# A User's Guide to the

# Lout

# Document Formatting System

Jeffrey H. Kingston

Version 3.07
April, 1996

 ISBN 0 86758 9515.

# Preface

This User's Guide brings together in one document everything needed for the day-to-day use of Version 3 of the Lout document formatting system.

There are three other documents describing Lout: the Expert's Guide [5], which you need if you want to add new features to Lout; a journal paper on the design and implementation of Lout [3]; and a set of overhead transparencies [4] that cover much the same ground as this Guide. These documents are all distributed with the software.

Lout is distributed free of charge under the GNU Public License. The primary source is directory

ftp://ftp.cs.su.oz.au/jeff/lout

in which may be found the main distribution, which is a gzipped tar file called lout.3.07.tar.gz or similar, and various other things including a PostScript version of this manual. The main distribution is redistributed on the comp.sources.misc newsgroup, which is archived at several sites around the world. The distribution contains source code, libraries, documentation, license, and installation instructions. Distribution by uuencoded email is also available, from me.

A mailing list has been set up for discussion of all topics related to Lout. To subscribe, send email to lout-request@niif.spb.su containing the word subscribe in the Subject line. To post an item, send email to lout@niif.spb.su; it will be forwarded to all subscribers via email. To unsubscribe, send email to lout-request@niif.spb.su containing the word unsubscribe in the Subject line.

Lout began in 1984 as a research project into the design of a high-level language for document formatting. At that time my name for the subject was 'document layout,' and this terminology survives in the names of some of the standard packages, and in 'Lout' itself. The initial design was strongly influenced by Brian W. Kernighan and Lorinda L. Cherry's eqn equation formatter [2], and also by Brian K. Reid's Scribe system [10]. That research phase ended in October 1991 with the first public release of Lout.

Since then the system has been steadily improved and extended. Optimal paragraph breaking and automatic hyphenation were copied from Donald E. Knuth's TeX system [6], and the optimal paragraph breaking algorithm was applied to the problem of producing optimal page breaks. The first implementations of horizontal galleys and optimal page breaking were by my student Gabor Inokai. The strongest influence during this period has come from Lout's users; indeed the number of people who have offered comments and suggestions is so great that it is quite out of my power to acknowledge them individually. I hope that seeing their ideas adopted will be thanks enough.

Jeffrey H. Kingston
Basser Department of Computer Science
The University of Sydney 2006, Australia
jeff@cs.su.oz.au

# Contents

# Chapter 1. The Basics

The Lout document formatting system has been designed with the needs of the ordinary user very much in mind. Although the features of Lout are virtually endless, and include mathematical equations, diagrams made from lines and shapes, bibliographic databases, and so on, the system is very simple to use.

## 1.1. Getting started

Suppose you want to produce the following little document:

> **Introduction by W. J. Harvey**
>
> For Virginia Woolf, *Middlemarch* was 'the magnificent book which for all its imperfections is one of the few English novels written for grown-up people.'
>
> She was, no doubt, thinking of George Eliot's unblinking but compassionate delineation of her characters, of the subtlety of psychological analysis and the maturity of moral comment which underlie this complex and varied novel of English provincial life in the early nineteenth century.

Unlike word processing and desktop publishing systems, with Lout you cannot see and edit your document on the screen in this finished form. Instead, you edit an ordinary text file, in which your text is augmented with symbols that mark out the headings, paragraphs, and so on. Although it would be nice to be able to see and edit the finished form, working with a text file and symbols does have some compensating advantages.

The first step in producing your introduction to *Middlemarch* is to use the text editor of your choice to construct this text file:

```
@SysInclude { doc }
@Doc @Text @Begin
@Display @Heading { Introduction by W. J. Harvey }
For Virginia Woolf, @I Middlemarch was ‘the magnificent book
which for all its imperfections is one of the few English novels
written for grown-up people.’
@PP
She was, no doubt, thinking of George Eliot’s unblinking but
compassionate delineation of her characters, of the subtlety of
psychological analysis and the maturity of moral comment which
underlie this complex and varied novel of English provincial
life in the early nineteenth century.
@End @Text
```

Comparing this with the finished form, it’s easy to guess that @I is a symbol that causes the following thing to be printed in italics, and that @PP starts a new paragraph. The other symbols are not much harder.

@SysInclude { doc } instructs Lout to read a *setup file* called doc, in which the symbols are defined. Setup files are the subject of Chapter 4, but you can go a long way without worrying about them. @Doc @Text @Begin and @End @Text have no visible effect, but they must bracket the document as a whole. Again, you don’t have to know what they are for.

That explains everything except the part that produces the heading. It’s an interesting glimpse of the way that Lout’s symbols cooperate with each other:

```
@Display @Heading { Introduction by W. J. Harvey }
```

The @Display symbol does the centring and leaves space above and below, while @Heading switches to a bold font. The braces group the words of the heading together so that these symbols apply to all of it; without them they would apply to just the first word. All this is explained in detail in Sections 1.2 and 1.3.

Once the file is ready, the next step is to get it processed by the Basser Lout interpreter. If the file’s name is intro, the command for this on the Unix[1] operating system is

```
lout intro > intro.ps
```

The output is the PostScript[2] file intro.ps, which is suitable for printing on many laser printers

[1]Unix is a trademark.

[2]PostScript is a trademark of Adobe Systems, Inc.

and other devices. There are programs that show you the result on your screen as well, although you won't be able to edit it there. You can also get plain text output (Section 3.6).

There are a few points that often confuse people as they begin, so we'll treat them briefly now with pointers to later sections where they are done properly.

Some characters are symbols that produce special effects – for example, { and } produce grouping – and to turn off these effects the characters must be enclosed in double quotes: "{" produces {. The complete set of these special characters is

```
/   |   &   {   }   #   @   ^   ~
```

Section 1.5 treats unusual characters in full detail, and you will need to read it before trying to produce \ or ".

Symbols like @Doc and @Text must be separated from each other by one or more spaces, otherwise Lout will think they are part of one symbol. See Section 1.3 for the details.

People familiar with other document formatting systems might expect that leaving a blank line would cause Lout to start a new paragraph; but this is not so, you must use a paragraph symbol. Lout will ordinarily take notice of how many spaces you type between words (Section 1.3, but it is willing to mimic the spacing rules of two other systems, troff and TeX, if you prefer (Section 1.18).

When Lout runs, you might see some error messages beginning with 'unresolved cross reference' – not on file intro above, but on more complicated ones (anything with a footnote, for example). These are nothing to worry about; they just mean that you have to run the lout command again to finish off the complicated things (Section 2.8), and they will gradually go away. Of course, if you see error messages about missing braces, unknown symbols, and so on, you need to revise your file. Lout will tell you the line number of the problem, and how far along the line it is.

***WARNING:*** Lout allows documents to cause arbitrary system commands to be executed. These typically do useful things such as format computer programs and uncompress graphics files, but it is possible for a malicious person to send you a document which includes a command to delete all your files, send abusive mail to the President of the United States in your name, etc. You can protect yourself against this possibility by using the 'safe execution' flag:

```
lout -S suspect.document > out.ps
```

Then no system commands will be executed; instead, Lout will print them so that you can confirm for yourself that they are safe before running again without the flag. These system commands are Lout's only potentially unsafe features, but you also need to worry about whether the resulting PostScript file contains malicious code, since the document may direct Lout to include arbitrary PostScript code in the output. The safe execution of PostScript programs is a matter for PostScript interpreters, not for Lout. For example, the popular Ghostview program has a -safer command line option, which is rumoured to disable unsafe PostScript features, although the manual entry available to the author does not say so.

## 1.2. Objects, symbols, options, and lengths

Lout is not concerned with the exact shapes of individual characters, only with the rectangular areas they occupy:

Biology

When letters join together into a word, the result is a larger rectangle enclosing them all:

Biology

When words join into lines we get even larger rectangles:

Biology is the study of living things.

and so on up through paragraphs and columns to the largest rectangles, which are pages. We call any such rectangle, whether made up of one character, one word, one line, one paragraph, one page, or anything else, an *object*.

We also often say, for example, 'the object @I { Hello world },' referring to a piece of Lout's input as an object. This makes sense because we are anticipating the result produced, in this case the object *Hello world*. It's true that if a line break happens to fall between *Hello* and *world*, the result of @I { Hello world } is not a single rectangle. We answer this by thinking of objects as existing before paragraph breaking rearranges them.

Not everything is an object, however. @I alone is not an object, merely a symbol with the potential of producing an object when given an object to work on. To understand this, ask yourself what rectangle @I alone could possibly represent: there is no such rectangle.

It helps to imagine the assembly of objects taking place before your eyes. Look at

```
Hello
```

and imagine the objects H, e, l, l, o being assembled into the larger object Hello; look at

```
Hello world
```

and imagine Hello and world being assembled into Hello world. When looking at

```
@I { Hello world }
```

you need to imagine the @I symbol consuming the following object, Hello world, and replacing it with the object *Hello world*. Here is another example:

```
@CurveBox { Hello world }
```

The @CurveBox symbol (Section 8.2) consumes Hello world and replaces it with the object

Hello world

This brings us to a basic principle of Lout: *Where you can put one object, you can put any object*. A few examples will show the vast range of possibilities opened up by this:

```
@CurveBox { @I Hello world }
```

produces

*Hello* world

It doesn't bother @CurveBox if one of the words inside it is in italics. Next:

```
@I @CurveBox { Hello world }
```

produces

*Hello world*

The object following @I cannot be just @CurveBox, since that is not an object by itself (it needs to be applied to some object first). So the object following @I is @CurveBox { Hello world }, and it is this that is consumed by @I and modified. The @I symbol is happy to hunt through the object looking for words to italicize. We could go on indefinitely in this way, producing

Hello world

for example by @CurveBox { @CurveBox Hello @CurveBox world }.

Symbols like @CurveBox often have *options*, which are subsidiary symbols that modify the result. For example, @CurveBox has margin and paint options:

```
@CurveBox
   margin { 0.5c }
   paint { lightgrey }
{ Hello world }
```

Options come immediately after the main symbol, before any following object. Each consists of the option name followed by the value we want the option to have, always enclosed in braces. Setting out options on separate lines as we have done above makes them easy to see but is not compulsory (end of line and space are the same to Lout). The result, naturally enough, is a curved box with a 0.5 centimetre margin around its contents, painted light grey:

Hello world

Options are optional: if you leave out an option, Lout supplies a sensible *default* value for it.

Options may be given in any order. They are a very useful way of adding flexibility to symbols without cluttering things up when they aren't needed. They also help with learning: you can learn the basic symbol first and worry about the options later.

Whenever a length is required, as in the margin option above, it may be given using any one of the following seven units of measurement:

| | |
|---|---|
| c | Centimetres |
| i | Inches (1i = 2.54c) |
| p | Points (72p = 1i) |
| m | Ems (12m = 1i) |
| f | 1f is the current font size |
| s | 1s is the current width of a space character |
| v | 1v is the current inter-line spacing |

The first four all define absolute distances and are strictly interchangeable, so use whichever seems convenient. It is traditional to measure font sizes in points; typical sizes are 12p and 10p, but fractional sizes are allowed.

If you use the f unit, the length will depend on the current font size. This can be very useful. For example, the default value of the margin option of @CurveBox is 0.3f (0.3 times the current font size). If you use a large font, for example in an overhead transparency, you get a correspondingly large margin without having to ask for it.

The s and v units are less useful. The v unit is used within paragraph symbols (Section 1.8) to ensure that the space between paragraphs widens with the inter-line spacing.

### 1.3. Spaces and braces

Every symbol in Lout either consists entirely of letters (@ is considered to be a letter) or entirely of punctuation characters. Here are some examples of each type:

| *From letters* | *From punctuation* |
|---|---|
| @PP | { |
| margin | } |

Now if two symbols made from letters are run together like this:

```
@CurveBox@I Hello
```
*(wrong!)*

Lout will take this to mean one word or symbol called @CurveBox@I, which is wrong. In the same way, a letter-type symbol cannot be run together with a word. However, punctuation-type symbols can be run together with anything. For example, in

```
@CurveBox{ Hello @I { world }}.
```

Lout understands that @CurveBox and { are separate, and it also sorts out }}. into two right brace symbols and a full stop. It might seem strange at first to treat punctuation and letters so differently, but these rules have been used in computer programming languages for many years, and they work well. This gives us the first use for spaces: to separate letter-type symbols from each other and from words.

To see the second use for spaces, consider two words side by side:

```
Hello world
```

We want this to produce Hello world, so a space between two words in the input must mean a space between them in the result. Apply the golden rule (where you can put one object, you can put any object) and you get this: *a space between two objects in the input produces a space between them in the result*. For example,

```
@CurveBox Hello @CurveBox world
```

produces

Hello world

The space between the two objects @CurveBox Hello and @CurveBox world appears between them in the result; the other two spaces do not separate objects so do not appear in the result.

Two objects may be separated by a number of spaces other than one. If they are separated by no spaces, they will appear immediately adjacent in the result; if separated by two spaces, they will appear two spaces apart; and so on. In English it is correct to leave two spaces between the end of one sentence and the beginning of the next, for example.[1]

Occasionally the two uses for spaces conflict. For example, to produce

Hello world

we need to have no spaces between the two objects, but then Hello and the following @CurveBox would be run together, which will not work. The solution is to use braces:

```
{ @CurveBox Hello }{ @CurveBox world }
```

None of the six spaces in this example lie between two objects. Enclosing one of the two objects in braces is actually sufficient.

However, the main use of braces is to inform Lout that the object within them is to be kept together, so that any nearby symbols are to apply to all of it. For example, leaving the braces out of @I { Hello world } would mean that @I applies only to Hello.

When an object-consuming symbol like @I is followed by an object enclosed in braces, that is the object consumed. For example,

[1]Lout offers two alternative ways to interpret white space between two objects, reminiscent of those used by the troff [9] and LaTeX [7] document formatting systems. Section 1.18 explains these alternatives and how to obtain them.

```
This is @I { absolutely necessary }, since otherwise ...
```

produces

> This is *absolutely necessary*, since otherwise ...

with the object `absolutely necessary` italicized, but not the following comma. If there are no braces, the object consumed is everything up to the next object-separating space:

```
This is @I necessary, since otherwise ...
```

produces

> This is *necessary,* since otherwise ...

with an undesirable italic comma. In practice, this means you can avoid braces only when italicizing a single word with no punctuation attached.

One common pitfall is to use unnecessary braces, like this:

```
@I { @CurveBox { Hello world } }
```
*(bad!)*

Another is to think that all spaces produce space in the result, and so write

```
@I{@CurveBox{Hello world}}
```
*(worse!)*

Use braces only when necessary, and add extra spaces where they do not separate objects, and your documents will be far easier to read while you are working on them. Don't be fooled by the argument that says it doesn't matter because it doesn't affect the final printed result. It matters to *you*, the author.

### 1.4. The empty object

It is possible to produce examples in which an object is clearly missing:

```
{ @I  }
```

The `@I` symbol is supposed to italicize the following object, but in this example there isn't one. A more plausible example is

```
@PP
@PP
```

There are supposed to be paragraph objects between paragraph symbols, but here there aren't.

Wherever an object is clearly missing, Lout inserts an *empty object*, which is a rectangle of size zero by zero that prints as nothing. Here are two other ways to get an empty object:

```
{}    ""
```

Braces always enclose an object, so Lout is obliged to insert an empty object between them; the two double quotes make a word with no characters in it, which is taken to be an empty object.

## 1.5. Characters

The usual way to get characters into a document is simply to type them as we have been doing all along. However, for some characters this is not possible, either because they have some special meaning, as { and } do, or because the keyboard has no button for them. This section explains how to get every possible character.[1] If it exists at all, you will find it here.

First up we have the characters that you get simply by typing them. The characters themselves are shown at the left, and what you type to get them at the right:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| ! | ! | $ | $ | % | % | ’ | ’ |
| ( | ( | ) | ) | * | * | + | + |
| , | , | - | - | 0 | 0 | 1 | 1 |
| 2 | 2 | 3 | 3 | 4 | 4 | 5 | 5 |
| 6 | 6 | 7 | 7 | 8 | 8 | 9 | 9 |
| : | : | ; | ; | < | < | = | = |
| > | > | ? | ? | A | A | B | B |
| C | C | D | D | E | E | F | F |
| G | G | H | H | I | I | J | J |
| K | K | L | L | M | M | N | N |
| O | O | P | P | Q | Q | R | R |
| S | S | T | T | U | U | W | W |
| X | X | Y | Y | Z | Z | [ | [ |
| ] | ] | _ | _ | ‘ | ‘ | a | a |
| b | b | c | c | d | d | e | e |
| f | f | g | g | h | h | i | i |
| j | j | k | k | l | l | m | m |
| n | n | o | o | p | p | q | q |
| r | r | s | s | t | t | u | u |
| v | v | w | w | x | x | y | y |
| z | z | | | | | | |

Next come characters that have buttons but have a special meaning if they are typed directly, and

[1]Technically, how to get every printable character in the ISO-LATIN-1 character set, every character in the Adobe Systems Symbol font, plus the characters “, ”, ﬁ, ﬂ, –, —, •, †, ‡, ƒ, and ⁄. ISO-LATIN-2 and Russian characters are available separately (consult directory software in the Lout distribution). In principle, there is no limit to the characters available, but to go beyond those given in this section requires expertise in defining encoding vectors and fonts [5].

consequently have to be enclosed in double quotes to turn off this meaning:

| | | |
|---|---|---|
| | " " | (space character) |
| " | "\"" | |
| # | "#" | |
| & | "&" | |
| / | "/" | |
| @ | "@" | |
| \ | "\\" | |
| ^ | "^" | |
| { | "{" | |
| \| | "\|" | |
| } | "}" | |
| ~ | "~" | |

If you think you want ", you probably really want “ and ”, for which see below. You can place whole sequences of characters, special or not, inside one pair of double quotes:

| | |
|---|---|
| jeff/includes/su_crest.eps | "jeff/includes/su_crest.eps" |
| "@PP" | "\"@PP\"" |

Next we have some miscellaneous characters which have been deemed sufficiently important to deserve their own symbols:

| | | | |
|---|---|---|---|
| “ | `` | ” | '' |
| – | -- | — | --- |
| • | @Bullet | ∗ | @Star |
| ¶ | @ParSym | § | @SectSym |
| † | @Dagger | ‡ | @DaggerDbl |
| · | @CDot | £ | @Sterling |
| ¥ | @Yen | ƒ | @Florin |
| ° | @Degree | ′ | @Minute |
| ″ | @Second | ◊ | @Lozenge |
| × | @Multiply | ÷ | @Divide |
| © | @CopyRight | ® | @Register |
| ™ | @TradeMark | | |

Next we have the complete ISO-LATIN-1 character set, whose members you get with the @Char symbol followed by the name of the character you want:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | @Char space | ! | @Char exclam | " | @Char quotedbl | # | @Char numbersign |
| $ | @Char dollar | % | @Char percent | & | @Char ampersand | ’ | @Char quoteright |
| ( | @Char parenleft | ) | @Char parenright | * | @Char asterisk | + | @Char plus |
| , | @Char comma | - | @Char hyphen | . | @Char period | / | @Char slash |
| 0 | @Char zero | 1 | @Char one | 2 | @Char two | 3 | @Char three |
| 4 | @Char four | 5 | @Char five | 6 | @Char six | 7 | @Char seven |
| 8 | @Char eight | 9 | @Char nine | : | @Char colon | ; | @Char semicolon |
| < | @Char less | = | @Char equal | > | @Char greater | ? | @Char question |
| @ | @Char at | A | @Char A | B | @Char B | C | @Char C |
| D | @Char D | E | @Char E | F | @Char F | G | @Char G |
| H | @Char H | I | @Char I | J | @Char J | K | @Char K |
| L | @Char L | M | @Char M | N | @Char N | O | @Char O |
| P | @Char P | Q | @Char Q | R | @Char R | S | @Char S |
| T | @Char T | U | @Char U | V | @Char V | W | @Char W |
| X | @Char X | Y | @Char Y | Z | @Char Z | [ | @Char bracketleft |
| \ | @Char backslash | ] | @Char bracketright | ^ | @Char asciicircum | _ | @Char underscore |
| ‘ | @Char quoteleft | a | @Char a | b | @Char b | c | @Char c |
| d | @Char d | e | @Char e | f | @Char f | g | @Char g |
| h | @Char h | i | @Char i | j | @Char j | k | @Char k |
| l | @Char l | m | @Char m | n | @Char n | o | @Char o |
| p | @Char p | q | @Char q | r | @Char r | s | @Char s |
| t | @Char t | u | @Char u | v | @Char v | w | @Char w |
| x | @Char x | y | @Char y | z | @Char z | { | @Char braceleft |
| \| | @Char bar | } | @Char braceright | ~ | @Char asciitilde | ı | @Char dotlessi |
| ` | @Char grave | ´ | @Char acute | ˆ | @Char circumflex | ˜ | @Char tilde |
| ¯ | @Char macron | ˘ | @Char breve | ˙ | @Char dotaccent | ¨ | @Char dieresis |
| ˚ | @Char ring | ¸ | @Char cedilla | ˝ | @Char hungarumlaut | ˛ | @Char ogonek |
| ˇ | @Char caron | | @Char space | ¡ | @Char exclamdown | ¢ | @Char cent |
| £ | @Char sterling | ¤ | @Char currency | ¥ | @Char yen | ¦ | @Char brokenbar |
| § | @Char section | ¨ | @Char dieresis | © | @Char copyright | ª | @Char ordfeminine |
| « | @Char guillemotleft | ¬ | @Char logicalnot | - | @Char hyphen | ® | @Char registered |
| ¯ | @Char macron | ° | @Char degree | ± | @Char plusminus | ² | @Char twosuperior |
| ³ | @Char threesuperior | ´ | @Char acute | µ | @Char mu | ¶ | @Char paragraph |
| · | @Char periodcentered | ¸ | @Char cedilla | ¹ | @Char onesuperior | º | @Char ordmasculine |
| » | @Char guillemotright | ¼ | @Char onequarter | ½ | @Char onehalf | ¾ | @Char threequarters |
| ¿ | @Char questiondown | À | @Char Agrave | Á | @Char Aacute | Â | @Char Acircumflex |
| Ã | @Char Atilde | Ä | @Char Adieresis | Å | @Char Aring | Æ | @Char AE |
| Ç | @Char Ccedilla | È | @Char Egrave | É | @Char Eacute | Ê | @Char Ecircumflex |
| Ë | @Char Edieresis | Ì | @Char Igrave | Í | @Char Iacute | Î | @Char Icircumflex |
| Ï | @Char Idieresis | Ð | @Char Eth | Ñ | @Char Ntilde | Ò | @Char Ograve |
| Ó | @Char Oacute | Ô | @Char Ocircumflex | Õ | @Char Otilde | Ö | @Char Odieresis |
| × | @Char multiply | Ø | @Char Oslash | Ù | @Char Ugrave | Ú | @Char Uacute |
| Û | @Char Ucircumflex | Ü | @Char Udieresis | Ý | @Char Yacute | Þ | @Char Thorn |
| ß | @Char germandbls | à | @Char agrave | á | @Char aacute | â | @Char acircumflex |
| ã | @Char atilde | ä | @Char adieresis | å | @Char aring | æ | @Char ae |
| ç | @Char ccedilla | è | @Char egrave | é | @Char eacute | ê | @Char ecircumflex |
| ë | @Char edieresis | ì | @Char igrave | í | @Char iacute | î | @Char icircumflex |
| ï | @Char idieresis | ð | @Char eth | ñ | @Char ntilde | ò | @Char ograve |
| ó | @Char oacute | ô | @Char ocircumflex | õ | @Char otilde | ö | @Char odieresis |
| ÷ | @Char divide | ø | @Char oslash | ù | @Char ugrave | ú | @Char uacute |
| û | @Char ucircumflex | ü | @Char udieresis | ý | @Char yacute | þ | @Char thorn |
| ÿ | @Char ydieresis | | | | | | |

Of course, many of these characters can also be typed directly, or with the aid of double quotes, as we have seen. If your keyboard has accented characters on it, you can type them directly too; if not, you need to use the @Char symbol, in which case you will probably need braces as well, like this:

gar{@Char ccedilla}on

to distinguish the @Char symbol and the character name from adjacent letters.

Finally we have the Adobe Systems Symbol font, a treasure trove of exotic characters obtained with the @Sym symbol:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | @Sym space | ! | @Sym exclam | ∀ | @Sym universal | # | @Sym numbersign |
| ∃ | @Sym existential | % | @Sym percent | & | @Sym ampersand | ∋ | @Sym suchthat |
| ( | @Sym parenleft | ) | @Sym parenright | ∗ | @Sym asteriskmath | + | @Sym plus |
| , | @Sym comma | − | @Sym minus | . | @Sym period | / | @Sym slash |
| 0 | @Sym zero | 1 | @Sym one | 2 | @Sym two | 3 | @Sym three |
| 4 | @Sym four | 5 | @Sym five | 6 | @Sym six | 7 | @Sym seven |
| 8 | @Sym eight | 9 | @Sym nine | : | @Sym colon | ; | @Sym semicolon |
| < | @Sym less | = | @Sym equal | > | @Sym greater | ? | @Sym question |
| ≅ | @Sym congruent | Α | @Sym Alpha | Β | @Sym Beta | Χ | @Sym Chi |
| Δ | @Sym Delta | Ε | @Sym Epsilon | Φ | @Sym Phi | Γ | @Sym Gamma |
| Η | @Sym Eta | Ι | @Sym Iota | ϑ | @Sym theta1 | Κ | @Sym Kappa |
| Λ | @Sym Lambda | Μ | @Sym Mu | Ν | @Sym Nu | Ο | @Sym Omicron |
| Π | @Sym Pi | Θ | @Sym Theta | Ρ | @Sym Rho | Σ | @Sym Sigma |
| Τ | @Sym Tau | Υ | @Sym Upsilon | ς | @Sym sigma1 | Ω | @Sym Omega |
| Ξ | @Sym Xi | Ψ | @Sym Psi | Ζ | @Sym Zeta | [ | @Sym bracketleft |
| ∴ | @Sym therefore | ] | @Sym bracketright | ⊥ | @Sym perpendicular | _ | @Sym underscore |
| ‾ | @Sym radicalex | α | @Sym alpha | β | @Sym beta | χ | @Sym chi |
| δ | @Sym delta | ε | @Sym epsilon | ϕ | @Sym phi | γ | @Sym gamma |
| η | @Sym eta | ι | @Sym iota | φ | @Sym phi1 | κ | @Sym kappa |
| λ | @Sym lambda | μ | @Sym mu | ν | @Sym nu | ο | @Sym omicron |
| π | @Sym pi | θ | @Sym theta | ρ | @Sym rho | σ | @Sym sigma |
| τ | @Sym tau | υ | @Sym upsilon | ϖ | @Sym omega1 | ω | @Sym omega |
| ξ | @Sym xi | ψ | @Sym psi | ζ | @Sym zeta | { | @Sym braceleft |
| \| | @Sym bar | } | @Sym braceright | ∼ | @Sym similar | ϒ | @Sym Upsilon1 |
| ′ | @Sym minute | ≤ | @Sym lessequal | ⁄ | @Sym fraction | ∞ | @Sym infinity |
| ƒ | @Sym florin | ♣ | @Sym club | ♦ | @Sym diamond | ♥ | @Sym heart |
| ♠ | @Sym spade | ↔ | @Sym arrowboth | ← | @Sym arrowleft | ↑ | @Sym arrowup |
| → | @Sym arrowright | ↓ | @Sym arrowdown | ° | @Sym degree | ± | @Sym plusminus |
| ″ | @Sym second | ≥ | @Sym greaterequal | × | @Sym multiply | ∝ | @Sym proportional |
| ∂ | @Sym partialdiff | • | @Sym bullet | ÷ | @Sym divide | ≠ | @Sym notequal |
| ≡ | @Sym equivalence | ≈ | @Sym approxequal | … | @Sym ellipsis | ⏐ | @Sym arrowvertex |
| ⎯ | @Sym arrowhorizex | ↵ | @Sym carriagereturn | ℵ | @Sym aleph | ℑ | @Sym Ifraktur |
| ℜ | @Sym Rfraktur | ℘ | @Sym weierstrass | ⊗ | @Sym circlemultiply | ⊕ | @Sym circleplus |
| ∅ | @Sym emptyset | ∩ | @Sym intersection | ∪ | @Sym union | ⊃ | @Sym propersuperset |
| ⊇ | @Sym reflexsuperset | ⊄ | @Sym notsubset | ⊂ | @Sym propersubset | ⊆ | @Sym reflexsubset |
| ∈ | @Sym element | ∉ | @Sym notelement | ∠ | @Sym angle | ∇ | @Sym gradient |
| ® | @Sym registerserif | © | @Sym copyrightserif | ™ | @Sym trademarkserif | ∏ | @Sym product |
| √ | @Sym radical | ⋅ | @Sym dotmath | ¬ | @Sym logicalnot | ∧ | @Sym logicaland |
| ∨ | @Sym logicalor | ⇔ | @Sym arrowdblboth | ⇐ | @Sym arrowdblleft | ⇑ | @Sym arrowdblup |
| ⇒ | @Sym arrowdblright | ⇓ | @Sym arrowdbldown | ◊ | @Sym lozenge | 〈 | @Sym angleleft |
| ® | @Sym registersans | © | @Sym copyrightsans | ™ | @Sym trademarksans | ∑ | @Sym summation |
| ⎛ | @Sym parenlefttp | ⎜ | @Sym parenleftex | ⎝ | @Sym parenleftbt | ⎡ | @Sym bracketlefttp |
| ⎢ | @Sym bracketleftex | ⎣ | @Sym bracketleftbt | ⎧ | @Sym bracelefttp | ⎨ | @Sym braceleftmid |
| ⎩ | @Sym braceleftbt | ⎪ | @Sym braceex | 〉 | @Sym angleright | ∫ | @Sym integral |
| ⌠ | @Sym integraltp | ⎮ | @Sym integralex | ⌡ | @Sym integralbt | ⎞ | @Sym parenrighttp |
| ⎟ | @Sym parenrightex | ⎠ | @Sym parenrightbt | ⎤ | @Sym bracketrighttp | ⎥ | @Sym bracketrightex |
| ⎦ | @Sym bracketrightbt | ⎫ | @Sym bracerighttp | ⎬ | @Sym bracerightmid | ⎭ | @Sym bracerightbt |

There is only one Symbol font; it does not come in bold or italic faces like the other fonts. Typing @B @Sym alpha is therefore useless, and anyway there is no bold α character in any font distributed with Lout (except see Section 7.8).

## 1.6. Fonts and font sizes

A *font* is a collection of characters that may be printed. For example, here is the Times Roman font:

```
 !"#$%&'()*+,-./0123456789:;<=>?@[\]^_‘
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
{|}~ı`´ˆ˜¯˘˙¨˚¸˝˛ˇ ¡¢£¤¥¦§¨©ª«¬-®¯°±²³´µ¶·¸¹º»¼½¾¿
ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞ
ßàáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþÿ
```

and here is the Times Italic font:

```
 !"#$%&'()*+,-./0123456789:;<=>?@[\]^_‘
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
{|}~ı`´ˆ˜¯˘˙¨˚¸˝˛ˇ ¡¢£¤¥¦§¨©ª«¬-®¯°±²³´µ¶·¸¹º»¼½¾¿
ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞ
ßàáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþÿ
```

As their names imply, these two fonts belong to the *Times family*, a collection of fonts designed to go well together. Every font has a *family name*, such as Times, Helvetica, or Courier, and a *face name*, such as Roman or Italic. To find out how to get the unusual characters, see Section 1.5.

Documents look best when they use just one font family, so the most common need is to change to a different face within the current family. We have already seen @I, which changes to the Italic face of the current family; there are five such symbols:

| | |
|---|---|
| @B { Hello World } | **Hello World** |
| @I { Hello World } | *Hello World* |
| @BI { Hello World } | ***Hello World*** |
| @S { Hello World } | HELLO WORLD |
| @R { Hello World } | Hello World |

The symbols' names stand for Bold, Italic, Bold-Italic, Small capitals, and Roman. It is conventional to use Bold for headings; Italic for emphasis, terms being defined, and subsidiary headings; and Roman for the rest. Small capitals are not really a different font; they are made on demand from the current font. So you can write, for example,

```
@I @S { Hello World }
```

and get *HELLO WORLD*.

The @R symbol is almost unnecessary, since the document as a whole is set in a Roman face; but it is occasionally useful:

```
@I { An Italic sentence with one @R Roman word }
```

produces

*An Italic sentence with one* Roman *word*

This illustrates the general principle that the effect of a font symbol on the following object is subject to font symbols within that object.

Changing families is a little more complicated. Here is the complete list of font families and their faces available with Basser Lout Version 3:

| | |
|---|---|
| AvantGarde | Base Slope Bold BoldSlope BoldObl Book BookOblique CondBold CondBook CondDemi CondMedium Demi DemiOblique ExtraLight ExtraLightObl Medium MediumObl |
| Bookman | Base Slope Bold BoldSlope BoldItalic Demi DemiItalic Light LightItalic Medium MediumItalic |
| Chancery | Base Slope Bold BoldSlope Roman Bold Italic Light Demi LightItalic MediumItalic |
| Courier | Base Slope Bold BoldSlope BoldOblique Oblique |
| Helvetica | Base Slope Bold BoldSlope Black BlackOblique BoldOblique Compressed Cond CondBlack CondBlackObl CondBold CondBoldObl CondLight CondLightObl CondOblique ExtraCompressed Light LightOblique Narrow NarrowBold NarrowBoldObl NarrowObl Oblique UltraCompressed |
| Schoolbook | Base Slope Bold BoldSlope BoldItalic Italic Roman |
| Palatino | Base Slope Bold BoldSlope BoldItalic BoldItalicOsF BoldOsF Italic ItalicOsF Roman SC |
| Symbol | Base Slope Bold BoldSlope |
| Times | Base Slope Bold BoldSlope BoldItalic BoldItalicOsF BoldSC ExtraBold Italic ItalicOsF Roman RomanSC Semibold SemiboldItalic |
| Dingbats | Base Slope Bold BoldSlope |

Lout understands all these fonts, but your printing device may not. Times, Helvetica, Courier, and Symbol at least seem to be ubiquitous, although not in every face. These fonts work only with languages that use the Latin1 character set; consult Section 1.14 for more information about this. It is not difficult for a Lout expert to extend this list [5].

