delete the storage box itself.

D.2.3 Lines

LaTeX can draw straight lines, but the range of slopes for lines is somewhat restricted. Further, very short lines cannot be drawn.

```
\line(<i,j>){<distance>}
```

The pair $\langle i, j \rangle$ specifies the *slope* of the line, and $\langle distance \rangle$ is a value that controls the length of the line. The line starts at its reference point (i.e., the place where it is `\put`). The

slope of the line is such that if a point on the line is slid along the line, then for every i units the point moves in the horizontal direction it will also have moved j units in the vertical direction. Negative values for i or j have the expected meaning. A move of -3 units in i means a move of 3 units to the left, and similarly a move of -4 units in j means a move of 4 units downwards. So, a line sloping up to the right will have positive values for i and j , while a line sloping up to the left will have a negative value for i and a positive value for j .

The `<distance>` argument specifies the length of the line in the x (horizontal) direction. One problem with this may have occurred to you: what if the line is vertical (i.e., $i = 0$)? In this case only, `<distance>` specifies the vertical length of the line. The `<distance>` argument must be a non-negative value. For horizontal and vertical lines only, the actual length of the line is `<distance>`. Figure D.2, which is produced from the code below, diagrams the line specification arguments.

```
\begin{figure}
\centering
\setlength{\unitlength}{1mm}
\begin{picture}(70,60)
\thicklines    % draw line and ref point
\put(10,20){\line(2,1){40}}
\put(10,20){\vector(1,-1){0}}
\thinlines     % draw axes
\put(0,0){\vector(1,0){60}} \put(63,0){x}
\put(0,0){\vector(0,1){50}} \put(0,53){y}
                % draw i and j vectors
\put(20,25){\vector(1,0){20}}
\put(30,22){\makebox(0,0)[t]{$i$}}
\put(40,25){\vector(0,1){10}}
\put(42,30){\makebox(0,0)[l]{$j$}}
                % draw distance vector
\put(30,10){\vector(-1,0){20}}
\put(30,10){\vector(1,0){20}}
\put(30,8){\makebox(0,0)[t]{\textit{distance}}}
\end{picture}
\setlength{\unitlength}{1pt}
\caption{Specification of a line or arrow}
\label{flpic:spec}
\end{figure}
```

Only a fixed number of slopes are available. This is because LaTeX uses a special font for drawing lines — a line actually consists of little bits of angled rules joined together. Thus, there is only a limited number of values for i and j . They must both be integers and in the range $-6 \leq i, j \leq 6$. Also, they must have no common divisor other than 1. In other words, the ratio between i and j must be in its simplest form. You cannot, for example, have (3, 6); instead it would have to be (1, 2). The shortest line that LaTeX can draw is about ten points (1/7 inch approximately) in overall length. You can, though, draw lines that are too long to fit on the page.

Figure D.3 shows the lines and arrows slanting upwards and to the right that can be

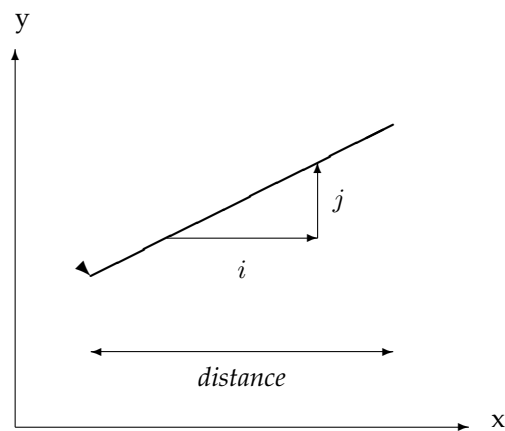


Figure D.2: Specification of a line or arrow

drawn in LaTeX. The slope (i, j) pair are shown to the right of the first set of lines and arrows, together with the j/i ratio which gives the slope of the line as a decimal number.

D.2.4 Arrows

As shown in Figure D.3 you can also draw a line with an arrowhead on it. These are specified by the `\vector` command.

```
\vector(<i,j>){<distance>}
```

This works exactly like the `\line` command and the arrowhead is put on the line at the end away from the reference point. That is, the arrow points away from the reference point. If the `<distance>` argument is too small (zero, for instance) the arrowhead only is drawn, with its point at the position where it is `\put`.

LaTeX is even more restrictive in the number of slopes that it can draw with arrows than it is with lines. The (i, j) slope specification pair must lie in the range $-4 \leq i, j \leq 4$. Also, as with the `\line` command, they must have no common divisor.

D.2.5 Circles

LaTeX can draw two kinds of circles. One is an open circle where only the perimeter is drawn, and the other is a solidly filled disk.

```
\circle{<diameter>}
\circle*{<diameter>}
```

The reference point for the open circle, drawn by the `\circle` command, and the disk, which is drawn by the `\circle*` command, is at the center of the circle. The argument to the commands is the `<diameter>` of the circle.

Source for example D.8

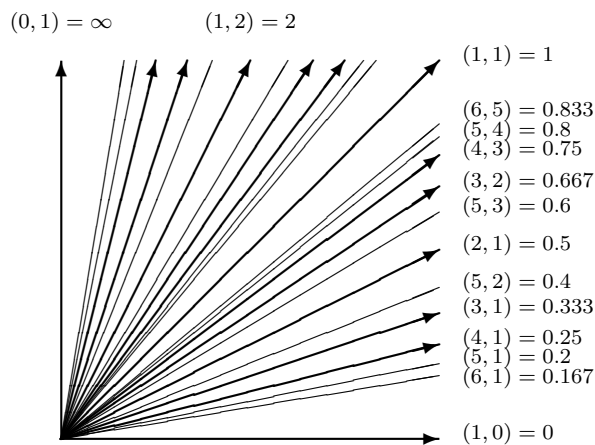
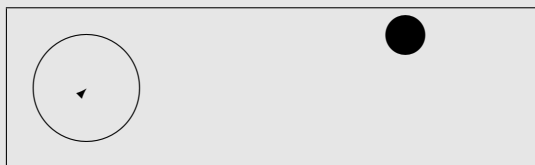


Figure D.3: Sloping lines and arrows

Typeset example D.8: Picture: circles



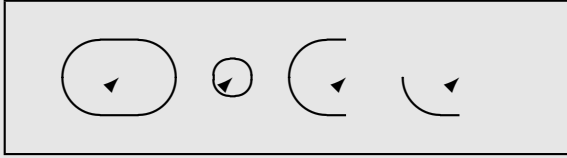
```

\setlength{\unitlength}{1pt}
\begin{picture}(200,60)
\put(0,0){\framebox(200,60){}}
\put(30,30){\circle{40}}
\put(30,30){\vector(1,1){0}}
\put(150,50){\circle*{20}}
\end{picture}
\setlength{\unitlength}{1pt}

```

Just as with the `\line` and `\vector` commands, there is only a limited range of circles that can be drawn. Typically, the maximum diameter of a `\circle` is about 40 points, while for a `\circle*` the maximum diameter is less, being about 15 points. LaTeX will choose

Typeset example D.9: Picture: ovals



the nearest sized circle to the one that you specify. Either consult your local guru to find what sized circles you can draw on your system, or try some experiments by drawing a range of circles to see what happens.

Quarter circles and boxes

In LaTeX an `\oval` is a rectangular box with rounded corners.

`\oval(<width,height>)[<portion>]`

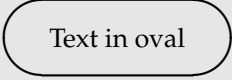
The `\oval` command has one required argument which specifies the width and height of the box. The normally sharp corners of the box are replaced by quarter circles of the maximum possible radius (which LaTeX figures out for itself). Unlike the boxes discussed earlier, the reference point is at the ‘center’ of the oval.

Source for example D.9

```
\setlength{\unitlength}{1mm}
\begin{picture}(75,20)
\thicklines
\put(0,0){\framebox(75,20){}}
\put(15,10){\oval(15,10)}      % complete oval
\put(15,10){\vector(1,1){0}}
\put(30,10){\oval(5,5)}      % small oval
\put(30,10){\vector(1,1){0}}
\put(45,10){\oval(15,10)[l]} % left half
\put(45,10){\vector(1,1){0}}
\put(60,10){\oval(15,10)[bl]} % bottom left quarter
\put(60,10){\vector(1,1){0}}
\end{picture}
\setlength{\unitlength}{1pt}
```

The `\oval` command also has one optional argument, *<portion>*, which comes after the required argument. Use of the optional argument enables either half or a quarter of the

Typeset example D.10: Picture: text in oval



complete rounded box to be drawn. The argument is a one or two letter code drawn from the following.

- l (left) Draw the left of the oval.
- r (right) Draw the right of the oval.
- t (top) Draw the top of the oval.
- b (bottom) Draw the bottom of the oval.

These are similar to the optional positioning argument in the box commands. A one letter code will draw the designated half of the oval, while a two letter code results in the designated quarter of the oval being drawn. In all cases the reference point is at the center of the ‘complete’ oval.

Source for example [D.10](#)

```
\setlength{\unitlength}{1mm}
\begin{picture}(30,10)
\thicklines
\put(15,5){\oval(30,10)}
\put(15,5){\makebox(0,0){Text in oval}}
\end{picture}
\setlength{\unitlength}{1pt}
```

Unlike the boxes described in §[D.2.2](#) there is no *<text>* argument for an `\oval`. If you want the rounded box to contain text, then you have to place the text inside the box yourself. The code in example [D.10](#) shows one way of doing this; a zero-sized box is used to center the text at the center of the oval.

D.3 REPETITIONS

The `\multiput` command is a convenient way to place regularly spaced copies of an object in a picture.

`\multiput(<x,y>)(<dx,dy>){<num>}{<object>}`

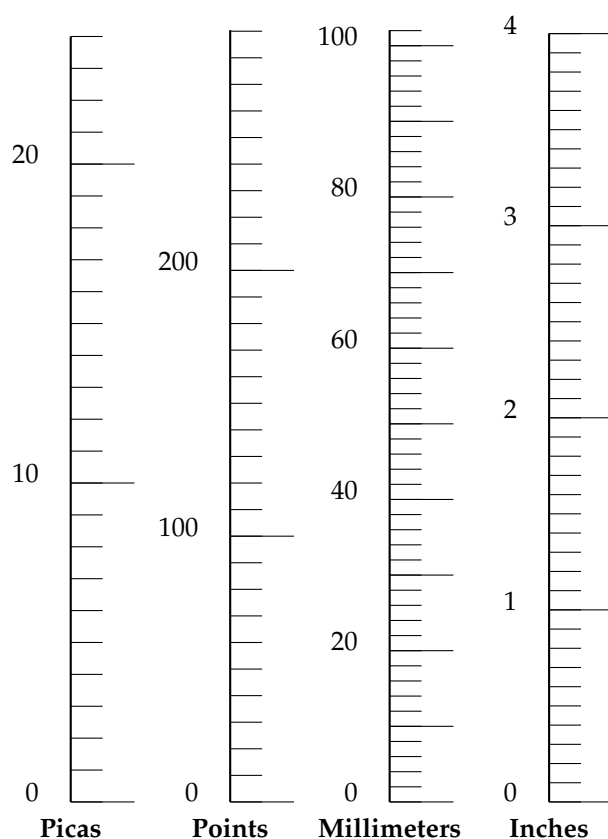


Figure D.4: Some measuring scales

As you can see, this is similar to the syntax for the `\put` command, except that there are two more required arguments, namely $\langle dx, dy \rangle$ and num .

The $\langle dx, dy \rangle$ argument is a pair of (decimal) numbers that specify the amount that the $\langle object \rangle$ shall be moved at each repetition. The first of this pair specifies the horizontal movement and the second the vertical movement. Positive values shift to the right or up, and negative numbers shift to the left or down. The $\langle num \rangle$ argument specifies how many times the $\langle object \rangle$ is to be drawn.

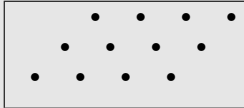
The code below produces Figure D.4. This example also shows that a `picture` can be placed inside another `picture`. Often it is useful to break a complex diagram up into pieces, with each piece being a separate `picture`. The pieces can then be individually positioned within the overall diagram.

```
\begin{figure}
\setlength{\unitlength}{1pc}
\centering
\begin{picture}(21,26)
% Draw Pica scale
```

```
\put(2,2){\begin{picture}(5,24)
  \put(0,-0.5){\makebox(0,0)[t]{\textbf{Picas}}}
  \thicklines \put(0,0){\line(0,1){24.0}}
  \thinlines \multiput(0,0)(0,1){25}{\line(1,0){1}}
             \multiput(0,0)(0,10){3}{\line(1,0){2}}
  \put(-1,0){\makebox(0,0)[br]{0}}
  \put(-1,10){\makebox(0,0)[br]{10}}
  \put(-1,20){\makebox(0,0)[br]{20}}
\end{picture}}
% Draw Points scale
\put(7,2){\begin{picture}(5,24)
  \put(0,-0.5){\makebox(0,0)[t]{\textbf{Points}}}
  \thicklines \put(0,0){\line(0,1){24.2}}
  \thinlines \multiput(0,0)(0,0.8333){30}{\line(1,0){1}}
             \multiput(0,0)(0,8.333){3}{\line(1,0){2}}
  \put(-1,0){\makebox(0,0)[br]{0}}
  \put(-1,8.333){\makebox(0,0)[br]{100}}
  \put(-1,16.667){\makebox(0,0)[br]{200}}
\end{picture}}
% Draw Millimeter scale
\put(12,2){\begin{picture}(5,24)
  \put(0,-0.5){\makebox(0,0)[t]{\textbf{Millimeters}}}
  \thicklines \put(0,0){\line(0,1){24.2}}
  \thinlines \multiput(0,0)(0,0.4742){15}{\line(1,0){1}}
             \multiput(0,0)(0,2.3711){11}{\line(1,0){2}}
  \put(-1,0){\makebox(0,0)[br]{0}}
  \put(-1,4.742){\makebox(0,0)[br]{20}}
  \put(-1,9.484){\makebox(0,0)[br]{40}}
  \put(-1,14.226){\makebox(0,0)[br]{60}}
  \put(-1,18.968){\makebox(0,0)[br]{80}}
  \put(-1,23.71){\makebox(0,0)[br]{100}}
\end{picture}}
% Draw Inch scale
\put(17,2){\begin{picture}(5,24)
  \put(0,-0.5){\makebox(0,0)[t]{\textbf{Inches}}}
  \thicklines \put(0,0){\line(0,1){24.1}}
  \thinlines \multiput(0,0)(0,0.60225){41}{\line(1,0){1}}
             \multiput(0,0)(0,6.0225){5}{\line(1,0){2}}
  \put(-1,0){\makebox(0,0)[br]{0}}
  \put(-1,6.0225){\makebox(0,0)[br]{1}}
  \put(-1,12.045){\makebox(0,0)[br]{2}}
  \put(-1,18.0675){\makebox(0,0)[br]{3}}
  \put(-1,24.09){\makebox(0,0)[br]{4}}
\end{picture}}

\end{picture}
\setlength{\unitlength}{1pt}
```

Typeset example D.11: Picture: repetitions



```
\caption{Some measuring scales} \label{flpic:scales}
\end{figure}
```

You can also make regular two-dimensional patterns by using a `\multiput` pattern inside another `\multiput`. As LaTeX will process each `\multiput` every time it is repeated it is often more convenient to store the results of the first `\multiput` in a bin and then use this as the argument to the second `\multiput`.