It is a convention in Lout that every font family should at least contain faces called Base, Slope, Bold, and BoldSlope, and these faces are what the @R, @I, @B, and @BI symbols give you. But this convention is something of a fiction for two reasons. First, some font families don't have faces that could reasonably be described as bold or whatever. In particular, the Symbol family contains just one face, and all four conventional face names produce that face. Second, the four conventional face names are not names that typographers actually use, Bold excepted. Slope produces an italic face in some families and an oblique one in others. As the table shows, the true names are available if you want to use them, but it is very convenient to have a Slope face that is guaranteed to exist no matter which family is used.

The @Font symbol changes the font of the following object. For example,

{ Helvetica Slope } @Font { Hello World }

produces

*Hello World*

When changing to a different family, a face name must follow the family name; but when changing face within a family, just the face name is sufficient.

To make the characters larger or smaller, you need to change the *font size*, which can also be done with the @Font symbol. Font sizes are traditionally measured in *points*: there are 72 points to one inch, and the most common font sizes are 12 point and 10 point. However, as Section 1.2 explains in detail, any length including fractional lengths is acceptable:

24p @Font { Hello World }

changes to 24 point size, producing

# Hello World

It is also possible to specify a font size relative to the current size: +2p means two points larger, -2p means two points smaller, and 1.5f means 1.5 times the current font size.

For the convenience of people who use fixed width fonts such as Courier, there is an @F symbol which switches to a fixed width font family:

@F { Hello world }

produces

`Hello world`

It is the same as writing { Courier Base -1p } @Font ... with the -1p included to compensate for the relatively large appearance of the Courier font.

The document as a whole will be set in Times Base 12p. To change this you need to change the @InitialFont option, for example to

@InitialFont { Helvetica Base 10p }

to get Helvetica 10 point. You must give all three parts in @InitialFont: family, face, size. If you are using your own setup file, as explained in Section 4.1, you can find the @InitialFont option there. If not, you can set it at the beginning of your document as explained in Section 3.1.

There are two features that make fonts look better on the page. *Ligatures* are pairs of letters run together; the most common ligatures are 'fi' and 'fl.' *Kerning* is moving adjacent letters closer together, for example in 'VA.' Lout considers ligatures and kerning to be integral parts of each font; you can prevent them from happening only by enclosing one of the letters in a @OneCol symbol, as in @OneCol { V }A.

## 1.7. Headings

The @Heading symbol makes the following object into a heading. Actually, all it does is change the font, so if you want a centred heading you have to display it as well:

```
@Display @Heading { A Centred Heading }
Following text
```

If you want a left-justified heading, use @LeftDisplay instead of @Display. Alternatively, you can use no display symbol at all, but then you will need paragraph symbols before and after:

```
@DP
@Heading { A Left-Justified Heading }
@PP
Following text
```

The font used is Bold in the current family, although you can change this by changing the @HeadingFont option in the setup file (Section 4.1).

The @Heading symbol may be used with any type of document, but it is really intended only for simple ones. In complex documents, large-scale structure symbols (Section 2.7) are usually more appropriate.

## 1.8. Starting a new line, paragraph, or page

The usual way to start a new paragraph is with the @PP 'plain paragraph' symbol. It produces a small vertical space and indents the first line of the new paragraph. Some document formatting systems interpret a blank line as a request to start a new paragraph. This is not the case with Lout: a blank line is two line-endings, equivalent to two spaces. The @LP 'left paragraph' symbol produces the same vertical space as @PP, but omits the indent.

The @LLP 'left line paragraph' symbol starts a new paragraph using the usual inter-line spacing and no indent, or in other words it starts a new line. If you are using it to create single lines, you need the lines paragraph breaking style instead (Section 1.9).

The @DP 'display paragraph' symbol produces a somewhat larger vertical space, equal to the amount used before and after displays (Section 2.1), with no indent. To get even larger vertical spaces, use @DP repeatedly.

The @NP 'new page' symbol causes the following paragraph to begin on a new page or column. Of course, Lout starts a new page or column automatically when the old one is full, so @NP is needed only rarely.

Occasionally Lout will start a new page or column directly after a heading, which looks very poor. The obvious answer is to place an @NP just before the heading, but when the document is later revised and the heading no longer falls near the page or column ending, this @NP will have to be taken away again.

A better answer is to precede the heading with a @CNP 'conditional new page' symbol, which checks whether enough space remains in the page or column for a heading and at least two lines of text. If so, @CNP does nothing; if not, @CNP causes a new page or column to be begun, like @NP. The recommended arrangement is

```
end of previous part.
@DP
@CNP
@Heading { A Heading }
@PP
First paragraph of next part ...
```

The @CNP symbol should be preceded by either @DP or @LP, preferably @DP, and this determines the amount of space when the @NP action does not occur.

The ultimate answer to the conditional new page problem is to recognise that the heading is the beginning of a new section of the document, and to use a large-scale structure symbol like @Section (Section 2.7). Conditional new page is just one of many services provided automatically by these symbols.

You can modify the effect of the paragraph symbols by changing options in the setup file. For general information about setup files and their options, consult Section 4.1; here we just explain how the relevant options work. The options and their default values are

```
@ParaGap { 1.30vx }
@ParaIndent { 2.00f }
@DisplayGap { 1.00v }
```

The values are lengths (Section 1.2), except that for reasons beyond our scope @ParaGap must be a length with an x appended, as shown. The @ParaGap option determines how much vertical space will be inserted by @PP and @LP. The default value, 1.30vx, is 30% more than the normal inter-line spacing; to get no extra spacing, change it to 1.00vx. The @ParaIndent option determines the width of the indent produced by @PP, and its default value is twice the current font size. The @DisplayGap option determines the amount of vertical space inserted by @DP, and many other things as well, such as the vertical space before and after displays and lists.

## 1.9. Paragraph breaking

*Paragraph breaking* is the process of taking a paragraph and inserting line breaks at the places appropriate to the column width. Lout works out suitable column widths and performs paragraph breaking automatically, finding an 'optimal' break with the method used by the TEX system. It offers eight styles of paragraph breaking, which we will explore with the aid of this example paragraph:

```
It is a truth universally
acknowledged, that a single man
in possession of a good fortune,
must be in want of a wife.
```

Changing the paragraph breaking style is similar to changing the font, colour, or language, and is done using the @Break symbol:

```
ragged @Break ...
```

This example causes every paragraph in the following object to be broken using the ragged style, of which more below.

The first two of the eight styles perform *line adjustment*, which means that they enlarge the spaces between the objects making up each line so as to fill the lines completely:

| | |
|---|---|
| adjust @Break ... | It is a truth universally acknowledged, that a single man in possession of a good fortune, must be in want of a wife. |
| outdent @Break ... | It is a truth universally acknowledged, that a single man in possession of a good fortune, must be in want of a wife. |

The adjust style is frequently used, so it has been chosen as the default style. Outdenting adds a small space at the start of each line except the first, and is much less common.

The next three styles do not adjust lines, leaving the paragraph *ragged*:

| | |
|---|---|
| ragged @Break ... | It is a truth universally acknowledged, that a single man in possession of a good fortune, must be in want of a wife. |
| cragged @Break ... | It is a truth universally acknowledged, that a single man in possession of a good fortune, must be in want of a wife. |
| rragged @Break ... | It is a truth universally acknowledged, that a single man in possession of a good fortune, must be in want of a wife. |

The paragraph is broken in the same places as adjust breaks it, but the resulting lines are left-justified, centred, or right-justified with respect to each other, rather than adjusted.

If you have a few words that must be kept together on one line, the recommended way is to separate them by an ~ symbol:

```
According to Prof.~Jones, the effect of ...
```

It's best not to bother about this until you actually get a bad line break, since chances are good that the words will fall on one line anyway.

The last three styles differ from the first five in breaking the paragraph at the points where it is broken in the original input:

| | |
|---|---|
| lines @Break ... | It is a truth universally<br>acknowledged, that a single man<br>in possession of a good fortune,<br>must be in want of a wife. |
| clines @Break ... | It is a truth universally<br>acknowledged, that a single man<br>in possession of a good fortune,<br>must be in want of a wife. |
| rlines @Break ... | It is a truth universally<br>acknowledged, that a single man<br>in possession of a good fortune,<br>must be in want of a wife. |

The lines are left-justified, centred, or right-justified with respect to each other in the same way as for the ragged styles.

When using the lines style, there are some fine points concerning the proper use of white space. Consider this example:

```
@IndentedDisplay lines @Break @I {
Teach me to hear Mermaides singing,
Or to keep off envies stinging,
    And finde
    What winde
Serves to'advance an honest minde.
}
```

The result is the indented display

*Teach me to hear Mermaides singing,*
*Or to keep off envies stinging,*
    *And finde*
    *What winde*
*Serves to'advance an honest minde.*

This style is the only one for which it is useful to indent individual lines in the input; as the result shows, such indents will be respected, as will blank lines. However, Lout's rule that only white space separating objects affects the result (Section 1.3) still holds, which means that indenting the first line is not effective:

```
@IndentedDisplay lines @Break @I {
    And finde
    What winde
Serves to'advance an honest minde.
}
```

produces

*And finde*
    *What winde*
*Serves to'advance an honest minde.*

This may seem awkward at first, but actually it is extremely convenient because you don't have to worry about whether the first line of the paragraph should appear on a new line as above, or immediately after the opening brace: space at that point does not separate two objects, so it has no effect on the result. The indent can be obtained by starting the first line with an empty object (Section 1.4):

```
@IndentedDisplay lines @Break @I {
{}    And finde
    What winde
Serves to'advance an honest minde.
}
```

The result is

    *And finde*
    *What winde*
*Serves to'advance an honest minde.*

as desired.

To set the entire document in a paragraph breaking style other than adjust, you need to change the @InitialBreak option, as explained at the end of Section 1.10.

### 1.10. Line spacing

The @Break symbol also controls the amount of space placed between the lines of paragraphs. This distance is best given using the v unit of measurement: 1v is the current line separation (see Section 1.2 for a description of lengths in general). For example,

```
2vx @Break ...
```

produces double spacing in the paragraphs of the following object, and

```
0.9vx @Break ...
```

produces cramped spacing, which can be useful in large tables that don't quite fit on one page. The x following the v is required, but its meaning is beyond our scope [5].

To set the entire document in a different line spacing from the default, you need to change the @InitialBreak option. If you are using your own setup file (Section 4.1), change it there. If not, you can change it at the beginning of your document, as described in Section 3.1.

The default value of the @InitialBreak option produces the adjust paragraph breaking style with a line spacing of 1.20 times the current (that is, the initial) font size, and hyphenation on:

```
@InitialBreak { adjust 1.20fx hyphen }
```

To get double spacing, change it to

```
@InitialBreak { adjust 2.40fx hyphen }
```

To get ragged paragraphs with hyphenation off, change it to

```
@InitialBreak { ragged 1.20fx nohyphen }
```

and so on. It is a good idea to define the initial line spacing using the f unit, since then if you change the initial font size the line spacing will change with it. However, any length (Section 1.2) with an x appended will do: 14px for 14 point, 0.5cx for 0.5 centimetres, etc. Don't use the v unit though, because it refers to some *previous* line spacing, whereas here we are defining the line spacing for the first time.

## 1.11. Hyphenation

The @Break symbol also controls hyphenation: hyphen @Break turns it on, nohyphen @Break turns it off. For example, ragged breaking is often done without hyphenation:

```
@IndentedDisplay { ragged nohyphen } @Break {
This little paragraph will appear with
ragged ends to its lines.
}
```

Lout's method of choosing hyphenation points is copied from the TEX system, except that Lout will never place a hyphen within a sequence of characters that form a ligature (fl and fi are the most common ligatures).

Hyphenation usually works well by itself; you should never need to interfere with its ideas of what to do. However, if you do want to tell Lout where you think a hyphen could be inserted, you can use the &- symbol:

```
after&-math
```

If &- occurs directly after a hyphen character, hyphenation will be permitted but no extra hyphen will be inserted. To prevent hyphenation of a word, enclose the word in a @OneCol symbol.

To turn hyphenation off throughout the document, you need to set the @InitialBreak option to nohyphen, as described at the end of Section 1.10.

## 1.12. Underlining

The @Underline symbol underlines the following object:

```
This little paragraph of text will have
@Underline { three underlined words } in it.
```

produces

> This little paragraph of
> text will have <u>three un-</u>
> <u>derlined words</u> in it.

The underlining is continuous unless a line break intervenes. You can't use this symbol to underline an arbitrary object: it is carefully designed to produce high-quality underlining of single words and parts of paragraphs, and it works only for those objects.

Each font contains information about how words in that font should be underlined: how far below the baseline the line should be drawn, and how thick. The @Underline symbol uses this information; the font it bases its underlining on is the font of the first object underlined if it is a word, or else the font of the enclosing paragraph.

## 1.13. The current date and time

The @Date and @Time symbols produce the current date and time:

```
It is now @Time on @Date.
```

produces something like

It is now 10.26 a.m. on 3 April, 1996.

The result depends on the current language.

Both symbols have a @Format option that changes the format of the result:

```
@Date @Format { @DayNum"/"@MonthNum"/"@ShortYear }
```

The result is the @Format option with the symbols replaced by the appropriate values:

3/4/96

The / characters have been enclosed in double quotes for the usual reason (Section 1.5).

Here is the full list of symbols that you can use within both @Format options:

| | |
|---|---|
| @Year | The year, e.g. 1994 |
| @ShortYear | The last two digits of the year, e.g. 94 |
| @Month | The month, e.g. December |
| @ShortMonth | The month abbreviated, e.g. Dec |
| @MonthNum | The number of the month, between 1 and 12 |
| @Day | The day of the week, e.g. Saturday |
| @ShortDay | The day abbreviated, e.g. Sat |
| @DayNum | The day of the month, between 1 and 31 |
| @MeriDiem | a.m. or p.m. |
| @ShortMeriDiem | am or pm |
| @Hour | The hour, between 00 and 23 |
| @ShortHour | The hour, between 0 and 23 |
| @TwelveHour | The hour, between 1 and 12 |
| @Minute | The minute, between 00 and 59 |
| @Second | The second, almost always between 00 and 59 |

The default format for @Date in English is

    @Date @Format { @DayNum @Month, @Year }

and the default format for @Time in English is

    @Time @Format { @TwelveHour.@Minute @MeriDiem }

Both default formats depend on the current language, and of course so do @Month, @ShortMonth, @Day, and @ShortDay.

## 1.14. Languages other than English

When part of a document is written in a language other than English, Lout should be informed of this using the @Language symbol:

    ... the garter, he said: French @Language { 'Honi soit qui mal y
    pense' }, and this saying ...

Changing language is quite analogous to changing font using the @Font symbol.

Since accented characters (Section 1.5) are always available irrespective of the language, at first sight it might seem that there is no need to bother informing Lout what language you are writing in. However, words are hyphenated differently depending on the language, and some symbols have different results in different languages. For example,

    Danish @Language @Date

produces

> 3. april, 1996

and the alphabetic list symbols of Section 2.2 also vary with the current language. So it's worth doing for the sake of knowing that non-English parts will appear as they should.

At the time of writing, the following languages were available as part of the standard distribution of Lout:

    Czech Cesky Cestina
    Danish Dansk
    Dutch Nederlands
    English
    Finnish Suomi
    French Francais Français
    German Deutsch
    Norwegian Norsk
    Polish Polski
    Russian
    Slovenian Slovenia Slovenija
    Spanish Español
    Swedish Svenska

As shown, most languages have alternative names, all equally acceptable to the @Language symbol.

If your entire document is in a language other than English, you need to change the @InitialLanguage option:

```
@InitialLanguage { Deutsch }
```

If you are using your own setup file (Section 4.1), you can change it there. If not, you can change it at the start of your document, as explained in Section 3.1.

Czech, Polish, and Slovenian use the ISO-LATIN-2 character set, and Russian uses Cyrillic characters. Consult directory software in the Lout distribution to find out how to get the extra things you need for these languages. Other left-to-right languages are easily added, so please consult the author if your language is not listed.

## 1.15. Superscripts and subscripts

There are @Sup and @Sub symbols for producing superscripts and subscripts:

```
2 @Sup nd
```

produces

$2^{\text{nd}}$

and the @Sub symbol works in a similar way. These symbols are probably never required in English language text, since the only uses for them are in footnotes, which produce the superscript automatically, and equations, which have their own versions of these symbols. Both symbols have a gap option which determines the vertical spacing.

## 1.16. Verbatim text

The @Verbatim symbol prints the following object exactly as it appears in the input file. All special meanings for characters, symbols, etc. are turned off; there is one result line for each input line. For example,

```
@IndentedDisplay @Verbatim {
A line of "verbatim" text
Another line, with a \ character
}
```

has result

A line of "verbatim" text
Another line, with a \ character

Use { Courier Base } @Font @Verbatim { ... } to get the result in a fixed-width font.

If the verbatim text contains { or } characters, then they should either be balanced or else you need to use the alternative form

```
@Verbatim @Begin
...
@End @Verbatim
```

so that there is no doubt about where the verbatim text ends.

The @Verbatim symbol relies on the sed command, which exists on Unix systems but not necessarily on others. To ensure that this manual can be formatted on any system, the result above was simulated (i.e. not actually produced by the @Verbatim symbol).

## 1.17. Drop capitals

There are two symbols for producing drop capitals, @DropCapTwo and @DropCapThree. Place the capital to be dropped just before the symbol, and the rest of the paragraph after it:

```
I @DropCapTwo {
it is a truth universally acknowledged, that a single man
in possession of a good fortune, must be in want of a wife.
}
```

produces the object

I it is a truth universally acknowledged, that a single man in possession of a good fortune, must be in want of a wife.

@DropCapThree is the same except that the capital is larger, and spreads over three lines instead of two.

Because Lout occasionally gets the height of the enlarged capital slightly wrong, there is a height option which allows you to change the height if you need to:

```
H @DropCapTwo height { 1.5v }
{
  ...
}
```

This shows the default value for the height of the capital in @DropCapTwo: 1.5 times the current inter-line spacing. The default height in @DropCapThree is 2.5v.

These symbols produce an object which may appear anywhere in the usual way. A paragraph symbol will be needed after the paragraph. The paragraph breaking style of the body of the paragraph will be adjust, and hyphenation will be off. These settings cannot be changed at present.

## 1.18. Alternative conventions for white space

As Section 1.3 explains, when two objects are separated by one or more white space characters (spaces, tabs, and newlines), this same amount of white space will separate the two objects in the output.

Two other conventions for interpreting these white spaces have been used in other document formatting systems. Roughly, they are:

troff Like Lout, except that at every point where a sentence ends at the end of an input line, add one extra space in the output.

TEX Replace all sequences of two or more white spaces by one. Then, at every point where a sentence ends, whether or not it is at the end of a line, add one extra space in the output.

Lout offers these two alternative conventions by means of the @InitialSpace option. This is similar to the @InitialFont option described at the end of Section 1.6, in that you can set it at the beginning of your document, like this:

```
@SysInclude { doc }
@Document
   @InitialSpace { lout }
//
@Text @Begin
...
@End @Text
```

or you can set it in the setup file. The above example shows the default value, lout, which produces Lout's usual spacing; the alternative values are troff and tex.

How to tell whether a sentence has ended is a vexed question. For the troff method, Lout looks for a word at the end of a line ending in one of '.', ':', '?', or '!' optionally followed by either a right quote character or a right parenthesis. Actually, this depends on the current language (Section 1.14); the rule just given is for English, and other languages may differ.

The tex rule for where a sentence ends is slightly more complicated. Lout looks for a word, not necessarily at the end of an input line, which ends as described for troff but in addition has a lower-case letter preceding that.

In all cases you must use a paragraph symbol, such as @PP or @LP, to separate your paragraphs. The common convention of other systems, that a blank line marks a paragraph, is never true of Lout.

Whatever rule is adopted, there are occasional exceptions where you will have to indicate explicitly whether you want an ordinary space or a between-sentences space. For this there are two symbols, ~ (ordinary space) and ~~ (between-sentences space). For example,

```
Dr.~Kingston
```

will produce an ordinary space between the two words, even with tex which would otherwise consider that spot to be the end of a sentence. Spaces adjacent to these two symbols have no effect on the result.

# Chapter 2. Adding Structure to Documents

## 2.1. Displays

The @Display symbol displays the following object in the centre of the page or column:

```
@Display @I { Invitation to Afternoon Tea }
```

has result

*Invitation to Afternoon Tea*

Space is inserted automatically above and below the display, so no paragraph symbols are needed anywhere near it.

To make the display appear at the left margin instead of centred, use @LeftDisplay instead of @Display. To make an indented display, use @IndentedDisplay or @QuotedDisplay; the latter indents at the right margin as well as at the left. There are also @CentredDisplay and @CenteredDisplay symbols which centre the display just like @Display does, and @RightDisplay which right-justifies the display.

If you use displays frequently you might prefer abbreviated forms of their names. These are made from @ and the capital letters of the full name: @D, @LD, @ID, @QD, and @CD. Owing to an unfortunate clash with the name of another symbol, there is no abbreviation for @RightDisplay.

Displays often need to be set using a different font, paragraph breaking style, and so on to the surrounding text. It's best to set out such displays like this:

```
@CentredDisplay @I clines @Break {
Invitation to Afternoon Tea
with
Mr. and Mrs. Gilbert Newington-Smith
}
```

You can have as many of these symbols as you like, including specialized symbols like @CurveBox, @Eq, and @Tab. The only rule is that the display symbol must come first: @I @Display ... is wrong.

It's not a good idea to have one display immediately followed by another one, because there will be too much vertical space between them. Use a list instead (Section 2.2). Displays at the ends of paragraphs look awkward and are best avoided.

A display may come out partly on one page or column and partly on the next, if it has places where it obviously can be broken in two. For example, a display which is an ordinary paragraph of text might be broken in two between any two lines. To force a display to keep together on one page or column, use the @OneRow symbol like this: @Display @OneRow { ... }.

There are other display symbols for producing more advanced kinds of displays: aligned and numbered displays, and raw displays (i.e. without vertical space). Although these symbols can display any object as usual, in practice they are used for displaying mathematics, so they are described in Section 7.5.

Three setup file options are provided for controlling the appearance of displays. (For a general introduction to setup files and their options, consult Section 4.1.) Here they are with their default values:

```
@DisplayGap { 1.00v }
@DefaultIndent { 0.5rt }
@DisplayIndent { 2.00f }
```

@DisplayGap is the amount of vertical space inserted before and after displays, and may be any length (Section 1.2). The default value, 1.00v, is equal to the current inter-line spacing.

@DefaultIndent is the indent produced by @Display; the default value, 0.5rt, produces centring, although why it does so is beyond our scope [5]. @DisplayIndent is the indent produced by @IndentedDisplay, and at both margins by @QuotedDisplay. Its default value, 2.00f, is twice the current font size.

## 2.2. Lists

The @List symbol introduces a sequence of items to be made into a displayed list:

```
preceding text
@List
@ListItem @I Emma
@ListItem @I { Mansfield Park }
@EndList
following text
```

After the initial @List symbol, each item is introduced by @ListItem, and the list ends with @EndList. The result here is

preceding text

> *Emma*
>
> *Mansfield Park*

following text

with @DP inserted automatically before, between, and after the items.

As the example shows, the @List symbol causes the items to be indented. Also available are @LeftList, @IndentedList, @QuotedList, @CentredList, and @CenteredList, which format the items like the corresponding display symbols do.

Other list symbols generate a *label* for each item. For example, @NumberedList causes the items to be numbered:

```
@Heading { Quiz }
@NumberedList
@ListItem { Which American statesman owned a two-storey clock? }
@ListItem { Which Yankee commander from the Civil War cut a
swathe of destruction through the State of Georgia? }
@EndList
```

has result

**Quiz**

1. Which American statesman owned a two-storey clock?

2. Which Yankee commander from the Civil War cut a swathe of destruction through the State of Georgia?

The generated labels are added at the left margin. Here is the full set of label-generating list symbols, showing the first label produced by each:

| | | | |
|---|---|---|---|
| 1. | @NumberedList | (1) | @ParenNumberedList |
| i. | @RomanList | (i) | @ParenRomanList |
| I. | @UCRomanList | (I) | @ParenUCRomanList |
| a. | @AlphaList | (a) | @ParenAlphaList |
| A. | @UCAlphaList | (A) | @ParenUCAlphaList |
| • | @BulletList | | |
| * | @StarList | | |
| – | @DashList | | |

The Roman numerals end at cc (200), but ordinary decimal numbers have no limit. The labels produced by the four alphabetical list symbols are determined by the current language; in English they start at a and end at z.

You may also supply your own labels using the @TaggedList symbol. Each item is introduced by @TagItem instead of @ListItem. Since such labels tend to be quite wide, there are @WideTaggedList and @VeryWideTaggedList symbols which leave extra space for them:

```
@WideTaggedList
@TagItem { 9 a.m. } { Breakfast in the Ipamena Lounge,
served with Irish coffee and fresh croissants. }
@TagItem { 10 a.m. } { Prof. A. Smith
speaks on ‘The Wealth of Nations.’ }
@EndList
```

Each @TagItem symbol is followed by the desired label between braces, and then the item proper. The label may be empty, but still its enclosing braces must be there. The result here is

9 a.m. Breakfast in the Ipamena Lounge, served with Irish coffee and fresh croissants.

10 a.m. Prof. A. Smith speaks on ‘The Wealth of Nations.’

An alternative way to accommodate wide labels is the ‘drop item,’ which looks like this:

10 a.m.
Prof. A. Smith speaks on ‘The Wealth of Nations.’

Individual items are dropped in this way by using @DropTagItem instead of @TagItem. There is also a @DropListItem symbol corresponding to @ListItem, but it is very rarely needed. Lout is not able to decide for itself whether a label is wide enough to require a drop item.

Each list has a ‘raw’ version which omits the preceding space, and @EndList has a raw version which omits the following space. These are mainly used when an item is itself a list:

```
@ParenNumberedList
@ListItem {
  @RawParenRomanList
  @ListItem { MV Nominees,
hereinafter called the vendor, ... }
  @RawEndList
}
@EndList
```

produces

(1) (i) MV Nominees, hereinafter called the vendor, ...

If @ParenRomanList had been used instead of @RawParenRomanList, (1) and (i) would have appeared on different lines; or if @EndList had been used instead of @RawEndList, there would have been too much space following the list.

A list item may come out partly on one page or column and partly on the next, if it has places where it obviously can be broken in two. For example, a list item which is an ordinary paragraph of text might be broken in two between any two lines. To force a list item to keep together on one page or column, use the @OneRow symbol like this: @ListItem @OneRow { ... }.

Every symbol introduced in this section has an abbreviated form consisting of @ followed by its capital letters only. For example, @RawNumberedList abbreviates to @RNL, and @ListItem to @LI. The sole exception is @RawList, which has no abbreviation because @RL is the abbreviation for @RomanList.

Expert users will be interested to learn that all of the list symbols described in this section are derived from the two basic ones, @List and @RawList, merely by setting options. Here are all the options, together with their default values:

```
@List
   style {}
   labelwidth { 2f }
   indent { 0c }
   rightindent { 0c }
   gap { 1v }
   start { 1 }
```

These options may be used with all of the list and raw list symbols, except that some combinations don't make sense, for example indent with @CentredList or style with @BulletList, since the list symbol has clearly already set the option.

The style option determines the format of the label, any num symbol within it being replaced by the number of the item. For example, @ParenNumberedList is just

```
@List
   style { (num) }
```

and @BulletList is just

```
@List
   style { @Bullet }
```

with num not mentioned since no number is wanted. It is not possible to change the style of @TaggedList and its variants, since it is set to produce the label supplied by the author.

The labelwidth option determines the width set aside for the labels; this is where @WideTaggedList and @VeryWideTaggedList differ from @TaggedList. The indent and rightindent options determine the space left blank at the left and right margins. The value given to these three options may be any length, for example 0.5i (half an inch), or 0.5f (half the current font size). Section 1.2 describes lengths in general. There are also three useful symbols denoting lengths: @DisplayIndent is the amount by which indented and quoted displays are indented; @WideIndent and @VeryWideIndent are the indents used by @WideTaggedList and @VeryWideTaggedList. Using these symbols helps to keep documents consistent.

The gap option determines the vertical space inserted between items. Once again this must be a length, although since it is vertical rather than horizontal, somewhat different kinds of lengths are appropriate: 1.5v for 1.5 times the current vertical space between lines, or the default value, @DisplayGap, which produces the amount of vertical space used before and after displays. There is no option for the space before or after the list as a whole; to change this, use a raw list and insert your own.

The start option is the number assigned to the first item. It must be decimal:

```
@ParenRomanList
   start { 25 }
```

looks strange, but it is the correct way to number the first item (xxv).

Here is a larger example of these options in action. Setting both indent and rightindent to @DisplayIndent produces an effect similar to @QuotedDisplay:

```
preceding text
@List
   style { @I {Item num}: }
   indent { @DisplayIndent }
   rightindent { @DisplayIndent }
   labelwidth { @WideIndent }
   start { 10 }
@ListItem { The vendor ... in the case of accident. }
@ListItem { The vendor ... adjacent to the facility. }
@EndList
following text
```

The result is

preceding text

*Item 10*: The vendor will not be liable for any injury caused by the escape of radiation or radioactive materials from the facility, nor for the costs of repair of any property damaged by nuclear blast or fallout in the case of accident.

*Item 11*: The vendor will not be liable for any injury caused by radioactive materials being transported to or from the facility, nor for injury caused by radioactive materials stored adjacent to the facility.

following text

You can change the *default* values of the labelwidth, indent, rightindent, and gap options, by setting options called @ListTagWidth, @ListIndent, @ListRightIndent, and @ListGap in the setup file (Section 4.1). These default values will then apply automatically to every list in the document unless overridden by an option, just as the usual default values do.

## 2.3. Footnotes and endnotes

A footnote is created by typing

```
@FootNote { Like this. }
```

after the word that the footnote refers to. It will be numbered automatically and placed at the foot of the page or column;[1] or, if space there is insufficient, it may start on or run onto the following page or column. The footnote must be enclosed in braces.

The @FootNote symbol has a @Location option which determines where it goes:

```
@FootNote
   @Location { ColFoot }
{ ... }
```

[1]Like this.

places the footnote at the bottom of the column, and

```
@FootNote
   @Location { PageFoot }
{ ... }
```

places it at the bottom of the current page, occupying the full page width even in a multi-column document (this is occasionally useful for footnotes to headings). Of course, in a single-column document there is no difference anyway. The default value of the @Location option is ColFoot.

Endnotes work in exactly the same way as footnotes, except that the symbol to use is @EndNote and they appear either at the end of the document or at the end of some major part of it, depending on the type of document (Chapter 3). Endnotes are always column width and so have no @Location option.

The language of a footnote or endnote will be the language of the document as a whole. This is not necessarily the same as the current language at the point where the footnote or endnote occurs, or even the language of the enclosing large-scale structure symbol. It may be necessary to enclose the body of the footnote in a language symbol, like this:

```
@FootNote { French @Language { ... } }
```

Doing it the other way (French @Language @FootNote ...) is not effective.

A footnote attached to the very last line of a chapter or appendix of a book occasionally runs onto the first page of the following chapter or appendix, and this looks very poor. If this happens, the solution is to place an @LP after the last line (including the footnote).

The setup file contains a number of options for controlling the appearance of footnotes. (See Section 4.1 for a general introduction to setup files and their options.) Here are all the options, with their default values:

```
@FootNoteThrough { No }
@FootNoteLocation { ColFoot }
@FootNoteNumbers { Arabic }
@FootNoteFont { 0.80f }
@FootNoteBreak { 1.20fx }
@FootLen { 2.00c }
@FootAboveGap { 1.00v }
@FootGap { 0.20c }
```

There are also setup file options for controlling endnotes. Since they are quite similar to the ones for footnotes, we won't say any more about them here.

@FootNoteThrough may be Yes or No; Yes means that the footnotes are numbered continuously through the document (or through each chapter in the case of books); No means that the numbering begins afresh on each page. @FootNoteLocation determines the default value of the @Location option mentioned above; it may be either ColFoot or ColPage. @FootNoteNumbers determines how the footnotes are numbered; it may be Arabic, Roman, UCRoman, Alpha, or UCAlpha.

@FootNoteFont and @FootNoteBreak determine the font and paragraph breaking style of footnotes. The default value of @FootNoteFont produces the same font family and face as the bulk of the document, but reduced to 0.8 times the original size.

@FootLen determines the length of the small horizontal line drawn above the footnotes; @FootAboveGap determines the minimum space to be left clear above this line; and @FootGap determines the vertical separation between footnotes. All three may be any length.

## 2.4. Margin notes and arbitrary placement

A note can be placed in the left margin by typing

A left note.

```
@LeftNote { A left note. }
```

after the word that the note refers to. The note will appear in the margin at the same height on the page as that word, unless that would cause it to overlap a previous margin note, in which case it will be shifted downwards (but never onto the next page). The note may be an arbitrary Lout object; for example, you might type

*A left note.*

```
@LeftNote @I { A left note. }
```

to make your note come out in italics.

A right note.

An outer note.

An inner note.

You can get a note in the right margin by using @RightNote instead of @LeftNote. To get a note in the outer margin (left on even pages, right on odd pages), use @OuterNote; and for the opposite, use @InnerNote.

By default, Lout produces margins that are 2.5 centimetres wide, which is not really enough to accommodate reasonable margin notes. To change these margins, you need to change options in the setup file, as explained in Section 4.3.

The appearance of the margin notes themselves is also determined by options in the setup file (for a general introduction to setup files and their options, consult Section 4.1). Here are the options and their default values:

```
@MarginNoteFont { 0.80f }
@MarginNoteBreak { ragged 1.10fx }
@MarginNoteHGap { 0.5c }
@MarginNoteVGap { 1.00v }
@MarginNoteWidth { 1.50c }
```

@MarginNoteFont determines the font used; the default value produces the current font scaled to 0.8 times the current size; you could write Slope 0.80f to get italic notes, and so on. @MarginNoteBreak is the paragraph breaking style to use, similar to the @InitialBreak setup file option.