Source for example D.11

```
\setlength{\unitlength}{1mm}
\begin{picture}(32,14)
\put(0,0){\framebox(32,14){}}
\savebox{\Mybox}(8,8){\multiput(0,0)(4,4){3}{\circle*{1}}}
\multiput(4,4)(6,0){4}{\usebox{\Mybox}}
\sbox{\Mybox}{}
\end{picture}
\setlength{\unitlength}{1pt}
```

Remember that a storage bin must have been declared via a `\newsavebox` command before it can be used. I originally declared and used the `\Mybox` bin in §D.2.2. As the above example shows, you can change the contents of a storage bin by utilising it in another `\savebox`. Storage bins can use up a lot of LaTeX's memory. After you have finished with a storage bin empty it via the `\sbox` command with an empty last argument, as shown in the example.

D.4 BEZIER CURVES

Standard LaTeX provides one further drawing command — the `\qbezier` command. This can be used for drawing fairly arbitrary curves.

```
\qbezier[⟨num⟩](⟨Xs, Ys⟩)(⟨Xm, Ym⟩)(⟨Xe, Ye⟩)
```

The command will draw what geometers call a *quadratic Bezier curve* from the point $(\langle Xs, Ys \rangle)$ to the point $(\langle Xe, Ye \rangle)$. The curve will pass somewhere near to the point $(\langle Xm, Ym \rangle)$.

Bezier curves are named after Pierre Bezier who first used them in 1962. They are widely used in Computer Aided Design (CAD) programs and other graphics and font

design systems. Descriptions, with varying degrees of mathematical complexity, can be found in many places: when I was a practicing geometer these included [FP80], [Mor85] and [Far90]; no doubt there are more recent sources available and there is a brief review in [Wil04a].

Figure D.5 shows two of these curves. The figure was produced by the code below.

```
\begin{figure}
\setlength{\unitlength}{1mm}
\centering
\begin{picture}(100,100)

\thicklines % first curve
\qbezier(10,50)(50,90)(50,50)
\thinlines % draw lines joining control points
\put(10,50){\line(1,1){40}}
\put(50,90){\line(0,-1){40}}
% label control points
\put(10,45){\makebox(0,0)[t]{\texttt{(10,50)}}}
\put(50,95){\makebox(0,0)[b]{\texttt{(50,90)}}}
\put(55,50){\makebox(0,0)[l]{\texttt{(50,50)}}}

\thicklines % second curve
\qbezier[25](50,50)(50,10)(90,50)
\thinlines % draw lines joining control points
% \put(50,50){\line(0,-1){40}}
% \put(50,10){\line(1,1){40}}
% label control points
\put(50,5){\makebox(0,0)[t]{\texttt{(50,10)}}}
\put(90,55){\makebox(0,0)[b]{\texttt{(90,50)}}}

\end{picture}
\setlength{\unitlength}{1pt}
\caption{Two Bezier curves}
\label{picf:bez}
\end{figure}
```

The three points used to specify the position and shape of the Bezier curve are called *control points*. The curve starts at the first control point and is tangent to the line joining the first and second control points. The curve stops at the last control point and is tangent to the line joining the last two control points.

In Figure D.5 the lines joining the control points for the first curve have been drawn in. The locations of all the control points for the two curves are labeled.

The second Bezier curve is the same shape as the first one, but rotated 180 degrees. The first control point of this curve is the same as the last control point of the first curve. This means that the two curves are joined at this point. The line, although it is not drawn, connecting the first two control points of the second curve is in the same direction as the line joining the last two control points of the first curve. This means that the two curves are also tangent at the point where they join. By stringing together several Bezier curves

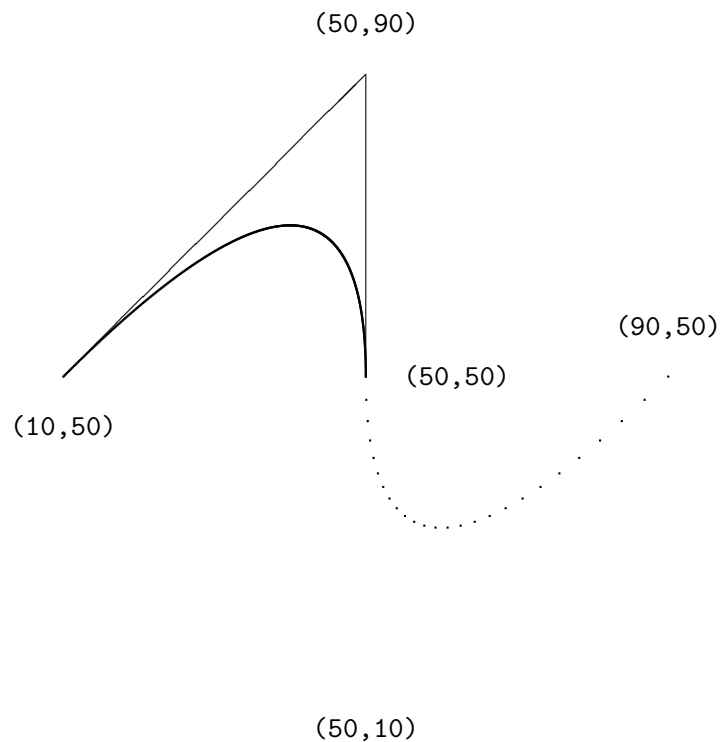


Figure D.5: Two Bezier curves

you can draw quite complex curved shapes.

`\qbeziermax`

The Bezier curves are actually drawn as a linearized form using a series of rectangular blobs of ink. Left to itself, LaTeX will attempt to pick the number of blobs to give the smoothest looking curve, up to a maximum number. (Each blob takes up space in LaTeX's internal memory, and it may run out of space if too many are used in one picture.) The maximum number of blobs per Bezier curve is set by the `\qbeziermax` command. This can be adjusted with the `\renewcommand` command. For example:

```
\renewcommand{\qbeziermax}{250}
```

will set the maximum number of blobs to be 250.

Another method of controlling the number of blobs is by the optional *<num>* argument to the `\qbezier` command. If used, it must be a positive integer number which tells LaTeX exactly how many blobs to use for the curve.

E

LaTeX and TeX

Strictly speaking, LaTeX is a set of macros built on top of the TeX program originally developed by Donald Knuth [Knu86, Knu84] in the early 1980's. TeX is undoubtedly one of the most robust computer programs to date.

Leslie Lamport says that most TeX commands can be used with LaTeX and lists those that cannot be used [Lam94, Appendix E]. Apart from this he says nothing about any TeX commands. I have used some TeX macros in the code examples and so I need to talk a little bit about these.

I like to think of the commands and macros as falling into one of several groups.

- TeX primitives. These are the basic constructs of the TeX language.
- TeX commands or macros. These are part of the plain TeX system and are constructed from the TeX primitives.
- LaTeX kernel commands or macros. These are defined in the LaTeX kernel and are based on plain TeX primitives or commands. In turn, some higher level kernel macros are constructed from more basic aspects of the kernel. The kernel does redefine some of the plain TeX commands.
- Class command. These are mainly built up on the kernel commands but may use some basic TeX.
- Package commands. These are similar to the class commands but are less likely to directly use TeX macros.
- User commands. Typically these are limited to the commands provided by the class and any packages that might be called for, but more experienced users will employ some kernel commands, like `\newcommand`, to make their authoring more efficient.

Although TeX is designed as a language for typesetting it is also a 'Turing complete' language which means that it can perform any function that can be programmed in any familiar programming language. For example, an interpreter for the BASIC language has been written in TeX, but writing this kind of program using TeX is something that only an expert¹ might consider.

Nevertheless, you may have to, or wish to, write a typesetting function of your own. This chapter presents a few of the programming aspects that you may need for this, such

¹Probably also a masochist with plenty of time.

as performing some simple arithmetic or comparing two lengths. For anything else you will have to read one or more of the TeX books or tutorials.

In England witnesses at a trial have to swear to ‘Tell the truth, the whole truth, and nothing but the truth’. I will try and tell the truth about TeX but, to misquote Hamlet

There are more things in heaven and TeX, Horatio,
Than are dreamt of in your philosophy.

E.1 THE T_EX PROCESS

As we are delving deeper than normal and because at the bottom it is the TeX language that does all the work, it is useful to have an idea of how TeX processes a source file to produce a dvi file. It is all explained in detail by Knuth [Knu84] and also perhaps more accessibly by Eijkhout [Eij92]; the following is a simplified description. Basically there are four processes involved and the output from one process is the input to the following one.

Input The input process, which Knuth terms the ‘eyes’, reads the source file and converts what it sees into *tokens*. There are essentially two kinds of token. A token is either a single character such as a letter or a digit or a punctuation mark, or a token is a control sequence. A *control sequence* consists of a backslash and either all the alphabetic characters immediately following it, or the single non-alphabetic following it. Control sequence is the general term for what I have been calling a macro or a command.

Expansion The expansion processor is what Knuth calls ‘TeX’s mouth’. In this process some of the tokens from the input processor are expanded. Expansion replaces a token by other tokens or sometimes by no token. The expandible tokens include macros, conditionals, and a number of TeX primitives.

Execution The execution process is TeX’s ‘stomach’. This handles all the tokens output by the expansion processor. Control sequences that are not expandible are termed *executable*, and the execution processor executes the executable tokens. Character tokens are neither expandible nor executable. It handles any macro definitions and also builds horizontal, vertical and mathematical lists.

Layout The layout processor (TeX’s ‘bowels’) breaks horizontal lists into paragraphs, mathematical lists into formulae, and vertical lists into pages. The final output is the dvi file.

In spite of the sequential nature implied by this description the overall process includes some feedback from a later process to an earlier one which may affect what that does.

It is probably the expansion processor that needs to be best understood. Its input is a sequence of tokens from the input processor and its output is a sequence of different tokens.

In outline, the expansion processor takes each input token in turn and sees if it is expandible; if it is not it simply passes it on to the output. If the token is expandible then it is replaced by its expansion. The most common expandible tokens are control sequences that have been defined as macros. If the macro takes no arguments then the macro’s name is replaced by its definition. If the macro takes arguments, sufficient tokens are collected

to get the values of the arguments, and then the macro name is replaced by the definition. The expansion processor then looks at the first token in the replacement, and if that is expandible it expands that, and so on.

Nominally, the eventual output from the expansion processor is a stream of non-expandible tokens. There are ways, however of controlling whether or not the expansion processor will actually expand an expandible token, and to control the order in which things get expanded, but that is where things get rapidly complicated.

The layout processor works something like this. Ignoring maths, TeX stores what you type in two kinds of lists, vertical and horizontal. As it reads your words it puts them one after another in a horizontal list. At the end of a paragraph it stops the horizontal list and adds it to the vertical list. At the beginning of the next paragraph it starts a new horizontal list and adds the paragraph's words to it. And so on. This results in a vertical list of horizontal lists of words, where each horizontal list contains the words of a paragraph.

It then goes through each horizontal list in turn, breaking it up into shorter horizontal lists, one for each line in the paragraph. These are put into another vertical list, so conceptually there is a vertical list of paragraphs, and each paragraph is a vertical list of lines, and each line is a horizontal list of words, or alternatively one vertical list of lines. Lastly it chops up the vertical list of lines into page sized chunks and outputs them a page at a time.

TeX is designed to handle arbitrary sized inserts, like those for maths, tables, sectional divisions and so forth, in an elegant manner. It does this by allowing vertical spaces on a page to stretch and shrink a little so that the actual height of the typeblock is constant. If a page consists only of text with no widow or orphan then the vertical spacing is regular, otherwise it is likely to vary to some extent. Generally speaking, TeX is not designed to typeset on a fixed grid, but against this other systems are not designed to produce high quality typeset mathematics. Attempts have been made to tweak LaTeX to typeset on a fixed grid but as far as I know nobody has been completely successful.

TeX works somewhat more efficiently than I have described. Instead of reading the whole document before breaking paragraphs into lines, it does the line breaking at the end of each paragraph. After each paragraph it checks to see if it has enough material for a page, and outputs a page whenever it is full. However, TeX is also a bit lazy. Once it has broken a paragraph into lines it never looks at the paragraph again, except perhaps to split it at a page break. If you want to change, say, the width of the typeblock on a particular page, any paragraph that spills over from a previous page will not be reset to match the new measure. This asynchronous page breaking also has an unfortunate effect if you are trying to put a note in say, the outside margin, as the outside is unknown until after the paragraph has been set, and so the note may end up in the wrong margin.

E.2 L^AT_EX FILES

The aux file is the way LaTeX transfers information from one run to the next and the process works roughly like this.

- The aux file is read at the start of the document environment. If `\nofiles` has not been specified a new empty aux file is then created which has the side effect of destroying the original aux file.

- Within the document environment there may be macros that write information to the aux file, such as the sectioning or captioning commands. However, these macros will not write their information if `\nofiles` has been specified.
- At the end of the document environment the contents of the aux file are read.

Under normal circumstances new output files are produced each time L^AT_EX is run, but when `\nofiles` is specified only the dvi and log files will be new — any other files are unchanged.

In the case of the sectioning commands these write macros into the aux file that in turn write information into a toc file, and the `\tableofcontents` command reads the toc file which contains the information for the Table of Contents. To make this a bit more concrete, as L^AT_EX processes a new document through the first two runs, the following events occur.

1. Initially there is neither an aux nor a toc file. At the start of the document environment a new empty aux file is created.
2. During the first run the `\tableofcontents` typesets the Contents heading and creates a new empty toc file.

During the run sectional commands write information into the new aux file. At the end of the document environment the aux file is read. Contents information in the aux file is written to the toc file. Lastly all the output files are closed.

3. For the second run the aux file from the previous run is read at the start of the document environment; no information can be written to a toc file because the toc file is only made available by the `\tableofcontents` command. The aux file from the previous run is closed and the new one for this run is created.