@MarginNoteHGap determines how far away from the adjacent text column the margin note will appear; the default value is 0.5 centimetres. @MarginNoteVGap is the minimum vertical separation between margin notes (i.e. it determines how far downwards a note will be shifted to avoid the previous one). @MarginNoteWidth determines the width of the column in which margin notes (both left and right) are set; the default value of 1.5 centimetres is suited to the 2.5 centimetre page margins that are the default, but if you widen the page margins you will be able to increase @MarginNoteWidth too.

Left notes extend into the left margin a total distance of @MarginNoteHGap plus @MarginNoteWidth, and it is up to you to make sure that this does not put them off the page. Similar remarks apply to right notes. And since notes are never shifted to the next page, only downwards, there is also a risk that a note will be shifted off the bottom of the page, if it is very long or if preceding notes obstruct it. Again, it is up to you to avoid this problem by keeping your notes small and not too close together.

Margin notes inside footnotes, figures and tables work well. Margin notes in multi-column documents are disastrous unless used very sparingly. Margin notes do not appear in plain text output (Section 3.6).

A more radical way to place objects at arbitrary points on the current page is provided by the @Place symbol:

```
@Place
   x { right - 1 cm - xsize }
   y { { foot + top } / 2 }
{
   @Box { Hello }
}
```

$\boxed{\text{Hello}}$

The placed object may be any object. This particular example produces a box whose *x* (horizontal) position is such that its right edge is one centimetre from the right edge of the page, and whose *y* (vertical) position is halfway up the page.

In addition to numbers, Lout lengths (Section 1.2), and Fig lengths (Section 9.4), the following symbols may be used inside the x and y options:

| | |
|---|---|
| left | The left edge of the page |
| right | The right edge of the page |
| foot | The foot edge of the page |
| top | The top edge of the page |
| + | Addition (positive is to the right and up) |
| - | Subtraction (negative is to the left and down) |
| * | Multiplication |
| / | Division |
| xsize | The width of the object being placed |
| xmark | The column mark of the object being placed (for expert users) |
| ysize | The height of the object being placed |
| ymark | The row mark of the object being placed (for expert users) |

The usual precedences and associativities apply to the mathematical operators; braces (not parentheses) may be used for grouping. It is best to give values to x and y that do not depend on any assumptions about where the coordinate system's origin is; this is true of the examples above. At the point where @Place occurs, the result is an empty object. As with margin notes, Lout has no idea that the translation is happening and will not lay out the rest of the page around the placed object.

## 2.5. Theorems, lemmas, corollaries, definitions, propositions, examples, and claims

A theorem is created like this:

```
@LD @Theorem
   @Title { Fermat's Last Theorem }
{
@Eq { a sup n + b sup n != c sup n } for all positive
integers @Eq { a }, @Eq { b }, @Eq { c } and @Eq { n }
when @Eq { n > 2 }.
@LP
@Proof I have a proof of this theorem, but the margin
is too small to contain it. @EndProof
}
```

where we have used the @LD 'left display' symbol from Section 2.1 to get a left-justified display, and the @Eq symbol from Chapter 7 for the equations. The result is

> **Theorem 2.1 (Fermat's Last Theorem):** $a^n + b^n \neq c^n$ for all positive integers $a$, $b$, $c$ and $n$ when $n > 2$.
>
> **Proof:** I have a proof of this theorem, but the margin is too small to contain it. □

The @Theorem symbol produces an object with no adjacent vertical space, hence it needs to be used in conjuction with display or paragraph symbols. The theorem is numbered automatically, with the title and number inserted at the start of the first paragraph. @Title may be omitted.

The @Proof symbol produces **Proof:** with the appropriate following space, and the @EndProof symbol produces a box at the end of the line. They may be used anywhere, not just within theorems.

There are seven symbols that produce independently numbered sequences in this way. They are @Theorem, @Definition, @Claim, @Proposition, @Lemma, @Corollary, and @Example.

The setup file contains options which determine whether the theorem numbers include a chapter number (@ChapterNumInTheorems), or a section number (@SectionNumInTheorems), and so on. A section number automatically includes a chapter number, etc. There are also options to change the word printed. For example, if you need a sequence of conjectures, change the

@ClaimWord setup file option to

```
@ClaimWord { Conjecture }
```

and use the @Claim symbol for your conjectures. You can even put

```
import @DocumentLayout
macro @Conjecture { @Claim }
```

into your mydefs file (Section 2.12) if you wish, so that you can write @Conjecture in your documents instead of @Claim.

Owing to problems behind the scenes, the @Theorem symbol and its companions have a potential efficiency problem: although all numbers are finalized on the second run, it takes Lout time proportional to the square of the highest theorem number to do this. So large numbers of theorems numbered together might be slow.

## 2.6. Figures and tables

Figures are created in a similar way to footnotes:

```
@Figure
   @Caption { Basser Lout }
@Fig {
   { @Box Lout }{ @HArrow { 2c @Wide } }{ @Box PostScript }
}
```

The @Figure symbol places the figure (which in this example is created using the advanced graphics features of Chapter 9) at the top of the following column or page, labelled by the @Caption option and automatically numbered. You can see this example at the top of page 38. Tables are obtained in the same way using @Table instead of @Figure.

@Figure and @Table each have an @InitialLanguage option which determines the language of the figure or table. If this is omitted, the language of the document as a whole will be used, not the language where the figure or table occurs.

If your document contains many figures, large figures, or multi-page figures, you are likely to encounter cases where Lout's assignment of figures to pages is not pleasing. In that case, you can improve things by moving the figures around within the body text, and by using the following options.

The @Figure symbol has a @Location option which determines where the figure will appear. Its possible values are

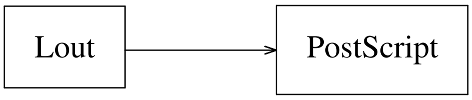



**Figure 2.1.** Basser Lout

PageTop — The figure will appear at the top of the following page, occupying the full page width; or, if there is insufficient space there (owing to other figures already present), at the top of the first subsequent page with sufficient space.

PageFoot — The figure will appear at the foot of the current page, occupying the full page width; or, if there is insufficient space there, at the top of the following page and so on as for PageTop.

ColTop — The figure will appear at the top of the following column, occupying the column width; or, if there is insufficient space there, at the top of the first subsequent column with sufficient space. This is different from PageTop only in multi-column documents.

ColFoot — The figure will appear at the foot of the current column, occupying the column width; or, if there is insufficient space there, at the top of the following column and so on as for ColTop. This is different from PageFoot only in multi-column documents.

ColEnd — The figure will appear in a column at the end of the document (or chapter, appendix etc. in the case of books). There is no PageEnd value corresponding to ColEnd.

AfterLine — The figure will appear as a column-width display immediately after the line in the final printed document in which it occurs.

TryAfterLine — The same as AfterLine unless there is insufficient space in the current column to hold the displayed figure, in which case it switches to ColTop instead.

Display — The figure will appear as a display at the point it occurs. There is no TryDisplay value corresponding to Display.

Raw — The figure will appear as an object, with no extra spacing, at the point it occurs. This is useful, for example, for getting two figures side by side in one display: use a displayed table containing two raw figures.

The @Table symbol also has this option. The default location is PageTop, but this can be changed by changing the @FigureLocation and @TableLocation setup file options.

The numbers assigned to figures and tables, and their ordering in any list of figures or tables, is based on where they appear in the final printed document, not on where they appear in the source files. This is better for the reader in the unusual case of a fixed figure being overtaken by a floating one.

@Figure and @Table each have a @OnePage option, whose value may be Yes or No. Setting @OnePage to Yes causes the figure or table and its caption to be kept together on one page or column (enclosing the body of the figure or table in @OneRow would have the same effect except that it would not incorporate the caption, hence the need for this option). You need to be certain that the whole assembly will fit on one page when setting @OnePage to Yes.

The *default* value of the @OnePage option for each figure or table depends on the value of its @Location option as follows:

| | |
|---|---|
| No | PageTop ColTop ColEnd Raw |
| Yes | PageFoot ColFoot Display AfterLine TryAfterLine |

These choices represent a guess that figures that the user is happy to see at the page foot or in a display are probably going to be small enough to keep on one page, but that other figures may not be. In any case, these are only default values and you may set @OnePage to whatever you prefer.

@Figure and @Table also have a @FullPage option, whose value may again be either Yes or No, with default value No. Setting this option to Yes causes the figure or table to occupy the full page (quite a different thing from being kept together on one page), by adding white space below the caption. This is mainly useful when the figure or table occupies nearly all of one or more pages, with one or two silly-looking lines of body text at the bottom. To be precise, this option does not prevent the figure or table from appearing on a page which already contains other things; rather, it ensures that nothing more will be placed on the page after the figure or table is. The only way to ensure that a figure or table starts on a fresh page is to make sure that there are no other figures and tables (other than full-page ones) closely preceding it in the document.

By default, the body of the figure will be centred, and this usually looks best, at least for small figures. @Figure and @Table each have a @Format option which controls this format:

```
@Figure
   @Format { @CurveBox @HExpand { @CentredDisplay @Body } }
```

Within the @Format option, the @Body symbol stands for the body of the figure or table. This example centres the figure inside a @CurveBox which is horizontally expanded (by the @HExpand symbol) to occupy the full width of the page or column, rather than fitting snugly around the figure. Actually the most useful format is probably just

```
@Figure
   @Format { @Body }
```

which turns off the default centring. This is the most practical format for multi-page figures, for example.

There are setup file options called @FigureNumbers and @TableNumbers that determine whether figures and tables are numbered automatically or not. Your choices for these options are None, Arabic, Roman, UCRoman, Alpha, and UCAlpha. Depending on the document type and where the figure or table occurs, the number might include a chapter number as well.

There are also setup file options called @CaptionFont and @CaptionBreak which determine the font and paragraph breaking style used in the captions of figures and tables. Their default values are empty, meaning to use the initial font and break styles; but, for example, you could have

```
@CaptionFont { -2p }
```

in your setup file to get a smaller font size in your captions.

There are setup file options called @FigureCaptionPos and @TableCaptionPos which determine whether the captions to figures and tables appear above or below them. The default value is Below and the alternative is Above. Also, @Figure and @Table have a @CaptionPos option:

```
@Figure
   @CaptionPos { Above }
```

which allows this to be set individually for each figure or table.

You can get a list of figures at the start of your document by setting the @MakeFigureContents setup file option to Yes. There is also a @MakeTableContents setup file option for making a separate list of tables. The format of these lists will follow the format of tables of contents. These lists are only available in books (Section 3.3).

Owing to problems behind the scenes, a figure or table on a page without body text at the end of a chapter may have an inappropriate running header. If this happens, the recommended workaround is either to move the figure to an earlier point in the document, so that it comes out on a page that also has some body text, or else to place one or more @NP (new page) symbols at the end of the chapter (or at the end of its last section, subsection, etc.).

## 2.7. Large-scale structure: chapters, sections, etc.

Lout's large-scale structure symbols vary with the type of document (@Chapter for books, @Overhead for overhead transparencies, etc.), but they all work in the same way. Here is a typical example, @Section, as it would actually be used:

```
@Section
   @Title { Allocation of teachers }
@Begin
@PP
Apart from the usual need to avoid clashes, the allocation of teachers must
ensure that no teacher teaches more than seven periods per day, or ...
@End @Section
```

First comes the symbol itself, then any options in the usual way, and then the following object, enclosed in @Begin and @End @Section. The following object, also called the body of the section, may contain paragraphs, displays, and all the other features as usual. The body should begin with a paragraph symbol, which may be @PP or @LP as you prefer. The result is a section like the present one, automatically numbered, with the @Title option for its heading, preceded by a conditional new page symbol (Section 1.8).

When @Section symbols are used within an ordinary document, they must be bracketed by @BeginSections and @EndSections symbols, like this:

```
@SysInclude { doc }
@Doc @Text @Begin
preceding text
@BeginSections
@Section ... @End @Section
@Section ... @End @Section
...
@Section ... @End @Section
@EndSections
@End @Text
```

This arrangement is reminiscent of the one for lists, and, as for lists, there may be no paragraph or new page symbols before, between, or after the sections. To change the gap between sections, you need to change the @SectionGap option in the setup file, as explained in Chapter 3.

The @Begin ... @End @Section that brackets the body of each section may be abbreviated to { ... }. However, the long form is recommmended because it helps Lout to detect missing or extra braces within the body of the section.

All large-scale structure symbols have a @Tag option, whose use is explained in Section 2.8, and a @RunningTitle option. If running page headers have been requested, @RunningTitle will be used if it is given, otherwise @Title will be used for the running header. For example, the present section begins like this:

```
@Section
   @Title { Large-scale structure:  chapters, sections, etc. }
   @RunningTitle { Large-scale structure }
   @Tag { largescale }
@Begin
...
```

The point is that the section title is rather long for a running title, and so we use @RunningTitle to get an abbreviated version of it.

All large-scale structure symbols also have an @InitialLanguage option which sets the current language for the duration of that symbol. However, footnotes, endnotes, figures, tables, references, and index entries are set in the initial language of the document as a whole, unless you change their language explicitly using the @Language symbol.

## 2.8. Cross references

Cross references are a useful feature of documents, but they are a problem for authors. Suppose that at one point of your document you have

```
We hold these truths to be self-evident, that all men are created equal,
that they are endowed by their Creator with certain inalienable Rights,
that among these are Life, Liberty, and the pursuit of Happiness...
```

and that at some other point, earlier or later, you have

```
The anti-slavery cause, founded as it was on the Declaration
of Independence (page 181), could appeal to patriotic as
well as moral sentiments...
```

This is a *cross reference*, and the problem is that as the document is revised, the Declaration of Independence might move to page 185, and the cross reference must be found and changed.

Lout has a simple solution to this problem. Instead of writing the page number, write

```
The anti-slavery cause, founded as it was on the Declaration
of Independence (page @PageOf { decl.of.ind }), could appeal to
patriotic as well as moral sentiments...
```

instead, and at the point referred to, write

```
We @PageMark decl.of.ind hold these truths to be self-evident, that...
```

Inserting @PageMark decl.of.ind will not affect the result, but Lout makes a note of the number of the page on which the word preceding it appears, and inserts that number in place of @PageOf decl.of.ind. The tag, decl.of.ind, may be any simple word (actually Lout will accept a multi-word tag, but you will find that they are very inconvenient and better avoided). The braces are used for the usual reason, to control grouping: we don't want the following punctuation characters in the tag.

One tag called last.page is created automatically for you. @PageOf last.page gives the number of the last page of the document. For example, the result for this document is 199.

Cross referencing also applies to large-scale structure symbols such as @Chapter and @Section (any symbol with a @Title option), as well as @FootNote, @EndNote, @Figure, @Table, the numbered display symbols, and @ListItem and @DropListItem (but not @TagItem and @DropTagItem). Each of these symbols has a @Tag option:

```
@Section
  @Title { Cross references }
  @Tag { cross }
@Begin
@PP
Cross references are a useful ...
```

Now you can use the @PageOf symbol to find the number of the page on which the symbol's result begins, and the @NumberOf symbol to find its number:

```
For further information on this point, please consult
Section @NumberOf cross (page @PageOf { cross }).
```

produces

For further information on this point, please consult Section 2.8 (page 41).

Like all tags, the value of the @Tag option should be a simple word (although Lout does accept multi-word tags). Cross referencing of list items works best with parenthesized lists, since, for example, (16) looks good inside a paragraph but 16. does not.

To work cross references out, Lout has to process your document more than once, storing information between runs in special files it creates whose names end in .li and .ld. A complex document like this Guide requires five runs, but since every run produces a perfectly good PostScript file suitable for proof reading, in fact you need two runs to start with and one run per cycle of revision thereafter, only one more than would have been necessary in any case.

The cross referencing system assumes that each Unix directory contains only one Lout document (possibly spread over many files). If you keep several documents in one directory you can turn off the cross referencing with the -s flag:

```
lout -s simple > simple.ps
```

Since this will cause question marks to replace footnote and section numbers, and other products of cross referencing, it is only feasible for simple documents. Alternatively, you can reset cross referencing when switching from one document to another, by removing file lout.li. You should also remove this file if your document changes radically – from a report to a book, say.

## 2.9. Tables of contents

Lout takes note of the titles of all your large-scale structure symbols (Section 2.7) and what pages they begin on, and it uses this information to produce a table of contents like the one at the start of the present document. It is totally automatic; you do nothing.

Some details of the appearance of the table of contents, including whether to make one or not, are controlled by options in the setup file. The default setting is to make one in books but not to in other types of documents, but by changing the setup file you can have a table of contents in any type of document.

Section 4.1 describes setup files in general and how to change the options within them. The options relevant to tables of contents and their default values are:

```
@MakeContents { No }
@ContentsGap { 0.20v }
@ContentsGapAbove { 0.80v }
@ContentsGapBelow { 0.00v }
@ContentsLeader { .. }
@ContentsLeaderGap { 4s }
@ContentsRightWidth { 3f }
```

The @MakeContents option may be Yes or No, and determines whether a table of contents is made or not. Its default value is No but it is set to Yes in the book setup file.

@ContentsGap determines how much vertical space to leave above each line of the table of contents, in addition to the usual single line spacing; its value may be any length (Section 1.2). The default value, 0.20v, is twenty percent of the current inter-line spacing.

Some entries, such as those for chapters and appendices in books, are more important than others. @ContentsGap does not apply to these entries; instead, @ContentsGapAbove and @ContentsGapBelow are used above and below each of them, again in addition to the usual single line spacing.

@ContentsLeader is the object which is repeated across the page to connect each entry with its page number; popular values are .. and . and the empty object. @ContentsLeaderGap determines how far apart these objects are; the default value, 4s, is four times the width of a space character. @ContentsLeaderGap may be 0s, but only if @ContentsLeader is non-empty.

@ContentsRightWidth reserves a certain amount of horizontal space at the far right for page numbers. Any entry wide enough to intrude into this space is broken into two or more lines to keep it clear.

In addition to these options, each document type has options that determine which large-scale structure symbols will be listed in the table of contents. For example, among the options to the @BookLayout symbol in the book setup file are these:

```
@ChapterInContents { Yes }
@SectionInContents { Yes }
@SubSectionInContents { Yes }
@SubSubSectionInContents { No }
@AppendixInContents { Yes }
@SubAppendixInContents { Yes }
@SubSubAppendixInContents { No }
```

Each may be either Yes or No; these default values produce entries for everything except sub-subsections and sub-subappendices.

## 2.10. Indexes

Although Lout is not clever enough to guess what entries should go in your index, it will do almost everything else for you: sort the entries and attach the correct page numbers automatically. As for tables of contents, the default setting is to have an index in books but not in other types of documents. This and a few aspects of the appearance of the index can be changed by changing the setup file, as explained at the end of this section.

Now, suppose you are discussing Galileo and you want his name in your index. Let's be ambitious and say that you want the index to contain something like this:

Galileo Galilei
  life of, 201
  telescope, his use of, 201–203
  trial of, 205–211, 242, 395

Each line shows off one of Lout's four tricks: the first is a *raw entry* (no page number attached); the second is a *sub-entry* (indented); the third has a *page number range* instead of a single page number; and the fourth is a *merged entry* (several page numbers or ranges within one entry).

We'll take each of them in turn in a moment, but first, let's see how to get a basic entry, like this one:

Galileo Galilei, 201

To get this into your index, type

```
galileo @Index { Galileo Galilei }
```

at the point where you mention Galileo. Nothing will be printed there, but the object following the @Index symbol will be placed in the index, with a comma and the correct page number appended automatically.

The object preceding the @Index symbol is a compulsory key which is used for sorting the index entries,[1] but which is not itself printed anywhere. It is best to construct these sorting keys from lower-case letters and the . character only, beginning with a letter, although multi-word keys are allowed. These sorting keys do not have to be distinct from the tags used in cross referencing; however, they do have to be distinct from each other, unless you want merged entries (see below).

Our first trick, raw entries (no page number attached), is very easy: just use @RawIndex instead of @Index. So the first line of our ambitious example is obtained by

```
galileo @RawIndex { Galileo Galilei }
```

This could go anywhere, since no page numbers are involved.

Another use for @RawIndex is to get blank lines into the index between the letters of the alphabet, by inserting phantom entries:

```
b @RawIndex {}
c @RawIndex {}
...
z @RawIndex {}
```

In fact there is a symbol called @IndexBlanks that makes exactly these 25 entries. Unfortunately, these blanks will occasionally appear at the top of a column, and if there are no tags beginning with x, for example, there will be two blank lines between the w and y entries. You can start off with @IndexBlanks and replace it later by the appropriate subset, if necessary.[2]

Our second trick, sub-entries, is also very easy, since a sub-entry differs from an ordinary entry only by having an indent. The symbol is @SubIndex, so the second line of our ambitious example is produced by

```
galileo.life @SubIndex { life of }
```

---

[1] The collating sequence used to decide what comes after what is the collating sequence used by the memcmp() library routine (just the underlying binary character codes). If this turns out not to be the one that you expected, the first thing to try is the replacement of all accented letters in index keys by unaccented ones. Sorting is quite an intractable problem: even if Lout knew the collating sequence for every language, which it doesn't, there would still remain the problem of sorting multilingual indexes.

[2] For Lout to solve this problem automatically, it would need to be told which letter each index entry belongs under, perhaps by symbols @AIndex, @BIndex, etc. The author felt that this would have been too tedious.

You should always give sub-entries the same sorting key as their corresponding main entries, plus a . and another word, because then you can be certain that the sorting will place sub-entries directly after their main entries. There is a @SubSubIndex symbol that produces a double indent, and there are @RawSubIndex and @RawSubSubIndex symbols.

For our third trick, page number ranges, we use the to option of the @Index, @SubIndex, and @SubSubIndex symbols. For example, to produce the sub-entry

telescope, his use of, 201–203

put

```
galileo.telescope @SubIndex to { gt.end } { telescope, his use of }
```

at the beginning of the range, and

```
@PageMark { gt.end }
```

at the end. You can use any tag you like inside the to option, as long as it differs from every other tag (notice that sorting keys do not have to differ from tags, but to options do: this is because to options go into @PageMark like other tags do, and if two tags are the same we would have an ambiguous result of @PageOf). If both ends of the range fall on the same page, the to option is ignored: you will never get 201–201.

Our fourth and final trick is the merged entry:

trial of, 205–211, 242, 395

The main thing to grasp is that this merged entry was originally three separate entries (sub-entries in this case):

trial of, 205–211
trial of, 242
trial of, 395

We already know how to produce these three entries, using three @SubIndex symbols, one with a to option. Now we have discovered that Lout is able to merge several entries into one entry. This raises two questions: how does Lout know which entries to merge? and given those entries, what does the merging produce?

The answer to the first question is that Lout merges entries whose sorting keys are equal. The merged entry above is produced by these three entries, placed in the appropriate places:

```
galileo.trial @SubIndex to { gtrial.end } { trial of }
galileo.trial @SubIndex { trial of }
galileo.trial @SubIndex { trial of }
```

The entries are merged because they have the same sorting key (galileo.trial), not because they happen to have the same content (trial of). In fact, once the page numbers are added the content is not the same at all.

Now, having decided that the three entries

trial of, 205–211
trial of, 242
trial of, 395

must be merged, what does Lout do? It takes the second entry, discards any initial part that is the same as the first entry (trial of, in this case), and, if anything remains, it adds a comma, a space, and the remainder to the first entry, producing

trial of, 205–211, 242

in this case. This process is repeated on this and the third entry, producing

trial of, 205–211, 242, 395

in this case, and so on. The entries are merged in the order in which their points of origin appear in the final printed document.

If nothing remains after discarding the common initial part, nothing is added to the growing merged entry; in effect, the entry that could contribute nothing new is deleted. With this in mind, let us return to our initial, ambitious example:

Galileo Galilei
  life of, 201
  telescope, his use of, 201–203
  trial of, 205–211, 242, 395

We now know how to produce all four of these entries, but one problem of some practical importance remains. Suppose we delete the section on the life of Galileo. If we had put the entry that produces 'Galileo Galilei' in that section, we might inadvertently delete it, and the other two sub-entries will lose their main entry. Before deleting anything, we must hunt through it for index entries and ponder their significance, an error-prone and time-wasting thing to do.

The solution is as follows. When an index entry has sub-entries, make it raw, and repeat it just before each of its sub-entries:

```
galileo @RawIndex { Galileo Galilei }
galileo.life @SubIndex { life of }
```

at the first place,

```
galileo @RawIndex { Galileo Galilei }
galileo.telescope @SubIndex { telescope, his use of }
```

at the second, and so on. Now it is easy to verify that every sub-entry has a main entry; and when deleting a sub-entry we can and should delete the adjacent main entry. After sorting, our index entries will be

| | |
|---|---|
| galileo | Galileo Galilei |
| galileo | Galileo Galilei |
| galileo | Galileo Galilei |
| galileo | Galileo Galilei |
| galileo | Galileo Galilei |
| galileo.life | life of, 201 |
| galileo.telescope | telescope, his use of, 201–203 |
| galileo.trial | trial of, 205–211 |
| galileo.trial | trial of, 242 |
| galileo.trial | trial of, 395 |

The first five entries have the same sorting key, and will be merged as required.

The language of the index entry will be the initial language of the document as a whole, which is not necessarily the language at the point where the index entry occurs. To get the correct language you will need a @Language symbol following the @Index symbol:

```
galileo. @Index French @Language { Galileo Galilei }
```

or whatever. If you don't do this your index entry might be hyphenated incorrectly.

Although the page numbers in index entries will be kept up to date automatically as the document changes, as all cross references are, it is best to refrain from inserting index entries until the document is complete and an overall plan of the structure of the index can be made.

The remainder of this section describes how to change the appearance of the index by setting options in the setup file. For setup files and their options in general, consult Section 4.1.

There are five setup file options for the index. Here they are with their default values:

```
@MakeIndex { No }
@IndexFont { }
@IndexBreak { outdent 1.2fx }
@IndexColumnNumber { 2 }
@IndexColumnGap { 1.00c }
```

The @MakeIndex option, which may be Yes or No, determines whether to produce an index or not. Although the default value is No, any type of document may be given an index just by changing it to Yes. This has already been done in the book setup file, but not in the others.

@IndexFont determines the font and font size of index entries (e.g. Times Base 12p). Leaving it empty as above produces the same font as the rest of the document. @IndexBreak is the paragraph breaking style applied to index entries; outdent is the traditional and best way.

@IndexColumnNumber and @IndexColumnGap determine the number of index columns per page, and the gap between them, and are exactly analogous to the @ColumnNumber and @ColumnGap options described in Section 2.11.

Lout offers the possibility of having up to three independent indexes (useful for glossaries, author indexes, etc.). The other two are called index A and index B, and they precede the main index in the output. Just replace Index by IndexA to refer to index A, and by IndexB to refer to index B. For example,

```
smith.j @IndexA { Smith, John }
```

will insert an index entry to index A, and @IndexBBlanks will insert the usual 25 blank entries into index B.

In large projects it might help to rename the @IndexA symbol to something else, such as @AuthorIndex. This can be done by placing

```
import @DocumentLayout
macro @AuthorIndex { @IndexA }
```

in the mydefs file. See Section 2.12 for an introduction to the mydefs file; the word macro is needed here instead of def because we are introducing a new name for an existing symbol, not defining a new symbol.

## 2.11. Multiple columns

You can change the number of columns of text per page, and the width of the gap between the columns, by changing these two setup file options:

```
@ColumnNumber { 1 }
@ColumnGap { 1.00c }
```

If you are using your own setup file (Section 4.1), you can find and change them there. If not, @ColumnNumber may be changed at the beginning of your document (Section 3.1).

@ColumnNumber may be any number between 1 and 10, with default value 1 as shown, and @ColumnGap may be any length (Section 1.2). The column width is derived from these options using the obvious formula

$$\mathit{columnwidth} = \frac{\mathit{pagewidth} - \mathit{margins} - (\text{@ColumnNumber} - 1) \times \text{@ColumnGap}}{\text{@ColumnNumber}}$$

You must ensure that this comes to something reasonable.

These two options do not apply to pages containing an index. For them there are similar setup file options called @IndexColumnNumber and @IndexColumnGap (Section 2.10).

Most document types permit you to have multiple columns, but certain things will be kept full width regardless of the @ColumnNumber option: figures and tables, chapter headings, and so on. The details vary with the document type, so are deferred to Chapter 3.

## 2.12. Defining new symbols

Whenever you find yourself typing the same thing repeatedly, you can save a lot of time by defining your own personal symbol to stand for that thing. For example, suppose you type your company's name, Batlow Food Distributors Pty. Ltd., frequently. You can define your own symbol, @Batlow say, so that

```
Concerning your crate supply contract with @Batlow, @Batlow wishes to ...
```

produces

> Concerning your crate supply contract with Batlow Food Distributors Pty. Ltd., Batlow Food Distributors Pty. Ltd. wishes to ...

You will never have to type Batlow Food Distributors Pty. Ltd. again.

The method is to create a file called mydefs in your current directory, containing definitions like this:

```
import @DocumentLayout
def @Batlow { Batlow Food Distributors Pty. Ltd. }
```

The meaning of the first line, import @DocumentLayout, will be explained shortly. After that comes def for 'define,' then the name of the symbol being defined, then its value between braces. So this example defines a symbol called @Batlow to stand for the object following it between braces. Lout will read this file during its setup phase (Section 4.1).

Your symbols may have any names you wish made from letters and @. However, it is good practice to have exactly one @, at the start, and to choose distinctive names that have no chance of being the same as the name of any existing symbol. @Batlow is a good choice, for example.

The object between braces is quite arbitrary; in particular, it may contain symbols. For example, suppose you frequently need a small grey box:

```
import @DocumentLayout
def @GreyBox { @Box paint { lightgrey } {} }
```

This defines a @GreyBox symbol that produces □. Most of the symbols in this guide are from the *DocumentLayout package*, which is why import @DocumentLayout is required: it makes these symbols available to the definition, and can actually be omitted before definitions like the one for @Batlow which do not use any symbols. However it does no harm, so we place it in front of every definition as a matter of course.[1]

Now suppose you frequently need a grey box, but enclosing different things: ENTRY one moment, EXIT the next. You could try omitting the {} from the definition above, but that does not work, because Lout notices the missing object while reading the definition, and inserts an empty object in the usual way (Section 1.4).

However, there is a way to define a @GreyBox symbol so that @GreyBox ENTRY produces ENTRY, @GreyBox EXIT produces EXIT, and so on:

```
import @DocumentLayout
def @GreyBox right x { @Box paint { lightgrey } x }
```

[1]Later chapters of this guide introduce specialized symbols for producing tables, equations, figures, graphs, and computer programs. You need a different import clause when using those symbols within a definition, because they are not from the DocumentLayout package. Examples may be found in the chapters concerned.

The addition of right x immediately after the symbol's name places @GreyBox into that class of symbols, like @I and @Box, which consume and transform the object to their right. The x in right x means that the object to the right will be referred to as x within the definition. So in

```
@GreyBox { Hello world }
```

@GreyBox consumes the following object, which becomes x, so that the value is

```
@Box paint { lightgrey } { Hello world }
```

which produces Hello world.

It is a good principle to choose symbol names that refer to what the symbol is for, rather than how it does what it does. Here is a good example:

```
import @DocumentLayout
def @Poetry right x { lines @Break @I x }
```

This kind of name is very pleasant to use, since it allows you to forget about what is going on behind the scenes:

```
@IndentedDisplay @Poetry {
Teach me to hear Mermaides singing,
Or to keep off envies stinging,
    And finde
    What winde
Serves to'advance an honest minde.
}
```

Most of Lout's symbols follow this principle.

You can define symbols that consume the object to their left as well as the object to their right, as the @Font, @Break, and @Colour symbols do:

```
import @DocumentLayout
def @HeadingBox left x right y
{  @Box { @CentredDisplay @Heading x y }
}
```

This definition occupies several lines only because it is long; as usual, end of line is the same as one space. Now

```
Cheating @HeadingBox {
The Department uses assignments ... of that student alone.
}
```

is much easier to type than the equivalent example in Section 8.2. The result is the same:

**Cheating**

The Department uses assignments both as a teaching device and as a major component of its assessment of each student. It therefore requires that all programs, exercises etc. handed in bearing an individual student's name be the work of that student alone.

Do not use a paragraph, display, or list symbol at the beginning or end of a definition, since the result is not what people who do it are hoping for.

# Chapter 3. Types of Documents

Particular types of documents have specialized formatting requirements: title pages in books, abstracts in technical reports, and so on. Producing such things using the general-purpose features of previous chapters can be quite awkward, so Lout provides a range of *document types* with the appropriate specialized features for each type.

At present there are six types: ordinary documents, technical reports, books, overhead transparencies, stand-alone illustrations, and plain text documents. The features of all other chapters are available within every document type, but the features of one document type are not available within other document types.

### 3.1. Ordinary documents

Ordinary documents are the simplest kind, consisting of a plain sequence of numbered pages containing whatever you want. To produce an ordinary document, use the doc setup file and the @Doc symbol:

```
@SysInclude { doc }
@Doc @Text @Begin
...
@End @Text
```

where ... stands for the body of your document. This is exactly the arrangement recommended in Section 1.1 for getting started.

There is an alternative way to begin an ordinary document, using the @Document symbol instead of @Doc:

```
@SysInclude { doc }
@Document
   @InitialFont { Times Base 12p }
   @InitialBreak { adjust 1.2fx hyphen }
   @InitialSpace { lout }
   @InitialLanguage { English }
   @PageHeaders { Simple }
   @FirstPageNumber { 1 }
   @ColumnNumber { 1 }
   @OptimizePages { No }
//
@Text @Begin
...
@End @Text
```

This shows the eight options of @Document, with their default values. As usual with options, the options of @Document may be given in any order, and only the ones that need to be changed need be given at all.

Notice the // after the last option. Its meaning is beyond our scope, but total disaster will ensue if it is forgotten. The @Doc symbol is an abbreviation for @Document //, which is why you don't need // with @Doc.

The eight options are a selection of setup file options (Section 4.1) that frequently need to be changed. If your changes to the overall formatting are confined to these options, you can change them here and avoid having your own setup file. If you already have your own setup file, change them in either place and omit them in the other.

@InitialFont is the font in which the bulk of the document will be set, and should contain a family, a face, and a size. The default value selects the Times family, the Base face, and the 12 point size.

@InitialBreak controls the behaviour of paragraph breaking in the bulk of the document. It should have three parts: a paragraph breaking style (adjust, ragged, etc.), an inter-line spacing (1.2fx for single spacing, 2.4fx for double spacing, and so on), and either hyphen or nohyphen for turning hyphenation on or off.

@InitialSpace determines how Lout treats white space between two objects, as described in Section 1.18. @InitialLanguage determines the language of the bulk of the document.

@PageHeaders determines the appearance of page headers and footers throughout the document. Its value may be None, Simple, Titles, or NoTitles. Section 4.4 has the details, but just briefly, None means no page headers at all, Simple means a simple style with a page number between hyphens at the top of each page except the first, Titles produces full running titles as in the present document, and NoTitles is like Titles with the running titles omitted, leaving just the page numbers.

@FirstPageNumber is the page number given to the first page.

@ColumnNumber is the number of columns per page in the bulk of the document, and may be anything from 1 (the default value) to 10. Irrespective of its value, all figures and tables will be printed full width; and it is possible to produce full-width ordinary text in a multi-column document as well, using the @FullWidth symbol:

```
@SysInclude { doc }
@Document
   @ColumnNumber { 2 }
//
@Text @Begin
@FullWidth {
@CentredDisplay @Heading { NOTICE TO TRESPASSERS }
}Trespassers are hereby notified that, pursuant to the
terms of the Trespassing Act, 1875, ...
@End @Text
```

This produces a full-width heading above a two-column body. The word Trespassers has been placed immediately after the closing brace of @FullWidth because (regrettably) any space here will appear before Trespassers in the output. Alternatively you could use a paragraph symbol:

```
@Text @Begin
@FullWidth {
@CentredDisplay @Heading { NOTICE TO TRESPASSERS }
}
@PP
Trespassers are hereby notified that, ...
@End @Text
```

You can have @FullWidth symbols repeatedly through the document, producing full-width text wherever you want. Just be aware that @FullWidth always causes a fresh page to be begun, it will never appear on the same page as a figure or table, and it is not able to hold a table of contents, a section, or an appendix.

Lout ordinarily places lines onto a page until space runs out, then moves to the next page and so on. This often produces ugly empty spaces at the bottoms of pages preceding large unbreakable displays. Setting the @OptimizePages option to Yes causes Lout to examine the overall situation and try to minimize the ugliness, using the TEX optimal paragraph breaking algorithm. It takes two runs to do this, with intermediate results stored in Lout's cross reference database (Section 2.8); so deleting file lout.li will reset it, which might be wise after major changes.

Within the @Text symbol, it is possible to have a sequence of sections:

```
preceding text
@BeginSections
@Section ... @End @Section
@Section ... @End @Section
...
@Section ... @End @Section
@EndSections
```

as described in detail in Section 2.7. Within any section, a similar arrangement produces subsections:

```
preceding text
@BeginSubSections
@SubSection ... @End @SubSection
@SubSection ... @End @SubSection
...
@SubSection ... @End @SubSection
@EndSubSections
```

Within any subsection, there may be sub-subsections, obtained in the same way only using @BeginSubSubSections, @SubSubSection, and @EndSubSubSections. There are no sub-sub-subsections.

Also within the @Text symbol only, there may be a sequence of appendices:

```
preceding text
@BeginAppendices
@Appendix ... @End @Appendix
@Appendix ... @End @Appendix
...
@Appendix ... @End @Appendix
@EndAppendices
```

These will be 'numbered' A, B, C etc. as is conventional. Within any appendix there may be a sequence of subappendices, obtained in the usual way using @BeginSubAppendices, @SubAppendix, and @EndSubAppendices. There are sub-subappendices as well, following the same pattern, but no sub-sub-subappendices.

In addition to the @Title option, each large-scale structure symbol (@Section, @SubSection, @SubSubSection, @Appendix, @SubAppendix, and @SubSubAppendix) has a @Tag option for cross referencing (Section 2.8), an @InitialLanguage option for changing the language of that part of the document, and a @RunningTitle option which will be used in place of @Title in running headers if given. This last is useful when the full title is rather long.

The features described in other chapters are all available within ordinary documents. Endnotes and references appear automatically at the end of the document. Figures are labelled Figure 1, Figure 2, etc., and tables are labelled Table 1, Table 2, etc.

To get a table of contents, set the @MakeContents option in the setup file to Yes, and insert the symbol @ContentsGoesHere at the point where you would like the table of contents to appear, anywhere before the first section:

```
@SysInclude { doc }
@Text @Begin
@CentredDisplay @Heading { Safety Procedures }
@Heading { Contents }
@DP
@ContentsGoesHere
@DP
...
@End @Text
```

You must supply your own heading, as well as paragraph symbols before and after. Regrettably, @ContentsGoesHere may not be placed inside a display, nor inside @FullWidth.

To get an index, set the @MakeIndex option in the setup file to Yes, and follow the instructions in Section 2.10. The index will appear automatically at the end of your document.

Within the doc setup file there is an @OrdinaryLayout symbol whose options control the appearance of features specific to ordinary documents (in other words, the features described in this section). Here is a representative sample of these options, showing their default values:

```
@Use { @OrdinaryLayout
  # @IndexWord { index }
  # @AppendixWord { appendix }
  # @SectionNumbers { Arabic }
  # @SectionHeadingFont { Bold }
  # @SectionGap { 2.00v }
  # @SectionInContents { Yes }
}
```

Section 4.1 explains how to make your own setup file and change its options.

The @IndexWord option determines what the index is called, if there is one. The default value, index, produces the word 'Index' in the current language. Any other value produces itself. The @AppendixWord option is similar; its default value is 'Appendix' in the current language.

@SectionNumbers determines how sections will be numbered, and may be None, Arabic, Roman, UCRoman, Alpha, or UCAlpha. The default value is Arabic for sections and also all other large-scale structure symbols except appendices, for which it is UCAlpha. This produces the appendices numbered in upper-case letters (A, B, C, etc.) that were mentioned earlier.

@SectionHeadingFont is the font used for section headings. The default value shown above produces the bold face from the family of the initial font. A family name or size is acceptable here as well:

```
@SectionHeadingFont { Helvetica Base +2p }
```

makes the section heading appear in the Helvetica font, two points larger than the initial size.

@SectionGap determines how much space is left blank before each section title; the default value shown above is twice the current inter-line spacing. There are similar options for other large-scale structure symbols, which determine how much space is left before each one.

@SectionInContents determines whether or not an entry is made in the table of contents for each section; it may be Yes or No, but would always be Yes. The default value of the corresponding options for sub-subsections and sub-subappendices, however, is No.

## 3.2. Technical reports

To make a technical report, start off with the report setup file and the @Report symbol:

```
@SysInclude { report }
@Report
   @Title {}
   @Author {}
   @Institution {}
   @DateLine { No }
   @CoverSheet { Yes }
   @InitialFont { Times Base 12p }
   @InitialBreak { hyphen adjust 1.2fx }
   @InitialSpace { lout }
   @InitialLanguage { English }
   @PageHeaders { Simple }
   @ColumnNumber { 1 }
   @FirstPageNumber { 1 }
   @OptimizePages { No }
//
```

This shows all the options of @Report with their default values. As usual with options, they may be given in any order, and only the ones whose values need to be changed need be given at all. The meaning of the // symbol after the last option is beyond our scope, but disaster will ensue if it is forgotten.

The @Title option holds the title of the report. It will be printed using the clines paragraph breaking style (Section 1.9), which centres each line, so it makes sense to have multi-line titles:

```
@Report
   @Title {
The solution of real instances of
the timetabling problem
}
   ...
```

With a multi-line title, each line after the first should begin at the left margin, not indented. It doesn't matter where the first line begins, because space following an open brace is ignored.

The @Author and @Institution options hold the author's name and institution or address, and will also be printed using the clines style. If there are several authors but only one institution, list all the authors in the @Author option:

```
@Author { Tim B. Cooper and Jeffrey H. Kingston }
```

With more authors, or with more than one institution, it is best to ignore the @Institution option and place all the information within the @Author option, enclosing institution information in @I symbols. In extreme cases, a table with columns of authors might be necessary (Chapter 6).

@DateLine may be set to No, meaning no dateline, Yes, meaning print the current date, or anything else, which is taken to be a date and printed:

```
@DateLine { 4 July, 1776 }
```

A good plan is to start off with @DateLine { Yes } and introduce a final date when the report is finalized.

The remaining options are a selection of setup file options (Section 4.1) that frequently need to be changed. If your changes to the overall formatting are confined to these options, you can change them here and avoid having your own setup file. If you already have your own setup file, change them in either place and omit them in the other.

If the @CoverSheet option is Yes, an unnumbered cover sheet will be produced containing the title, author, institution, abstract, and dateline. Otherwise these things will appear on the first page.

@InitialFont is the font in which the bulk of the report will be set, and should contain a family, a face, and a size. The default value selects the Times family, the Base face, and the 12 point size.

@InitialBreak controls the behaviour of paragraph breaking in the bulk of the report. It should have three parts: a paragraph breaking style (adjust, ragged, etc.), an inter-line spacing (1.2fx for single spacing, 2.4fx for double spacing, and so on), and either hyphen or nohyphen for turning hyphenation on or off.

@InitialSpace determines how Lout treats white space between two objects, as described in Section 1.18. @InitialLanguage determines the language of the bulk of the report.

@PageHeaders determines the appearance of page headers and footers throughout the report. Its value may be None, Simple, Titles, or NoTitles. Section 4.4 has the details, but just briefly, None produces no page headers, Simple produces a centred page number between hyphens on every page except the cover sheet and the first page, Titles produces full running titles as in the present document, and NoTitles is like Titles with the running titles omitted, leaving just the page numbers.

@ColumnNumber is the number of columns per page in the bulk of the report, and may be anything from 1 (the default value) to 10. However, there is nothing analogous to the @FullWidth symbol of ordinary documents. Instead, the cover sheet, title material, and all figures and tables will be printed full width, and the rest will be set in columns. There is a separate @IndexColumnNumber option in the setup file which determines the number of columns in the index (Section 2.10).

@FirstPageNumber is the page number given to the first page.

Lout ordinarily places lines onto a page until space runs out, then moves to the next page and so on. This often produces ugly empty spaces at the bottoms of pages preceding large unbreakable displays. Setting the @OptimizePages option to Yes causes Lout to examine the overall situation and try to minimize the ugliness, using the TeX optimal paragraph breaking algorithm. It takes two runs to do this, with intermediate results stored in Lout's cross reference database (Section 2.8); so deleting file lout.li will reset it, which might be wise after major changes.

After the compulsory // comes an optional abstract:

```
@Abstract
   @Title { Abstract }
@Begin
@LP
...
@End @Abstract
```

This will be printed either on the cover sheet or on the first page, depending on the @CoverSheet option of @Report (see above). The default value of @Title is ‘Abstract’ in the current language, so you can usually omit @Title.

Next comes the body of the report in the form of a sequence of sections:

```
@Section
   @Title { Introduction }
@Begin
@PP
...
@End @Section
```

No @BeginSections or @EndSections symbols are needed. The general rule is that you need these bracketing symbols when you are inside something else, but not when you aren’t. Sections lie inside @Text in ordinary documents, but they don’t lie inside anything else in technical reports.

Each section may contain subsections, bracketed by @BeginSubSections and @EndSubSections:

```
preceding text
@BeginSubSections
@SubSection ... @End @SubSection
@SubSection ... @End @SubSection
...
@SubSection ... @End @SubSection
@EndSubSections
```

Within each subsection there may be sub-subsections, each introduced by @SubSubSection, with the whole sequence bracketed by @BeginSubSubSections and @EndSubSubSections:

```
preceding text
@BeginSubSubSections
@SubSubSection ... @End @SubSubSection
@SubSubSection ... @End @SubSubSection
...
@SubSubSection ... @End @SubSubSection
@EndSubSubSections
```

There are no sub-sub-subsections.

After the sections comes an optional sequence of appendices, each introduced by @Appendix in the usual way:

```
@Appendix
   @Title { Derivation of the renewal formula }
@Begin
@PP
...
@End @Appendix
```

No @BeginAppendices or @EndAppendices symbols are needed, because (like the sections above) these appendices do not lie inside any other large-scale structure symbol. The appendices are numbered A, B, C, etc., as is conventional for them. Within each appendix there may be a sequence of subappendices, obtained with the @SubAppendix symbol and bracketed by @BeginSubAppendices and @EndSubAppendices:

```
preceding text
@BeginSubAppendices
@SubAppendix ... @End @SubAppendix
@SubAppendix ... @End @SubAppendix
...
@SubAppendix ... @End @SubAppendix
@EndSubAppendices
```

There are sub-subappendices following the same pattern, but no sub-sub-subappendices.

The report ends with the last section or appendix; any reference list or index will be appended automatically. Although we have described how to create reports as though everything was in one large file, in practice it is much better to divide the report into multiple files, following the method given in Section 3.7.

In addition to the @Title option, each large-scale structure symbol (@Abstract, @Section, @SubSection, @SubSubSection, @Appendix, @SubAppendix, and @SubSubAppendix) has a @Tag option for cross referencing (Section 2.8), an @InitialLanguage option for changing the language of that part of the document, and a @RunningTitle option which will be used in place of @Title in running headers if given. This last is useful when the full title is rather long.

The features described in other chapters are all available within technical reports. To get a table of contents, change the @MakeContents option in the setup file to Yes; the rest is automatic, and you don't need the @ContentsGoesHere symbol from ordinary documents. To get an index, again you need only change the @MakeIndex setup file option to Yes. Endnotes and references appear at the end of the report. Figures and tables are numbered 1, 2, 3, etc.

Within the report setup file there is a @ReportLayout symbol whose options control the appearance of features specific to reports (in other words, the features described in this section). Section 4.1 explains setup files and their options in general; here is a representative sample of these options, showing their default values:

```
@Use { @ReportLayout
  # @CoverSheet { Yes }
  # @DateLine { No }
  # @ReferencesBeforeAppendices { No }
  # @AbstractWord { abstract }
  # @ContentsWord { contents }
  # @SectionNumbers { Arabic }
  # @SectionHeadingFont { Bold }
  # @SectionGap { 2.00v }
  # @SectionInContents { Yes }
}
```

@CoverSheet and @DateLine are as for @Report; you can set them in either place as you prefer. @ReferencesBeforeAppendices determines whether the reference list is printed out before or after any appendices. @AbstractWord determines the value of the title of the abstract if none is given there; its default value, abstract, prodcues 'Abstract' in the current language. @ContentsWord is similar; its default value produces 'Contents' in the current language. The other four options control the appearance of sections, and there are similar options for controlling the other large-scale structure symbols.

@SectionNumbers determines how sections will be numbered, and may be None, Arabic, Roman, UCRoman, Alpha, or UCAlpha. The default value is Arabic for sections, and also for all large-scale structure symbols except appendices, for which it is UCAlpha. This produces the appendices numbered in upper-case letters (A, B, C, etc.) that were mentioned earlier.

@SectionHeadingFont is the font used for section headings. The default value shown above produces the bold face from the family of the initial font. A family name and size is acceptable here as well:

```
@SectionHeadingFont { Helvetica Base +2p }
```

produces section headings in the Helvetica font, two points larger than the initial font size.

@SectionGap determines how much space is left blank between sections; the default value is twice the initial inter-line spacing. @SectionInContents determines whether or not an entry is made in the table of contents for each section; it may be Yes or No.

## 3.3. Books

To produce a book, start off with the book setup file and the @Book symbol:

```
@SysInclude { book }
@Book
  @Title {}
  @Author {}
  @Edition {}
  @Publisher {}
  @BeforeTitlePage {}
  @AfterTitlePage {}
  @InitialFont { Times Base 12p }
  @InitialBreak { adjust 1.2fx hyphen }
  @InitialSpace { lout }
  @InitialLanguage { English }
  @PageHeaders { Titles }
  @ColumnNumber { 1 }
  @FirstPageNumber { 1 }
  @IntroFirstPageNumber { 1 }
  @OptimizePages { No }
//
```

This shows all the options of @Book with their default values. As usual with options, these may be given in any order, and only those that must be changed need be given at all. The meaning of the // symbol after the last option is beyond our scope, but total disaster will ensue if it is forgotten.

The @Title, @Author, and @Edition options will appear on the title page, in the clines paragraph breaking style which centres each line (Section 1.9). The @Publisher option will appear at the foot of the title page.

The @BeforeTitlePage option will come out on the page (or pages) preceding the title page. This is where publishers traditionally advertise other books of a similar kind, perhaps from a series. If this option is empty or omitted, there will be no such pages. If it is non-empty, there will be a pre-title page (a title page with just the title on it) first.

The @AfterTitlePage option will come out on the page (or pages) following the title page. This is where publishers traditionally put copyright notices, information about production, and cataloguing-in-publication data. If this option is empty or omitted, there will be no such pages.

The remaining options are a selection of setup file options (Section 4.1) that frequently need to be changed. If your changes to the overall formatting are confined to these options, you can change them here and avoid having your own setup file. If you already have your own setup file, change them in either place and omit them in the other.

@InitialFont is the font in which the bulk of the book will be set, and should contain a family, a face, and a size. The default value selects the Times family, the Base face, and the 12 point size.

@InitialBreak controls the behaviour of paragraph breaking in the bulk of the book. It should have three parts: a paragraph breaking style (adjust, ragged, etc.), an inter-line spacing (1.2fx for single spacing, 2.4fx for double spacing, and so on), and either hyphen or nohyphen for turning hyphenation on or off.

@InitialSpace determines how Lout treats white space between two objects, as described in

Section 1.18. @InitialLanguage determines the language of the bulk of the book.

@PageHeaders determines the appearance of page headers and footers throughout the book. Its value may be None, Simple, Titles, or NoTitles. Section 4.4 has the details, but just briefly, None and Simple are not really suitable for books, Titles produces full running titles as in the present document, and NoTitles is like Titles with the running titles omitted, leaving just the page numbers.

@ColumnNumber is the number of columns per page in the bulk of the book, and may be anything from 1 (the default value) to 10. Irrespective of its value, all prefatory material, all chapter and appendix headings, and all figures and tables will be printed full width. There is a separate @IndexColumnNumber option in the setup file which determines the number of columns in the index (Section 2.10).

@FirstPageNumber is the page number given to the first non-introductory page. @IntroFirstPageNumber is the page number of the first introductory page (it will ordinarily appear in Roman but must be given in Arabic).

Lout ordinarily places lines onto a page until space runs out, then moves to the next page and so on. This often produces ugly empty spaces at the bottoms of pages preceding large unbreakable displays. Setting the @OptimizePages option to Yes causes Lout to examine the overall situation and try to minimize the ugliness, using the TEX optimal paragraph breaking algorithm. It takes two runs to do this, with intermediate results stored in Lout's cross reference database (Section 2.8); so deleting file lout.li will reset it, which might be wise after major changes.

After the compulsory // comes an optional preface:

```
@Preface
   @Title { About this book }
@Begin
@PP
...
@End @Preface
```

Since the title of most prefaces is simply Preface, that is the default value in English of the @Title option. After the preface there will automatically appear a table of contents listing the introduction, chapters, sections, subsections, appendices, sub-appendices, bibliography, and index as appropriate.

The pages up to this point will be numbered in lower case Roman numerals; subsequent pages will be numbered in Arabic starting from the @FirstPageNumber option of @Book. There is a setup file option for changing this to a single numbering sequence (see below).

Next comes an optional introduction, exactly like the preface except that its name is @Introduction and the default title in English is Introduction:

```
@Introduction
@Begin
@PP
...
@End @Introduction
```

After that comes a sequence of chapters in the usual style:

```
@Chapter
   @Title { Australian Native Plants }
@Begin
@PP
...
@End @Chapter
```

No @BeginChapters or @EndChapters symbols are needed, because these chapters are not inside any other large-scale structure symbol. Within a chapter, there may be a sequence of sections, each introduced by @Section in the usual way, all bracketed by @BeginSections and @EndSections:

```
preceding text
@BeginSections
@Section ... @End @Section
@Section ... @End @Section
...
@Section ... @End @Section
@EndSections
```

Within each section there may be subsections, each introduced by @SubSection, and the sequence as a whole bracketed by @BeginSubSections and @EndSubSections:

```
preceding text
@BeginSubSections
@SubSection ... @End @SubSection
@SubSection ... @End @SubSection
...
@SubSection ... @End @SubSection
@EndSubSections
```

The subsections may contain sub-subsections following the same pattern. There are no sub-sub-subsections.

After the chapters comes an optional sequence of appendices, each introduced by @Appendix in the usual way:

```
@Appendix
   @Title { Climatic Regions of Australia }
@Begin
@PP
...
@End @Appendix
```

No @BeginAppendices or @EndAppendices symbols are needed, because (like chapters) these appendices do not lie inside any other large-scale structure symbol. The appendices are numbered A, B, C, etc., as is conventional for them. Within each appendix there may be a sequence of subappendices, obtained with the @SubAppendix symbol and bracketed by

@BeginSubAppendices and @EndSubAppendices:

```
preceding text
@BeginSubAppendices
@SubAppendix ... @End @SubAppendix
@SubAppendix ... @End @SubAppendix
...
@SubAppendix ... @End @SubAppendix
@EndSubAppendices
```

There are sub-subappendices following the same pattern, but no sub-sub-subappendices.

The book ends with the last chapter or appendix; any reference list or index will be appended automatically. Although we have described how to create books as though everything was in one large file, in practice it is much better to divide the book into multiple files, following the method given in Section 3.7.

In addition to the @Title option, each large-scale structure symbol (i.e. @Preface, @Introduction, @Chapter, @Section, @SubSection, @SubSubSection, @Appendix, @SubAppendix, and @SubSubAppendix) has a @Tag option for cross referencing (Section 2.8), an @InitialLanguage option for changing the language of that part of the document, and a @RunningTitle option which will be used in place of @Title in running headers if given. This last is useful when the full title is rather long.

The @Chapter symbol has three additional options for dividing the book into parts:

```
@Chapter
   @PartNumber { Part A }
   @PartTitle { The Ancient World }
   @PartText { ... }
```

Any chapter with a non-empty @PartTitle option will become the first chapter of a part. It will be preceded by two pages containing the part number, part title, and part text, and there will also be an entry made in the table of contents. @PartNumber and @PartText may be omitted if desired. Parts are *not* numbered automatically: you have to supply your own numbers or letters as in the example above.

The features described in other chapters are all available within books. A table of contents and index will appear automatically, and you will need to change the setup file to avoid them. Endnotes will appear at the end of the enclosing preface, introduction, chapter, or appendix. The numbering of figures and tables includes a chapter or appendix number: the first figure of Appendix C will be Figure C.1, and so on. Figures and tables within the preface or introduction are numbered 1, 2, 3, etc. A figure or table will never appear on the same page as the beginning of a chapter or appendix. References work as described in Chapter 5. As explained there, it is possible to have a list of references at the end of each chapter as well as at the end of the book.

Within the book setup file there is a @BookLayout symbol whose options control the appearance of features specific to books (in other words, the features described in this section). Here is a representative sample of these options, showing their default values:

```
@Use { @BookLayout
   # @TitlePageFont { Helvetica Base }
   # @SeparateIntroNumbering { Yes }
   # @ReferencesBeforeAppendices { No }
   # @ChapterStartPages { Any }
   # @ChapterWord { chapter }
   # @ChapterNumbers { Arabic }
   # @ChapterHeadingFont { Bold 2.00f }
   # @ChapterHeadingBreak { ragged 1.2fx nohyphen }
   # @ChapterHeadingFormat { number @DotSep title }
   # @AboveChapterGap { 3.00f }
   # @ChapterInContents { Yes }
}
```

Section 4.1 explains how to make your own setup file and change its options. @TitlePageFont is the font used on the title page of the book, not including a size. @ChapterStartPages determines what kinds of pages chapters and other major components of the book may begin on, and may be Any, Odd, or Even, meaning any page, odd-numbered pages only, or even-numbered pages only. @SeparateIntroNumbering determines whether the introductory part of the book is to have a separate numbering sequence or not. @ReferencesBeforeAppendices determines whether any final list of references appears before or after any appendices. @ChapterWord determines the word used in chapter titles; its default value, chapter, produces ‘Chapter’ in the current language. The other six options control the appearance of chapters, and there are similar options for controlling the other large-scale structure symbols.

@ChapterNumbers determines how chapters will be numbered, and may be None, Arabic, Roman, UCRoman, Alpha, or UCAlpha. The default value is Arabic for chapters and also for all large-scale structure symbols except appendices, for which it is UCAlpha. This produces the appendices numbered in upper-case letters (A, B, C, etc.) that were mentioned earlier.

@ChapterHeadingFont is the font used for chapter headings. The default value shown above produces the bold face from the family of the initial font, at twice the initial size. A family name is acceptable here as well. @ChapterHeadingBreak is the break style for chapter headings.

@ChapterHeadingFormat allows you to change the format of the heading. The symbol number within it will be replaced by the number of the chapter (actually including the word Chapter as well in the current language, e.g. Chapter 12); the symbol title within it will be replaced by the title. So you could write, say,

```
@ChapterHeadingFormat { @Box paint { lightgrey } { number @DP title } }
```

to get the title below the number, both enclosed in a box. The default value uses the @DotSep symbol from Section 4.4 to produce the number and title separated by a dot and two spaces, roughly the same as

```
@ChapterHeadingFormat { number.  title }
```

except when there is no number. For consistency this option is applied to other major headings as well: in the preface, introduction, table of contents, appendices, reference list, and index. In all these other cases, number is an empty object, except for appendices, when it contains Appendix A or whatever.

There is a @PartHeadingFormat option for determining the format of part headings. It works in the same way as @ChapterHeadingFormat, with number and title symbols standing for the @PartNumber and @PartTitle options of the relevant @Chapter symbol. The default value is

```
@PartHeadingFormat { @CD number @DP @CD title }
```

which centres the number and title. The default paragraph breaking style is clines, but you may include a @Break symbol to change this.

The example of boxed titles for chapters given above suffers from two practical deficiencies. First, the box won't extend right across the page, and second, when there is no number we don't want the @DP either. Here is a value for @ChapterHeadingFormat that solves both of these problems and looks good in practice:

```
@ChapterHeadingFormat {
   number @Case {
      {} @Yield @Box paint { lightgrey } @HExpand { title }
      else @Yield @Box paint { lightgrey } @HExpand { number @DP title }
   }
}
```

This uses the @Case symbol (described in the Expert's Guide [5]) to distinguish between the cases where number is empty and non-empty; and it uses the @HExpand symbol to expand the horizontal space occupied by the heading to the maximum possible, so that when the box is drawn around it it will occupy the full page width. The format object can be as complicated as you like, and there is no need to squeeze it all onto one line; as always in Lout, the end of a line is the same as one space.

Every chapter and appendix begins on a new page. @AboveChapterGap determines how much space is left blank above the chapter title; the default value is three times the initial font size. There are similar options for other large-scale structure symbols, which determine how much space is left before each one.

@ChapterInContents determines whether or not an entry is made in the table of contents for each chapter; it may be Yes or No, but would always be Yes. The default value of the corresponding options for sub-subsections and sub-subappendices, however, is No.

### 3.4. Overhead transparencies

To produce overhead transparencies (hereafter called overheads), start off with the slides setup file and the @OverheadTransparencies symbol:

```
@SysInclude { slides }
@OverheadTransparencies
   @Title {}
   @RunningTitle {}
   @Author {}
   @Institution {}
   @DateLine { No }
   @InitialFont { Times Base 20p }
   @InitialBreak { ragged 1.2fx nohyphen }
   @InitialSpace { lout }
   @InitialLanguage { English }
   @PageHeaders { Titles }
   @FirstPageNumber { 1 }
   @FirstOverheadNumber { 1 }
   @FirstLectureNumber { 1 }
   @OptimizePages { No }
//
```

This shows all the options of @OverheadTransparencies with their default values. As usual with options, they may be given in any order, and only the ones whose values need to be changed need be given at all. The meaning of the // symbol after the last option is beyond our scope, but disaster will ensue if it is forgotten.

If @Title is not empty, an initial overhead will be produced containing the @Title, @Author, @Institution, and @DateLine options. @DateLine may be set to No, meaning no dateline, Yes, meaning print the current date, or anything else, which is taken to be a date and printed.

Each overhead has a running header printed in small type at the top left. The @RunningTitle option goes into this header, or, if there is no @RunningTitle option, @Title is used instead.

The remaining options are a selection of setup file options (Section 4.1) that frequently need to be changed. If your changes to the overall formatting are confined to these options, you can change them here and avoid having your own setup file. If you already have your own setup file, change them in either place and omit them in the other.

@InitialFont is the font in which the overheads will be set, and should contain a family, a face, and a size. Experience shows that 20 points is a good font size for overheads, so that is the default size.

@InitialBreak controls the behaviour of paragraph breaking in the overheads. It should have three parts: a paragraph breaking style (adjust, ragged, etc.), an inter-line spacing (1.2fx for single spacing, 2.4fx for double spacing, and so on), and either hyphen or nohyphen for turning hyphenation on or off. Adjusted lines and hyphenated words are difficult to read from overheads, so the default is not to have them.

@InitialSpace determines how Lout treats white space between two objects, as described in Section 1.18. @InitialLanguage determines the language of the overheads.

@PageHeaders determines the appearance of page headers and footers. Its value may be None, Simple, Titles, or NoTitles. Section 4.4 has the details, but just briefly, None produces no page headers, Simple produces page numbers only, Titles produces full running titles, and NoTitles is similar to Simple in this context.

@FirstPageNumber is the number given to the first page, @FirstOverheadNumber is the number given to the first overhead, and @FirstLectureNumber is the number given to the first lecture, of which more below.

Lout ordinarily places lines onto a page until space runs out, then moves to the next page and so on. This often produces ugly empty spaces at the bottoms of pages preceding large unbreakable displays. Setting the @OptimizePages option to Yes causes Lout to examine the overall situation and try to minimize the ugliness, using the TeX optimal paragraph breaking algorithm. It takes two runs to do this, with intermediate results stored in Lout's cross reference database (Section 2.8); so deleting file lout.li will reset it, which might be wise after major changes.

After the compulsory // come the overheads themselves. There are two alternatives here: you may either choose to make a simple series of overheads, corresponding to a single lecture, or you may choose to make a series of series of overheads, corresponding to a series of lectures. To make a simple series of overheads, use this arrangement:

```
@SysInclude { slides }
@OverheadTransparencies
   ...
//
@Overhead ... @End @Overhead
@Overhead ... @End @Overhead
...
@Overhead ... @End @Overhead
```

@Overhead is a large-scale structure symbol, similar to @Section, with the usual options:

```
@Overhead
   @Title { Trends in investment since 1980 }
   @RunningTitle { Investment }
   @Tag { investment }
   @InitialLanguage { English }
@Begin
...
@End @Overhead
```

If @Title is given it will appear as a centred, bold display at the beginning of the overhead. As usual, these options may be given in any order or omitted altogether.

The body of the overhead is quite arbitrary. Typically one tends to use lists and displays more than paragraphs, but all the usual features are available. Each overhead begins on a fresh page, but it may occupy more than one page.

Lout does not provide any special support for overlays. A good way to make them is to first produce one overhead containing all the layers simultaneously. Once this is correct, enclose the entire body of the overhead in white @Colour, make one copy of the text of the overhead for each layer, separating the copies with @NP (new page) symbols, and, in each copy, enclose the parts that are to appear in that layer in black @Colour (or any other colour). This works because white @Colour makes an object invisible without altering its size.

We turn now to the second major alternative, which is to make a series of lectures, each of which is a series of overheads. Use this arrangement:

```
@SysInclude { slides }
@OverheadTransparencies
   ...
//
@Lecture ... @End @Lecture
@Lecture ... @End @Lecture
...
@Lecture ... @End @Lecture
```

@Lecture is a large-scale structure symbol, again with the usual options:

```
@Lecture
   @Title { Macro-Economic Policies for the Nineties }
   @RunningTitle { Macro-economic policies }
   @Tag { macro-economics }
   @InitialLanguage { English }
@Begin
...
@End @Lecture
```

If @Title is non-empty the series of overheads will begin with an overhead containing the title alone, centred on the page using the clines paragraph breaking style. This means that it makes sense to have a multi-line title.

Within the body of @Lecture, place a series of overheads bracketed by @BeginOverheads and @EndOverheads:

```
@BeginOverheads
@Overhead ... @End @Overhead
@Overhead ... @End @Overhead
...
@Overhead ... @End @Overhead
@EndOverheads
```

The @Overhead symbol is exactly as described earlier.

The features described in other chapters are available with overheads. Endnotes and references appear automatically at the end of the overheads. You can have a table of contents, by setting the @MakeContents option of the setup file to Yes. It will appear automatically after any title overhead. However, it is not possible to have an index, and it is not possible to have multiple columns.

Within the slides setup file there is an @OverheadLayout symbol whose options control the appearance of features specific to overheads (in other words, the features described in this section). Here are all these options and their default values:

```
@Use { @OverheadLayout
  # @DateLine { No }
  # @FirstOverheadNumber { 1 }
  # @FirstLectureNumber { 1 }
  # @ContentsWord { contents }
  # @LectureNumbers { Arabic }
  # @OverheadNumbers { Arabic }
  # @TitlePageFont { Helvetica Base }
  # @LectureHeadingFont { Bold 1.20f }
  # @OverheadHeadingFont { Bold }
  # @LectureInContents { Yes }
  # @OverheadInContents { No }
  # @ReferencesInContents { Yes }
}
```

For general information about setup files and their options, consult Section 4.1. The first four options are as for @OverheadTransparencies as described above. ContentsWord determines table of contents heading; its default value, contents, produces 'Contents' in the current language. @LectureNumbers and @OverheadNumbers determine the style of numbering of lectures and overheads, and may be None, Arabic, Roman, UCRoman, Alpha, or UCAlpha as usual. Next come options for controlling the font used for the overall title page, the title page of each lecture, and the heading of one overhead, and finally three options which determine which entries are made in the table of contents, if there is one.