This time the `\tableofcontents` reads toc file that was created during the previous run which contains the typesetting instructions for the contents, and then starts a new toc file.

And so the process repeats itself.

The aux file mechanism means that, except for the simplest of documents, L^AT_EX has to be run at least twice in order to have all the information to hand for typesetting. If sections are added or deleted, two runs are necessary afterwards to ensure that everything is up to date. Sometimes three, or even more, runs are necessary to guarantee that things are settled.

E.3 SYNTAX

The L^AT_EX syntax that you normally see is pretty regular. Mandatory arguments are enclosed in curly braces and optional arguments are enclosed in square brackets. One exception to this rule is in the `picture` environment where coordinate and direction pairs are enclosed in parentheses.

The T_EX syntax is not regular in the above sense. For example, if in L^AT_EX you said

```
\newcommand*{\cmd}[2]{#1 is no. #2 of}
\cmd{M}{13} the alphabet. % prints: M is no. 13 of the alphabet
```

Then in TeX you would say

```
\def\cmd#1#2{#1 is no. #2 of}
```

and you could then use either of the following calls:

```
\cmd M{13} the alphabet. % prints: M is no. 13 of the alphabet
\cmd{M}{13} the alphabet. % prints: M is no. 13 of the alphabet
```

A simplistic explanation of the first TeX call of `\cmd` is as follows. A control sequence starts with a backslash, followed by either a single character, or one or more of what TeX thinks of as letters (normally the 52 lower- and upper-case alphabetic characters); a space or any non-letter, therefore, ends a multiletter control sequence. TeX and LaTeX discard any spaces after a macro name. If the macro takes any arguments, and `\cmd` takes two, TeX will then start looking for the first argument. An argument is either something enclosed in braces or a single token. In the example the first token is the character ‘M’, so that is the value of the first argument. TeX then looks for the second argument, which is the ‘13’ enclosed in the braces. In the second example, both arguments are enclosed in braces.

Here are some TeX variations.

```
\cmd B{2} the alphabet. % prints: B is no. 2 of the alphabet.
\cmd B2 the alphabet.   % prints: B is no. 2 of the alphabet.
\cmd N14 the alphabet.  % prints: N is no. 1 of4 the alphabet.
```

The result of `\cmd B{2}` is as expected. The results of `\cmd B2` and `\cmd N14` should also be expected, and if not take a moment to ponder why. The ‘B’ and ‘N’ are the first arguments to `\cmd` in the two cases because a single character is a token. Having found the first argument TeX looks for the second one, which again will be a token as there are no braces. It will find ‘2’ and ‘1’ as the second arguments and will then expand the `\cmd` macro. In the case of `\cmd B2` this gives a reasonable result. In the case of `\cmd N14`, TeX expands `\cmd N1` to produce ‘N is in position 1 of’, then continues printing the rest of the text, which is ‘4 the alphabet’, hence the odd looking result.

E.4 (LA)TEX COMMANDS

I have used some TeX commands in the example code and it is now time to describe these. Only enough explanation is given to cover my use of them. Full explanations would require a doubling in the size of the book and a concomitant increase in the price, so for full details consult the *TeXbook* which is the definitive source, or one of the TeX manuals listed in the Bibliography. I find *TeX by Topic* particularly helpful.

I have also used LaTeX commands that are not mentioned by Lamport. LaTeX uses a convention for command names; any name that includes the @ character is an ‘internal’ command and may be subject to change, or even deletion. Normal commands are meant to be stable — the code implementing them may change but their effect will remain unaltered. In the LaTeX kernel, and in class and package files the character @ is automatically treated as a letter so it may be used as part of a command name. Anywhere else you have to use `\makeatletter` to make @ be treated as a letter and `\makeatother` to make @ revert to its other meaning. So, if you are defining or modifying or directly using any command that includes an @ sign then this must be done in either a .sty file or if in the document itself it must be surrounded by `\makeatletter` and `\makeatother`.

The implication is ‘don’t use internal commands as they may be dangerous’. Climbing rocks is also dangerous but there are rock climbers; the live ones though don’t try climbing Half Dome in Yosemite or the North Face of the Eiger without having first gained experience on friendlier rocks.

The L^AT_EX kernel is full of internal commands and a few are mentioned in Lamport. There is no place where you can go to get explanations of all the L^AT_EX commands, but if you run L^AT_EX on the `source2e.tex` file which is in the standard L^AT_EX distribution you will get the commented kernel code. The index of the commands runs to about 40 double column pages. Each class and package introduce new commands over and above those in the kernel.

L^AT_EX includes `\newcommand`, `\providecommand` and `\renewcommand` as means of (re-)defining a command, but T_EX provides only one method.

`\def<cmd><arg-spec>{<text>}`

`\def` specifies that within the local group the command `\cmd` is defined as `<text>`, and any previous definitions of `<cmd>` within the group are overwritten. Neither the `<text>` nor any arguments can include an end-of-paragraph. The L^AT_EX equivalent to `\def` is the pair of commands `\providecommand*` followed by `\renewcommand*`.

The `<arg-spec>` is a list of the argument numbers (e.g., `#1#2`) in sequential order, the list ending at the ‘`’` starting the `<text>`. Any spaces or other characters in the argument list are significant. These must appear in the actual argument list when the macro is used.

`\long \global
\gdef<cmd><arg-spec>{<text>
\edef<cmd><arg-spec>{<text>
\xdef<cmd><arg-spec>{<text>`

If you use the `\long` qualifier before `\def` (as `\long\def...`) then the `<text>` and arguments may include paragraphs. The L^AT_EX version of this is the unstarred `\providecommand` followed by `\renewcommand`.

To make a command global instead of local to the current group, the `\global` qualifier can be used with `\def` (as `\global\def...`) when defining it; `\gdef` is provided as a shorthand for this common case.

Normally any macros within the replacement `<text>` of a command defined by `\def` are expanded when the command is called. The macro `\edef` also defines a command but in this case any macros in the replacement `<text>` are expanded when the command is defined. Both `\long` and `\global` may be used to qualify `\edef`, and like `\gdef` being shorthand for `\global\def`, `\xdef` is short for `\global\edef`.

There is much more to the `\def` family of commands than I have given; consult elsewhere for all the gory details.

`\let<cmda>=<cmdb>`

The `\let` macro gives `<cmda>` the same definition as `<cmdb>` at the time the `\let` is called. The `=` sign is optional. `\let` is often used when you want to save the definition of a command.

Here is a short example of how some of `\def` and `\let` work.

```
\def\name{Alf}  
\let\fred = \name  
  \name, \fred.           % prints Alf, Alf.
```

```

\def\name{Fred}
\name, \fred.           % prints Fred, Alf.
\def\name{\fred red}
\name, \fred.           % prints Alfred, Alf.

```

`\csname <string>\endcsname`

If you have ever tried to define commands like `\cmd1`, `\cmd2` you will have found that it does not work. TeX command names consists of either a single character or a name composed solely of what TeX thinks of as alphabetic characters. However, the `\csname` `\endcsname` pair turn the enclosed `<string>` into the control sequence `\string`, which means that you can create `\cmd1` by

```
\csname cmd1\endcsname
```

Note that the resulting `\cmd1` is not defined (as a macro).

`\@namedef{<string>}`
`\@nameuse{<string>}`

The kernel `\@namedef` macro expands to `\def\<string>`, where `<string>` can contain any characters. You can use this to define commands that include non-alphabetic characters. There is the matching `\@nameuse` macro which expands to `\<string>` which then lets you use command names that include non-alphabetic characters. For example:

```

\@namedef{fred2}{Frederick-II}
...
\makeatletter\@nameuse{fred2}\makeatother reigned from ...

```

At any point in its processing TeX is in one of six *modes* which can be categorized into three groups:

1. horizontal and restricted horizontal;
2. vertical and internal vertical;
3. math and display math.

More simply, TeX is in either horizontal, or vertical, or math mode. In horizontal mode TeX is typically building lines of text while in vertical mode it is typically stacking things on top of each other, like the lines making up a paragraph. Math gets complicated, and who can do with more complications at this stage of the game?

`\hbox to <dimen>{\<text>}` `\hb@xt@<dimen>{\<text>}`
`\vbox to <dimen>{\<text>}`

With `\hbox`, `<text>` is put into a horizontal box, and similarly `\vbox` puts `<text>` into a vertical box. The sizes of the boxes depend on the size of the `<text>`. The optional `to <dimen>` phrase sets the size of the box to the fixed `<dimen>` value. If the `<text>` does not fit neatly inside a fixed size box then TeX will report `overfull` or `underfull` warnings. LaTeX supplies the `\hb@xt@` command as a shorthand for `\hbox to`.

Inside a horizontal box TeX is in restricted horizontal mode which means that everything in the box is aligned horizontally. Inside a vertical box TeX is in internal vertical mode and the contents are stacked up and aligned vertically.

```
\dp<box> \ht<box> \wd<box>
```

The depth, height and width of a box are returned by the macros `\dp`, `\ht` and `\wd` respectively.

```
\leavevmode
```

T_EX may be in either vertical or horizontal mode and there are things that can be done in one mode while T_EX reports an error if they are attempted in the other mode. When typesetting a paragraph T_EX is in horizontal mode. If T_EX is in vertical mode, `\leavevmode` makes it switch to horizontal mode, but does nothing if T_EX is already in horizontal mode. It is often used to make sure that T_EX is in horizontal mode when it is unclear what state it might be in.

E.5 CALCULATION

L^AT_EX provides some methods for manipulating numbers and these, of course, are composed from T_EX's more basic methods. Sometimes it is useful to know what T_EX itself provides. We have met most, if not all, of L^AT_EX's macros earlier but I'll collect them all here for ease of reference.

E.5.1 Numbers

In L^AT_EX a counter is used for storing an integer number.

```
\newcounter{<counter>}
\setcounter{<counter>}{<number>}
\stepcounter{<counter>} \refstepcounter{<counter>}
```

A new counter called `<counter>`, without a backslash, is created using `\newcounter`. Its value can be set to a `<number>` by the `\setcounter` command and `\stepcounter` increases its value by one. If the counter is to be used as the basis for a `\label`, its value must be set using `\refstepcounter`, neither `\stepcounter` nor `\setcounter` will work as expected in this case.

Internally, a L^AT_EX *counter* is represented by a T_EX *count* — the `\newcounter` macro creates a T_EX count named `\c@<counter>`, and the other `\...counter` macros similarly operate on the `\c@<counter>` count.

```
\newcount<count>
```

The T_EX `\newcount` command creates a new count, `<count>`, which *does* include an initial backslash. For example

```
\newcount\mycount
```

T_EX's method of assigning a number to a count uses nothing like `\setcounter`.

```
<count> [=] <number>
```

The `[=]` enclosing the `=` sign are there only to indicate that the `=` sign is optional. For example:

```
\mycount = -24\relax % \mycount has the value -24
\mycount 36\relax    % now \mycount has the value 36
```

Table E.1: Some internal macros for numbers

<code>\m@ne</code>	-1	<code>\z@</code>	0	<code>\@ne</code>	1
<code>\tw@</code>	2	<code>\thr@@</code>	3	<code>\sixt@@n</code>	16
<code>\@xxxii</code>	32	<code>\@ccclv</code>	255	<code>\@ccclvi</code>	256
<code>\@m</code>	1000	<code>\@Mi</code>	10001	<code>\@Mii</code>	10002
<code>\@Miii</code>	10003	<code>\@Miv</code>	10004	<code>\@MM</code>	20000

I have added `\relax` after the digits forming the number for safety and efficiency. When TeX is reading a number it keeps on looking until it comes across something that is not part of a number. There are things that TeX will treat as part of a number which you might not think of, but `\relax` is definitely not part of a number. See, for example, [Eij92, chapter 7] for all the intricate details if you need them.

There are some numbers that are used many times in the LaTeX kernel and class codes. To save having to use `\relax` after such numbers, and for other reasons of efficiency, there are commands that can be used instead of typing the digits. These are listed in Table E.1. The command `\z@` can be used both for the number zero and for a length of 0pt. Do not use the commands to print a number.

TeX has a limited vocabulary for arithmetic. It can add to a count, and can multiply and divide a count, but only by integers. The result is always an integer. This may be disconcerting after a division where any remainder is discarded. The syntax for these operations is:

```
\advance<count> [ by ] <number>
\multiply<count> [ by ] <number>
\divide<count> [ by ] <number>
```

The `by` is a TeX keyword and the brackets are just there to indicate that it can be missed out. Some examples:

```
\advance\mycount by -\mycount % \mycount is now 0
\mycount = 15\relax          % \mycount is now 15
\divide\mycount by 4\relax   % \mycount is now 3
\multiply\mycount 4\relax     % \mycount is now 12
\advance\mycount by \yourcount % \mycount is now \yourcount + 12
```

The value of a count can be typeset by prepending the count by the `\the` command, e.g., `\the\mycount`.

E.5.2 Lengths

Every length has an associated unit. For convenience I'll use '*dimension*' as shorthand for a number and a length unit.

```
dimension: <number><length-unit>
```

For example, a *dimension* may be 10pt, or 23mm, or 1.3pc.

Unlike LaTeX, TeX distinguishes two kinds of lengths. A TeX `\dimen` is a length that is fixed; in LaTeX's terms it is a *rigid* length. On the other hand a TeX `\skip` is a length that may stretch or shrink a little; it is what LaTeX calls a *rubber* length.

```
\newdimen<dimen> \newskip<skip>
```

The TeX macros `\newdimen` and `\newskip` are used for creating a new *<dimen>* or a new *<skip>*. For instance:

```
\newdimen\mydimen
\newskip\myskip
```

The value of a `\dimen` is a *dimension* and the value of a `\skip` is what TeX calls *glue*. It so happens that LaTeX's `\newlength` always creates a new skip — all LaTeX lengths are created as rubber lengths. Glue has at least one and possibly as many as three parts.

glue: *dimension* [*plus dimension*] [*minus dimension*]

The optional *plus* part is the amount that the glue can stretch from its normal size and the optional *minus* part is the amount the glue can shrink below its normal size. Both *plus* and *minus* are TeX keywords. Glue can never shrink more than the *minus dimension* and it normally does not stretch more than the *plus dimension*.

`\@plus \@minus`

LaTeX supplies `\@plus` and `\@minus` which expand to *plus* and *minus* respectively. Writing `\@plus` instead of *plus* uses one instead of four tokens, saving three tokens, and `\@minus` in place of *minus* saves four tokens — remember that a TeX token is either a control sequence (e.g. `\@minus`) or a single character (e.g., *m*). TeX's memory is not infinite — it can only hold so many tokens — and it makes sense for kernel and class or package writers to use fewer rather than more to leave sufficient space for any that authors might want to create.

In TeX, assigning a value to a length (`\dimen` or `\skip`) is rather different from the way it would be done in LaTeX.

<dimen> [=] *<dimension>*
<skip> [=] *<glue>*

The [and] enclosing the = sign are there only to indicate that the = sign is optional. For example:

```
\newdimen\mydimen
\mydimen = 3pt      % \mydimen has the value 3pt
\mydimen  -13pt    % now \mydimen has the value -13pt
\myskip = 10pt plus 3pt minus 2pt % \myskip can vary between
                                % 8pt and 13pt (or more)
\myskip = 10pt plus 3pt          % \myskip can vary between
                                % 10pt and 13pt (or more)
\myskip = 10pt minus 2pt        % \myskip can vary between
                                % 8pt and 10pt
\myskip = 10pt                 % \myskip is fixed at 10pt
```

Like counts, the value of a length can be typeset by prepending the length by the `\the` command, e.g., `\the\myskip`.

TeX's lengths can be manipulated in the same way as a count, using the `\advance`, `\multiply` and `\divide` macros. Ignoring some details, lengths can be added together but may only be multiplied or divided by an integer number.


```

▷ \Wdimen = 10pt ⇒
                                Wdimen = 10.0pt
▷ \Wskip = 15pt plus 5pt minus 3pt ⇒
                                Wskip = 15.0pt plus 5.0pt minus 3.0pt
▷ \advance\Wskip by \Wskip ⇒
                                Wskip = 30.0pt plus 10.0pt minus 6.0pt
▷ \multiply\Wskip by 3 ⇒
                                Wskip = 90.0pt plus 30.0pt minus 18.0pt
▷ \divide\Wskip by 17 ⇒
                                Wskip = 5.29411pt plus 1.7647pt minus 1.05882pt
▷ \advance\Wskip by \Wdimen ⇒
                                Wskip = 15.29411pt plus 1.7647pt minus 1.05882pt
▷ \advance\Wdimen by \Wskip ⇒
                                Wdimen = 25.29411pt

```

A length can be multiplied by a fractional number by prepending the length with the number. For example:

```

▷ \Wdimen = 0.5\Wdimen ⇒
                                Wdimen = 12.64705pt
▷ \Wskip = 0.5\Wskip ⇒
                                Wskip = 7.64705pt

```

When `\multiply` or `\divide` is applied to a `\skip` all its parts are modified, both the fixed part and any elastic components. However, if a `\skip` is multiplied by a fractional number then it loses any elasticity it might have had. In the same vein, if a `\skip` is added to a `\dimen` any elasticity is lost. A `\skip` can be coerced into behaving like a `\dimen` but a `\dimen` is always rigid. For example, typing `'\Wdimen = 10pt plus 2pt minus 1pt'` results in: 'plus 2pt minus 1pt'.

`\newlength{<len>}`

LaTeX's `\newlength` macro creates a new rubber length (internally it uses `\newskip`); there is no LaTeX specific macro to create a rigid length (i.e., a `\dimen`).

LaTeX has a variety of macros for setting or changing its length values.

`\setlength{<len>}{<glue>}`

The LaTeX `\setlength` macro assigns the value `<glue>` to the rubber length `<len>`. Some examples of this are:

```

▷ \setlength{Wlen}{10pt} ⇒
                                Wlen = 10.0pt
▷ \setlength{Wlen}{10pt plus 2pt} ⇒
                                Wlen = 10.0pt plus 2.0pt
▷ \setlength{Wlen}{10pt minus 1pt} ⇒
                                Wlen = 10.0pt minus 1.0pt
▷ \setlength{Wlen}{10mm plus 2pt minus 1pt} ⇒
                                Wlen = 28.45274pt plus 2.0pt minus 1.0pt

```

As shown in the last example above where both mm and pt are used as a length unit, the `\the` applied to a length always prints the value in pt units.

```
\settowidth{<len>}{<text>}
\settoheight{<len>}{<text>}
\settodepth{<len>}{<text>}
```

These put the `<text>` into a box and then set the `<len>` to the width, height and depth respectively of the box.

```
\addtolength{<len>}{<glue>}
```

LaTeX's `\addtolength` macro is the equivalent of TeX's `\advance` command. There are no equivalents to TeX's `\multiply` or `\divide` but in any case a length can still be multiplied by prepending it with a fractional number.

```
\z@
fil fill filll
```

`\z@` is a very useful LaTeX command when specifying lengths. Depending on the context it either stands for the number 0 (zero) or 0pt (zero length). TeX has three kinds of infinitely stretchy length units that can be used in the plus or minus parts of a skip. `fil` is infinitely more flexible than any fixed amount, but `fill` is infinitely more flexible than `fil` and `filll` is infinitely more flexible than anything else at all. These infinite glues can be used to push things around.

```
\hskip<skip>
\vskip<skip>
```

The TeX command `\hskip` inserts `<skip>` horizontal space and likewise `\vskip` inserts `<skip>` vertical space.

```
\hfil \hfill \hfilneg \hss
```

These commands are all TeX primitives and are equivalent to horizontal skips with some kind of infinite glue, as indicated below (note the use of `fil` as a length unit, it being preceded by a number):

```
\hfil    -> \hskip 0pt plus 1fil
\hfill   -> \hskip 0pt plus 1fill
\hfilneg -> \hskip 0pt          minus 1fil
\hss     -> \hskip 0pt plus 1fil minus 1fil
```

```
\vfil \vfill \vfilneg \vss
```

These commands are all TeX primitives and are equivalent to vertical skips with some kind of infinite glue, as indicated below:

```
\vfil    -> \vskip 0pt plus 1fil
\vfill   -> \vskip 0pt plus 1fill
\vfilneg -> \vskip 0pt          minus 1fil
\vss     -> \vskip 0pt plus 1fil minus 1fil
```

E.6 PROGRAMMING

One of the commonest programming operations is to possibly do one thing if something is true and to possibly do another thing if it is not true. Generally speaking, this is called an ‘if-then-else’ or *conditional* statement.

```
\if... <test> <true-text> [ \else <false-text> ] \fi
```

TeX has several kinds of ‘if-then-else’ statements which have the general form shown above. The statement starts with an `\if...` and is finished by a matching `\fi`. As usual, the brackets enclose optional elements, so there need be no `\else` portion. The `<true-text>`, if it exists, is processed if the `<test>` is true otherwise the `<false-text>`, if both the `\else` clause and `<false-text>` are present, is processed.

The simplest kind of `\if...` is defined by the `\newif` macro.

```
\newif\if<name>
```

`\newif` creates three new commands, the `\ifname` and the two declarations, `\nametrue` and `\namefalse`, for setting the value of `\ifname` to true or false respectively. In this case the `<test>` is embedded in the `\if...`. For example:

```
\newif\ifpeter
...
\ifpeter
  My name is Peter.
\else
  Call me Ishmael.
\fi
```

or a more likely scenario is

```
\newif\ifmine
  \minetrue % or \minefalse
\newcommand{\whose}{%
  \ifmine It's mine. \else I don't know whose it is. \fi}
```

Here are some of the other more commonly used kinds of ifs.

```
\ifdim <dimen1> <rel> <dimen2>
\ifnum <number1> <rel> <number2>
\ifodd <number>
```

The `<rel>` in `\ifnum` and `\ifdim` is one of the three characters: `<` (less than), `=` (equals), or `>` (greater than). `\ifdim` results in true if the two lengths are in the stated relationship otherwise it results in false. Similarly `\ifnum` is for the comparison of two integers. The `\ifodd` test is true if the integer `<number>` is an odd number, otherwise it results in false.

Among other things, the LaTeX class code that organizes the page layout checks if the length values are sensible. The following code is a snippet from the layout algorithm. It checks that the sum of the margins and the width of the typeblock is the same as the width of the page after trimming. `\@tempdima` and `\@tempdimb` are two ‘scratch’ lengths used in many calculations.

```
\@tempdimb=-1pt          % allow a difference of 1pt
\@tempdima=\paperwidth    % paperwidth
```

```

\advance\@tempdima by -\foremargin % minus the foremargin
\advance\@tempdima -\textwidth      % minus the textwidth
\advance\@tempdima -\spinemargin    % minus the spinemargin
\ifdim\@tempdima < \@tempdimb       % should be close to zero
  %% error                          % otherwise a problem
\fi

```

Changing the subject, on the offchance that you might want to see how the Fibonacci sequence progresses, the first thirty numbers in the sequence are: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229 and 832040. I got L^AT_EX to calculate those numbers for me, and it could have calculated many more. They were produced by just saying `\fibseries{30}`. The French mathematician Édouard Lucas (1842–1891) studied sequences like this and was the one to give it the name Fibonacci. Lucas also invented the game called the Tower of Hanoi with Henri de Parville (1838–1909), supplying the accompanying fable [dP84, RBC74]:

In the great temple at Benares beneath the dome that marks the center of the world, rests a brass plate in which are fixed three diamond needles, each a cubit high and as thick as the body of a bee. On one of these needles, at the creation, God placed sixty-four discs of pure gold, the largest disc resting on the brass plate, and the others getting smaller and smaller up to the top one. This is the tower of Bramah. Day and night unceasingly the priests transfer the discs from one diamond needle to another according to the fixed and immutable laws of Bramah, which require that the priest on duty must not move more than one disc at a time and that he must place this disc on a needle so that there is no smaller disc below. When the sixty-four discs shall have been thus transferred from the needle which at creation God placed them, to one of the other needles, tower, temple, and Brahmins alike will crumble into dust and with a thunderclap the world will vanish.

The number of separate transfers of single discs is $2^{64} - 1$ or just under eighteen and a half million million moves, give or take a few, to move the pile. At the rate of one disc per second, with no mistakes, it would take more than 58 million million years before we would have to start being concerned.

In his turn, Lucas has a number sequence named after him. There are many relationships between the Fibonacci numbers F_n and the Lucas numbers L_n , the simplest, perhaps, being

$$L_n = F_{n-1} + F_{n+1} \tag{E.1}$$

$$5F_n = L_{n-1} + L_{n+1} \tag{E.2}$$

The first 15 numbers in the Lucas sequence are: 2, 1, 3, 4, 7, 11, 18, 29, 47, 76, 123, 199, 322, 521 and 843. These were produced by saying `\gfibseries{2}{1}{15}`. The Lucas numbers are produced in the same manner as the Fibonacci numbers, it's just the starting pairs that differ.

However, it is the definition of the `\fibseries` and `\gfibseries` macros that might be more interesting in this context.

First, create four new counts. `\fibtogo` is the number of terms to be calculated, `\fib` is the current term, and `\fibprev` and `\fibprevprev` are the two prior terms.

```
\newcount\fib
\newcount\fibprev
\newcount\fibprevprev
\newcount\fibtogo
```

The argument to `\fibseries` is the number of terms. The counts `\fibprevprev` and `\fibprev` are set to the starting pair in the sequence. Provided the number of terms requested is one or more the macro `\@fibseries` is called to do the work.

```
\newcommand*{\fibseries}[1]{%
  \fibprevprev=1\relax
  \fibprev=1\relax
  \ifnum #1>0\relax
    \@fibseries{#1}%
  \fi}
```

The macro `\@fibseries` calculates and prints the terms.

```
\newcommand*{\@fibseries}[1]{%
  \fibtogo=#1\relax
```

It's simple if no more than two terms have been asked for — just print them out.

```
\ifnum \fibtogo=\@ne
  \the\fibprevprev
\else
  \ifnum \fibtogo=\tw@
    \the\fibprevprev{} and \the\fibprev
  \else
```

Three or more terms have to be calculated. We reduce the number to be calculated by 2, and print the first two terms.

```
    \advance\fibtogo by -\tw@
    \the\fibprevprev, \the\fibprev
```

We now have to calculate the rest of the terms, where each term is the sum of the two previous terms. The macro `\@fibnext` calculates the next term, prints it out and reduces the number of terms left to be calculated (`\fibtogo`) by one. If there are terms left to be done then the process is repeated until they have all been printed.

```
    \loop
      \@fibnext
      \ifnum \fibtogo>\z@
        \repeat
    \fi
\fi}
```

The `\@fibnext` macro calculates a term in the series, uses `\printfibterm` to print it, and decrements the `\fibtogo` count.

```
\newcommand*{\@fibnext}{%
```

```
\fib=\fibprev
\advance\fib by \fibprevprev
\fibprevprev=\fibprev
\fibprev=\fib
\printfibterm
\advance\fibtogo \m@ne}
```

The last of the macros, `\printfibterm`, typesets a term in the sequence. If the term is the last one print an ‘and’ otherwise print a ‘,’ then a space and the term.

```
\newcommand*{\printfibterm}{%
  \ifnum \fibtogo=\@ne \space and \else , \fi
  \the\fib}
```

You have met all of the macros used in this code except for TeX’s `\loop` construct. I find the syntax for this a little unusual.

```
\loop <text1> \if... <text2> \repeat
```

The construct starts with `\loop` and is ended by `\repeat`; the `\if...` is any conditional test, but without the closing `\fi`. TeX processes `<text1>`, then if the `\if...` is true it processes `<text2>` and repeats the sequence again starting with `<text1>`. On the other hand, as soon as the result of the `\if...` is false the loop stops (i.e., TeX jumps over `<text2>` and goes on to do whatever is after the `\repeat`).

The `\gfibseries` macro that I used for the Lucas numbers is a generalisation of `\fibseries`, where the first two arguments are the starting pair for the sequence and the third argument is the number of terms; so `\gfibseries{1}{1}{...}` is equivalent to `\fibseries{...}`.

```
\newcommand*{\gfibseries}[3]{%
  \fibprevprev=#1\relax
  \fibprev=#2\relax
  \ifnum #3>0\relax
    \@fibseries{#3}%
  \fi}
```

The calculation of the terms in the Fibonacci and in the generalised sequences is the same so `\@fibseries` can be used again.

I used the TeX `\loop` construct in the `\@fibseries` macro but LaTeX has a similar construct.

```
\@whilenum <ifnum test> \do {<body>}
\@whiledim <ifdim test> \do {<body>}
```

As long as the appropriate `<test>` is true the `<body>` is processed.