Other setup file options exist which permit you to have a box drawn around each overhead, and to change the page size, margins, and orientation. These are described in Chapter 4.

Section 4.4 describes the setup file options that control the appearance of page headers and footers. With overhead transparencies, the values given to the @MajorTitle, @MinorTitle, @MajorNum, and @MinorNum symbols used within those options are as follows. If @Lecture is being used:

| | |
|---|---|
| @MajorTitle | The @RunningTitle option of @OverheadTransparencies, or its @Title option if @RunningTitle is absent; |
| @MinorTitle | The @RunningTitle option of the current @Lecture, or else its @Title option if @RunningTitle is absent; |
| @MajorNum | The number of the current @Lecture; |
| @MinorNum | A two-part number, for example 5.2, containing the number of the current @Lecture and the number within that lecture of the current overhead. |

If @Lecture is not being used:

| | |
|---|---|
| @MajorTitle | The @RunningTitle option of @OverheadTransparencies, or its @Title option if @RunningTitle is absent; |
| @MinorTitle | Empty; |
| @MajorNum | Empty; |
| @MinorNum | The number of the current overhead. |

The first page occupied by any overhead is a Start page; subsequent pages are NonStart pages. There are no Intro pages.

## 3.5. Stand-alone illustrations

This section describes how to use Lout to produce an illustration for inclusion in some other document, which may itself be a Lout document but need not be. The opposite process, the inclusion of an illustration in a Lout document, is the subject of Section 8.5.

Suppose you want to produce the following logo as an illustration for inclusion in some other document:


This is just an object, and it is not hard to make it using Lout's graphics features:

```
45d @Rotate @CurveBox { ARMY @LP 180d @Rotate ARMY }
```

The problem is that objects ordinarily come out on pages with margins, page numbers, and so forth, which we don't want here. The solution is to use the illustration document type, whose file name, curiously enough, is picture:

```
@SysInclude { picture }
@Illustration {
   45d @Rotate @CurveBox { ARMY @LP 180d @Rotate ARMY }
}
```

After the usual @SysInclude line comes one @Illustration symbol. Following it is an arbitrary object which becomes the entire result, with no pages and no margins, ready for inclusion in some other document as an illustration.

The @Illustration symbol has options for setting the initial font, paragraph breaking style, colour, and language. Here they are with their default values:

```
@Illustration
   @InitialFont { Times Base 12p }
   @InitialBreak { adjust 1.2fx hyphen }
   @InitialSpace { lout }
   @InitialLanguage { English }
   @InitialColour { black }
{
   ...
}
```

You can specify any colour from the list in Section 8.1, for example blue, and then your illustration will have that colour wherever it is included.

Because there are no pages, the width and height of the result are indeterminate, depending on how large the object turns out to be. This makes things very awkward for filled paragraphs and centring, which depend on knowing how much space is available to be occupied. So you should either avoid filled paragraphs and all displays and lists altogether in illustrations, or else enclose your object in a @Wide symbol:

```
@Illustration 5c @Wide {
    ...
}
```

to make clear how wide you want your illustration to be.

The technical name for a file containing a stand-alone illustration is 'encapsulated PostScript file' or 'EPS file' for short. To get Lout to produce an encapsulated PostScript file instead of an ordinary PostScript file, you have to use the -EPS Unix command line flag. For example, suppose the Lout file containing our example illustration is called army; then the appropriate Unix command for formatting it is

```
lout -EPS army > army.eps
```

An EPS file is supposed to contain only one 'page', so Lout will refuse to generate any second or subsequent pages when the -EPS flag is given. There is also a minor difference in format between ordinary and encapsulated PostScript files, which is the main reason why the -EPS flag is needed at all.

### 3.6. Plain text documents

Occasionally you may need to produce an output file containing plain text rather than PostScript, for example for an online manual entry or to send as electronic mail. Any document that can be produced by Lout in PostScript can be produced in plain text as well, by adding a -p flag to the Unix command line:

```
lout -p simple
```

No other changes are required, although one is recommended (use of the pdoc setup file, see below). Here we are sending the output directly to the screen, but it can be redirected to a file, or piped through the more command for viewing one page at a time, etc.

Of course, plain text is an extremely limited medium of communication compared with PostScript, and this forces Lout to make some rather drastic compromises:

- Symbols like @Bullet, which stand for unusual characters, produce printable characters which approximate the PostScript ones. For example, @Bullet produces o. However, the @Char and @Sym symbols often produce unprintable characters, and are best avoided;
- All font and size changes are ignored, since plain text has only one font and size. Every character is taken to be $^1/_{10}$ inch wide and $^1/_6$ inch high;
- No underlines are printed;

- No margin notes are printed;
- Scaled objects are not printed unless the scale factor happens to be 1;
- Rotated objects are not printed unless the angle happens to be zero degrees. This means that page orientations (Section 4.2) other than Portrait do not work;
- Ruled lines are not printed, and paint and colour options are ignored. This spoils the graphics and graphs of Chapters 8, 9, and 10.

Despite the problems, many things work surprisingly well. Tables, for example, lose their ruled lines but otherwise look very good. It does no harm to try things and see if they work out.

The worst problem with plain text is that characters cannot be placed at arbitrary points on the page. A superscript, for example, is impossible to place correctly, so Lout uses a different layout for footnote labels (and makes a mess of equations, which are best avoided). Because of this problem it's best to make all horizontal lengths multiples of $^1/_{10}$ inch (conveniently expressed as 1s), and all vertical lengths multiples of $^1/_6$ inch (conveniently expressed as 1f). To help you do this, there is a setup file called pdoc which is exactly like doc except that the values of the setup file options are chosen to suit plain text output. Use it like this:

```
@SysInclude { pdoc }
@Doc @Text @Begin
...
@End @Text
```

No plain text versions of the setup files for other document types are supplied, but they could easily be made following the example of pdoc.

If you use lout -P instead of lout -p, the plain text output will contain a form-feed character (control-L) after each page except the last. This character causes most printing devices to start a new page, which is very useful when your page height is not exactly right.

### 3.7. Organizing large documents

It is not a good plan to store a large document in a single large file. It takes too long to find things in it, and if some catastrophe occurs, you lose the lot. Lout encourages you to break documents into pieces by its willingness to read a sequence of files (lout file1 file2 ...). For large documents, the following plan is recommended.

Suppose you are making a book whose third chapter contains sections on banksias, grevilleas, acacias, and eucalypts. Place each section, from @Section to @End @Section, in a separate file, making four files called, say, banksias, grevilleas, acacias, and eucalypts. Then make a single file for the chapter as a whole whose contents are as follows:

```
@Chapter
   @Title { Australian Native Plants }
@Begin
Australian native plants provide a distinctive identity to the
garden.  Although less colourful than their European alternatives,
some banksias and grevilleas do flower strongly, and of course the
acacias (wattles) are unsurpassable in late winter.
@BeginSections
@Include { banksias }
@Include { grevilleas }
@Include { acacias }
@Include { eucalypts }
@EndSections
@End @Chapter
```

The @Include symbol causes Lout to read the file whose name follows it between braces, just as though the contents of that file had been included at that point.

With this arrangement you can easily rearrange the order of the sections: just swap their @Include lines. You should be using Lout's automatic cross referencing features (Section 2.8), so you don't have to worry about keeping cross references up to date. You can also temporarily delete a section by placing a # character at the start of its line:

```
# @Include { acacias }
```

This works because # is the *comment character*: Lout will ignore this character (unless enclosed in double quotes) and everything following it up to the end of the line. You can even temporarily delete every section except the one you are working on at the moment, using these comments.

Suppose now that this chapter file is called natives, and you have others called preface, flowers, etc. Then you can make one file (call it garden) for the whole book like this:

```
@SysInclude { book }
@Book
   @Title { The Australian Garden }
   @Author { Martha S. Vineyard }
//
@Include { preface }
@Include { flowers }
@Include { shrubs }
@Include { natives }
@Include { trees }
```

You can play the same tricks here: swap chapters around, or temporarily delete one or more with a #. When a chapter is finished you can temporarily delete it to save formatting time and paper, and bring it back at the end. To format the book, use lout garden > out.ps in Unix. Lout will read each @Include file as it comes to it, and if it finds an @Include of a section while reading a chapter file, it will read the section too.

If you decide to store chapters in separate Unix directories, make sure that any / characters in the file names are enclosed in double quotes:

```
@Include { "natives.dir/acacias" }
```

Be careful not to give the directory the same name as your chapter file. You might also find it useful to construct your book *top-down*, as computer scientists call it, laying out all the chapters and sections as empty skeletons and filling their contents in later.

# Chapter 4. Changing the Overall Format

The symbols of Lout make many decisions behind the scenes. Even the humble @PP symbol has to decide how much vertical space to leave, and how far to indent the first line of the paragraph. How to change these decisions is the subject of this chapter.

## 4.1. Setup files

As mentioned briefly in Section 1.1, each Lout document begins with an instruction to include (i.e. to read) a *setup file*:

```
@SysInclude { doc }
```

The setup file's name in this example is doc, and the Sys in @SysInclude means that doc is stored in the *Lout system include directory*, which is where all the standard setup files are kept. Each document type (Chapter 3) has its own setup file.

To change the overall format of a document, you need to create your own setup file by copying and modifying one of the standard ones. We will assume that you are making an ordinary document, with the doc setup file, but a similar procedure works for any setup file.

You first need to find out the name of the Lout system include directory, by typing

```
lout -V
```

in Unix. This causes Lout to print out various facts about itself. Then, supposing that this tells you that the Lout system include directory is /usr/lout/include, type the Unix command

```
cp /usr/lout/include/doc mydoc
```

to place a copy of the doc setup file in your directory, renaming it mydoc. Since doc is read-only, you may also need to change the mode of mydoc to be writable (by chmod +w mydoc in Unix). Now replace

```
@SysInclude { doc }
```

at the beginning of your document by

```
@Include { mydoc }
```

and Lout will read mydoc as the setup file instead of doc. Since the two files are at present identical, this has changed nothing so far; but now any changes you make to mydoc will affect your document. Notice the use of @Include rather than @SysInclude; @Include will search your current directory for mydoc, whereas @SysInclude searches only the system directory.

The remainder of this section is a tour through doc, explaining the various parts and how to modify them. The first lines that actually do anything are these:

```
@SysInclude { fontdefs }
@SysInclude { langdefs }
@SysInclude { dl }
@SysInclude { docf }
```

We already know that @SysInclude causes Lout to read a file from the Lout system include directory. Files fontdefs and langdefs tell Lout what fonts and languages there are. File dl contains the definition of the DocumentLayout package, in which all the symbols of the first four chapters of this guide are defined. File docf contains extra definitions specific to ordinary documents (as distinct from technical reports, books, or the other document types of Chapter 3). So this fourth line will be different in the setup files for those other types.

Now let's look at the next six lines of doc:

```
# @SysInclude { tab }
# @SysInclude { eq }
# @SysInclude { fig }
# @SysInclude { graph }
# @SysInclude { cprint }
# @SysInclude { pas }
```

These lines, each beginning with #, are *comments*: when Lout encounters a # character not enclosed in double quotes, it ignores it and everything following it up to and including the end of that line. So these lines do nothing as they stand, but by deleting one or more # characters you can have the @SysInclude lines for these files, which contain the definitions of Lout's specialized formatting packages, in your setup file, rather than at the beginning of your document.

The next line is

```
@Include { mydefs }
```

This searches your current directory for a file called mydefs, which (as Section 2.12 explains) is intended to hold your own personal set of definitions of new symbols. It does no harm if there is no mydefs file in your current directory, because @Include then searches the Lout system include directory for it, and there is an empty mydefs file there. When using your own setup file, you might prefer to delete @Include { mydefs } and put your definitions in its place, so that you have one file of setup-type material rather than two.

Next we come to the main thing: the DocumentLayout @Use clause. It looks like this:

```
@Use { @DocumentLayout
  # @InitialFont { Times Base 12p }
  # @InitialBreak { adjust 1.20fx hyphen }
  # @InitialLanguage { English }
  # @InitialColour { black }
  # @HeadingFont { Bold }
  # @ParaGap { 1.30vx }
}
```

@DocumentLayout is a symbol, and @InitialFont, @InitialBreak, etc. are its options. There are over one hundred options altogether; the display above just shows the first six. You change the overall format of your document by changing these options.

As it stands, the options are all hidden within comments, so the default values (shown within braces) are in force. To change an option, delete the # and change the value between braces. For example, to set the document in Helvetica 10 point font, change the @InitialFont line to

```
@InitialFont { Helvetica Base 10p }
```

We won't go through all the options now, since they are the subject of following sections.

The standard setup files are all much the same up to this point; the main variation is that in some files, some options are already set. The slides setup file, for example, contains

```
@InitialFont { Times Base 20p }
```

so that overhead transparencies will have a large font size. However, now comes a second @Use clause whose symbol and options depend on the document type. For ordinary documents (i.e. in the doc setup file) this clause is

```
@Use { @OrdinaryLayout
  # @IndexWord { index }
  # @AppendixWord { appendix }
  # @SectionNumbers { Arabic }
  # @SubSectionNumbers { Arabic }
  # @SubSubSectionNumbers { Arabic }
  # @AppendixNumbers { UCAlpha }
  # @SubAppendixNumbers { Arabic }
  # @SubSubAppendixNumbers { Arabic }
  # @SectionHeadingFont { Bold }
}
```

Once again this is just some of the options. In the slides setup file for overhead transparencies, we find this:

```
@Use { @OverheadLayout
  # @DateLine { No }
  # @ContentsWord { contents }
  # @FirstLectureNumber { 1 }
  # @FirstOverheadNumber { 1 }
  # @LectureNumbers { Arabic }
  # @OverheadNumbers { Arabic }
  # @TitlePageFont { Helvetica Base 1.5f }
  # @LectureHeadingFont { Bold 1.20f }
  # @OverheadHeadingFont { Bold }
  # @LectureInContents { Yes }
  # @OverheadInContents { No }
  # @ReferencesInContents { Yes }
}
```

In general this second @Use clause assigns values to options specific to the document type we are using, whereas the first @Use clause assigns values to options that are relevant to many or all document types.

The setup files used with C and C++ program printing (cdoc, creport, etc.) contain a third @Use clause, providing options specific to printing C and C++ programs. The setup file ends with a comment identifying the spot where database declarations should be put, and one such declaration, for reference printing styles.

## 4.2. Page size and page orientation

This section explains how to use the setup file options that determine page size and page orientation. Here they are with their default values:

```
@PageType { A4 }
@PageWidth {}
@PageHeight {}
@PageOrientation { Portrait }
```

The usual way to determine the page size is to set the @PageType option to the name of the paper you use:

| | *width in points* | *height in points* |
|---|---|---|
| @PageType { Letter } | 612p | 792p |
| @PageType { Tabloid } | 792p | 1224p |
| @PageType { Ledger } | 1224p | 792p |
| @PageType { Legal } | 612p | 1008p |
| @PageType { Statement } | 396p | 612p |
| @PageType { Executive } | 540p | 720p |
| @PageType { A3 } | 842p | 1190p |
| @PageType { A4 } | 595p | 842p |
| @PageType { A5 } | 420p | 595p |
| @PageType { B4 } | 729p | 1032p |
| @PageType { B5 } | 516p | 729p |
| @PageType { Folio } | 612p | 936p |
| @PageType { Quarto } | 610p | 780p |
| @PageType { 10x14 } | 720p | 1008p |

This will automatically assign the widths and heights shown above to the @PageWidth and @PageHeight options, so you don't have to worry about those options.

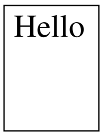
If your paper size is not on this list, set @PageType to Other and supply your own width and height:

```
@PageType { Other }
@PageWidth { 12.0c }
@PageHeight { 18.0c }
```

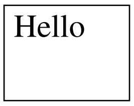
The width and height may each be any length (Section 1.2), and do not have to be in points.

The basic page orientations are *portrait* (shorter side at the top) and *landscape* (longer side at the top):

@PageOrientation { Portrait }



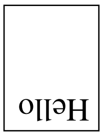

@PageOrientation { Landscape }




When changing to Landscape, do not change the page type, page width, or page height, and do not change the way you feed your paper into the printer. Lout knows what to do.

Two other orientations are provided which are 180° rotations of the basic ones:

@PageOrientation { ReversePortrait }




@PageOrientation { ReverseLandscape }


ReverseLandscape might be useful when post-processing the PostScript output to print two landscape pages per sheet of paper.

### 4.3. Page margins, page boxes, and page backgrounds

There are six options for setting the top and bottom margins on each page, and the left and right margins on odd and even pages. Here they are with their default values:

```
@TopMargin { 2.50c }
@FootMargin { 2.50c }
@OddLeftMargin { 2.50c }
@OddRightMargin { 2.50c }
@EvenLeftMargin { 2.50c }
@EvenRightMargin { 2.50c }
```

When setting these options you must ensure that

```
@OddLeftMargin + @OddRightMargin = @EvenLeftMargin + @EvenRightMargin
```

In other words, the total margin on odd pages must be the same as on even pages.

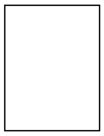
You can have a box drawn around each page if you wish. Here are the relevant options and their default values:

```
@PageBoxType { None }
@PageBoxMargin { 1.00c }
@PageBoxLineWidth {}
@PageBoxPaint { None }
@PageBoxShadow { 0.06c }
```

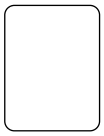
You get boxes by changing the @PageBoxType option:

@PageBoxType { None } (no box)

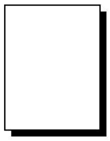
@PageBoxType { Box }



@PageBoxType { CurveBox }



@PageBoxType { ShadowBox }



Page boxes reduce the amount of space available to the page contents, so your columns will become somewhat narrower and shorter when you introduce them.

The @PageBoxMargin, @PageBoxLineWidth, @PageBoxPaint, and @PageBoxShadow options affect the page box exactly as the margin, linewidth, paint, and shadow options described for other boxes in Section 8.2 do. For example,

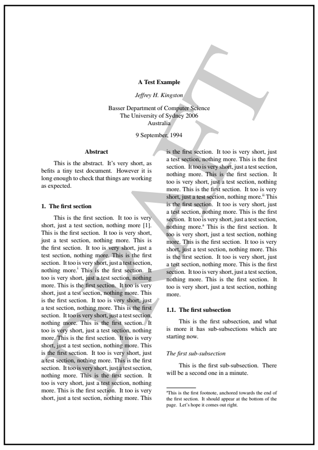
```
@PageBoxType { CurveBox }
@PageBoxMargin { 1.0c }
@PageBoxPaint { grey }
```

draws a curved box, painted grey, around each page, with a one centimetre margin between its boundary and the page contents. If the left margin is 2.5 centimetres, say, this gives a total left margin from the page edge to the page contents of 3.5 centimetres.

Finally, it is possible to have something other than the usual white background on the page, using the @PageBackground option:

```
@PageBackground { @Scale 60d @Rotate lightgrey @Colour DRAFT }
```

The value of the option is an object which is drawn on each page, within the margins, before the page contents are drawn. This example draws a large word DRAFT in light grey diagonally across each page:



DRAFT

**A Test Example**

*Jeffrey H. Kingston*

Basser Department of Computer Science
The University of Sydney 2006
Australia

9 September, 1994

**Abstract**

This is the abstract. It's very short, as befits a tiny test document. However it is long enough to check that things are working as expected.

**1. The first section**

This is the first section. It too is very short, just a test section, nothing more [1]. This is the first section. It too is very short, just a test section, nothing more. This is the first section. It too is very short, just a test section, nothing more. This is the first section. It too is very short, just a test section, nothing more.[i] This is the first section. It too is very short, just a test section, nothing more. This is the first section. It too is very short, just a test section, nothing more. This is the first section. It too is very short, just a test section, nothing more. This is the first section. It too is very short, just a test section, nothing more. This is the first section. It too is very short, just a test section, nothing more. This is the first section. It too is very short, just a test section, nothing more. This is the first section. It too is very short, just a test section, nothing more. This is the first section. It too is very short, just a test section, nothing more. This is the first section. It too is very short, just a test section, nothing more. This is the first section. It too is very short, just a test section, nothing more. This is the first section. It too is very short, just a test section, nothing more.[ii] This is the first section. It too is very short, just a test section, nothing more. This is the first section. It too is very short, just a test section, nothing more.[a] This is the first section. It too is very short, just a test section, nothing more. This is the first section. It too is very short, just a test section, nothing more. This is the first section. It too is very short, just a test section, nothing more. This is the first section. It too is very short, just a test section, nothing more. This is the first section. It too is very short, just a test section, nothing more.

**1.1. The first subsection**

This is the first subsection, and what is more it has sub-subsections which are starting now.

*The first sub-subsection*

This is the first sub-subsection. There will be a second one in a minute.

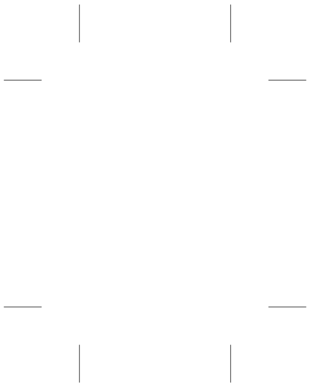
[a]This is the first footnote, anchored towards the end of the first section. It should appear at the bottom of the page. Let's hope it comes out right.

You have to find a suitable angle by experiment. As Section 8.4 explains, @Scale with no scale factor only takes account of the available horizontal space, not the available vertical space, so if your angle is too steep the result will be too tall for the page and you will get a regrettably obscure warning message about a 'broken size constraint.' The solution is to try a smaller angle.

Another useful page background draws marks to show where the margins lie:

```
@PageBackground { @BoundaryMarks }
```

produces something like this around each page:



The @BoundaryMarks symbol has options for controlling the line width (thickness), the line length, and the gap between the ends of the lines and the corner of the text area:

```
@PageBackground {
   @BoundaryMarks
      linewidth { 0.2p }
      length { 0.5c }
      gap { 0.5c }
}
```

This shows the default values: 0.2 points for line width, 0.5 centimetres for the others.

## 4.4. Page numbers and running headers

A *page header* is a line at the top of a page containing a page number or running title. A *page footer* is a similar line at the bottom of a page. This section describes the setup file options that control the appearance of page headers and footers.

There are four basic styles, selected by the @PageHeaders option:

| | |
|---|---|
| @PageHeaders { None } | No page headers, no page footers. |
| @PageHeaders { Simple } | No footers, and a centred page number between hyphens for header on every page whose number is not 0 or 1. |
| @PageHeaders { Titles } | Full running titles as in the present document. |
| @PageHeaders { NoTitles } | Page numbers placed as for Titles, but with the titles themselves blanked out. |

Titles and NoTitles use Lout's cross-referencing machinery, so will require a few runs to settle down. None and Simple do not, so they work first time and may be used with the -s command line flag. Section 2.8 has a fuller discussion of these ramifications of cross referencing.

The next step is to set the page numbers, using the @PageNumbers and @FirstPageNumber options. There are two useful values for @PageNumbers:

| | |
|---|---|
| @PageNumbers { Arabic } | Arabic page numbers |
| @PageNumbers { Roman } | Lower-case Roman page numbers |

although the full range of choices is None, Arabic, Roman, UCRoman, Alpha, and UCAlpha. @FirstPageNumber is the number of the first page. Its default value is of course 1, although

```
@FirstPageNumber { 0 }
```

might be useful if the first page is really an unnumbered cover sheet. @FirstPageNumber must be an Arabic number even if @PageNumbers is set to something other than Arabic.

Some types of documents, such as books, have a separate introductory sequence of pages preceding the main sequence. For the page numbers on introductory pages there are two options, @IntroPageNumbers and @IntroFirstPageNumber, which are exactly analogous to @PageNumbers and @FirstPageNumber. It is traditional to number introductory pages using Roman numerals, so Roman is the default value of @IntroPageNumbers.

Let's summarize the five options so far by looking at their values in the book setup file, which was used to produce the present document:

```
@PageHeaders { Titles }
@PageNumbers { Arabic }
@FirstPageNumber { 1 }
@IntroPageNumbers { Roman }
@IntroFirstPageNumber { 1 }
```

The remainder of this section goes beyond these basic choices to explain how to change the detailed appearance of page headers and footers. Inevitably it gets quite a lot harder.

Pages are classified by the page header options in three ways:

1. *Odd vs. even.* The first page is odd, the second is even, the third is odd, and so on. If @FirstPageNumber is set to an even number, the first page will have that number, but it will still be classified as odd.

2. *Start vs. non-start.* A start page is the first page of some major part of the document (a chapter, say); other pages are non-start. The Simple header type uses a simpler definition: a page whose number is 0 or 1 is a start page, all others are non-start.

3. *Intro vs. non-intro.* Intro pages form a separate sequence of pages that precede the main (non-intro) sequence. They typically contain prefatory material such as a title page, preface, and table of contents. There will always be an even number of Intro pages, even if it means that the last one is empty.

These classifications are quite independent of each other: a page could be a non-intro start odd page, or an intro non-start even page, and so on. This makes eight ($2 \times 2 \times 2$) possibilities altogether. Depending on the type of document there may also be pages that Lout will never place a page header or footer on. For example, no page headers or footers will appear on pages containing part titles in books.

If you choose @PageHeaders { None }, there are no page headers or footers, so there is nothing more to say. If you choose @PageHeaders { Simple }, then eight options become relevant for controlling the page headers on each of the eight kinds of pages. Here they are with their default values:

```
@OddTop { @Centre { - @PageNum - } }
@EvenTop { @Centre { - @PageNum - } }
@StartOddTop { @Null }
@StartEvenTop { @Null }
@IntroOddTop { @Null }
@IntroEvenTop { @Null }
@IntroStartOddTop { @Null }
@IntroStartEvenTop { @Null }
```

If the word Start is missing from an option name, the option applies to non-start pages; if Intro is missing, it applies to non-intro pages. Another eight options control footers in the same way:

```
@OddFoot { @Null }
@EvenFoot { @Null }
@StartOddFoot { @Null }
@StartEvenFoot { @Null }
@IntroOddFoot { @Null }
@IntroEvenFoot { @Null }
@IntroStartOddFoot { @Null }
@IntroStartEvenFoot { @Null }
```

The value of the option is an object which becomes the header or footer. It may be any object, but there are some peculiarities that will be explained now.

The full set of symbols of the DocumentLayout package unfortunately cannot be used within page header options, because we are still setting up this package at this point; the symbols become available only after the @Use clause. However, the following symbols *can* be used:

> @Underline @Font @Break @Language @Wide @Plain @Sym @R @I @B @BI
> @S @Date @Time “ ” -- --- @Bullet @ParSym @SectSym @Dagger @DaggerDbl
> @CDot @Sterling @Yen @Florin @Star @Degree @Minute @Second @Multiply
> @Divide @Lozenge @Register @CopyRight @TradeMark @Colour @Color
> @PageOf @Box @CurveBox @ShadowBox @Rotate @Scale @IncludeGraphic

And there are five symbols of special relevance to page headers and footers: @Null, @Centre, @Center, @Right, and @PageNum.

The @Null symbol is similar to the empty object in printing as nothing, but in addition it removes the vertical space that ordinarily separates the header line from the page body. If there is no header there should be no vertical space either, so always use @Null rather than the empty object in header and footer options.

@Centre and @Center centre the following object, and @Right right-justifies it:

```
at left  @Centre { - 27 - }  @Right { at right }
```

produces

| | | |
|---|---|---|
| at left | - 27 - | at right |

The objects should be enclosed in braces if they contain spaces.

The @PageNum symbol produces the number of the current page, in Arabic, Roman, etc. as specified by the @PageNumbers or @IntroPageNumbers option. @PageNum is available only within page header and footer options.

At this point you might like to pause and verify that the default values of the sixteen options given above produce what we said they would: no footers, and a centred page number between hyphens on every page whose number is not 0 or 1. It should be clear now what to do if you want to remove the hyphens, move the numbers to the page footer, make them bold, have them at the left on even pages and at the right on odd pages, and so on.

A different set of sixteen options applies when @PageHeaders is set to Titles or NoTitles. Here are the eight options for headers, with their default values:

```
@RunningOddTop { @I { @MinorNum @DotSep @MinorTitle }
    @Right @B @PageNum }
@RunningEvenTop { @B @PageNum
    @Right @I { @MajorNum @DotSep @MajorTitle } }
@RunningStartOddTop { @Null }
@RunningStartEvenTop { @Null }
@RunningIntroOddTop { @Null }
@RunningIntroEvenTop { @Null }
@RunningIntroStartOddTop { @Null }
@RunningIntroStartEvenTop { @Null }
```

Some options occupy two lines, but only because they are long: as usual, the end of a line is the same as one space. Here are the options for footers:

```
@RunningOddFoot { @Null }
@RunningEvenFoot { @Null }
@RunningStartOddFoot { @Centre { Bold 0.8f } @Font @PageNum }
@RunningStartEvenFoot { @Centre { Bold 0.8f } @Font @PageNum }
@RunningIntroOddFoot { @Right @PageNum  }
@RunningIntroEvenFoot { @PageNum }
@RunningIntroStartOddFoot { @Null }
@RunningIntroStartEvenFoot { @Null }
```

All these options are similar to the earlier ones, in providing one option for each of the eight kinds of pages. The names are the same except that Running is added to each. Remember that a start page is now one that begins a major part of the document.

In addition to the symbols described earlier for simple page headers and footers, these running header options may contain the symbols @MajorNum, @MajorTitle, @MinorNum, @MinorTitle, @DotSep, and @DotJoin.

The exact values of @MajorNum, @MajorTitle, @MinorNum, and @MinorTitle depend on the document type, but they are intended to describe what is on the current page. Here are some values typical of books:

| | |
|---|---|
| @MajorNum | Chapter 2 |
| @MajorTitle | Adding Structure to Documents |
| @MinorNum | 2.7 |
| @MinorTitle | Tables of contents |

It is not possible to change the values assigned to these symbols, but the sixteen options allow you to choose whether to use them and how to arrange them, in the usual way.

The @DotSep symbol consumes the objects to its left and right and produces them separated by a dot and two spaces:

```
@MinorNum @DotSep @MinorTitle
```

is the same as

```
@MinorNum.  @MinorTitle
```

However, if either object is empty, the dot and two spaces are omitted. It's a fine point, needed mainly for unnumbered chapters and sections. @DotJoin is the same as @DotSep but without the two spaces.

The present document was produced using @PageHeaders { Titles } with the default values of the sixteen options unchanged, as you might like to verify. @PageHeaders { NoTitles } is identical to @PageHeaders { Titles } except that @MajorNum, @MajorTitle, @MinorNum, and @MinorTitle are always replaced by empty objects. The description given at the beginning of this section, 'like Titles but with the titles blanked out,' is therefore accurate.

# Chapter 5. References

The simple way to make a list of references is to put them in a numbered or tagged list at the end of your document. If you use references only rarely, that is probably the best way, but if you use them frequently this chapter will save you hours of work in the long run.

Some good general principles and many examples have been given by van Leunen [12]. Broadly speaking Lout follows her recommendations, with some unification and scaling back as is inevitable with software. The reference formatting of Scribe [10] and LaTeX [7] were derived from the first edition of the same source, so there is a family resemblance which is close enough to make translation from Scribe and LaTeX references fairly straightforward.

### 5.1. Setting up a bibliographic database

The basic idea is to store your references in a separate *database file*, in a form which does not include formatting details such as font changes. This makes it easy to use the same references in many documents, and it leaves the formatting to Lout.

Here is an example of a reference as it would appear in a database file:

```
{ @Reference
    @Tag { vanleunen1992 }
    @Type { Book }
    @Author { Mary-Claire van Leunen }
    @Title { A Handbook for Scholars }
    @Publisher { Oxford }
    @Edition { Revised Edition }
    @Year { 1992 }
}
```

@Reference is a symbol, and @Tag, @Type, @Author, and so on are its options. The database file as a whole consists of a sequence of references, each enclosed in braces as shown.

The @Tag option is compulsory: since you cite a reference by giving its tag, there must be one. The @Type option is also compulsory, since it says whether the reference is to a book, a journal article, or whatever, and this determines what other options are required. Section 5.4 describes all the types provided by Lout, and Section 5.6 explains how to add your own.

Database file names are supposed to end in .ld for 'Lout database,' so now let's suppose that you have made a database file called refs.ld and put it in the same directory as your document. The next step is to place

```
@Database @Reference { refs }
```

at the start of your document, just before @Doc, @Document, @Report, or whatever. Alternatively, you may place it at the end of your setup file. It informs Lout that you might be referring to @Reference symbols in database refs (that is, in file refs.ld).

If you want to maintain a central database, used by many documents, you won't want it in the same directory as any one of them. A Unix pathname will be more appropriate:

```
@Database @Reference { "/usr/jeff/lib/refs" }
```

or whatever. Quotes are needed because of the / characters.

With the database file created and the @Database line in place, you are ready to start citing references. The first time that the database is used, Lout will create an *index file* whose purpose is to speed up the retrieval of your references. Thanks to this file you can have hundreds or even thousands of references in your database, without slowing Lout down very much. However, whenever you change your database file *you must remove its corresponding index file*, so that Lout knows to create it afresh.[1] The index file is stored in the same directory as the database file, and it has the same name except that it ends in .li rather than .ld (e.g. refs.li).

If a separate database file is not convenient for some reason, perhaps because you need a self-contained document in a single file, the @Reference symbols may be incorporated into the document itself, anywhere that ordinary text may appear. Nothing will appear where they are typed in, but Lout will notice them and treat them as if they had come from a database file. In this case no @Database symbol is needed unless you are referring to a database as well.

You may have multiple databases. For example, you might use a public database augmented with some entries of your own:

```
@Database @Reference { myrefs }
@Database @Reference { "/usr/pub/refs/theoryrefs" }
```

Lout will search the databases in the order you list them.

## 5.2. Citation

To cite one or more references, use the @Cite symbol like this:

```
This feature is beyond our scope @Cite { $kingston1995lout.expert, page 97 }.
```

The following object must be enclosed in braces. It may be an arbitrary object as usual. Within it the $ character is a symbol with a special meaning: it causes a citation to be made of the reference whose @Tag option is the word following the $ symbol:

This feature is beyond our scope [5, page 97].