In `\@fibseries` I used `\ifnum`s to check for 3 possible values. There is another `\if...` form that can be used for this type of work.

```
\ifcase <number> <text for 0> \or <text for 1> \or <text for 2>
...
\or <text for N> [ \else <text for anything else> ] \fi
```

If the `<number>` is 0 then `<text for 0>` is processed, but if `<number>` is 1 then `<text for 1>` is processed, but if `<number>` is ... Each `<text for ...>` is separated by an `\or`. If `<number>` is

anything other than the specified cases (i.e., less than zero or greater than N) then if the `\else` is present (*text for anything else*) is processed.

Here's another version of the `\@fibseries` macro using `\ifcase` and `\@whilenum`.

```
\renewcommand*{\@fibseries}[1]{%
  \fibtogo=#1\relax
  \ifcase \fibtogo % ignore 0
  \or % \fibtogo=1
    \the\fibprevprev
  \or % \fibtogo=2
    \the\fibprevprev{} and \the\fibprev
  \else % fibtogo > 2
    \advance\fibtogo by -\tw@
    \the\fibprevprev, \the\fibprev
    \@whilenum \fibtogo > \z@ \do {% must kill space after the {
      \fibnext}%
    }
  \fi}
```

TeX has more programming constructs than I have shown here and these will be explained in any good TeX book. LaTeX also has more than I have shown but in this case the best place to look for further information is in the LaTeX kernel code, for example in `ltxcntrl.dtx`.

F

The terrors of errors

No matter how conscientious you are a mistake or two will occasionally creep into your document source. The good news is that whatever happens TeX will not destroy your files — it may produce some odd looking output, or even no output at all, but your work is safe. The bad news is that you have to correct any errors that TeX finds. To assist you in this TeX stops whenever it comes across what it thinks is an error and tells you about it. If you're not sure what to do it will also provide some possibly helpful advice.

TeX underlies LaTeX which underlies classes and packages. You may get messages than originate from TeX, or from LaTeX, or from the class and any packages you may be using. I'll describe the TeX, LaTeX, and class messages below.

In general, you will see a message on your terminal and LaTeX will stop and wait for you to respond. It prints a question mark and is expecting you to type one of the following:

- *<return>* (or *<enter>* or what is the equivalent on your keyboard): LaTeX will continue processing the document.
- H (help): the help message is output and LaTeX waits for you to respond again.
- S (scroll): Continue processing, outputting any further error messages, but not stopping.
- Q (quiet): Continue processing without stopping and with no further messages.
- R (run): Like the Q option but not even stopping if your document requires some user input.
- I (insert): To insert some material for TeX to read but no changes are made to the source file.
- E (edit): This may return you to an editor so you can change the file. What actually happens is system dependent.
- X (exit): Stop this LaTeX run.

On the system I am used to the case of the characters does not matter. I must admit that the only ones I have used are *<return>*, q, h and x, in approximately that order of frequency.

All messages are output to the log file so you can study them later if you need to.

F.1 TEX MESSAGES

The following is an alphabetical list of some of TeX's messages, abbreviated in some cases, together with their corresponding remarks. As an example of how these appear on your terminal, if you had a line in your source that read:

resulting in x^3^4 .

then TeX would output this:

```
! Double superscript
1.102 resulting in  $x^3^
4^$.$ 
```

?

If you typed `h` in response to this you would then see:

I treat ' x^{1^2} ' essentially like ' $x^{1\{ \}^2}$ '.

TeX's messages start with `!` followed by the particular message text. The second line starts `1.` and a number, which is the number of the line in your file where the error is. This is followed by the text of the line itself up to the point where the error was detected, and the next line in the report shows the rest of the erroneous line. The last line of the report is a `?` and TeX awaits your response.

In the listing I have used this font for the error message and *this font for the comment message*.

! A box was supposed to be here.

I was expecting to see `\hbox` or `\vbox` or `\copy` or `\box` or something like that. So you might find something missing in your output. But keep trying; you can fix this later.

! Argument of ... has an extra `}`.

I've run across a '`}`' that doesn't seem to match anything. For example, '`\def\aa#1{...}`' and '`\aa}`' would produce this error. If you simply proceed now, the `\par` that I've just inserted will cause me to report a runaway argument that might be the root of the problem. But if your '`}`' was spurious, just type '`2`' and it will go away.

In LaTeX terms, the example can be translated into '`\newcommand{\aa}[1]{...}`' and '`\aa}`'.

If you can't find the extra `}` it might be that you have used a fragile command in a moving argument. Footnotes or math in division titles or captions are a fruitful source for this kind of error. You shouldn't be putting footnotes into titles that will get listed in the ToC. For maths, put `\protect` before each fragile command.

! Arithmetic overflow.

I can't carry out that multiplication or division, since the result is out of range.

The maximum number that TeX can deal with is 2,147,483,647 and it balks at dividing by zero.

! Dimension too large.

I can't work with sizes bigger than about 19 feet. Continue and I'll use the largest value I can.

! Display math should end with \$\$.

The '\$' that I just saw supposedly matches a previous '\$\$'. So I shall assume that you typed '\$\$' both times.

Although \$\$ is one of TeX's methods for starting and ending display math, do *not* use it in LaTeX.

! Double subscript.

I treat 'x_1_2' essentially like 'x_1{}_2'.

This would produce x_{12} . If you were after say, x_{23} instead, type $x_{2\{3\}}$.

! Double superscript.

I treat 'x^1^2' essentially like 'x^1{}^2'.

This would produce x^{12} . If you were after say, x^{23} instead, type $x^{2\{3\}}$.

! (\end occurred inside a group at level ...).

This message is output at the end of a run. It means that you have not ended all the groups that you started; a group can be started by a simple open brace ($\{$), but there are other starting mechanisms as well, such as $\begin\{\dots\}$. If the problem is a missing $\end\{\dots\}$, LaTeX is kind enough to tell you what the mismatch is.

! (\end occurred when ... was incomplete).

! Extra \fi. or Extra \else. or Extra \or.

I'm ignoring this; it doesn't match any \if.

! Extra \endcsname.

I'm ignoring this, since I wasn't doing a \csname.

! Extra \right.

I'm ignoring a \right that had no matching \left.

! Extra }, or forgotten \endgroup, \$, or \right.

I've deleted a group closing symbol because it seems to be spurious, as in '\$x}\$'. But perhaps the } is legitimate and you forgot something else, as in '\hbox{\$x}' . In such cases the way to recover is to insert both the forgotten and the deleted material, e.g., by typing 'I\$'.

The braces or math mode delimiters didn't match. You might have forgotten a {, \[, \{ or \$.

! Extra ...

Things are pretty mixed up, but I think the worst is over.

! Extra alignment tab has been changed to \cr.

You have given more \span or & marks than there were in the preamble to the \halign or \valign now in progress. So I'll assume that you meant to type \cr instead.

Internally, LaTeX uses \halign for its array and tabular environments. The message means that you have too many column entries in a row (i.e., too many & before the end of the row). Perhaps you have forgotten to put \\ at the end of the preceding row.

! File ended while scanning or Forbidden control sequence found while scanning

I suspect you have forgotten a '}', causing me to read past where you wanted me to stop. I'll try to recover; but if the error is serious you'd better type 'E' or 'X' now and fix your file.

! Font ...not loadable: Metric (TFM) file not found.

! Font ...not loadable: Bad metric (TFM) file.

I wasn't able to read the size data for this font, so I will ignore the font specification. [Wizards can fix TFM files using TFtoPL/PLtoTF.] You might try inserting a different font spec; e.g., type 'I\font<same font id>=<substitute font name>'.

LaTeX can't find a font you have asked for.

! Huge page cannot be shipped out.

The page just created is more than 18 feet tall or more than 18 feet wide, so I suspect something went wrong.

! I can't find file '...', please type another.

TeX couldn't find the file you asked it to read. You can also get this message with LaTeX if you have missed the braces around the argument to \input.

! I can't go on meeting you like this.

One of your faux pas seems to have wounded me deeply... in fact, I'm barely conscious. Plase fix it and try again.

! I can't write on file '...', please type another.

TeX couldn't write on a file, you might have misspelled the name or not have permission to use it.

! Illegal parameter number in definition of

You meant to type ## instead of #, right? Or maybe a } was forgotten somewhere earlier, and things are all screwed up? I'm going to assume that you meant ##.

This is probably due to a command defining command like \newcommand or \renewcommand or \providecommand, or an environment defining command like \newenvironment or \renewenvironment, where a # has been used incorrectly. Apart from the command \#, a # can only be used to indicate an argument parameter, like #3 which denotes the third argument. You cannot use an argument parameter, like the #3 in the last argument of either the \newenvironment or the \renewenvironment commands.

You get the same error if you try to include any of the above defining commands inside another one.

! Illegal unit of measure (replaced by filll).

I dddon't go any higher than filll.

You have tried to use a filll with more than 3 'l's.

! Illegal unit of measure (mu inserted).

The unit of measurement in math glue must be mu. To recover gracefully from this error it's best to delete the erroneous units; e.g., type '2' to delete two letters. (See Chapter 27 of The TeXbook.)

TeX was in math mode and expecting a length, which must be in mu units.

! Illegal unit of measure (pt inserted).

Dimensions can be in units of em, ex, in, pt, pc, cm, mm, dd, cc, bp, or sp; but yours is a new one! I'll assume you meant to say pt, for printers' points. To recover gracefully from this error it's best to delete the erroneous units; e.g., type '2' to delete two letters. (See Chapter 27 of The TeXbook.)

TeX was expecting a length but it found just a number without a known length unit. For example you wrote 2ib instead of 2in.

! Improper \hyphenation will be flushed.

Hyphenation exceptions must contain only letters and hyphens. But continue; I'll forgive and forget.

! Incomplete ...all text was ignored after line

A forbidden control sequence occurred in skipped text. This kind of error happens when you say '\if...' and forget the matching '\fi'. I've inserted a '\fi'; this might work.

! Infinite glue shrinkage found in a paragraph.

The paragraph just ended includes some glue that has infinite shrinkability, e.g., '\hskip 0pt minus 1fil'. Such glue doesn't belong there—it allows a paragraph of any length to fit on one line. But it's safe to proceed, since the offensive shrinkability has been made finite.

! Limit controls must follow a math operator.

I'm ignoring this misplaced \limits or \nolimits command.

! Misplaced &. or Misplaced \cr. or Misplaced \span.

I can't figure out why you would want to use a tab mark or \cr or \span here. If you just want an ampersand the remedy is simple: Just type 'I\&' now. But if some right brace up above has ended a previous alignment prematurely, you're probably due for more error messages, and you might try typing 'S' now just to see what is salvageable.

In LaTeX the most likely of these messages is the Misplaced &. You can only use a naked & in environments like array and tabular as column separators. Anywhere else you have to use \&.

! Misplaced \noalign.

I expect to see \noalign only after the \cr of an alignment. Proceed, and I'll ignore this case.

! Misplaced \omit.

I expect to see \omit only after the tab marks or the \cr of an alignment. Proceed, and I'll ignore this case.

! Missing \cr inserted.

I'm guessing that you meant to end an alignment here.

You might have missed a `\\` at the end of the last row of a `tabular` or `array`.

! Missing = inserted for

I was expecting to see '<', '=', or '>'. Didn't.

! Missing # inserted in alignment preamble.

There should be exactly one # between &'s, when an \halign or \valign is being set up. In this case you had none, so I've put one in; maybe that will work.

If you get this in LaTeX then there are problems with the argument to an `array` or `tabular`.

! Missing \$ inserted. or Missing \endgroup inserted. or Missing \right inserted. or Missing } inserted.

I've inserted something that you may have forgotten. (See the <inserted text> above.) With luck, this will get me unwedged, But if you really didn't forget anything, try typing '2' now; then my insertion and my current dilemma will both disappear.

This is a general response to the above messages. There is also a more specific response for each of the messages, as listed below.

! Missing \$ inserted.

I've inserted a begin-math/end-math symbol since I think you left one out. Proceed with fingers crossed.

Certain commands can only be executed in math mode and there are others that cannot be used in math mode. TeX has come across a command that cannot be used in the current mode, so it switches into, or out of, math mode on the assumption that that was what you had forgotten to do.

! Missing \endcsname inserted.

The control sequence marked <to be read again> should not appear between \csname and \endcsname.

! Missing { inserted.

A left brace was mandatory here, so I've put one in. You might want to delete and/or insert some corrections so that I will find a matching right brace soon. If you're confused by all this, try typing 'I}' now.

! Missing { inserted.

Where was the left brace? You said something like \def\ a}', which I'm going to interpret as \def\ a{}'.

In LaTeX terms, the example wrongdoing would be `\newcommand{\ a}{}}`

! Missing { inserted.

I've put in what seems necessary to fix the current column of the current alignment. Try to go on, since this might almost work.

It seems that a `{` might have been missing in a `tabular` or `array` entry.

! Missing control sequence inserted.

*Please don't say '`\def cs{...}`', say '`\def\cs{...}`'. I've inserted an inaccessible control sequence so that your definition will be completed without mixing me up too badly. You can recover graciously from this error, if you're careful; see exercise 27.2 in *The TeXbook*.*

! Missing delimiter(. inserted).

I was expecting to see something like '(' or '{' or '\{' or '\}' here. If you typed, e.g., '{' instead of '\{' you should probably delete the '{' by typing '1' now, so that braces don't get unbalanced. Otherwise just proceed. Acceptable delimiters are characters whose `\delcode` is nonnegative, or you can use '`\delimiter <delimiter code>`'.

! Missing number, treated as zero.

*A number should have been here; I inserted '0'. (If you can't figure out why I needed to see a number, look up 'weird error' in the index to *The TeXbook*.)*

In LaTeX this is often caused by a command expecting a number or a length argument but not finding it. You might have forgotten the argument or an opening square bracket in the text might have been taken as the start of an optional argument. For example, the `\` (newline) command takes an optional length argument, so the following will produce this error:

```
... next line\
[Horatio:] ...
```

! Not a letter.

Letters in `\hyphenation` words must have `\lccode>0`.

One or more characters in the argument to the `\hyphenation` command should not be there.

! Number too big.

I can only go up to $2147483647 = '17777777777 = "7FFFFFFF$, so I'm using that number instead of yours.

These all represent the same value, firstly in decimal, secondly in octal, and lastly in hexadecimal notations.

! Output loop-- ...consecutive dead cycles.

I've concluded that your `\output` is awry; it never does a `\shipout`, so I'm shipping `\box255` out myself. Next time increase `\maxdeadcycles` if you want me to be more patient!

TeX appears to be spinning its wheels, doing nothing.

! Overfull \hbox (...pt too wide).

This is a warning that TeX couldn't cram some text into the allotted horizontal space.

! Overfull \vbox (...pt too high).

This is a warning that TeX couldn't find a good place for a pagebreak, so it has put too much onto the current page.

! Paragraph ended before ...was complete.

I suspect you've forgotten a '}', causing me to apply this control sequence to too much text. How can we recover? My plan is to forget the whole thing and hope for the best.

Either a blank line or a `\par` command appeared in the argument to a macro that cannot handle paragraphs (e.g., a macro that was defined using `\newcommand*`).

! Please type a command or say '`\end`'.

This is the message that causes me the most trouble. My computer always ignores whatever I say to it and even typing `\end` has no effect. What I usually do, after having tried a few variations like `\end{document}`, is to kill the program by whatever means the operating system provides. Some other possible responses include:

- Type `\stop`
- Type `\csname @@end\endcsname` (LaTeX stores TeX's version of `\end` as `\@@end`)
- Type some macro that you think is unknown, perhaps `\qwertyuiod`, then respond to the error message: `Undefined control sequence`.
- Sometimes nothing works except killing the program. If you are sure you know how to kill a program, try the following highly contrived code:

```
\documentclass{article}
\newif\ifland
\newif\ifprint
\newcommand{\Xor}[2]{\ifx #1 #2}
\begin{document}
% \Xor{\ifland}{\ifprint}% try uncommenting this
\iffalse
\end{document}
```

! Runaway argument. or Runaway definition. or Runaway preamble. or Runaway text.

! Sorry, but I'm not programmed to handle this case.

I'll just pretend that you didn't ask for it. If you're in the wrong mode, you might be able to return to the right one by typing '`I`' or '`I$`' or '`I\par`'.

! TeX capacity exceeded, sorry [...].

If you absolutely need more capacity, you can ask a wizard to enlarge me.

This is dealt with in more detail below.

! Text line contains an invalid character.

A funny symbol that I can't read has just been input. Continue, and I'll forget that it ever happened.

The input file contains a nonprinting (control) character; only printing characters should be in the file. Some programs, like word processors, insert invisible characters into their output file. If you have used one of these to prepare your input file, make sure you save it as a plain text file (also known as an ASCII file).

- ! That makes 100 errors; please try again.
- ! This can't happen (...).
I'm broken. Please show this to someone who can fix can fix
This is the message you should never see!
- ! Too many }'s.
You've closed more groups than you opened. Such booboos are generally harmless, so keep going.
There are more closing braces (}) than there are opening braces ({).
- ! Unbalanced output routine.
Your sneaky output routine has fewer real {'s than }'s. I can't handle that very well; good luck.
A package or class has done nasty things to one of LaTeX's most delicate parts — the output routine.
- ! Unbalanced write command.
On this page there's a \write with fewer real {'s than }'s. I can't handle that very well; good luck.
- ! Undefined control sequence.
The control sequence at the end of the top line of your error message was never \def'ed. If you have misspelled it (e.g., '\hobx'), type 'I' and the correct spelling (e.g., 'I\hbox'). Otherwise just continue, and I'll forget whatever was undefined.
TeX has come across a macro name that it does not know about. Perhaps you misspelled it, or it is defined in a package you did not include. Another possibility is that you used a macro name that included the @ character without enclosing it between \makeatletter and \makeother (see §E.4). In this case TeX would think that the name was just the portion up to the @.
- ! Underfull \hbox (badness ...).
This is a warning. There might be some extra horizontal space. It could be caused by trying to use two \newline or \\ commands in succession with nothing intervening, or by using a \linebreak command or typesetting with the \sloppy declaration.
- ! Underfull \vbox (badness ...).
This is a warning that TeX couldn't find a good place for a pagebreak, so it produced a page with too much whitespace on it.
- ! Use of ...doesn't match its definition.
If you say, e.g., '\def\al{...}', then you must always put 'I' after '\a', since the control sequence names are made up of letters only. The macro here has not been followed by the required stuff, so I'm ignoring it.
- ! You can't use '...' in '...'.

This often manifests itself in the form

You can't use '\spacefactor' in vertical mode

the cause is usually trying to use a macro with @ in its name, typically in the preamble (see §E.4). The solution is to enclose the macro within \makeatletter and \makeatother.

Another version is

You can't use 'macro parameter character #' in ... mode.

In this case you have used a naked # in ordinary text; it can only be used in the definition of a macro. In ordinary text you have to use \#.

F.1.1 TeX capacity exceeded

TeX has run out of computer space before it finished processing your document. The most likely cause is an error in the input file rather than there really not being enough space — I have processed documents consisting of more than 1400 pages without any capacity problems.

You can very easily make TeX run out of space. Try inputting this:

```
\documentclass{article}
\newcommand*{\fred}{Fred}          % should print 'Fred'
% try to make it print 'Frederick' instead
\renewcommand{\fred}{\fred erick}
\begin{document}
  His name is \fred.
\end{document}
```

and TeX will tell you that it has run out of stack space:

```
! TeX capacity exceeded, sorry [input stack size=15000].
\fred ->\fred
      erick
1.5 His name is \fred
.
```

No pages of output.

Transcript written on errors.log.

The offending code above tries to define \fred in terms of itself, and TeX just keeps chasing round and round trying to pin down \fred until it is exhausted.

At the end of the log file for a run, TeX prints the memory space it has used. For example:

```
Here is how much of TeX's memory you used:
2432 strings out of 60985
29447 string characters out of 4940048
106416 words of memory out of 8000001
5453 multiletter control sequences out of 10000+65535
8933 words of font info for 31 fonts out of 1000000 for 1000
276 hyphenation exceptions out of 1000
26i,11n,21p,210b,380s stack positions out of
      15000i,4000n,6000p,200000b,40000s
```

The error message says what kind of space it exhausted (input stack size in the example above). The most common are:

buffer size Can be caused by too long a section or caption title appearing in the ToC, LoF, etc. Use the optional argument to produce a shorter entry.

exception dictionary There are too many words listed in `\hyphenation` commands. Remove any that are not actually used and if that doesn't work, remove the less common ones and insert `\-` in the words in the text.

hash size The document defines too many command names and/or uses too many cross-referencing `\labels`.

input stack size Typically caused by a self-referencing macro definition.

main memory size There are three main things that cause TeX to run out of main memory:

- Defining a lot of very long complicated macros.
- Having too many `\index` or `\glossary` commands on a page.
- Creating such a complicated page that TeX cannot hold all it needs to process it.

The solution to the first two problems is to simplify and eliminate. The third is more problematic.

Large tabulars, arrays and pictures (the `\qbezier` command is a memory hog) can gobble up memory. A queue of floats also demands memory space. Try putting a `\clearpage` just before the place where the error occurs and if it still runs out of room then there may be an error in your file, otherwise you did exceed the capacity.

If you have a long paragraph or a long `verbatim` environment try breaking it up, as TeX keeps these in memory until it is ready to typeset them. If you have a queue of floats make sure that you have done your best to help LaTeX find a way to output them (see §??) and try adding `\clearpage` at appropriate places to flush the queue.

pool size Typically caused by having too many characters in command names and label names.

It can also be caused by omitting the right brace that ends the argument of a counter command (`\setcounter` or `\addtocounter`) or of a `\newenvironment` or `\newtheorem` command.

save stack size This happens if commands or environments are nested too deeply. For instance a picture that contains a picture that includes a `\multipt` that includes a picture that includes a ...

F.2 LATEX ERRORS

LaTeX errors introduce themselves differently from those that TeX finds. For example, if you ever happened to use the `\caption` command outside a float, like:

```
\caption{Naked}
```

you would get the message:

```
! LaTeX Error: \caption outside float.
```

```
See the LaTeX manual or LaTeX Companion for explanation.
```

```
Type H <return> for immediate help.
```

```
...
```

```
1.624 \caption
      {Naked}
```

```
?
```

If you then typed H in response you would get the following helpful message:

```
You're in trouble here. Try typing <return> to proceed.
```

```
If that doesn't work, type X <return> to quit.
```

```
?
```

The majority of LaTeX's help messages follow this formula, so I have not noted them in the alphabetical listing below.

`\<` in mid line

A `\<` appears in the middle of a line in a tabbing environment; it should only come at the start of a line.

... allowed only in math mode

You have tried to use a math command in a non-math mode.

Bad `\line` or `\vector` argument

A `\line` or `\vector` has a negative length argument or the slope is not within the allowed range.

Bad math environment delimiter

If in math mode there is a start math mode command like `\(` or `\[` or if in LR or paragraph mode there is an end math mode command like `\)` or `\]`. The basic problem is unmatched math mode delimiters or unbalanced braces.

`\begin{...}` ended by `\end{...}`

The name of the `\begin` argument is not the same as the name of the `\end` argument. This could be caused by a typo or a missing `\end`.

Can only be used in the preamble

Some commands can only be used in the preamble, such as `\usepackage`, but there was one of these after the `\begin{document}`.

\caption outside float

You have used the `\caption` command outside a float, such as a figure or table environment.

Command \... already defined or name \end... illegal

This is normally because you have used one of the `\new...` commands to define a command or environment or counter name that has already been used; remember also that defining an environment `foo` automatically defines the macro `\foo`. Either choose a new name or use the appropriate `\renew...`; also, see §8.1. In the unlikely event that you have tried to define something beginning with `\end...`, choose another name.

Command ... invalid in math mode

You have used a non-math command in math mode.

Command ... not provided in base LaTeX2e

You have tried to use a symbol that is not part of basic LaTeX. Try loading the `latexsym` or `amsfonts` package which might define the symbol.

Counter too large

You are using a non-numeric counter representation, such as letters or footnote symbols, and the counter has exceeded the allowed number (for example there are only 26 alphabetic characters).

Environment ... undefined

LaTeX does not know the name of the argument of a `\begin`. You have probably misspelled it.

File not found. Type X to quit or <RETURN> to proceed or enter new name (Default extension: ...)

LaTeX cannot find the file you requested. The extension `tex` results from a problematic `\input` or `\include`; the extension `sty` from a `\usepackage` and an extension `cls` from a `\documentclass`.

Float(s) lost

Usually caused by having too many `\marginpars` on a page.

Illegal character in array argument

There is an illegal character in the argument of an `array` or `tabular` environment, or in the second argument of a `\multicolumn` command.

\include cannot be nested

A file that is `\included` cannot `\include` any other files.

\LoadClass in package file

This is an error in a package file you are using (you can only use `\LoadClass` in a class file). Complain to the author.

Lonely `\item` -- perhaps a missing list environment

An `\item` command appears to be outside any list environment.

Missing `\begin{document}`

If you haven't forgotten `\begin{document}` then there is something wrong in the preamble as LaTeX is trying to typeset something before the document starts. This is often caused by missing the backslash from a command, misplaced braces round an argument, a stray character, or suchlike.

Missing `@-exp` in array argument

The `@` character is not followed by an `@-expression` in the argument of an array or tabular environment, or in the second argument of a `\multicolumn` command.

Missing `p-arg` in array argument

There is a `p` not followed by braces in the argument of an array or tabular environment, or in the second argument of a `\multicolumn` command.

No counter ... defined

The argument to a `\setcounter` or `\addtocounter` command, or in the optional argument to `\newcounter` or `\newtheorem` is not the name of a counter. Perhaps you misspelled the name. However, if the error occurred while an aux file was being read then you might well have used a `\newcounter` in an `\included` file.

No room for a new ...

TeX is limited in the numbers of different things it can handle. You might not recognize the thing that the message mentions as some of them are hidden in LaTeX. The LaTeX counter uses a TeX `\count` for example, and a length is a TeX `\skip`. Most things are limited to a maximum of 256 but there can be no more than 16 files open for reading and 16 for writing.

No `\title` given

You did not put a `\title` command before using `\maketitle`.

Not in outer par mode

There is a float (e.g., a figure or a `\marginpar`) in math mode or in a parbox (e.g., in another float).

Option clash for ...

The same package was used twice but with different options. It is possible for one package to use another package which might be the cause if you can't see anything obvious.

Page height already too large

You are trying to use `\enlargethispage` when the page is already too large.

`\pushtabs` and `\poptabs` don't match

There are unmatched `\pushtabs` and `\poptabs` in a tabbing environment.

`\RequirePackage` or `\LoadClass` in Options Section

This is a problem in a class or package file. Complain to the author.

Something's wrong -- perhaps a missing `\item`

This can be caused by not starting a list environment, such as `itemize` with a `\item` command, or by omitting the argument to the `thebibliography` environment. There are many other non-obvious causes, such as calling some macro that ends up using `\addvspace` or `\addpenalty` when not in `vmode`.

Suggested extra height (...) dangerously large

LaTeX is concerned that you are trying to increase the page size too much with the `\enlargethispage` command.

Tab overflow

In the tabbing environment a `\=` has exceeded LaTeX's maximum number of tab stops.

The file needs format ... but this is ...

The document uses a document class or package that is not compatible with the version of LaTeX you are using. If you are using only standard files then there is a problem with your LaTeX installation.

There's no line to end here

A `\newline` or `\\` appears in vertical mode, for example between paragraphs. Or perhaps you have tried to put `\\` immediately after an `\item` to start the text on a new line. If this is the case, then try this:

```
\item \mbox{} \\
...
```

This may be a LaTeX bug

This is a message you don't want to see as it is produced by the output routine — perhaps the most obscure part of LaTeX. It is probably due to an earlier error. If it is the first error, though, and you can't see anything wrong, ask for somebody's help.

Too deeply nested

There are too many list environments nested within each other. At least four levels are usually available but some list environments are not obvious (for example the quotation environment is actually a list).

Too many columns in `eqnarray` environment

An `eqnarray` environment has three `&` column separators with no `\\` between.

Too many unprocessed floats

There may be too many `\marginpars` to fit on a page, but it's more likely that LaTeX hasn't been able to find locations for printing all the figures or tables. If one float cannot be placed, all later ones are saved until LaTeX runs out of storage space. See §?? for details on how LaTeX decides to place a float.

Two `\documentclass` commands

Your document has two `\documentclass` commands; only one is permitted.

Two `\LoadClass` commands

This is an error in the class file. Complain to the author.

Undefined `tab` position

A `\>`, `\+`, `\-`, or `\<` tabbing command is trying to move to a tab position that has not been defined by a `\=` command.

Unknown option ... for class/package ...

You have asked for an option that the class or package does not know about. Perhaps you have misspelled something, or omitted a comma.

`\usepackage` before `\documentclass`

In general, the `\usepackage` command can only be used in the preamble.

`\verb` ended by end of line

The argument of a `\verb` command runs past the end of the line. Perhaps you forgot to put in the correct ending character.

`\verb` illegal in command argument

A `\verb` cannot be part of the argument to another command.

F.3 LATEX WARNINGS

Most warnings are given at the point in the document where a potential problem is discovered, while others are output after the document has been processed.