The reference itself will appear automatically in a reference list at the end of the document, and the citation(s) will be enclosed in brackets as shown. There is no need to write ${kingston1995lout.expert}, as would normally be the case, because within @Cite special arrangements are made to prevent commas and semicolons from being a nuisance.

[1]Depending on how it was installed on your system, Lout may be able to use the time of last modification of the database file and its index file to determine automatically whether the index file needs to be created afresh, thus saving you the trouble of removing it. You can find out whether this is true of your system by typing the command lout -V.

A reference may be cited many times, but it will appear in the reference list only once. The references will ordinarily be sorted by tag and labelled with Arabic numbers, although this can be changed by setting options in the setup file (Section 5.5).

If you are making a book, there is a @ChapCite symbol which is the same as @Cite except that its references come out at the end of the current preface, introduction, chapter, or appendix, rather than at the end of the document.

It is quite all right to cite a reference from within a footnote, figure, table, or index entry. The reference will appear in the closest reference list following the citation point in the final printed document, or if there is no such list, the closest preceding reference list. This is fine in documents with just one reference list; but when using @ChapCite in books, if the citation point appears after the intended reference list (because the footnote or figure has floated past the reference list at the end of the chapter), the reference will come out in the wrong list.

Although it is frowned upon by the authorities, some people include references which are not cited anywhere in the body of their document. For this there is @NoCite:

```
... our scope @NoCite { $kingston1995lout.expert, $kingston1993lout.design }.
```

produces

... our scope.

with the @NoCite symbol and any preceding space removed. The references will nevertheless appear in the reference list as usual. There is a @NoChapCite symbol that combines @NoCite and @ChapCite. For compatibility with previous versions of Lout, there is a @Ref symbol:

```
@Ref kingston1995lout.expert
```

is the same as @Cite { $kingston1995lout.expert } except that it produces no brackets, just one bare label, at the point of citation. There are analogous @ChapRef, @NoRef, and @NoChapRef symbols, which are not recommended.

Sometimes you need to print a reference outside the concluding reference list. For this there is a @RefPrint symbol:

```
@RefPrint kingston1995lout.expert
```

has result

Jeffrey H. Kingston. *An Expert's Guide to the Lout Document Formatting System (Version 3).* Basser Department of Computer Science, University of Sydney, 1995.

unrelated to any reference list. For example,

```
@Heading { Journal Articles }
@NumberedList
@LI @RefPrint kingston1985tree
...
@LI @RefPrint kingston1993lout.design
@EndList
```

might appear in someone's resume.

### 5.3. Labelled (as opposed to numbered) references

Lout ordinarily assigns a number to each reference, and prints this number beside the reference in the reference list and at the point(s) of citation. There is a way to make Lout use a label of your choice instead of a number for each reference. First change the following setup file options to the values shown (these options are explained in Section 5.5):

```
@RefCiteLabels { @Label }
@RefListLabels { @Label. }
@RefListLabelWidth { 4.00f }
@RefListSortKey { @Label }
```

Then make sure that every reference you cite has a @Label option:

```
{ @Reference
   @Tag { kingston1995lout.expert }
   @Type { TechReport }
   @Label { Kin94 }
   ...
}
```

@Label may contain several words, and even font changes, but not an arbitrary object.

The effect of these changes is that your references will now be labelled with their @Label options instead of with numbers, and they will be sorted by label instead of by tag. However, tags are still used when citing.

The big problem with labels is that they vary from document to document, either because of a change of style or because the usual first few letters of the authors' names plus year has to be augmented with a, b, c etc. to distinguish publications by the same authors in the same year. To help you overcome these problems, the $ symbol has a label option:

```
@Cite { $ label { Kin94a } kingston1995lout.expert, ... }
```

The @Ref and @ChapRef symbols also have a label option. If you use this option, it will be used to label the reference instead of the @Label option from the @Reference symbol (indeed, the @Reference symbol need have no @Label option in this case). But note that using label does not itself give you labelled references; you get them with the setup file options as explained above.

If your labels turn out to be too wide for the space allowed for them in the reference list, you have two alternatives. One is to increase the @RefListLabelWidth setup file option shown above, since it determines this space. The other is to change the @RefListFormat setup file option to DropLabels, which produces drop items:

Kin94a.
    Jeffrey H. Kingston. *An Expert's Guide to the Lout Document Formatting System (Version 3).* Basser Department of Computer Science, University of Sydney, 1995.

Then it won't matter how wide your labels are.

## 5.4. Constructing database entries

Here is the complete, fixed list of options that you may give to the @Reference symbol, with a brief description of what each is for:

```
{ @Reference
    @Tag {}              Used to cite this reference
    @Type {}             The type of reference, for example Book, Article
    @Abstract {}         Not used, intended to hold an abstract
    @Address {}          The address of a publisher, organization, or institution
    @Annote {}           Not used, intended for annotations
    @Author {}           The author(s) or editor(s)
    @Day {}              The day of the month, for newspaper articles
    @Edition {}          The edition, for example Second Edition
    @HowPublished {}     How something strange has been published
    @InAuthor {}         The author of the work that the cited work appears within
    @InTitle {}          The title of the work that the cited work appears within
    @Institution {}      The institution or school
    @Journal {}          The journal name
    @Keywords {}         Not used, intended to hold keywords
    @Label {}            The label of a labelled reference
    @Month {}            The month of publication or writing
    @Note {}             Any additional helpful information
    @Number {}           The number of a technical report
    @Organization {}     The organization sponsoring the work
    @Page {}             Page number if only one, for example 23
    @Pages {}            Page numbers if more than one, for example 23--47
    @Pinpoint {}         A point or part of the work, for example Chapter VI
    @Publisher {}        The publisher of the work
    @Title {}            The title of the work
    @TitleNote {}        Additional title information (series, editor, etc.)
    @TRType {}           The type of a technical report, for example Research Note
    @URL {}              The URL of the reference
    @Volume {}           The volume of a journal
    @Year {}             The year of publication or writing
}
```

Every reference may contain any of these options, although, depending on the @Type option, only some will be printed. You can't give an option twice; in particular, multiple authors must be placed within one @Author option, arranged as you want them to appear. Here is the complete set of values that you may give to the @Type option:

| | | | |
|---|---|---|---|
| Book | TechReport | Article | InBook |
| Proceedings | MastersThesis | | InProceedings |
| PhDThesis | Misc | | |

Each column represents one broad category of reference type: the first contains large works; the second contains small works not appearing within anything else (although possibly part of a series); the third contains small works appearing within an ongoing forum for such works; and the fourth contains small works appearing within large works. In each case, the reference may be to the work as a whole, or to one point or part of it (known as pinpointing).

Some care is needed when choosing the @Tag option, since references are both cited and sorted by tag. It is best to choose a three-part tag consisting of the first author's surname and possibly initial, the year of publication, and a brief reminder of the contents:

```
@Tag { kingston1995lout.expert }
```

Keep to lower-case letters, since mixed cases confuse the sorting, and give the full four digits of the year to avoid trouble in the year 2000. Multi-word tags are possible but not recommended.

Unusually for Lout, you can have / characters inside the @URL option without enclosing them in quotes:

```
@URL { ftp://ftp.cs.su.oz.au/jeff/lout }
```

In fact it is better not to use quotes because then Lout will be able to break lines at / characters, which is very useful since URLs tend to be long and prone to causing bad line breaks.

Since the types within each broad category are similar, our plan is to give one example of each and briefly note how the others differ. Here is a Book entry showing all its options:

```
{ @Reference
    @Tag { homer.odyssey }
    @Type { Book }
    @Author { Homer }
    @Title { The Odyssey }
    @TitleNote { Translated by E. V. Rieu }
    @Pinpoint { Chapter VI }
    @Pages { 102--111 }
    @Page { 102 }
    @Publisher { Penguin Books }
    @Address { Harmondsworth, Middlesex }
    @Edition { Penguin Classics Edition }
    @Month { August }
    @Year { 1942 }
    @Note { The date of composition is unknown,
but is thought to be about the tenth century BC }
}
```

And here is what it produces:

> Homer. *The Odyssey*, Chapter VI, pages 102–111, page 102. Translated by E. V. Rieu. Penguin Books, Harmondsworth, Middlesex. Penguin Classics Edition, August 1942. The date of composition is unknown, but is thought to be about the tenth century BC.

The only compulsory options are @Tag, @Type, and @Title, and Lout will carefully adjust the formatting to the right thing when you omit others. A basic book would have just @Tag, @Type, @Author, @Title, @Publisher, and @Year options.

Proceedings is very similar to Book, except that you may have an @Organization or @Institution option for the sponsoring organization if you wish, and the author will either be absent or an editor:

```
@Author { P. W. Lamb, editor }
```

There is no option specifically for editors, translators, and so forth.

PhDThesis is very similar again, with @Institution instead of @Publisher, and the phrase 'Ph.D. thesis' appearing by magic in the right spot. Like all words and phrases introduced automatically by Lout, it will be translated into the current language if this is not English.

Moving now to the second broad category, here is a typical TechReport:

```
{ @Reference
   @Tag { christofides1976tsp }
   @Type { TechReport }
   @Author { Christofides, N. }
   @Title { Worst-case analysis of a new heuristic
for the travelling salesman problem }
   @Number { 388 }
   @Institution { Graduate School of Industrial
Administration, Carnegie-Mellon University }
   @Address { Pittsburgh, PA }
   @Year { 1976 }
}
```

Here is the result:

> N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Tech. Rep. 388 (1976), Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA.

The two novelties here are the @Number option, which is the number of the report, and the 'Tech. Rep.' phrase. If you need some other phrase instead, use the @TRType option:

```
@TRType { Programmer's Manual }
```

or whatever. The phrase will be 'Master's Thesis' in the current language for type MastersThesis, and absent in type Misc. You may use the pinpointing options (@Pinpoint, @Page, and @Pages) and @TitleNote, @Month, and @Note in the same way as for books.

Journal articles are referenced by journal name, volume, number, and page(s):

```
{ @Reference
    @Tag { kingston1993lout.design }
    @Type { Article }
    @Author { Jeffrey H. Kingston }
    @Title { The design and implementation of the
Lout document formatting language }
    @Journal { Software---Practice and Experience }
    @Volume { 23 }
    @Pages { 1001--1041 }
    @Year { 1993 }
}
```

The result of this is

> Jeffrey H. Kingston. The design and implementation of the Lout document formatting language. *Software—Practice and Experience* **23**, 1001–1041 (1993).

All are optional, as usual. Notice that @Pages and @Page refer to the whole article so are not available for pinpointing here, but you may still use @Pinpoint.

Finally, small works that appear within large works have @Author and @Title options for the work itself, and @InAuthor and @InTitle for the work that it appears within:

```
{ @Reference
    @Tag { rieu1942intro }
    @Type { InBook }
    @Author { E. V. Rieu }
    @Title { Introduction to @I { The Odyssey } }
    @InAuthor { Homer }
    @InTitle { The Odyssey }
    @Publisher { Penguin }
    @Year { 1942 }
}
```

@InAuthor would often be absent or an editor. The result is

> E. V. Rieu. Introduction to *The Odyssey*. In Homer, *The Odyssey*. Penguin, 1942.

The other options are as for large works. Type InProceedings is similar to InBook.

A database usually has a long life, and some day it might find itself used in a document whose language is not the one its original compiler had in mind. For this reason, a truly meticulous compiler of database entries would enclose *all* language-specific options in @Language symbols:

```
{ @Reference
   @Tag { zimand1986size.sets.strings }
   @Type { Article }
   @Author { French @Language { M. Zimand } }
   @Title { English @Language { On the topological size of sets of random strings } }
   @Journal { German @Language { Zeitschr. f. math. Logik und Grundlagen d. Math. } }
   @Volume { 32 }
   @Pages { 81--88 }
   @Year { 1986 }
}
```

(My apologies to M. Zimand if he or she is not French.) This ensures correct hyphenation whatever the language of the document in which the reference appears.

## 5.5. Changing the appearance of citations and the reference list

By default, citations appear like this [5], and the reference list appears like the one at the end of this document, with the entries numbered, and sorted by their @Tag options. This section explains how to change all this, by setting options in the setup file.

For a general introduction to setup files and their options, see Section 4.1. Here we just describe the setup file options that relate to references. Here they are, with their default values:

```
@MakeReferences { Yes }
@RefCiteStyle { [cite] }
@RefCiteLabels { @RefNum }
@RefNumbers { Arabic }
@RefListFormat { Labels }
@RefListLabels { [@RefNum] }
@RefListTitle { references }
@ChapRefListTitle { references }
@RefListIndent { 0c }
@RefListRightIndent { 0c }
@RefListGap { 1.00v }
@RefListFont { }
@RefListBreak { }
@RefListLabelWidth { 2.00f }
@RefListSortKey { @Tag }
```

Setting @MakeReferences to No will cause Lout to ignore all citation symbols and omit all reference lists.

@RefCiteStyle and @RefCiteLabels work together to determine the appearance of citations. The result of each @Cite symbol is the value of @RefCiteStyle with the cite symbol replaced by the object following the @Cite symbol. For example, the default value shown above encloses each citation in brackets. The cite symbol must appear exactly once within @RefCiteStyle.

@RefCiteLabels determines the appearance of each label within the citation. Within it, the @RefNum symbol will produce the number of the reference, and you may also use any of the options of the @Reference symbol listed at the beginning of Section 5.4. Here are some examples:

| | |
|---|---|
| @RefCiteLabels { @RefNum } | [3] |
| @RefCiteLabels { @Label } | [Kin93] |
| @RefCiteLabels { @Author, @Year } | [Jeffrey H. Kingston, 1993] |

The value of @RefCiteLabels may be any object. The @Label symbol will produce the label option of $ or @Ref if there is one, rather than the @Label option of the reference; this label option is explained in Section 5.3.

@RefNumbers determines the kind of numbering produced by the @RefNum symbol used within @RefCiteLabels above and @RefListLabels below. Its value may be Arabic, Roman, UCRoman, Alpha, or UCAlpha, as usual for numbering in Lout. If you don't use @RefNum, @RefNumbers has no effect.

The remaining eleven setup file options are all concerned with the appearance of the reference list. The first, @RefListFormat, determines the overall format of the list. Here is what its four possible values do:

| | |
|---|---|
| @RefListFormat { NoLabels } | William Strunk and E. B. White. *The Elements of Style*. Macmillan. Third Edition, 1979. |
| @RefListFormat { Labels } | 11. William Strunk and E. B. White. *The Elements of Style*. Macmillan. Third Edition, 1979. |
| @RefListFormat { DropLabels } | 11.<br>William Strunk and E. B. White. *The Elements of Style*. Macmillan. Third Edition, 1979. |
| @RefListFormat { InLabels } | 11. William Strunk and E. B. White. *The Elements of Style*. Macmillan. Third Edition, 1979. |

@RefListFormat is not concerned with the appearance of the labels and references, only with where they appear.

@RefListLabels determines the appearance of the labels in the reference list (and so has no effect if @RefListFormat is NoLabels). It is a combination of @RefCiteStyle and @RefCiteLabels; you can use @RefNum and all the options of @Reference within it. The default value,

```
@RefListLabels { @RefNum. }
```

produces a numbered reference list in the style of @NumberedList. Another useful value is

```
@RefListLabels { [@Label] }
```

which produces the @Label option of the reference, or the label option of the citation if there is one, enclosed in brackets. If you do switch to non-numeric labels you will need to either use DropLabels or else increase the @RefListLabelWidth option described below.

@RefListTitle determines the heading placed just before the reference list at the end of the document:

```
@RefListTitle { Further Reading }
```

Two special values, references and bibliography, produce References and Bibliography in English and their equivalents in other languages. @ChapRefListTitle is the same as @RefListTitle, but applied to the reference list at the end of each chapter of a book when @ChapCite is used.

@RefListIndent, @RefListRightIndent, and @RefListGap determine the left indent, right indent, and gap between reference list items, analogously to the indent, rightindent, and gap options of the @List symbol (Section 2.2). @RefListFont and @RefListBreak determine the font and paragraph breaking style of the reference list. For example,

```
@RefListFont { -2p }
@RefListBreak { 1.2fx outdent }
```

switches to a smaller size with outdented paragraphs (these work well with NoLabels). The empty default values produce the same font and break style as in the document as a whole.

@RefListLabelWidth determines the distance from the left edge of the labels to the left edge of the references, when @RefListFormat is Labels or DropLabels (it has no effect when @RefListFormat is NoLabels or InLabels). This is different to @RefListIndent, which determines the distance from the edge of the column to the left edge of the item.

Particular care is needed when @RefListFormat is Labels and the labels are non-numeric, for then if the labels are too wide they will overstrike the references. The default value, 2.00f, is twice the current font size. It may be changed to any length (Section 1.2). Regrettably, Lout is not clever enough to choose a good value by itself.

Finally, @RefListSortKey determines the sorting key used when ordering the reference list. The default value,

```
@RefListSortKey { @Tag }
```

sorts by tag; the other popular possibility is to sort by the @Label option:

```
@RefListSortKey { @Label }
```

As usual @Label will use the value of a label option to the citation if there is one. There is no way to sort by order of first appearance in the document.

The value of @RefListSortKey may be any sequence of words and options from the @Reference symbol, but not @RefNum for obvious reasons. A possible more elaborate sorting key is

```
@RefListSortKey { @Author @Year }
```

sorting first by author, then by year within each author. However you are supposed to choose tags which have this effect, and that is more reliable since the modern practice is to put the authors' surnames after their given names. There seems to be little practical use for sorting keys other than @Tag and @Label.

### 5.6. Creating your own entry types and formats

Although the set of options to the @Reference symbol (@Tag, @Type, @Author, etc.) is fixed, you can add your own reference types and change the formatting of existing types.

To do this you must be using your own setup file, as explained in Section 4.1. At the end of the setup file you will find this line:

```
@SysDatabase @RefStyle { refstyle }
```

This tells Lout to consult a database file of reference styles called refstyle.ld. These are not references, they are formatting styles, one for each reference type. The Sys in @SysDatabase means that this file is stored in the *Lout system database directory*, which is where all the standard databases are kept. To change the formatting of a reference type, or to add your own types, you need to create your own reference styles database file by copying and modifying refstyle.ld.

To find out the name of the Lout system database directory, type the Unix command

```
lout -V
```

Then, supposing that the Lout system database directory is /usr/lout/data, type

```
cp /usr/lout/data/refstyle.ld mystyle.ld
```

to place a copy of the refstyle.ld database file in your directory, renaming it mystyle.ld. Since refstyle.ld is read-only, you may also need to change the mode of mystyle.ld to be writable (by chmod +w mystyle.ld in Unix). Now replace

```
@SysDatabase @RefStyle { refstyle }
```

at the end of your setup file by

```
@Database @RefStyle { mystyle }
```

and Lout will read its reference styles from mystyle.ld instead of from refstyle.ld. Since the two database files are at present identical, this has changed nothing so far; but now any changes you make to mystyle.ld will affect your document. Changing @SysDatabase to @Database tells Lout to search your current directory for mystyle.ld, whereas @SysDatabase searches only the system directory.

In practice you will probably want to store your database of reference styles in some library directory of your own, so that it can be used by many documents. A Unix pathname is appropriate for this:

```
@Database @RefStyle { "/usr/jeff/lib/mystyle" }
```

Quotes are needed because of the / characters.

The database entries within refstyle.ld and mystyle.ld might look something like this:

```
{ Book @RefStyle @Style
  { @Reference&&reftag @Open
    {
      @Author. @I @Title. @Publisher, @Year.
    }
  }
}
```

The meaning of the first two lines is beyond our scope, except that Book on the first line means that this is the entry which defines how references of type Book will be printed. Fortunately, apart from this one word these two lines are the same in every reference style entry so you don't need to understand them. The important part is in the middle:

```
@Author. @I @Title. @Publisher, @Year.
```

The meaning should be clear: first print the author option and a full stop, then the title option and another full stop in italics, and so on. To change the formatting of books, change this object. To create a new reference type, copy the entire database entry, change Book to a new name of your choice, and change the middle part. Don't forget to delete the index file mystyle.li afterwards, if there is one, so that Lout knows to generate it afresh.

Although the entry shown above is perfectly viable, the real entry for Book is much more complicated, in part because there are more options than those basic four, but mainly because the real entry goes to great lengths to do the right thing when options are omitted:

```
{ Book @RefStyle @Style
  { @Reference&&reftag @Open
    {
      { @Author. {}                 } @If @Author
      { @I @Title                   } @If @Title
      { @Word&&notitle              } @If @Not @Title
      { , @Pinpoint                 } @If @Pinpoint
      { , @Word&&pages @Pages       } @If @Pages
      { , @Word&&page @Page         } @If @Page
      { . @TitleNote                } @If @TitleNote
      { . @HowPublished             } @If @HowPublished
      { . @Publisher                } @If @Publisher
      { . @Organization             } @If @Organization
      { . @Institution              } @If @Institution
      { , @Address                  } @If @Address
      { . @Edition                  } @If @Edition
      { , @Month @Year              } @If @Year @And @Month
      { , @Year                     } @If @Year @And @Not @Month
      { .                           } @If @True
      { {} URL @URL.                } @If @URL
      { {} @Note.                   } @If @Note
    }
  }
}
```

The meaning is that each object to the left of an @If will be printed only if the condition to the right of the @If is true. The condition may contain options, which are considered to be true if they are not omitted (non-empty), and it may contain @And, @Or, @Not, and @True with the usual precedence and meaning. Sub-conditions may be enclosed in braces if desired, although it is best to keep the conditions as simple as possible given the complexity of the whole setup.

The objects subject to @If are printed with no space preceding them; any space in the final print will be the result of space within them, not between them. This is why @If @True is not redundant.

The object @Word&&notitle produces No title in the current language; @Word&&pages produces pages in the current language, and so on. Consult database standard.ld for other standard words and phrases.

# Chapter 6. Tables

This chapter explains how to produce tables like this one:

| Value of mathematical formulae (millions of dollars) | | |
|---|---|---|
| *Quadratic formula* | $x = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ | 3.5 |
| *Binomial theorem* | $(a+b)^n = \sum_{k=0}^{\infty} \binom{n}{k} a^k b^{n-k}$ | 12 |

As the example shows, the tables may contain spanning columns, aligned columns, and rules, and the entries may be arbitrary objects.

### 6.1. Introduction

The Lout definitions for table formatting are kept in a file called tab, which you must include at the start of your document if you want tables, like this:

```
@SysInclude { tab }
@SysInclude { doc }
@Doc @Text @Begin
...
@End @Text
```

Files of definitions, like tab, must be included before the setup file. Alternatively, if you are using your own setup file, you may place the include commands within it; the place to put them is clearly marked in all the standard setup files.

To begin with a very simple example, the table

| | | |
|---|---|---|
| Austen | Chaucer | Donne |
| Balzac | Darwin | Goethe |
| Byron | Dickens | Homer |

is the result of the following Lout input:

```
@Tab
   @Fmta { @Col A  !  @Col B  !  @Col C }
{
   @Rowa A { Austen } B { Chaucer } C { Donne }
   @Rowa A { Balzac } B { Darwin } C { Goethe }
   @Rowa A { Byron } B { Dickens } C { Homer }
}
```

Immediately after the @Tab symbol, which introduces the table, comes a *format option*, @Fmta, describing the format of each row. It says that each row contains three columns: @Col A, @Col B, and @Col C. The format option may have up to 26 columns, with names chosen freely from the upper-case letters from A to Z. The symbol ! separates each pair of columns.

After the format option comes the body of the table, enclosed in braces. It consists entirely of a sequence of rows, each introduced by the @Rowa symbol and containing one entry for each column of the format option, as shown (the row may occupy any number of lines of the input file). The entries may be arbitrary Lout objects, such as words, paragraphs, equations, figures, and so on without restriction. An entry may be omitted altogether if it is empty. Lout will choose suitable widths for the columns, and break paragraphs in the entries to the right widths.

The result of the @Tab symbol is an object. As usual with Lout, this object may appear at any point in the document,[1] even within a paragraph or another table. Most commonly, though, tables are displayed using the @IndentedDisplay and @CentredDisplay symbols (Section 2.1):

```
@CentredDisplay @Tab ...
```

or else they go into the @Table symbol (Section 2.6):

```
@Table
   @Caption { ... }
@Tab ...
```

which centres them at the top of the following page and adds a caption. Note the difference between @Tab, which builds a table, and @Table, which places an arbitrary object in an appropriate place.

## 6.2. Changing the style of entries and columns

The style of an entry is mainly affected by three symbols: @Font, @Break, and @Wide. We will review them only briefly here, since they are treated elsewhere in this Guide, then explain how they are used within tables.

The @Font symbol changes the font and font size of an object. For example,

```
Slope @Font { hello world }
```

[1] In rare cases, when the table occupies an entire paragraph but is not displayed, a bug in the current version of Basser Lout causes the second column to appear much too far to the right. You can work around this problem by replacing @Tab by @OneCol @Tab when it occurs.

produces *hello world*, and

```
{ Bold +2p } @Font { hello world }
```

produces

**hello world**

which is two points larger than it otherwise would have been, as well as set in bold. The abbreviations @I and @B stand for Slope @Font and Bold @Font.

The @Break symbol affects paragraph breaking. Most relevant here are clines @Break and rlines @Break, which will centre or right-justify one or more lines in the column:

```
clines @Break {
A small centred
paragraph.
}
```

produces

A small centred
paragraph.

for example. Actually, there is a small problem with these symbols: they only work on paragraphs of at least two words, so they will fail to centre or right-justify a solitary word. @Tab provides two symbols, @CC and @RR, which do essentially the same thing, but which work on single words as well.

The @Wide symbol forces an object to have a given width, by padding it with white space on the right if it is too small, or breaking its paragraphs if it is too wide:

```
1.5i @Wide {
This small paragraph will be broken to
a width of one and one half inches.
}
```

produces

This small paragraph
will be broken to a
width of one and one
half inches.

Widths may be specified in centimetres, points or ems using the letters c, p and e instead of i, as explained in Section 1.2.

Although these three symbols are the most frequently used in table formats, more exotic symbols may be used equally well, for example @Rotate which rotates any object (the entire table, perhaps, or one entry) by any angle (Section 8.3). Symbols defined by the user may also be used.

To change the style of an individual entry, enclose it in the symbols required:

```
@Rowa
   A { @CC { one entry of the table } }
```

More commonly, the same style is to be applied to every entry in a particular column, and this is done by placing the symbols for the style inside the format option, between @Col and the column letter:

```
@Tab
   @Fmta { @Col 3c @Wide @CC A  !  @Col @I 90d @Rotate B  }
```

Any @CC or @RR symbol should appear last, immediately before the column letter. This example centres each entry of column A in a three centimetre space (excluding margins), and prints every entry of column B in italics, rotated 90 degrees.

It is best not to use the @Wide symbol at first, since Lout is quite good at choosing appropriate column widths: it leaves narrow columns unbroken, and breaks all wide ones to the same column width. If the result is not satisfactory, and the table is wide enough to require paragraph breaking within its entries, @Wide should be used only to reduce the width of the wider columns, not to increase the width of the narrower ones.

## 6.3. Changing the style of rows

Sometimes different rows need different formats, usually because they are headings. Although this can be done with style-changing symbols in the entries, it is probably better to use multiple format options:

```
@Tab
   @Fmta { @Col @Heading A  !  @Col @Heading B }
   @Fmtb { @Col A  !  @Col @CC B }
{
   @Rowa A { Course } B { Enrolment }
   @Rowb A { Software Engineering } B { 174 }
   @Rowb A { Complexity Theory } B { 37 }
}
```

has result

| **Course** | **Enrolment** |
|---|---|
| Software Engineering | 174 |
| Complexity Theory | 37 |

The @Heading symbol is from Section 1.7. Each row appears in the format of the corresponding format option: @Rowa rows in the format of @Fmta, @Rowb rows in the format of @Fmtb, and so on. There may be up to ten different format options, from @Fmta to @Fmtj, and they may be used repeatedly and in any order.

There is an @Over symbol for constructing *spanning columns*: columns that spread over two or more following columns. For example, the Lout input

```
@I @Tab
   @Fmta { @Col @CC X @Over A,B  !  @Col C }
   @Fmtb { @Col A  !  @Col B  !  @Col C }
{
   @Rowa X { X } C { C }
   @Rowb A { A } B { B } C { C }
}
```

produces the table

| *X* | | *C* |
|---|---|---|
| *A* | *B* | *C* |

(We will see how to get the lines later.) The @Over symbol always comes directly after the column letter, and it is followed by one or more letters separated by commas indicating which columns in the following row this column is to span over.

With multiple row formats, especially those with spanning columns, it is important to take care that each row is compatible with the one preceding it, in the sense of having the same number of columns, taking @Over symbols into account. For example, two successive @Rowa rows would not be compatible in the example above, since the second has only two columns but the first prepares the way for three.

@Tab does not provide a way to have spanning columns underneath the columns they span, so spanning columns are useful only for headings. This is unavoidable, because it is due to compromises made in the way that the Lout interpreter handles spanning columns.

## 6.4. Aligned columns

Columns of numbers are often presented with decimal points aligned:

```
   5.46
   3.4159
5772
```

You can produce this by placing a ^ symbol just before the alignment point in each entry:

```
@Tab
   @Fmta { @Col A }
{
   @Rowa A { 5^.46 }
   @Rowa A { 3^.4159 }
   @Rowa A { 5772^ }
}
```

The equals signs of equations can be aligned in the same way. The format of such columns should not contain @CC or @RR.

The simplest way to get a heading over an aligned column is to use the ^ symbol in the heading as well. If it is not possible to do this, for example because the heading entry is a paragraph of text broken over more than one line, the most viable alternative is to use @Over in the heading line's format to make the heading column span over the aligned column:

```
@Tab
   @Fmta { @Col @CC H @Over A }
   @Fmtb { @Col A }
{
   @Rowa H { Heading }
   @Rowb A { 5^.46 }
   @Rowb A { 3^.4159 }
   @Rowb A { 5772^ }
}
```

for example produces

```
              Heading
                 5.46
                 3.4159
           5772
```

Unfortunately, the aligned entries are left-justified in the column, not centred.

## 6.5. Margins

@Tab places a *vertical margin* above and below each row, and a *horizontal margin* to the left and right of each column. (If there are no side rules, the leftmost and rightmost margins are omitted.) The size of these margins is determined by two options to the @Tab symbol:

```
@Tab
   hmargin { 0.3c }
   vmargin { 0.2v }
   @Fmta { ... }
...
```

Shown here are the default values, 0.3 centimetres for the horizontal margins, and 0.2 times the current inter-line spacing (denoted by the letter v) for the vertical margins.

The horizontal margin is fixed, but each row may have its own vertical margin:

```
@Rowa
   vmargin { 0.5vx }
   A { ... }
```

If omitted, the vmargin value is taken from the @Tab option.

The value shown here, 0.5vx, requests a vertical margin of half the current inter-line spacing, measured from the baseline of the entry rather than from its edge (this is what the x means). This value is a good choice when the entries contain simple words or lines of text (i.e. no paragraphs, large equations, figures, etc.):

| **Course** | **Enrolment** |
|---|---|
| Software Engineering | 174 |
| Complexity Theory | 37 |

since the lines of the table will then be separated evenly like the lines in paragraphs.

### 6.6. Rules, boxes, multi-page tables, etc.

@Tab draws four kinds of rules. *Above rules* go across the table above a specified row; *below rules* go across the table below a row. *Between rules* go down the table between any two columns, but not before the first or after the last; and *side rules* go down the two sides:

```
@Tab
   above { no }
   below { yes }
   between { single }
   side { double }
   @Fmta { ... }
...
```

Each option may have the value no (no rules), yes or single (draw single rules), or double (draw double rules). Omitting an option is equivalent to giving it the value no. A rule below one row exactly overstrikes a rule above the next, unless a page break intervenes.

These options to @Tab will draw all possible rules of the given type. If only some are wanted, there are above and below options (but no side option) for each row symbol:

```
@Rowf
   above { yes }
   below { double }
   ...
```

and there is a between option for the ! symbol that separates columns in the format options:

```
@Col A  ! between { double }  @Col B
```

The symbols !! and !!! are abbreviations for ! between { single } and ! between { double }. All these options override the corresponding @Tab options when they are present; when they are absent, the @Tab options hold sway.

It is not hard to assemble a single box around a table, using side rules, a rule above the first row, and a rule below the last:

```
@Tab
   side { single }
{
   @Rowa above { single } ...
   ...
   @Rowj below { single } ...
}
```

Double boxes around a table should be assembled using the following recipe:

```
@Tab
   side { double }
{
   @Rowa above { doubleboxabove } ...
   ...
   @Rowj below { doubleboxbelow } ...
}
```

This is a little tedious, but the result is good.

The symbols @Line and @DoubleLine produce horizontal single and double lines of the width of the column. Two other symbols, @Rule and @DoubleRule, produce lines which extend into the adjacent margins as well. For example,

| |
|---|
| 5.46 |
| 3.4159 |
| 5772 |
| 5909.0759 |

is produced by

```
@Tab
   @Fmta { @Col A }
{
   @Rowa A { 5^.46 }
   @Rowa A { 3^.4159 }
   @Rowa A { 5772^ }
   @Rowa A { @Line } vmargin { 0i }
   @Rowa A { 5909^.0759 }
}
```

Owing to problems behind the scenes, @Rule and @DoubleRule fail in aligned columns.

The @Tab symbol has a linewidth option which determines the thickness of all the lines:

```
@Tab
   linewidth { 0.5p }
```

shows the default value, 0.5 points. This cannot be overridden within the table. Double rules are drawn with their centres three line widths apart. Tab thinks that rules and double rules occupy no space, so thick rules may require larger margins.

The tables produced by @Tab permit page breaks between any two rows. The choice of page breaks can either be left to Lout, or it can be forced by placing the new page symbol @NP between two rows. To ensure that each part of a multi-page table is boxed, the rows preceding the page breaks must have below rules, and the rows following them must have above rules. It works best to display a multi-page table at the left margin by preceding and following it with @DP. To prevent page breaks within a small table, use @OneRow @Tab ....

The @HAdjust symbol from raw Lout can be used to widen a table's columns until it occupies the full column width. The recommended arrangement is

```
@DP
@HAdjust @Tab
  ...
@DP
```

since @HAdjust is not effective within displays.