For example, the following code

```
... \ref{joe}... \cite{FRED96} ...
```

may produce warnings like

```
Latex Warning: Reference 'joe' on page 12 undefined
                on input line 881.
```

```
Latex Warning: Citation 'FRED96' on page 12 undefined
                at lines 890--897.
```

during the document processing, and then at the end there will also be the warning:

```
LaTeX Warning: There were undefined references.
```

Some warning messages pinpoint where a problem might lie, as in the citation warning above, while others make no attempt to do so. In the alphabetical listing that follows I have not included such information, even if it is supplied.

Citation ... on page ... undefined

The key in a `\cite` command was not defined by any `\bibitem`.

Citation ... undefined

The key in a `\cite` command was not defined by any `\bibitem`.

Command ... invalid in math mode

The command is not permitted in math mode but was used there anyway. Remember that font size commands and `\boldmath` or `\unboldmath` cannot be used in math mode.

Float too large for page by ...

A float (table or figure) is too tall to fit properly on a page by the given amount. It is put on a page by itself.

Font shape ... in size ... not available size ... substituted

You asked for a font size that was not available. The message also says what font is being used instead.

Font shape ... undefined using ... instead

You asked for a font shape that was not available. The message also says what font is being used instead.

h float specifier changed to ht or !h float specifier changed to !ht

A float has an optional `h` or `!h` argument but as it wouldn't fit on the current page it has been moved to the top of the next page.

Label ... multiply defined

Two `\label` or `\bibitem` commands have the same argument (at least during the previous LaTeX run).

Label(s) may have changed. Rerun to get cross-references right

This is only output at the end of the run.

One of the numbers printed by `\cite`, `\ref` or `\pageref` commands might be incorrect because the correct values have changed since the preceding LaTeX run.

Marginpar on page ... moved

A `\marginpar` was moved down the page to avoid overwriting an earlier one. The result will not be aligned with the `\marginpar` call.

No `\author` given

There is no `\author` command before calling `\maketitle`.

No positions in optional float specifier. Default added (so using 'tbp')

You have used an empty optional argument to a float, for example:

```
\begin{figure}[]
```

so it has used

```
\begin{figure}[tbp]
```

instead.

Optional argument of `\twocolumn` too tall on page ...

The contents of the optional argument to `\twocolumn` was too long to fit on the page.

`\oval`, `\circle`, or `\line` size unavailable

You have asked for too large (or too small) an oval or circle, or too short a line, in a picture.

Reference ... on page ... undefined

The argument of a `\ref` or `\pageref` has not been defined on the preceding run by a `\label` command.

Size substitutions with differences up to ... have occurred.

Please check the transcript file carefully and redo the format generation if necessary!

This is only output at the end of the run.

Some fonts have had to be used as substitutes for requested ones and they are a different size.

Some shapes were not available, defaults substituted

This is only output at the end of the run.

At least one font had to be substituted.

Text page ... contains only floats

The page should have included some textual material but there was no room for it.

There were multiply defined labels

This is only output at the end of the run.

Two or more `\label` or `\cite` commands had the same argument.

There were undefined references

This is only output at the end of the run.

There was at least one `\ref` or `\pageref` or `\cite` whose argument had not been defined on the preceding run by a `\label` or `\biblabel` command.

Unused global option(s) [...]

The listed options were not known to the document class or any packages you used.

You have requested release ... of LaTeX but only release ... is available

You are using a class or package that requires a later release of LaTeX than the one you are using. You should get the latest release.

You have requested version ... of class/package ... but only version ... is available

You (or the class or one of the packages you are using) needs a later release of a class or package than the one you are using. You should get the latest release.

F.4 CLASS ERRORS

The class errors introduce themselves differently from those that LaTeX finds. Instead of starting with

`! LaTeX Error:`

the class errors start with

`! Class memoir Error:`

After that, it is indistinguishable from a LaTeX error. For example, if you ever happened to input the next line as line 954 in your document you would get the error message that follows

`\sidecapmargin{either}`

`! Class memoir Error: Unrecognized argument for \sidecapmargin.`

See the memoir class documentation for explanation.

Type H <return> for immediate help.

...

1.954 `\sidecapmargin{either}`

?

If you then typed H (or h) in response you would get the following helpful message:

Try typing <return> to proceed.

If that doesn't work, type X <return> to quit.

?

The majority of the help messages follow this formula, so I have not noted them in the alphabetical listing below.

... is negative

The value is negative. It should be at least zero.

... is not a counter

An argument that should be the name of a counter is not.

... is zero or negative

The value must be greater than zero.

>{...} at wrong position: token ignored

A >{...} in the argument to an array or tabular is incorrectly placed and is being ignored.

<{...} at wrong position: changed to !{...}

A <{...} in the argument to an array or tabular is incorrectly placed. It has been changed to !{...} instead.

A pattern has not been specified

You are trying to use the patverse or patverse* environment without having first defined a pattern.

Argument to `\setsidecappos` is not t or c or b

The argument will be assumed to be c.

Argument to `\overridesidecapmargin` neither left nor right

The argument to `\overridesidecapmargin` must be either left or right. The attempted override will be ignored.

Cannot change a macro that has delimited arguments

You are using `\patchcmd` on a macro that has delimited arguments.

Empty preamble: 'l' used

The argument to an array or tabular is empty. The specification {l} is being used instead.

Font command ... is not supported

You have tried to use a deprecated font command. Either replace it with the current font command or declaration or use the `\oldfontcommands` class option.

`\footskip` is too large for `\lowermargin` by ...

The `\footskip` is too large for the `\lowermargin`. Either increase the `\lowermargin` or decrease the `\footskip`.

`\headheight` and/or `\headsep` are too large for `\uppermargin` by ...

The sum of the `\headheight` and the `\headsep` is larger than the `\uppermargin`. Either increase the `\uppermargin` or reduce the others.

Illegal pream-token (...): 'c' used

An illegal character is used in the argument to an array or tabular. The 'c' specifier is being used instead (which centers the column).

Index ... outside limits for array ...

Trying to access an index for the array data structure that is not between the low and high indices.

Limits for array ... are in reverse order

The low index is not less than the high index in `\newarray`.

Missing arg: token ignored

The argument to a column specifier for a array or tabular is missing.

No array called ...

You have tried to access an unknown array data structure.

Not defined: ...

You are using `\patchcmd` on a macro that is not defined.

Not redefinable: ...

You are using `\patchcmd` on a macro that it is unable to modify.

Only one column-spec. allowed

There can only be one column specifier in a `\multicolumn`.

Optional argument is not one of: classic, fixed, lines, or nearest.
I will assume the default.

You have provided an unknown name for the optional argument to `\checkthelayout`. The default classic will be used instead.

`\paperheight` and/or `\trimtop` are too large for `\stockheight` by ...

The sum of the `\paperheight` and the `\trimtop` is larger than the `\stockheight`.
Either increase the `\stockheight` or reduce the others.

`\paperwidth` and/or `\trimedge` are too large for `\stockwidth` by ...

The sum of the `\paperwidth` and the `\trimedge` is larger than the `\stockwidth`.
Either increase the `\stockwidth` or reduce the others.

`\spinemargin` and/or `\textwidth` and/or `\foremargin` are too large for
`\paperwidth` by ...

The sum of the `\spinemargin` and the `\textwidth` and the `\foremargin` is larger
than the `\paperwidth`. Either increase the `\paperwidth` or reduce the others.

The combination of argument values is ambiguous. The lengths will
be set to zero

The combination of values in the arguments to one of the commands for page layout
does not make sense.

The 'extrafontsizes' option is required to use the '...pt' option

If you want to use a '...pt' class option greater than 25pt you also have to use the
extrafontsizes option. The class will use the 17pt option.

Unknown document division name (...)

You have used an unknown division name in the argument to `\settocdepth` or
`\setsecnumdepth` and friends. If you haven't mistyped it you will have to use
`\setcounter` instead.

Unknown mark setting type '...' for ...mark

In `\createmark` or `\createplainmark` the mark setting type should have been
left or both or right. The class will use both.

Unknown numbering type ... for ...mark

In `\createmark` the class expected either `shownumber` or `nonumber` for displaying
the number. It will use `shownumber`.

Unrecognized argument for `\sidecapmargin`

The argument to `\sidecaption` should be left or right or inner or outer.

`\uppermargin` and/or `\textheight` and/or `\lowermargin` are too large for `\paperheight` by ...

The sum of the `\uppermargin` and the `\textheight` and the `\lowermargin` is larger than the `\paperheight`. Either increase the `\paperheight` or reduce the others.

You have used the `'*pt'` option but file ... can't be found

You have used the `*pt` option but the corresponding `clo` file can't be found. Check your definitions of `\anyptfilebase` and `\anyptsize`. The `mem10.clo` file will be used instead.

XeTeX is required to process this document

The document needs to be processed via XeTeX. Try using `xelatex` instead of `(pdf)latex`, or try removing any XeTeX packages from the document.

F.5 CLASS WARNINGS

These are introduced by Class memoir Warning:

For example `\addtodef{alf}{\joe}{fred}` will produce a message along the lines of:

Class memoir Warning: `'alf'` is not a macro on input line 91.

while `\addtodef{\joe}{alf}{fred}` might produce:

Class memoir Warning: `'\joe'` is not a macro on input line 97.

The following is an alphabeticised list of the class warnings.

... at index ... in pattern ... is not a digit

The character at the given position in the verse pattern is not a digit.

... is not a macro

Using `\addtodef` or `\addtoiargdef` you have tried to extend the definition of an unknown macro.

... is not an input stream

You are trying to access a non-existent input stream.

... is not an output stream

You are trying to access a non-existent output stream.

Bad `\sidebarmargin` argument

The argument to `\sidebarmargin` is not recognized. The class will use right.

Characters dropped after `\end{...}`

At the end of a verbatim environment there should be no characters after the `\end{...}` on the same line.

Column ... is already defined

The column type has been defined by a previous `\newcolumnntype`.

Counter ... already defined

For information only, the counter in `\providecounter` is already defined.

Do not use `\footnote` in `\maketitle`. Use `\thanks` instead

You cannot use `\footnote` in any of the `\maketitle` elements (i.e., `\title` or `\author` or `\date`) but you can use `\thanks`.

Empty ‘thebibliography’ environment

There are no `\bibitems` in the `thebibliography` environment.

Environment ... already defined

For information only, the environment in `\provideenvironment` is already defined.

Index ... for pattern ... is out of bounds

The index for the verse pattern is either too low or too high.

Input stream ... is already defined

You are trying to use `\newinputstream` to create an already existing input stream.

Input stream ... is not open

You are trying to access or close an input stream that is closed.

Input stream ... is open

You are trying to open an input stream that is already open.

Length ... already defined

For information only, the length in `\providelength` is already defined.

Marginpar on page ... moved by ...

A marginal note has been lowered by the given amount to avoid overwriting a previous note; the moved note will not be aligned with its `\marginpar`. (This is a more informative message than the normal LaTeX one.)

No more to read from stream ...

There is nothing left in the stream to be read.

Optional argument of `\twocolumn` too tall on page ...

The contents of the optional argument to `\twocolumn` was too long to fit on the page.

Output stream ... is already defined

You are trying to use `\newoutputstream` to create an already existing output stream.

Output stream ... is not open

You are trying to access or close an output stream that is closed.

Output stream ... is open

You are trying to open an output stream that already open.

Redefining primitive column ...

The argument to `\newcolumntype` is one of the basic column types.

Stream ... is not open

You are trying to access a stream, either input or output, that is closed.

The ... font command is deprecated. Use ... or ... instead

You are using a deprecated font command. Consider using one of the alternatives.

The counter will not be printed. The label is: ...

The optional *<style>* argument to the `enumerate` environment does not include one of the special characters.

Undefined index file ...

You are trying to add an index entry to an unknown `idx` file.

Unknown `toclevel` for ...

The division name you have used for `\settocdepth` is not recognized.

`\verb` may be unreliable inside `tabularx`

A `\verb` in a `tabularx` may work, but may not.

X columns too narrow (table too wide)

The width of the X columns in a `tabularx` had to be made too narrow.

G

Comments

G.1 ALGORITHMS

Over time we may use this section to explain, or list some of the algorithms for some of the macros in the class. The information may be useful to some.

G.1.1 Autoadjusting `\marginparwidth`

This algorithm is used within `\fixthelayout` unless the user have used `\setmarginnotes`.

```
if twocolumn then
  marginparwidth = min{inner margin,outer margin}
else
  if twoside then
    if marginpar always left or always right then
      marginparwidth = min{inner margin,outer margin}
    else if marginpar in outer margin then
      marginparwidth = outer margin
    else if marginpar in inner margin then
      marginparmargin = inner margin
    end if
  else
    if marginpar in left margin then
      marginparwidth = inner margin
    else
      marginparwidth = outer margin
    end if
  end if
end if
marginparwidth = marginparwidth - 2marginparsep
if marginparwidth < 1pt then
  marginparwidth = 1pt
end if
```

Notes

CHAPTER 2 LAYING OUT THE PAGE

[Put no mark ... finally printed] ([page 7](#)) This manual uses both footnotes and endnotes. For identifying the endnotes I have used the 'words' method for identifying the parent location of an endnote, so as not to start confusing the reader with two sets of note marks in the body of the text. Typically either footnotes or endnotes are used, not both, so the question of distinguishing them does not normally arise.

Bibliography

CTAN is the Comprehensive TeX Archive Network. Information on how to access CTAN is available at <http://www.tug.org>.