Expert users might occasionally find a use for the @MarkRow symbols. There is one of them for each @Row symbol (@MarkRowa for @Rowa, etc.). A @MarkRow symbol causes the principal mark of the table to pass through its row, permitting alignment with that row.

## 6.7. A final example

In conclusion, here is the input for a relatively large table:

```
@LeftDisplay 10p @Font 1.15fx @Break @Tab
   linewidth { 1.0p }
   vmargin { 0.4v }
   @Fmta { @Col @B A  !  @Col @B B  !  @Col @B C  !  @Col @B D }
   @Fmtb { @Col outdent @Break A  !  @Col 1.8c @Wide ragged @Break B !
         @Col C  !  @Col outdent @Break D }
{

@Rowa above { single } below { single }
   A { Scholarship }
   B { Value $ }
   C { Closing date }
   D { Qualifications }

@Rowa
   A { 1.  Tenable at the University of Sydney }

@Rowb
   A { Commonwealth and University Postgraduate Research Awards }
   B { 10,415 (12,724 to 16,433 from 1 January 1990) }
   C { 31 October }
   D { Graduates with Hons I or Hons II Div. 1.  For research in any field }

@Rowb
   A { Commonwealth Postgraduate Course Awards }
   B { 10,415 }
   C { 30 September }
   D { Graduates with honours degrees or very good pass degrees.  For
Master's degrees undertaken by coursework }
```

```
@Rowb
   A { R. and M. Bentwich Scholarship }
   D { Graduate who holds a postgraduate research scholarship and who
requires a supplementary grant }

@Rowa
   A { 2.  Awarded by external bodies }

@Rowb
   A { Caltex }
   B { 24,000 }
   C { 30 September }
   D { Female graduates completing degree or diploma in year of application }

@Rowb
   A { Rhodes Scholarship }
   B { @Sterling3500 + return air-fare }
   C { 1 October }
   D { Age limit 25.  For tenure at the University of Oxford }
   below { single }
}
```

The most interesting things about it are the use of outdented paragraphs, and the @Wide symbol which restricts the width of the second column, which would otherwise be as wide as the first and last. Here is the result:

| **Scholarship** | **Value $** | **Closing date** | **Qualifications** |
|---|---|---|---|
| **1. Tenable at the University of Sydney** | | | |
| Commonwealth and University Postgraduate Research Awards | 10,415 (12,724 to 16,433 from 1 January 1990) | 31 October | Graduates with Hons I or Hons II Div. 1. For research in any field |
| Commonwealth Postgraduate Course Awards | 10,415 | 30 September | Graduates with honours degrees or very good pass degrees. For Master's degrees undertaken by coursework |
| R. and M. Bentwich Scholarship | | | Graduate who holds a postgraduate research scholarship and who requires a supplementary grant |
| **2. Awarded by external bodies** | | | |
| Caltex | 24,000 | 30 September | Female graduates completing degree or diploma in year of application |
| Rhodes Scholarship | £3500 + return air-fare | 1 October | Age limit 25. For tenure at the University of Oxford |

# Chapter 7. Equations

This chapter explains how to produce mathematical formulas in Lout, using the @Eq symbol like this:

```
@Eq { big int supp 1 on 0 ‘ dx over sqrt {1 - x sup 2} = pi over 2 }
```

This example produces

$$\int_0^1 \frac{dx}{\sqrt{1-x^2}} = \frac{\pi}{2}$$

The @Eq symbol looks after all the details of spacing for you, and it provides several hundred mathematical symbols.

## 7.1. Introduction

The Lout definitions for the @Eq symbol are kept in a file called eq, which you must include at the start of your document if you want equations, like this:

```
@SysInclude { tab }
@SysInclude { eq }
@SysInclude { doc }
@Doc @Text @Begin
...
@End @Text
```

This shows what to do if you want both tables and equations, but you may leave out the line for tables if you don’t want them. Files of definitions, like tab and eq, must be included before the setup file, but may be included in any order. Alternatively, if you are using your own setup file, you may place the include commands within it; the place to put them is clearly marked in all the standard setup files.

With the eq file included, you may write

```
@Eq { ... }
```

at any point in your document, and the symbols of @Eq will be available between the braces. Any symbols available outside continue to be available inside, which means that equations may be freely mixed with other symbols, without restriction.

Equations may appear within a paragraph of text, or they may be displayed. @Eq’s job is to produce an object containing the equation; it neither knows nor cares where this equation goes. To get an equation within a paragraph, simply place @Eq { ... } at the desired point. To prevent it spreading over two lines, use @OneCol @Eq { ... }. To display an equation, use a display symbol like @IndentedDisplay or @CentredDisplay (Section 2.1). For example,

```
@CentredDisplay @Eq { int supp pi on 0 sin ` x = 0 }
```

produces

$$\int_0^\pi \sin x = 0$$

There are also symbols for aligned and numbered displays, which are very commonly used with equations. These symbols are the subject of Section 7.5.

In this chapter we show the Lout input at the left, and its result at the right:

| | |
|---|---|
| `@Eq { {x sup 2 + y sup 2} over 2 }` | $\frac{x^2 + y^2}{2}$ |

Subsequent examples will omit the enclosing @Eq { ... }.

### 7.2. Symbols

@Eq prints characters in the fonts appropriate for mathematics:

| | |
|---|---|
| `x - 2` | $x - 2$ |

Here $x$ is in Italic, 2 is in Roman, and – is from the Symbol font. The character - is a *symbol* which stands for –, and 2 is also a symbol, standing for 2. @Eq includes a vast number of symbols:

| | |
|---|---|
| `Omega delta int partial club` | $\Omega\delta\int\partial\clubsuit$ |

The summary at the end of this chapter has the complete list.

Symbols whose names are made from letters should be separated from each other by at least one space or end of line, as was done above, or else @Eq will become confused:

| | |
|---|---|
| `Omegadelta` | *Omegadelta* |

Symbols whose names are made from digits and punctuation characters can, however, be run together with each other and with symbols made from letters:

| | |
|---|---|
| `Omega-delta<=2` | $\Omega - \delta \leq 2$ |

This rule applies throughout Lout (Section 1.3).

Some symbols join objects together in mathematical ways:

| | |
|---|---|
| `x sub 2` | $x_2$ |

Here the sub symbol has taken the object just to its left, and the object just to its right, and joined them into one object in the form of a subscript. The two objects are called the left and right parameters of sub, and they may be arbitrary Lout objects.

Other symbols of a similar kind include sup for superscripting, over for built-up fractions, and from and to for the lower and upper limits of sums, products, etc. These symbols may be used together to produce complicated equations very easily:

```
big sum from i=0 to n r sup i
= {r sup n+1 - 1} over r-1
```

$$\sum_{i=0}^{n} r^i = \frac{r^{n+1} - 1}{r - 1}$$

Here sum is just the $\sum$ symbol; from and to do all the work of placing the limits. They are quite independent, so either or both may be omitted. To get a superscript directly over a subscript, use the supp and on symbols:

```
A supp b on a
```

$$A_a^b$$

These two symbols should always be used together as shown.

Sometimes a subscript appears to be too far to the right, owing to the slope of italic letters: in $W_n$, for example. You can fix this by using 'tucked' subscripts, like this:

```
W tsub n
```

$$W_n$$

```
W supp b ton a
```

$$W_a^b$$

The tsub and ton symbols are exactly like sub and on except for this tucking-in effect.

As usual in Lout, braces are used to group something into an indivisible object. Leaving them out creates ambiguities:

```
a sup b over c
```

There are two possible interpretations for this:

```
{a sup b} over c
```

$$\frac{a^b}{c}$$

```
a sup {b over c}
```

$$a^{\frac{b}{c}}$$

@Eq chooses between them in the following way. Every symbol that takes a parameter also has a *precedence*, which is a number. For example, sup has precedence 60 and over has precedence 54. The symbol with the highest precedence wins the object lying between them, so in the above case the first interpretation is chosen. If two symbols of equal precedence compete for an object, the association is towards the left:

```
a sup b sub 2
```

$${a^b}_2$$

In this case it is more probable that the following right association was actually wanted:

```
a sup { b sub 2 }
```

$$a^{b_2}$$

When in doubt, use braces to make the grouping clear.

White space between two objects is considered to be a symbol with precedence 7, which is lower than the precedence of any @Eq symbol; but if the two objects are immediately adjacent and neither is enclosed in braces the precedence is 102, which is higher than the precedence of any @Eq symbol. Compare these three examples:

```
big sum from i=0 to n
```
$$\sum_{i=0}^{n}$$

```
big sum from {i = 0} to n
```
$$\sum_{i=0}^{n}$$

```
big sum from i = 0 to n
```
$$\sum_{i} = \overset{n}{0}$$

and you will see that some care is needed on this point. Braces can always be used to override precedence and associativity, and when in doubt the easiest course is to insert them. Although Lout allows symbols to associate towards the left or right, @Eq chooses to have only left associative symbols. The summary at the end of this chapter gives the precedence of every symbol.

The matrix symbol builds an array of objects:

```
matrix
  atleft { blpar }
  atright { brpar }
{
 row col x sup 2 col y sup 2 col z sup 2
 row col x col y col z
 row col 1 col 1 col 1
}
```

$$\begin{pmatrix} x^2 & y^2 & z^2 \\ x & y & z \\ 1 & 1 & 1 \end{pmatrix}$$

The atleft and atright options place vertically scaled versions of their values at each side; if either is omitted the value is taken to be an empty object of zero width by default. Although we have used blpar and brpar here, since the options are vertically scaled to the correct size some people prefer simply

```
matrix
  atleft { ( }
  atright { ) }
```

The right parameter of matrix is the array itself. It must be enclosed in braces, and it is a sequence of rows introduced by row symbols; each row is a sequence of objects introduced by col symbols.[1] The row and col symbols have low precedence, but not as low as white space between two objects. Therefore, unless the entries in the array are very simple, it is safest to enclose each of them in braces.

Entries built with the col symbol have their objects centred in the column. Also available

[1]Older versions of Lout use different symbols, above and nextcol, at this point. For backward compatibility these symbols are still available, but they are obsolete and no longer documented.

are lcol for left-justified entries, ccol meaning the same as col, rcol for right-justified entries, and mcol for alignment along column marks. Each column may contain entries of different kinds, except that mcol does not work well with any other sort.

Some symbols have been added which produce ‘matrices’ with commonly needed atleft and atright options already set for you. Here are these symbols, on the left, with the equivalent matrix symbol and, on the right, the result produced:

| | | |
|---|---|---|
| pmatrix | matrix atleft { ( } atright { ) } { M } | $\left( M \right)$ |
| bmatrix | matrix atleft { blbrack } atright { brbrack } { M } | $\left[ M \right]$ |
| brmatrix | matrix atleft { blbrace } atright { brbrace } { M } | $\left\{ M \right\}$ |
| fmatrix | matrix atleft { blfloor } atright { brfloor } { M } | $\left\lfloor M \right\rfloor$ |
| cmatrix | matrix atleft { blceil } atright { brceil } { M } | $\left\lceil M \right\rceil$ |
| amatrix | matrix atleft { blangle } atright { brangle } { M } | $\left\langle M \right\rangle$ |

For example:

| | |
|---|---|
| fmatrix { (n+1) over 2 } | $\left\lfloor \dfrac{(n+1)}{2} \right\rfloor$ |

As this example shows, these symbols are very useful for getting large scaled delimiters around things that aren’t necessarily matrices at all.

Each of the @Eq symbols that takes parameters also has a gap option, which controls the amount of space inserted by the symbol:

| | |
|---|---|
| x over y | $\dfrac{x}{y}$ |
| x over gap { 3p } y | $\dfrac{x}{y}$ |

@Eq usually gets the spacing right without help.

## 7.3. Vertical positioning

Every equation and every object within every equation has an *axis* running through it which is used to position it vertically with respect to nearby objects. In the Expert’s Guide to Lout [5] this is called a *row mark*, but we’ll stick with axis. Here are some examples with the axis shown as a dotted line:

$$x^2 \qquad + \qquad \frac{1}{\sqrt{1-4x^2}}$$

When these objects are placed adjacent to one another, their marks are merged, giving the correct vertical positioning:

$$x^2 + \frac{1}{\sqrt{1 - 4x^2}}$$

Most of the time you do not need to think about vertical positioning, because for most objects there is just one sensible place for the axis to go, and Lout puts it there.

Matrices and the delimiters that enclose them are the two exceptions. Lout makes the axis of a matrix pass through its exact centre, and it shifts the axes of delimiters so that they exactly enclose the thing delimited. These choices are never disastrous, but there are other possibilities that might be better sometimes.

The axis of a matrix could reasonably be set to the axis of any of its rows:

$$\begin{matrix} x^3 & y^3 & z^3 \\ x^2 & y^2 & z^2 \\ x & y & z \end{matrix} \qquad \begin{matrix} x^3 & y^3 & z^3 \\ x^2 & y^2 & z^2 \\ x & y & z \end{matrix} \qquad \begin{matrix} x^3 & y^3 & z^3 \\ x^2 & y^2 & z^2 \\ x & y & z \end{matrix}$$

Alternatively, it could be set to where Lout usually places it, through the exact centre:

$$\begin{matrix} x^3 & y^3 & z^3 \\ x^2 & y^2 & z^2 \\ x & y & z \end{matrix}$$

Delimiters could reasonably keep the axes that they naturally have (approximately through their centres, but not exactly):

$$\left( \frac{1}{\sqrt{1 - 4x^2}} \right)$$

or they could have their axes moved in the way that Lout usually does, to the point which allows them to evenly cover the thing delimited:

$$\left( \frac{1}{\sqrt{1 - 4x^2}} \right)$$

Altogether then there are four possibilities when these two alternatives interact:

| | Matrix axis uses row axis | Matrix axis passes through centre |
|---|---|---|
| Delimiter keeps its axis | $\left( \frac{1}{\sqrt{1 - 4x^2}} \right)$ | $\left( \frac{1}{\sqrt{1 - 4x^2}} \right)$ |
| Delimiter axis shifted | $\left( \frac{1}{\sqrt{1 - 4x^2}} \right)$ | $\left( \frac{1}{\sqrt{1 - 4x^2}} \right)$ |

To supply these possiblities, the matrix symbol and all its variants (pmatrix etc.) have two options whose values may be yes or no:

```
matrix
    userow { no }
    shiftdelim { yes }
{
    ...
}
```

The userow option determines whether the axis of the matrix will use a row axis; the default is not to, i.e. to centre the axis instead. The shiftdelim option determines whether the axis of the delimiter will be shifted so that the delimiter evenly covers the thing delimited; the default is to do this.

If userow is yes, the next question is which row's axis to use to make the overall axis. If you do nothing, the first (or only) row's axis becomes the overall axis. To select some other row instead, replace the row symbol that precedes the row by axisrow:

```
matrix userow { yes } {
    row      col {  x sup 3  }  col {  y sup 3  }  col {  z sup 3  }
    axisrow  col {  x sup 2  }  col {  y sup 2  }  col {  z sup 2  }
    row      col {  x        }  col {  y        }  col {  z        }
}
```

The result of this is

$$\begin{matrix} x^3 & y^3 & z^3 \\ x^2 & y^2 & z^2 \\ x & y & z \end{matrix}$$

with the axis through the second row as desired.

## 7.4. Spacing

There is a basic rule governing the use of white space characters (space, tab, and newline) in the input to Lout: white space between two objects affects the result; white space between a symbol and its parameter does not. This is explained at length in Section 1.3.

Although this rule is just right most of the time, it is not adequate for equation formatting. Getting the horizontal spacing right in equations is a very fiddly business, involving four different sizes of space (zero, thin, medium, and thick), and different rules for spacing within superscripts and subscripts to those applying outside, according to a leading authority [6]. @Eq therefore takes the spacing decisions upon itself, and consequently chooses to ignore all white space in its input, even between two objects. (The simplest way to restore the effect of white space to part of an equation is to enclose that part in a @Font symbol.)

Every symbol provided by @Eq has a *full name*, which denotes the symbol without any space attached. Many symbols also have a *short name*, which denotes the same symbol with what @Eq considers to be an appropriate amount of space for that symbol attached to it. For example, ≤ has full name lessequal and short name <=:

| | |
|---|---|
| a lessequal b | $a{\leq}b$ |
| a <= b | $a \leq b$ |

@Eq puts a thick space around relation symbols like <=, a medium space around binary operator symbols like +, and a thin space after punctuation symbols (; and ,); except that in places where the symbols appear in a smaller size (superscripts, subscripts, etc.), these spaces are omitted. No other horizontal space is ever inserted.

The short names have been carefully designed to produce good-looking mathematics most of the time. It is best to rely on them in the first instance and only think about spacing when the result is not pleasing. In that case, @Eq's space can be removed by using the full names, and thin, medium and thick space can be added using the following symbols:

| | |
|---|---|
| ‘ | 0.18f (0.018f in subscripts, etc.) |
| “ | 0.24f (0.024f in subscripts, etc.) |
| “‘ | 0.30f (0.030f in subscripts, etc.) |

where 1f is the current font size. These symbols have low precedence. The & symbol from raw Lout is also available; the s unit has value 0 and so is not very useful, but one can write &2m for example for a two em space. The full names are tedious to remember, so @Eq provides a non symbol which removes spaces from its right parameter; thus non <= is equivalent to lessequal. There are also rel, bin, and punct symbols for telling @Eq to add space to the following symbol as though it was a relation symbol, binary operator, or punctuation symbol.

## 7.5. Displaying equations

The result of the @Eq symbol is an object which, according to the golden rule (Section 1.2), may appear anywhere: inside a paragraph, inside a table, and so on. In particular, equations are often displayed using the @CentredDisplay or @IndentedDisplay symbols from Section 2.1:

```
@IndentedDisplay @Eq { ... }
```

Now displayed equations are often numbered, and often aligned with one another on their equals signs. For this there are special display symbols which are the the subject of this section. These symbols can align and number any display at all, but since in practice they seem to be used only with equations, we discuss them here rather than in Section 2.1 where they really belong.

Let's begin by looking at a first example of a numbered display:

$$F_n = F_{n-1} + F_{n-2} \tag{7.5.1}$$

After the display we might have some more text for a while, and then we might want a second display, aligned on its equals sign with the first, and also numbered in sequence with it:

$$F_n - F_{n-1} = F_{n-2} \tag{7.5.2}$$

Notice that the two displays are centred as a block as well as aligned. Altogether there are four ways in which displays vary:

- A display can be raw or not raw (see below);
- It can be a @Display, @LeftDisplay, @IndentedDisplay, @QuotedDisplay, @CentredDisplay, @CenteredDisplay, or @RightDisplay;
- It can be aligned or not aligned;
- It can be numbered or not numbered.

All possible combinations are allowed. The display that has everything is called

```
@RawCentredAlignedNumberedDisplay
```

By leaving out some or all of Raw, Aligned, and Numbered, and by changing or leaving out Centred, we get all these combinations. Here then is how the two displays given earlier were made:

```
... a first example of a numbered display:
@BeginAlignedDisplays
@CentredAlignedNumberedDisplay
  @Tag { fibeq }
@Eq { F sub n ^= F sub { n-1 } + F sub { n-2 } }
After the display we might ... numbered in sequence with it:
@CentredAlignedNumberedDisplay @Eq { F sub n - F sub { n-1 } ^= F sub { n-2 } }
@EndAlignedDisplays
Notice that the two displays are centred ...
```

All numbered displays have an optional @Tag option which is used for cross referencing (see Section 2.8). Alignment and numbering work quite independently; they don't have to start or end together, and there can be non-aligned and non-numbered displays among the others.

When aligned displays are used, it is necessary to indicate where the aligned group begins and ends, by placing @BeginAlignedDisplays just before the first, and @EndAlignedDisplays just after the last. The alignment points are indicated by preceding them by the symbol ^, so you aren't restricted to aligning at equals signs. @BeginAlignedDisplays and @EndAlignedDisplays cannot span across several sections or subsections: the equations aligned by them must lie within a single large-scale structure symbol.

In our example of aligned and numbered displays, the two displays were separated by some ordinary text. Very often, though, aligned displays follow directly after each other. This is a problem, because if you have one display directly following another there will be too much vertical space between them. This problem was mentioned in Section 2.1, and the recommended solution was to use a list. However, there are no aligned or numbered (in this sense) lists.

To solve this problem, each display symbol has a 'raw' version, which means that no space is inserted above or below the display. Instead, you must insert it yourself using paragraph symbols:

```
preceding text
@DP
@RawAlignedDisplay @Eq { ... }
@DP
@RawAlignedNumberedDisplay @Eq { ... }
@DP
following text
```

You get the right spacing by placing @DP symbols before, between, and after each display; and you get to use the specialized displays that you need. Raw and non-raw displays may be numbered and aligned together.

Numbered displays are numbered automatically. Depending on where in the document they appear, the number might include a chapter number or section number, etc. This is controlled by options in the setup file; for example, setting @ChapterNumInDisplays to Yes ensures that numbered displays will be numbered afresh at the beginning of each chapter, and that the number will include a chapter number. There is also a @DisplayNumStyle option which controls the style of displays; the default value, (num), encloses the number in parentheses as is conventional when numbering equations.

Every display symbol has an abbreviated form consisting of @ followed by its capital letters only. For example, @BeginAlignedDisplays may be abbreviated to @BAD, and the display that has everything to @RCAND. Owing to an unfortunate clash between the initial letters of 'raw' and 'right', @RightDisplay and the other right displays have no abbreviations.

### 7.6. Defining new equation formatting symbols

Whenever you type particular equations or parts of equations repeatedly, you can save time by using definitions. Definitions are the subject of Section 2.12, so here we will just give a few examples of their use in equation formatting.

Suppose for example that $p_i \log_2 p_i$ occurs frequently in your document. Then

```
def epi { p sub i ` log sub 2 ` p sub i }
```

makes the symbol epi stand for the object between the braces:

| | |
|---|---|
| big sum from i=1 to n ` epi | $\sum_{i=1}^{n} p_i \log_2 p_i$ |

Parameters are very useful when parts of the symbol vary:

```
def ep
  right x
{ p sub x ` log sub 2 ` p sub x
}
```

The parameter x will be replaced by the object just to the right of ep:

```
big sum from i=1 to k ` ep i +
big sum from j=k+1 to n ep j
```

$$\sum_{i=1}^{k} p_i \log_2 p_i + \sum_{j=k+1}^{n} p_j \log_2 p_j$$

The precedence of the symbols you define will be 100 by default.

To make the symbols of @Eq available within such definitions, each must be preceded by import @Eq. As explained in Section 2.12, the definitions go into a file called mydefs, which might then look like this:

```
import @Eq
def epi { p sub i ` log sub 2 ` p sub i }

import @Eq
def ep right x { p sub x ` log sub 2 ` p sub x }
```

Use of epi and ep outside @Eq will cause an error.

## 7.7. Summary

This section is a complete list of the symbols provided by @Eq. We divide them into auxiliary, parameterized, short names (further divided into relations, binary operators, and punctuation), and full names. The auxiliary symbols are:

| | |
|---|---|
| ` | Thin space |
| `` | Medium space |
| ``` | Thick space |
| bin x | Treat x as a binary operator |
| rel x | Treat x as a relation |
| punct x | Treat x as a punctuation symbol |
| non x | Remove spaces normally put into x |
| vctr x | Centre x vertically |
| big x | Make x larger |

Here are all the parameterized symbols, shown in groups of equal precedence, with the precedence itself at right:

matrix pmatrix bmatrix brmatrix fmatrix cmatrix amatrix not (100)
dot dotdot hat tilde vec dyad overbar underbar (62)
sup sub tsub supp (60) on ton (61)
from to widefrom wideto (58)
sqrt root (56)
over frac (54)
col lcol ccol rcol mcol (52)
row axisrow (50)

See Section 7.2 for examples of matrices. Here are some examples of the other symbols:

x dot $\qquad \dot{x}$

| | |
|---|---|
| x dotdot | $\ddot{x}$ |
| x hat | $\hat{x}$ |
| x tilde | $\tilde{x}$ |
| x vec | $\vec{x}$ |
| x dyad | $\overleftrightarrow{x}$ |
| x+y overbar | $\overline{x + y}$ |
| x+y underbar | $\underline{x + y}$ |

These marks are centred over the preceding object, except the last two which are extended to the width of the object.

| | |
|---|---|
| a sup b | $a^b$ |
| a sub b | $a_b$ |
| W tsub b | $W_b$ |
| a supp b on c | $a_c^b$ |
| W supp b ton c | $W_c^b$ |

Note that supp and on (or ton) must be used together as shown; tsub and ton are exactly like sub and on except that the subscript is tucked in.

| | |
|---|---|
| big sum from i | $\sum\limits_i$ |
| big prod to j | $\prod\limits^j$ |
| {a, ... , z} widefrom {90d @Rotate blbrace} | $\underbrace{a, \ldots, z}$ |
| {a, ... , z} wideto minus | $\overline{a, \ldots, z}$ |

widefrom and wideto are like from and to except that they horizontally scale the right parameter to the width of the left.

| | |
|---|---|
| sqrt {x over y} | $\sqrt{\dfrac{x}{y}}$ |
| 3 root {x over y} | $\sqrt[3]{\dfrac{x}{y}}$ |

The left parameter of root may be any object. Here are four ways to denote division:

| | |
|---|---|
| 2 over 3 | $\frac{2}{3}$ |
| 2 frac 3 | $^{2}/_{3}$ |
| 2 div 3 | $2 \div 3$ |
| 2 slash 3 | $2/3$ |

The div symbol is a binary operator (see below), and slash is the full name for the / character from the Adobe Symbol font. You can't use / itself, because it is one of Lout's special symbols.

The following short names define relations (that is, they have a thick space on each side):

| | | | | | |
|---|---|---|---|---|---|
| < | < | > | > | = | = |
| <= | ≤ | prec | ≺ | preceq | ⪯ |
| << | << | subset | ⊂ | subseteq | ⊆ |
| sqsubseteq | ⊑ | in | ∈ | vdash | ⊢ |
| smile | ⌣ | frown | ⌢ | >= | ≥ |
| succ | ≻ | succeq | ⪰ | >> | >> |
| supset | ⊃ | supseteq | ⊇ | sqsupseteq | ⊒ |
| ni | ∋ | dashv | ⊣ | mid | $\mid$ |
| parallel | ‖ | == | ≡ | ~ | ∼ |
| -~ | ≃ | asymp | ≍ | ~~ | ≈ |
| =~ | ≅ | bowtie | ⋈ | propto | ∝ |
| models | ⊨ | doteq | ≐ | perp | ⊥ |
| notsub | ⊄ | notin | ∉ | != | ≠ |
| <-> | ↔ | <-- | ← | --> | → |
| up | ↑ | down | ↓ | <=> | ⇔ |
| <== | ⇐ | ==> | ⇒ | dblup | ⇑ |
| dbldown | ⇓ | : | : | :: | : : |
| := | := | | | | |

These can be negated by enclosing them in braces and preceding them with not, as in not ==, for example, which yields ≢ . The following short names define binary operators (medium space on each side):

| | | | | | |
|---|---|---|---|---|---|
| + | + | - | − | +- | ± |
| -+ | ∓ | setminus | \ | cdot | · |
| times | × | * | ∗ | circ | ○ |
| div | ÷ | cap | ∩ | cup | ∪ |
| uplus | ⊎ | sqcap | ⊓ | sqcup | ⊔ |
| triangleleft | ◁ | triangleright | ▷ | wr | ≀ |
| bigcirc | ◯ | bigtriangleup | △ | bigtriangledown | ▽ |
| vee | ∨ | wedge | ∧ | oplus | ⊕ |
| ominus | ⊖ | otimes | ⊗ | oslash | ⊘ |
| odot | ⊙ | dagger | † | daggerdbl | ‡ |
| amalg | ∐ | | | | |

The following names define punctuation symbols (thin space on the right-hand side):

| | | | | | |
|---|---|---|---|---|---|
| ; | ; | , | , | col | : |

The following symbols are used in ways typified by the large sum and product symbols. In display equations they should be preceded by the big symbol:

| | | | | | |
|---|---|---|---|---|---|
| sum | $\sum$ | prod | $\prod$ | coprod | $\coprod$ |
| int | $\int$ | oint | $\oint$ | bcap | $\bigcap$ |
| bcup | $\bigcup$ | bvee | $\bigvee$ | bwedge | $\bigwedge$ |
| bodot | $\bigodot$ | botimes | $\bigotimes$ | boplus | $\bigoplus$ |
| buplus | $\biguplus$ | | | | |

The following symbols are defined so that they will appear in Roman, as is conventional for them in equations:

| | | | | | |
|---|---|---|---|---|---|
| arccos | arccos | arcsin | arcsin | arctan | arctan |
| arg | arg | cos | cos | cosh | cosh |
| cot | cot | coth | coth | csc | csc |
| deg | deg | det | det | dim | dim |
| exp | exp | gcd | gcd | hom | hom |
| inf | inf | ker | ker | lg | lg |
| lim | lim | liminf | lim inf | limsup | lim sup |
| ln | ln | log | log | max | max |
| min | min | Pr | Pr | sec | sec |
| sin | sin | sinh | sinh | supr | sup |
| tan | tan | tanh | tanh | mod | mod |

The following symbols are also defined to ensure that they will appear in Roman:

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 2 | 2 |
| 3 | 3 | 4 | 4 | 5 | 5 |
| 6 | 6 | 7 | 7 | 8 | 8 |
| 9 | 9 | ! | ! | ? | ? |
| % | % | ( | ( | ) | ) |
| [ | [ | ] | ] | | |

The following symbols make good atleft and atright parameters of the matrix symbol:

| | | | | | |
|---|---|---|---|---|---|
| lpar | ( | blpar | $\Bigg($ | rpar | ) |
| brpar | $\Bigg)$ | lbrack | [ | blbrack | $\Bigg[$ |
| rbrack | ] | brbrack | $\Bigg]$ | lbrace | { |

| | | | | | |
|---|---|---|---|---|---|
| blbrace | $\Big\{$ | rbrace | $\}$ | brbrace | $\Big\}$ |
| lfloor | $\lfloor$ | blfloor | $\Big\lfloor$ | rfloor | $\rfloor$ |
| brfloor | $\Big\rfloor$ | lceil | $\lceil$ | blceil | $\Big\lceil$ |
| rceil | $\rceil$ | brceil | $\Big\rceil$ | langle | $\langle$ |
| blangle | $\Big\langle$ | rangle | $\rangle$ | brangle | $\Big\rangle$ |

Here are some miscellaneous symbols:

| | | | | | |
|---|---|---|---|---|---|
| hbar | $\hbar$ | Re | $\Re$ | Im | $\Im$ |
| partial | $\partial$ | infty | $\infty$ | prime | $\prime$ |
| nabla | $\nabla$ | surd | $\surd$ | top | $\top$ |
| bot | $\bot$ | dbar | ‖ | triangle | $\triangle$ |
| backslash | $\backslash$ | forall | $\forall$ | exists | $\exists$ |
| neg | $\neg$ | circle | $\bigcirc$ | square | $\square$ |
| ldots | $\ldots$ | cdots | $\cdots$ | vdots | $\vdots$ |
| ddots | $\ddots$ | del | $\nabla$ | grad | $\nabla$ |
| ... | $\ldots$ | ,..., | $, \ldots ,$ | half | $^1\!/_2$ |
| third | $^1\!/_3$ | | | empty | $\emptyset$ |

Finally, here is the long list of full names from the Adobe Symbol font; these are the same characters as you get with the @Sym symbol of Section 1.5, but within equations you don't need to type @Sym:

| | | | | | |
|---|---|---|---|---|---|
| space | | exclam | ! | universal | $\forall$ |
| numbersign | # | existential | $\exists$ | percent | % |
| ampersand | & | suchthat | $\ni$ | parenleft | ( |
| parenright | ) | asteriskmath | $*$ | plus | $+$ |
| comma | , | minus | $-$ | period | . |
| slash | / | zero | 0 | one | 1 |
| two | 2 | three | 3 | four | 4 |
| five | 5 | six | 6 | seven | 7 |
| eight | 8 | nine | 9 | colon | : |
| semicolon | ; | less | $<$ | equal | $=$ |
| greater | $>$ | question | ? | congruent | $\cong$ |
| Alpha | A | Beta | B | Chi | X |
| Delta | $\Delta$ | Epsilon | E | Phi | $\Phi$ |
| Gamma | $\Gamma$ | Eta | H | Iota | I |
| thetaone | $\vartheta$ | Kappa | K | Lambda | $\Lambda$ |
| Mu | M | Nu | N | Omicron | O |

| | | | | | |
|---|---|---|---|---|---|
| Pi | Π | Theta | Θ | Rho | Ρ |
| Sigma | Σ | Tau | Τ | Upsilon | Υ |
| sigmaone | ς | Omega | Ω | Xi | Ξ |
| Psi | Ψ | Zeta | Ζ | bracketleft | [ |
| therefore | ∴ | bracketright | ] | perpendicular | ⊥ |
| underscore | _ | radicalex | ‾ | alpha | α |
| beta | β | chi | χ | delta | δ |
| epsilon | ε | phi | φ | gamma | γ |
| eta | η | iota | ι | phione | ϕ |
| kappa | κ | lambda | λ | mu | μ |
| nu | ν | omicron | ο | pi | π |
| theta | θ | rho | ρ | sigma | σ |
| tau | τ | upsilon | υ | omegaone | ϖ |
| omega | ω | xi | ξ | psi | ψ |
| zeta | ζ | braceleft | { | bar | \| |
| braceright | } | similar | ∼ | Upsilonone | ϒ |
| minute | ′ | lessequal | ≤ | fraction | ⁄ |
| infinity | ∞ | florin | ƒ | club | ♣ |
| diamond | ♦ | heart | ♥ | spade | ♠ |
| arrowboth | ↔ | arrowleft | ← | arrowup | ↑ |
| arrowright | → | arrowdown | ↓ | degree | ° |
| plusminus | ± | second | ″ | greaterequal | ≥ |
| multiply | × | proportional | ∝ | partialdiff | ∂ |
| bullet | • | divide | ÷ | notequal | ≠ |
| equivalence | ≡ | approxequal | ≈ | ellipsis | … |
| arrowvertex | ⏐ | arrowhorizex | ⎯ | carriagereturn | ↵ |
| aleph | ℵ | Ifraktur | ℑ | Rfraktur | ℜ |
| weierstrass | ℘ | circlemultiply | ⊗ | circleplus | ⊕ |
| emptyset | ∅ | intersection | ∩ | union | ∪ |
| propersuperset | ⊃ | reflexsuperset | ⊇ | notsubset | ⊄ |
| propersubset | ⊂ | reflexsubset | ⊆ | element | ∈ |
| notelement | ∉ | angle | ∠ | gradient | ∇ |
| registerserif | ® | copyrightserif | © | trademarkserif | ™ |
| product | ∏ | radical | √ | dotmath | ⋅ |
| logicalnot | ¬ | logicaland | ∧ | logicalor | ∨ |
| arrowdblboth | ⇔ | arrowdblleft | ⇐ | arrowdblup | ⇑ |
| arrowdblright | ⇒ | arrowdbldown | ⇓ | lozenge | ◊ |
| angleleft | 〈 | registersans | ® | copyrightsans | © |
| trademarksans | ™ | summation | ∑ | parenlefttp | ⎛ |
| parenleftex | ⎜ | parenleftbt | ⎝ | bracketlefttp | ⎡ |
| bracketleftex | ⎢ | bracketleftbt | ⎣ | bracelefttp | ⎧ |
| braceleftmid | ⎨ | braceleftbt | ⎩ | braceex | ⎪ |
| angleright | 〉 | integral | ∫ | integraltp | ⌠ |
| integralex | ⎮ | integralbt | ⌡ | parenrighttp | ⎞ |
| parenrightex | ⎟ | parenrightbt | ⎠ | bracketrighttp | ⎤ |
| bracketrightex | ⎥ | bracketrightbt | ⎦ | bracerighttp | ⎫ |

| | | | |
|---|---|---|---|
| bracerightmid | ⎬ | bracerightbt | ⎭ |

The names given are the same as Adobe's, as used by the @Sym symbol, except in a few places where the Adobe name contains a digit, which is not possible for a symbol name in Lout.