- [AHK90] Paul W. Abrahams, Kathryn Hargreaves and Karl Berry. *TeX for the Impatient*. Addison-Wesley, 1990. (Available at <ftp://tug.org/tex/impatient>)
- [Ars99] Donald Arseneau. *The url package*. February, 1999. (Available from CTAN via </macros/latex/contrib/url/>)
- [Ars01a] Donald Arseneau. *The titleref package*. April, 2001. (Available from CTAN via </macros/latex/contrib/titleref/>)
- [Ars01b] Donald Arseneau. *The chapterbib package*. September, 2001. (Available from CTAN via </macros/latex/contrib/cite/>)
- [Ars07] Donald Arseneau. *The framed package* v0.95. October, 2007. (Available from CTAN via </macros/latex/contrib/framed/>)
- [Ber02] Jens Berger. *The titlesec and titletoc packages*. September, 2002. (Available from CTAN via </macros/latex/contrib/titlesec/>)
- [Bez99] Javier Bezos. *The titlesec and titletoc packages*. February, 1999. (Available from CTAN via </macros/latex/contrib/titlesec/>)
- [Bir04] Derek Birdsall. *notes on book design*. Yale University Press, 2004. ISBN 0-300-10347-6.
- [Bra97] Johannes Braams. *The alltt environment*. June, 1997. (Available as `alltt.dtx` and `alltt.ins` from CTAN via </macros/latex/base/>)
- [Bri99] Robert Bringhurst. *The Elements of Typographic Style*. Hartley & Marks, second edition, 1999. ISBN 0-88179-033-8.
- [Car14] David Carlisle. *The delarray package*. October, 2014. (Available from CTAN via </macros/latex/required/tools/>)
- [Car95] David Carlisle. *The afterpage package*. October, 1995. (Available from CTAN via </macros/latex/required/tools/>)
- [Car98b] David Carlisle. *The longtable package*. May, 1998. (Available from CTAN via </macros/latex/required/tools/>)

- [Car98c] David Carlisle. *The enumerate package*. August, 1998. (Available from CTAN via [/macros/latex/required/tools/](#))
- [Car16] David Carlisle. *The tabularx package*. February, 2016. (Available from CTAN via [/macros/latex/required/tools/](#))
- [CR99] David Carlisle and Sebastian Rahtz. *The graphicx package*. February, 1999. (Available from CTAN via [/macros/latex/required/graphics/](#))
- [Car14] David Carlisle. *The dcolumn package*. May, 2001. (Available from CTAN via [/macros/latex/required/tools/](#))
- [Car04] David Carlisle. *The textcase package*. October, 2004. (Available from CTAN in [/macros/latex/contrib/textcase](#))
- [Car05] David Carlisle. *Packages in the graphics bundle* (includes the color package). November, 2005. (Available from CTAN via [/macros/latex/required/graphics/](#))
- [CB99] Warren Chappell and Robert Bringhurst. *A Short History of the Printed Word*. Hartley & Marks, 1999. ISBN 0-88179-154-7.
- [CH88] Pehong Chen and Michael A. Harrison. ‘Index Preparation and Processing’. *Software: Practice and Experience*, 19:8, pp. 897–915, September, 1988. (Available from CTAN via [/indexing/makeindex/paper/](#))
- [Chi93] *The Chicago Manual of Style*, Fourteenth Edition. The University of Chicago, 1993. ISBN 0-226-10389-7.
- [Dal99a] Patrick W. Daly. *Natural Sciences Citations and References*. May, 1999. (Available from CTAN via [/macros/latex/contrib/natbib/](#))
- [Dal99b] Patrick W. Daly. *Customizing Bibliographic Style Files*. August, 1999. (Available from CTAN via [/macros/latex/contrib/custom-bib](#))
- [Dow96] Geoffrey Dowding. *Finer Points in the Spacing & Arrangement of Type*. Hartley & Marks, 1996. ISBN 0-88179-119-9.
- [Dow00] Michael J. Downes. *The patchcmd package*. July, 2000. (Available from CTAN via [/macros/latex/contrib/patchcmd/](#))
- [Eij92] Victor Eijkhout. *TeX by Topic*. Addison-Wesley, 1992. ISBN 0-201-56882-9. (Available from <http://www.eijkhout.net/tbt/>).
- [Eij99] Victor Eijkhout. *comment.sty*. October, 1999. (Available from CTAN via [/macros/latex/contrib/comment/](#))
- [Fai00] Robin Fairbairns. *footmisc — a portmanteau package for customising footnotes in LaTeX2e*. March, 2000. (Available from CTAN via [/macros/latex/contrib/footmisc/](#))
- [FAQ] Robin Fairbairns. *The UK TeX FAQ*. (Available from CTAN via <http://faq.tug.org/>)
- [Far90] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design — A Practical Guide*. Academic Press, 2 edition, 1990.
- [FP80] I. D. Faux and M. J. Pratt. *Computational Geometry for Design and Manufacture*. Ellis Horwood, 1980.

- [Fea16] Simon Fear. *Publication quality tables in LaTeX*. April, 2016. (Available from CTAN via [/macros/latex/contrib/booktabs](#))
- [Fly98] Peter Flynn. *Formatting Information: A Beginner's Introduction to Typesetting with LaTeX2*. 2002. (Available from CTAN via [/info/beginlatex/](#))
- [GM⁺07] Michel Goossens, Frank Mittelbach, et al. *The LaTeX Graphics Companion: Second edition*. Addison-Wesley, 2007. ISBN 0-321-50892-0.
- [GR99] Michel Goossens and Sebastian Rahtz (with Eitan Gurari, Ross Moore and Robert Suitor). *The LaTeX Web Companion: Integrating TeX, HTML and XML*. Addison-Wesley, 1999. ISBN 0-201-43311-7.
- [Hoe98] Alan Hoenig. *TeX Unbound: LaTeX and TeX strategies for fonts, graphics, and more*. Oxford University Press, 1998. ISBN 0-19-509686-X.
- [Jon95] David M. Jones. *A new implementation of LaTeX's indexing commands*. September, 1995. (Available from CTAN via [/macros/latex/contrib/camel](#))
- [Keh98] Roger Kehr. *xindy: A flexible indexing system*. February, 1998. (Available from CTAN via [/indexing/xindy/](#))
- [Ker07] Uwe Kern. *Extending LaTeX's color facilities: the xcolor package*. January, 2007. (Available from CTAN via [/macros/latex/contrib/xcolor/](#))
- [Kha10] Vafa Khalighi. *The Bidi package*. 2010. (Available from CTAN via [/macros/latex/contrib/bidi/](#))
- [Knu84] Donald E. Knuth. *The TeXbook*. Addison-Wesley, 1984. ISBN 0-201-13448-9.
- [Knu86] Donald E. Knuth. *TeX: The Program*. Addison-Wesley, 1986. ISBN 0-201-13437-3.
- [Knu87] Donald E. Knuth. *Computer Modern Typefaces*. Addison-Wesley, 1987. ISBN 0-201-134446-2.
- [Knu92] Donald E. Knuth. *The METAFONT Book*. Addison-Wesley, 1992. ISBN 0-201-13444-6.
- [Lam94] Leslie Lamport. *LaTeX: A Document Preparation System*. Addison-Wesley, 1994. ISBN 0-201-52983-1.
- [LEB04] Leslie Lamport, Victor Eijkhout and Johannes Braams. *NTG document classes for LaTeX version 2e*. June, 2004. (Available from CTAN via [/macros/latex/contrib/ntgclass/](#))
- [Lea03] Christopher League. *TeX support for the FontSite 500 CD*. May 2003. (Available from <http://contrapunctus.net/fs500tex>)
- [Leh04] Philipp Lehman. *The Font Installation Guide*. December 2004. (Available from CTAN via [/info/Type1fonts/fontinstallationguide](#))
- [Leu92] Mary-Claire van Leunen. *A Handbook for Scholars*. Oxford University Press, 1992. ISBN 0-19-506954-4.
- [Lon91] F. W. Long. *multind*. August, 1991. (Available from CTAN as [/macros/latex209/contrib/misc/multind.sty](#))

- 230

- [Rahtz02] Sebastian Rahtz. *Hypertext marks in LaTeX*. May, 2002. Now maintained and frequently updated by Heiko Oberdiek. (Available from CTAN via [/macros/latex/contrib/hyperref/](#))
- [Rec97] Keith Reckdahl. *Using Imported Graphics in LaTeX2e*. December, 1997. Updated in 2006. (Available from CTAN via [/info/epslatex.pdf](#))
- [Rei07] Edward M. Reingold. ‘Writing numbers in words in TeX’. *TUGboat*, 28, 2 pp 256–259, 2007.
- [RBC74] W. W. Rouse Ball and H. S. M. Coxeter. *Mathematical Recreations and Essays*. University of Toronto Press, twelfth edition, 1974.
- [SW94] Douglas Schenck and Peter Wilson. *Information Modeling the EXPRESS Way*. Oxford University Press, 1994. ISBN 0–19–508714–3.
- [SRR99] Rainer Schöpf, Bernd Raichle and Chris Rowley. *A New Implementation of LaTeX’s verbatim and verbatim* Environments*. December, 1999. (Available from CTAN via [/macros/latex/required/tools/](#))
- [Sch07] Martin Scharrer, *Version Control of LaTeX Documents with svn-multi*. *The PracT_EX Journal*, 3, 2007. ISSN 1556-6994.
- [Sch09] Martin Scharrer, *The svn-multi package*, 2009. (Available from CTAN via [/macros/latex/contrib/svn-multi/](#))
- [Sne04] Maarten Sneep. *The atmosphere in the laboratory: cavity ring-down measurements on scattering and absorption*. Phd thesis, Vrije Universiteit, Amsterdam, 2004.
- [Tal06] Nicola L. C. Talbot. *datetime.sty: Formatting Current Date and Time*. December, 2006. (Available from CTAN via [/macros/latex/contrib/datetime/](#))
- [Thi98] Christina Thiele. ‘Hey — it works: Ornamental rules’. *TUGboat*, vol. 19, no. 4, p 427, December 1998.
- [Thi99] Christina Thiele. ‘The Treasure Chest: Package tours from CTAN’, *TUGboat*, vol. 20, no. 1, pp 53–58, March 1999.
- [TJ05] Kresten Krab Thorup, Frank Jensen (and Chris Rowley). *The calc package — Infix notation arithmetic in LaTeX*. August, 2005. (Available from CTAN via [/macros/latex/required/tools/](#))
- [Tob00] Geoffrey Tobin. *setspace.sty*. December, 2000. (Available from CTAN via [/macros/latex/contrib/setspace/](#))
- [Tsc91] Jan Tschichold. *The Form of the Book*. Lund Humphries, 1991. ISBN 0–85331–623–6.
- [Ume99] Hideo Umeki. *The geometry package*. November, 1999. (Available from CTAN in [/macros/latex/contrib/geometry/](#))
- [Wil00] Graham Williams. *The TeX Catalogue*. (Latest version on CTAN as [/help/Catalogue/catalogue.html](#))
- [Wil93] Adrian Wilson. *The Design of Books*. Chronicle Books, 1993. ISBN 0–8118–0304–X.
- [Wil99b] Peter Wilson. *The tocvsec2 package*. January, 1999. (Available from CTAN via [/macros/latex/contrib/tocvsec2/](#))

- [Wil00a] Peter Wilson. *The epigraph package*. February, 2000. (Available from CTAN via [/macros/latex/contrib/epigraph/](#))
- [Wil00b] Peter Wilson. *LaTeX files for typesetting ISO standards*. February, 2000. (Available from CTAN via [/macros/latex/contrib/isostds/iso/](#))
- [Wil00c] Peter Wilson. *The nextpage package*. February, 2000. (Available from CTAN as [/macros/latex/contrib/misc/nextpage.sty](#))
- [Wil00e] Peter Wilson. *The xtab package*. April 2000. (Available from CTAN via [/macros/latex/contrib/xtab](#))
- [Wil01a] Peter Wilson. *The abstract package*. February, 2001. (Available from CTAN via [/macros/latex/contrib/abstract/](#))
- [Wil01d] Peter Wilson. *The ccaption package*. March, 2001. (Available from CTAN via [/macros/latex/contrib/ccaption/](#))
- [Wil01e] Peter Wilson. *The chngcntr package*. April, 2001. (Available from CTAN via [/macros/latex/contrib/chngcntr/](#))
- [Wil01f] Peter Wilson. *The hanging package*. March, 2001. (Available from CTAN via [/macros/latex/contrib/hanging/](#))
- [Wil01g] Peter Wilson. *The titling package*. March, 2001. (Available from CTAN via [/macros/latex/contrib/titling/](#))
- [Wil01h] Peter Wilson. *The tocbibind package*. April, 2001. (Available from CTAN via [/macros/latex/contrib/tocbibind/](#))
- [Wil01i] Peter Wilson. *The tocloft package*. April, 2001. (Available from CTAN via [/macros/latex/contrib/tocloft/](#))
- [Wil03a] Peter Wilson. *The layouts package* November, 2003. (Available from CTAN in [/macros/latex/contrib/layouts/](#))
- [Wil03b] Peter Wilson. *ledmac: A presumptuous attempt to port EDMAC and TABMAC to LaTeX* November, 2003. (Available from CTAN via [/macros/latex/contrib/ledmac/](#))
- [Wil04a] Peter Wilson. *The bez123 and multiply packages*, April 2004. (Available from CTAN in [/macros/latex/contrib/bez123/](#))
- [Wil04b] Peter Wilson. *The pagenote package* September, 2004. (Available from CTAN via [/macros/latex/contrib/pagenote/](#))
- [Wil07a] Peter Wilson. *Some Examples of Title Pages*. Herries Press, 2007. (Available from CTAN via [/info/latex-samples/TitlePages/](#))
- [Wil07e] Peter Wilson. ‘Glisterings’, *TUGboat*, 28(2):229–232, 2007.
- [Wil08a] Peter Wilson. *The changepage package*. March, 2008. (Available from CTAN via [/macros/latex/contrib/changepage/](#))
- [Wil08b] Peter Wilson. ‘Glisterings’, *TUGboat*, 29(2):324–327, 2008.
- [Wil09a] Peter Wilson. *The fonttable package* April, 2009. (Available from CTAN via [/macros/latex/contrib/fonttable/](#))

- [Wil09b] Peter Wilson (with the assistance of Lars Madsen). *The LaTeX memoir class for configurable book typesetting: source code* July, 2009. (Available from CTAN via [/macros/latex/contrib/memoir/](#))
- [Wil07c] Peter Wilson (with the assistance of Lars Madsen). *The Memoir Class for Configurable Typesetting — User Guide* August, 2009. Regularly updated. (Available from CTAN via [/macros/latex/contrib/memoir/](#))
- [Wil09d] Peter Wilson. *A Few Notes on Book Design* August, 2009. (Available from CTAN via [/info/memdesign/](#))
- [Wil??] Peter Wilson. *A Rumour of Humour: A scientist's commonplace book*. To be published.
- [Wri18] Joseph Wright. *Siunitx — A comprehensive (SI) units package* May, 2018. (Available from CTAN via [/macros/latex/contrib/siunitx](#))
- [Zan98] Timothy Van Zandt. *Documentation for fancybox.sty: Box tips and tricks for LaTeX*, November, 1998. (Available from CTAN via [/macros/latex/contrib/fancybox/](#))

Colophon

This manual was typeset using the LaTeX typesetting system created by Leslie Lamport and the memoir class.

The body text is set 10/12pt on a 33pc measure with Palatino designed by Hermann Zapf, which includes italics and small caps. Other fonts include Sans, Slanted and Typewriter from Donald Knuth's Computer Modern family.