## 7.8. An alternative version that uses TEX's fonts

There is an alternative version of the @Eq symbol that uses fonts taken from the TEX document formatting system. These fonts are said to produce better-looking mathematics than the Adobe Systems Symbol font used by the standard @Eq symbol.

The fonts were converted from TEX form to PostScript form by Basil K. Malyshev, who has attached a license to them permitting non-commercial use only. This is a much more stringent license than the one attached to Lout itself. For this reason, the files needed to use these TEX fonts are distributed separately from the rest of Lout, although you can get them from the same place (see the preface of this guide).

Once these files are installed, you change from the standard @Eq symbol to the TEX version by changing the initial @SysInclude { eq } to @SysInclude { teq }. Do absolutely nothing else.

Unfortunately, the TEX fonts are not usually resident on PostScript printing devices, which means that Lout is obliged to include them in its PostScript output file. You don't have to do anything to make this happen, but the cost is fairly large: changing to @SysInclude { teq } increases the size of the PostScript output file by 252 kilobytes.

It is possible to gain access to characters in the TEX fonts that are not accessible directly from @Eq, mainly script capitals and bold-italic Greek letters. For example, you can use {cmsy Base} @Font @Char "A" to get a script A, and {cmmi Bold} @Font @Char "pi" to get a bold-italic $\pi$. For the full story, consult file teq in the Lout system include directory for the names of these fonts, and then look in Lout's font directory for their font metrics files, which show the names and encodings of all the characters.

# Chapter 8. Basic Graphics

This chapter introduces some basic graphics symbols for colour, rotation, scaling, and included illustrations. These are all from the standard DocumentLayout package, so no @SysInclude line is needed to get them.

## 8.1. Colour

Colour is obtained in much the same way that fonts and language changes are, using the @Colour (or equivalently @Color) symbol:

```
grey @Colour { Hello, world }
```

produces

Hello, world

The @Colour symbol will accept any of the following colours:

| | | |
|---|---|---|
| darkred | red | lightred |
| darkgreen | green | lightgreen |
| darkblue | blue | lightblue |
| darkcyan | cyan | lightcyan |
| darkmagenta | magenta | lightmagenta |
| darkyellow | yellow | lightyellow |
| darkgrey | grey | lightgrey |
| darkgray | gray | lightgray |
| black | white | |

Monochrome output devices will render them as shades of grey. Colouring something white makes it invisible, which is sometimes useful.

In addition to the list of colours given above, there is a special colour called nochange which produces the colour you already happen to be using.

Whether or not the colours produced by @Colour actually correspond with the names depends on the output device; the same nominal colour can look quite different on screen and on paper. The standard Lout @SetColour symbol can provide many more colours [5], although they must be specified using numbers rather than names.

## 8.2. Boxes

The @Box symbol causes the following object to be enclosed in a box:

```
@QuotedDisplay @Box {
@CentredDisplay @Heading Cheating
The Department uses assignments ... of that student alone.
}
```

The result of this is

> **Cheating**
>
> The Department uses assignments both as a teaching device and as a major component of its assessment of each student. It therefore requires that all programs, exercises etc. handed in bearing an individual student's name be the work of that student alone.

showing that a box may enclose an arbitrarily complicated object.

The @Box symbol has a margin option which determines the margin between the box and what it encloses. For example,

```
@Box
   margin { 0.1c }
{}
```

requests a box with a 0.1 centimetre margin enclosing an empty object, so the result is a square whose width and height are 0.2 centimetres:

□

If the margin option is omitted, it is assigned the default value 0.3f, which means 0.3 times the current font size. It is very useful to tie the margin to the font size in this way, because large headings (in overhead transparencies, say) need large margins.

There is a linewidth option which determines the width (thickness) of the line drawn around the boundary of the box:

```
@Box
   linewidth { 0.1c }
{ Hello world }
```

produces

Hello world

Lout does not take the line width into account when working out how large everything is: as far as Lout is concerned, the line always has width zero. If you draw really thick lines you might need a larger margin and more space near the box. The default value of linewidth is empty, which means to use whatever width the PostScript interpreter in your output device thinks is a good default value.

There is also a paint option which paints a background of the nominated colour:

@Box paint { grey } WARNING!

has result

WARNING!

This is quite different from grey @Colour @Box WARNING!, which produces

WARNING!

The paint option may be given any colour from the list in Section 8.1; its default value is none, which is a special value (not a colour) which means no painting. White paint comes into its own inside painted boxes:

@Box paint { nochange } white @Colour { Hello world }

produces a box painted in whatever colour we happen to be using at the moment, with white text inside:

Hello world

This works because the box is painted before the object it encloses is drawn on the page.

There are @CurveBox and @ShadowBox symbols that produce other kinds of boxes:

A curve box    A shadow box

These also have margin and paint options, and @ShadowBox has a shadow option which determines the thickness of the shadow (its default value is 0.2f).

Boxes are quite at home inside paragraphs, as a box, a curve box, and a shadow box show. Simply proceed as usual:

... paragraphs, as @Box { a box }, @CurveBox { a curve box }, ...

Boxes within paragraphs are never broken across two lines.

## 8.3. Rotation

The @Rotate symbol rotates the following object by any positive or negative angle, measured in degrees:

45d @Rotate @Box WARNING!

has result

WARNING!

As usual, the object to be rotated may be arbitrary. However, it is difficult for Lout to choose appropriate column widths for paragraphs inside rotated objects, so if a rotated object contains paragraphs that should be broken it is best to define the object's width explicitly, using the @Wide symbol:

```
-90d @Rotate 4.5c @Wide {
Papal initiatives and influence from the crowning of
Charlemagne to the First Crusade
}
```

The result here is

Papal initiatives and influ-
ence from the crowning of
Charlemagne to the First
Crusade

The @Wide symbol fixes the width of the following object, in this example to the length 4.5 centimetres, which is all Lout needs to decide the column widths of any paragraphs within it.

## 8.4. Scaling

The @Scale symbol performs a geometrical scaling of the following object:

```
0.5 @Scale @Box WARNING!
```

produces

WARNING!

A scale factor of 0.5 means half the original size, 2.0 means double size, and so on. No unit of measurement appears in the scale factor, because it makes no sense to have one. As usual, the object to be scaled may be arbitrary.

It is also possible to supply two scale factors, in which case the first is applied horizontally and the second vertically:

```
{0.5 2.0} @Scale @Box WARNING!
```

has result

WARNING!

Practical uses for this kind of scaling are rare.

If an empty object is given instead of a scale factor, like this:

```
{} @Scale @Box WARNING!
```

the @Scale symbol will choose the largest scale factor that does not overrun the available horizontal space. It is often possible to omit the {}, since Lout inserts an empty object automatically whenever an object is clearly missing (see Section 1.2). For example,

```
@QuotedDisplay @Scale @Box WARNING!
```

produces

WARNING!

@QuotedDisplay and @LeftDisplay go well with this form of @Scale. However, some care is needed because Lout foolishly takes no account of the available *vertical* space when choosing the scale factor. The chosen scale factor could enlarge the vertical size so much that the object no longer fits on the page, with disastrous results.

By using a @Wide symbol to restrict the available horizontal space, this form of scaling can also be used to scale to a nominated width. For example,

```
5c @Wide @Scale @Box WARNING!
```

produces

WARNING!

which is 5 centimetres wide.

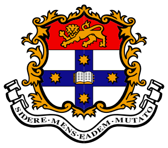
### 8.5. Including an illustration

The @IncludeGraphic symbol incorporates into a Lout document an illustration (technically, an encapsulated PostScript or EPS file) produced by other means. The opposite process, using Lout to produce an illustration for inclusion in some other document, is the subject of Section 3.5.

For example, suppose the encapsulated PostScript file su_crest.eps contains the University of Sydney crest. Then

```
@IncludeGraphic su_crest.eps
```

produces



In general, the result produced by @IncludeGraphic is an object that may be scaled, rotated, made into a display or placed within a paragraph, just like any other object. Accolades for this remarkable flexibility should go to the PostScript page description language, whose extraordinary power makes the provision of this feature in Lout almost trivial.

The @IncludeGraphic command understands that files ending with any of the suffixes .gz, -gz, .z, -z, _z, and .Z are compressed files, and it will uncompress such files using the gunzip command before including them. The uncompressed version is stored in a file called lout.eps in the current directory, and removed after being copied into the output file.

If you place an included illustration in a line of text, or anywhere where you care about its alignment with things on either side of it, it will be positioned with its centre at the same height as the centre of the letter x. If this is not where you want it, use the @VShift symbol:

```
... +0.5f @VShift @IncludeGraphic ...
```

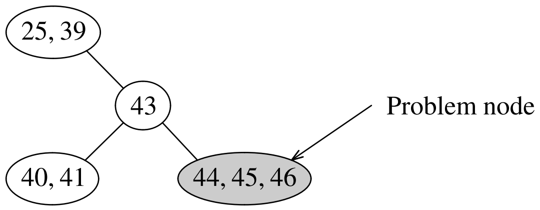
prints the illustration half of the current font size higher on the page than would otherwise have been the case, and

```
... -0.5f @VShift @IncludeGraphic ...
```

prints it half the current font size lower. Any length (Section 1.2) is allowed, and the object following @VShift may in fact be arbitrary as usual.

# Chapter 9. Advanced Graphics

This chapter describes the use of the @Fig symbol, which draws, colours, and positions arbitrary shapes made from straight lines, circular and elliptical arcs, and Bezier curves:




The graphical objects may be rotated and scaled; they may enclose, and be enclosed by arbitrary Lout objects (text, equations, tables, other graphical objects, etc.) without restriction. A convenient algebra of points and a method of labelling points assist positioning.

## 9.1. Introduction

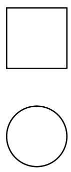
The Lout definitions for the @Fig symbol are kept in a file called fig, which you must include at the start of your document if you want advanced graphics, like this:

```
@SysInclude { fig }
@SysInclude { doc }
@Doc @Text @Begin
...
@End @Text
```

Files of definitions, like fig, must be included before the setup file. Alternatively, if you are using your own setup file, you may place the include commands within it; the place to put them is clearly marked in all the standard setup files.

Throughout this chapter we will show the Lout text on the left and the corresponding result on the right, like this:

```
@Fig {
  @Square
  //0.5c
  @Circle
}
```

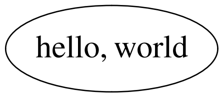


Subsequent examples will omit the enclosing @Fig.

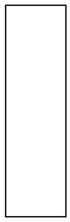
## 9.2. Basic shapes

@Fig has a repertoire of basic shapes, whose size varies depending on what they enclose:

```
@Ellipse { hello, world }
```



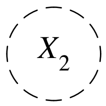

```
@Box { 2c @High }
```



There are eight such shapes: @Box, @CurveBox, @ShadowBox, @Square, @Diamond, @Polygon, @Ellipse, and @Circle; the result of each is the following object, enclosed in a small margin, with the shape around it.

There are options for changing the appearance of these shapes. The boundary line's style may be solid, dashed, cdashed, dotted, or noline (that is, no line is drawn), and the length of the dashes may be changed:

```
@Circle
  linestyle { cdashed }
  dashlength { 0.2 cm }
@Eq { X sub 2 }
```




If the line style is not mentioned, it becomes solid by default. The dashed option makes all dashes the same length; cdashed halves the length of the first and last dash on each segment, which looks better in some cases. The distance between dashes or dots will be at most dashlength, reduced to make the dashes or dots fit evenly.

Shapes may be painted any of the colours listed in Section 8.1, using the paint option:

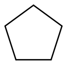
```
@Box
  margin { 0c }
  paint { grey }
@Diamond
  linestyle { noline }
  paint { white }
{ hello, world }
```


The default value of paint is nopaint, a special value (not a colour) meaning don't use any paint. In this example, the right parameter of @Box is a diamond containing hello, world. There is no limit to the amount of this sort of nesting; the following object may be any object.

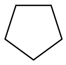
When painting it is important to know what order things are done in, because anything put down earlier will disappear under the paint. This is why nopaint and white are different. Painting is done first, then boundaries, and finally the right parameter.

The @Polygon shape has a sides option for specifying the number of sides, and an angle option for specifying what angle anticlockwise from vertically beneath the centre the first corner will appear at:

```
@Polygon
  sides { 5 }
{}
```



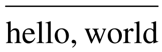
```
@Polygon
  sides { 5 }
  angle { 0 dg }
{}
```



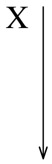
The defaults are 3 sides and the angle that gives the polygon a horizontal base (i.e. 180 degrees divided by the number of sides). Thus the two cases with symmetry about a vertical axis are obtained by the default angle and 0 dg respectively, which is convenient.

Although lines and arrows do not enclose things in the way that boxes and circles do, @Fig treats them as it does the other shapes. The line or arrow is drawn along the mark of the right parameter, either horizontally or vertically:

```
@HLine { //0.2c hello, world }
```




```
@VArrow { 2c @High X ^|0.2c }
```

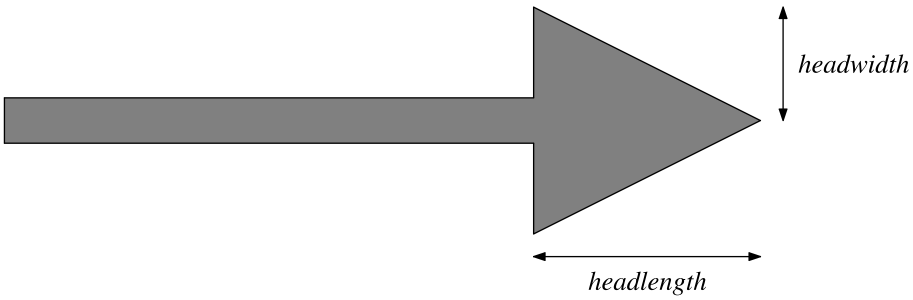



The usual line style options are available; the default margin is 0c. Arrows can be forward (the default), back, both, or noarrow (which just draws a line); the style of the arrowhead can be open (the default), halfopen, or closed:

```
@HArrow
  arrow { both }
  headstyle { closed }
{ 3c @Wide }
```



It is also possible to change the shape of the arrowhead, using the headwidth and headlength options:

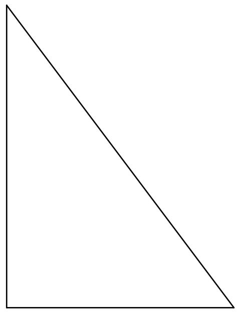



Their default values are 0.05 cm and 0.15 cm respectively.

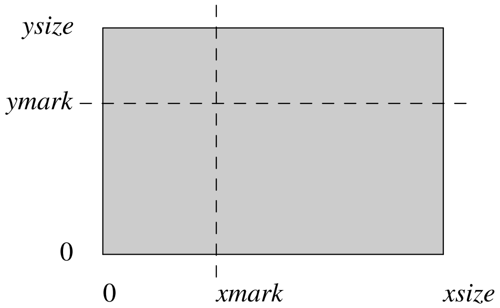
## 9.3. Creating new shapes

If the shape you need is not provided by @Fig, you can create it using the @Figure symbol (within @Fig, the meaning of @Figure is not that of Section 2.6). @Figure takes all the options we have already seen, plus another one called shape:

```
@Figure
  shape {
    0 0  xsize 0
    0 ysize  0 0
  }
{ 4c @High 3c @Wide }
```



The expression 4c @High 3c @Wide {} (recall that an empty object is inserted automatically when required) produces a four centimetre high, three centimetre wide blank space. The pairs of numbers define points in a coordinate system whose origin is the lower left corner of the right parameter; the upper right corner is xsize ysize, and the point where the right parameter's marks cross is xmark ymark:[1]




A sequence of points defines a shape, like the triangle above. Arrowheads are drawn pointing forwards from the last segment and backwards from the first, as requested; the margin option has default value 0c.

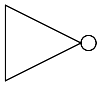
Normally, the points are connected by straight lines to form the shape, which is then painted and drawn in the usual way, depending on the other options.

If two points in the shape are separated by [], no line will be drawn between them. This permits a shape to consist of two or more disconnected parts.

If two points in the shape are separated by [$x$ $y$], where $x$ and $y$ are numbers, the two points will be joined by an anticlockwise arc whose centre is the point $(x, y)$. This arc will be circular if possible, otherwise it will be part of an ellipse whose axes are oriented horizontally and vertically. The notation [$x$ $y$ clockwise] makes the arc go clockwise. For example,

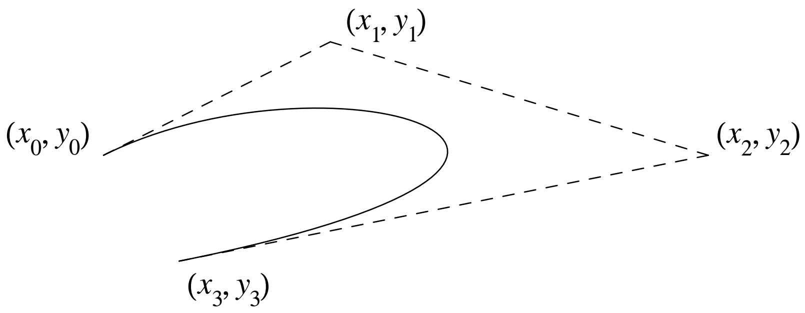
[1]Marks are used for aligning an object with nearby objects; they are explained in the expert's guide [5]. Expert users will note that this arrangement is identical with that for the @Graphic symbol of raw Lout.

```
@Figure
  shape {
    0 -0.5 cm
    0 0.5 cm
    1 cm 0  0 -0.5 cm []
    1 cm 0 [ 1.1 cm 0 ]
    1 cm 0
  }
{}
```



We have recklessly disregarded the size of the following object when drawing this shape, a dangerous thing to do since Lout thinks that the figure is the same size as the following object.

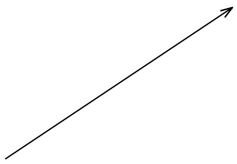
Finally, two points may be separated by [$x_1$ $y_1$ $x_2$ $y_2$], which requests that a Bezier curve be drawn between them with control points $(x_1, y_1)$ and $(x_2, y_2)$:




The curve is attracted toward the control points, without reaching them; it is tangent to the straight line from the start point to the first control point, and from the second control point to the finishing point, and it lies wholly inside the quadrilateral formed by the four points. Owing to the author's laziness, dashes and dots do not fit as neatly onto Bezier curves as they do onto lines and arcs.

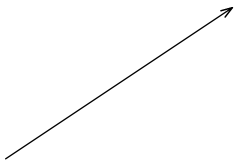
Lines, arrows and arcs at any angle can be produced using @Figure:

```
@Figure
  shape { 0 0  xsize ysize }
  arrow { forward }
{ 2c @High 3c @Wide }
```



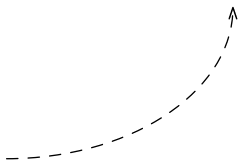
However, for convenience there are @Line and @Arrow symbols which have two options, from and to, for specifying the endpoints:

```
@Arrow
  from { 0 0 }
  to { xsize ysize }
{ 2c @High 3c @Wide }
```



There is also an @Arc symbol, which draws a circular or elliptical arc from one point to another about a given centre, with the usual options:

```
@Arc
  from { 0 0 }
  to { xsize ysize }
  ctr { 0 ysize }
  direction { anticlockwise }
  arrow { forward }
  linestyle { dashed }
{ 2c @High 3c @Wide }
```



The arc goes either clockwise (the default) or anticlockwise about the centre, depending on the direction option. Any arrowhead will point in a direction tangent to the arc.

### 9.4. Lengths, angles, and points

We already know that two lengths placed side by side define a point. This is only the simplest of a number of such geometrical combinations.

The symbol @Distance returns the length of the line joining two points:

{0 0} @Distance {3 cm  4 cm}

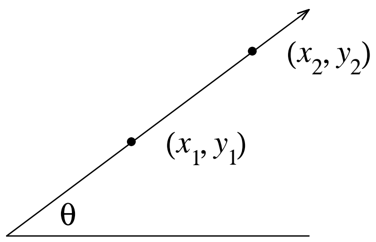
is equivalent to the length 5 cm. The result of @Distance is never negative. Notice that braces must enclose the two points. The symbol @XDistance returns the distance in the $x$ direction from one point to another:

$\{x_1\ y_1\}$ @XDistance $\{x_2\ y_2\}$

has for its result the length $x_2 - x_1$, and so may be negative. There is an analogous @YDistance symbol.

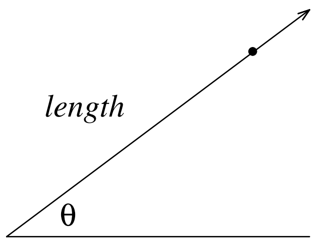
The symbol @Angle returns the angle $\theta$ from one point to another:

$\{x_1\ y_1\}$ @Angle $\{x_2\ y_2\}$




The result will be 0 if the two points are equal. The symbol << returns the point at a given distance and angle from $(0, 0)$:

{*length* << $\theta$}



For example, { 5 cm << 45 dg } is the point 5 cm from (0, 0) at 45 degrees.

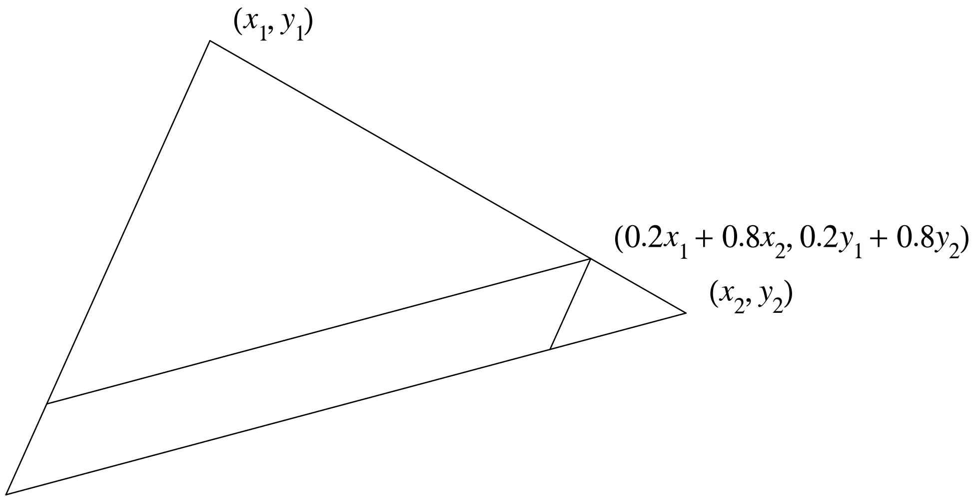
Points may be added, subtracted, and multiplied by a number:

| | | |
|---|---|---|
| $\{x_1\ y_1\}$ ++ $\{x_2\ y_2\}$ | is | $(x_1 + x_2, y_1 + y_2)$ |
| $\{x_1\ y_1\}$ -- $\{x_2\ y_2\}$ | is | $(x_1 - x_2, y_1 - y_2)$ |
| $\{x_1\ y_1\}$ ** $k$ | is | $(x_1k, y_1k)$ |

For example,

$\{x_1\ y_1\}$ ** 0.2 ++ $\{x_2\ y_2\}$ ** 0.8

is the point eight tenths of the way from $(x_1, y_1)$ to $(x_2, y_2)$ on the line joining them:




When using **, the point must be on the left and the number on the right. It would be more convenient to name these symbols +, -, and *, but these names are often taken by equation formatters, and - appears in lengths, so we don't. There are @Max and @Min symbols:

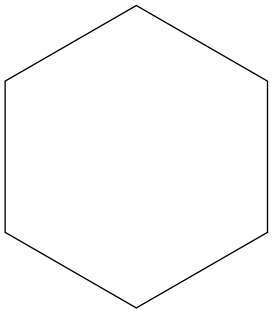
| | | |
|---|---|---|
| $\{x_1\ y_1\}$ @Max $\{x_2\ y_2\}$ | is | $(\max(x_1, x_2), \max(y_1, y_2))$ |
| $\{x_1\ y_1\}$ @Min $\{x_2\ y_2\}$ | is | $(\min(x_1, x_2), \min(y_1, y_2))$ |

Note carefully that these apply to points, not to numbers.

The result of adding two points together depends on where the origin is at the time, as well as on the points themselves. This can lead to unexpected results, as the author has found to his cost more than once. Within the shape option of @Figure, the origin is the lower left corner of the result of the @Figure. In cases like the example on page 147, where points are added outside of any @Figure symbol, the origin is usually at the bottom left corner of the figure as a whole. A label always denotes a particular point on the printed page, regardless of where the origin happens to be.

The symbol @Prev within the shape option of @Figure denotes the previous point of the shape, ignoring points within []. This makes it easy to specify each point relative to the previous one:

```
shape {
  0 0
  { 2 cm << 30 }
  @Prev ++ { 2 cm << 90 }
  @Prev ++ { 2 cm << 150 }
  @Prev ++ { 2 cm << 210 }
  @Prev ++ { 2 cm << 270 }
  0 0
}
```

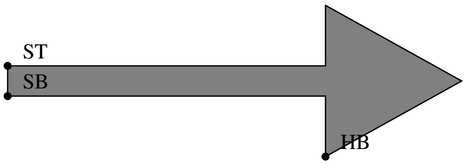


There is a @Label symbol, available only within the shape option of @Figure, for attaching a label to a point in the shape, like this:

```
{xsize ysize} ** 0.5 @Label CTR
```

The point may then be referred to more concisely by its label, CTR. For example, the large arrow appearing in Section 9.3 was built like this:

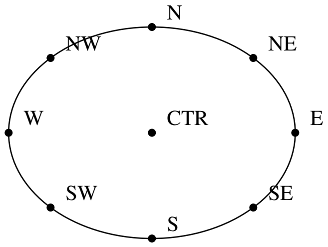
```
@Figure
  shape {
    {0 ysize} ** 0.4 @Label SB
    {0 ysize} ** 0.6 @Label ST
    {xsize 0} ** 0.7 @Label HB
    SB
    SB ++ HB
    HB
    {xsize 0} ++ {0 ysize} ** 0.5
    HB ++ {0 ysize}
    HB ++ ST
    ST
    SB
  }
  paint { grey }
{ 6c @Wide 2c @High }
```




Incidentally, the labels of a figure can be displayed as above by putting the symbol @ShowLabels at the end of the figure. Labels can save a lot of effort. They should contain only digits, upper-case letters and @, because @Fig and Lout itself use labels of their own made from lower-case letters.

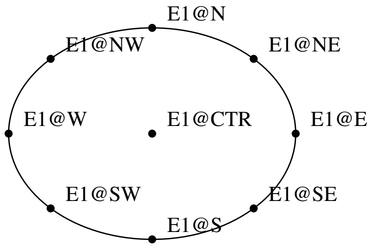
The standard shapes have standard labels; for example, the labels of @Ellipse are

```
@Ellipse
{ 3c @Wide 2c @High }
```



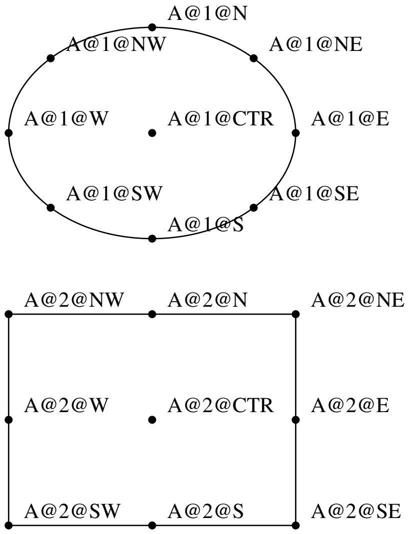

There is a symbol, ::, for *relabelling* an object. Each label in the right parameter is relabelled in the following way:

```
E1:: @Ellipse
{ 3c @Wide 2c @High }
```




Within the right parameter of :: the original names hold sway; but afterwards the names are changed by prefixing the label and @ to them. These composite labels may be used exactly like other labels. Relabelling can be nested to arbitrary depth:

```
A::
{   1:: @Ellipse
    { 3c @Wide 2c @High }
    //1c
    2:: @Box
    { 3c @Wide 2c @High }
}
```




The right parameter of :: may be any object.

The eight standard shapes (@Box, @CurveBox, @ShadowBox, @Square, @Diamond, @Polygon, @Ellipse, and @Circle) have a special CIRCUM label, not displayed by @ShowLabels. The expression

*angle* CIRCUM

yields the point on the boundary of the shape at the given angle from the centre, in a coordinate system with the centre for origin. Thus, given a labelled standard shape such as

```
A :: @Ellipse ...
```

the point on its boundary at an angle of 45 degrees from the centre is

```
A@CTR ++ {45 A@CIRCUM}
```

The braces must be present. Regrettably, there is no way to produce a CIRCUM label for shapes defined by the user in any reasonable time.

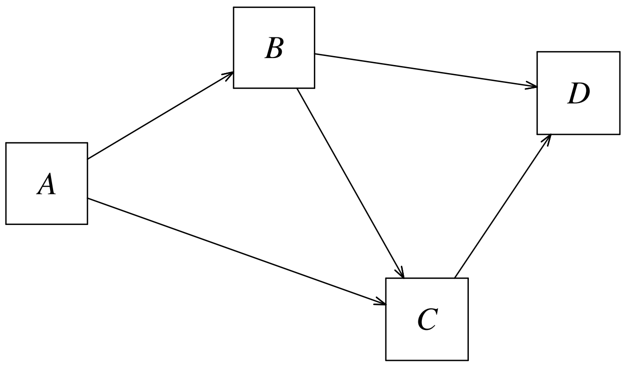
If the same label is used twice, as is inevitable with unlabelled standard shapes, only the most recent value is remembered. There is a limit on the maximum number of distinct labels in any one figure, which can be changed by means of an option to the @Fig symbol:

```
@Fig
  maxlabels { 500 }
{ ... }
```

The default value is 200.[1] Large values could cause the printing device to run out of memory. Memory is reclaimed at the end of each figure.

## 9.5. Putting it all together

In this section we consider the problem of linking individual shapes together to form complex diagrams like this one:




We already have several aids to hand: the basic graphics symbols for rotation and scaling, the ability to nest text, equations, and other figures (in fact arbitrary Lout objects) within our shapes; and the ability to define our own symbols.

Two other useful aids are the horizontal and vertical concatenation symbols from raw Lout:

```
A |1.0c B
```

causes A and B to appear side by side, one centimetre apart (this is horizontal concatenation). In general, A and B may be arbitrary objects, and any length is acceptable immediately after the

[1]In recent versions of PostScript, a change has been made which has the effect of automatically increasing the value of maxlabels when required. In those versions you can safely forget about maxlabels.

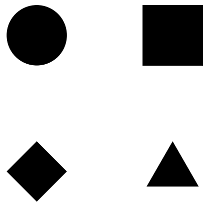
| symbol. Similarly,

```
A /1.0c B
```

makes A and B appear with B directly beneath A (this is vertical concatenation), again with a one centimetre separation. You could use tables to get the same effects. Vertical concatenation often appears below in the special form //, which roughly means vertical concatenation with zero separation.

The default values of the various options – solid for linestyle, noarrow for arrow, and so on – may be changed by giving options to the @Fig symbol:

```
@Fig
  linestyle { noline }
  paint { black }
{
  @Circle |1c @Square
  /1c @Diamond | @Polygon
}
```



A complete list of options is given in the next section.

@Fig provides another aid: the symbols @BaseOf and @MarkOf. The right parameter of @BaseOf is an arbitrary object, and its left parameter is a point. As far as Lout is concerned, the result of @BaseOf is always an empty object; but the right parameter will appear on the page with the bottom left-hand corner of its base at the point denoted by the left parameter. We stress that Lout has no idea that this is happening, and so cannot prevent the shifted object from writing over other objects or even going off the edge of the page. Of course, this lack of discipline is just what is needed very often in diagrams.

The @MarkOf symbol works in a similar way, except that the point where the object's alignment marks cross (rather than its bottom left-hand corner) will appear on the page at the point denoted by the left parameter.

We can set up a coordinate system for a figure:

```
@Figure shape { xsize 0 @Label X  0 ysize @Label Y }
{ 10c @Wide 6c @High }
```

In fact, Fig contains this shape under the name @Frame, so we need only write

```
@Frame { 10c @Wide 6c @High }
```

Of course, the right parameter may contain an arbitrary Lout object.

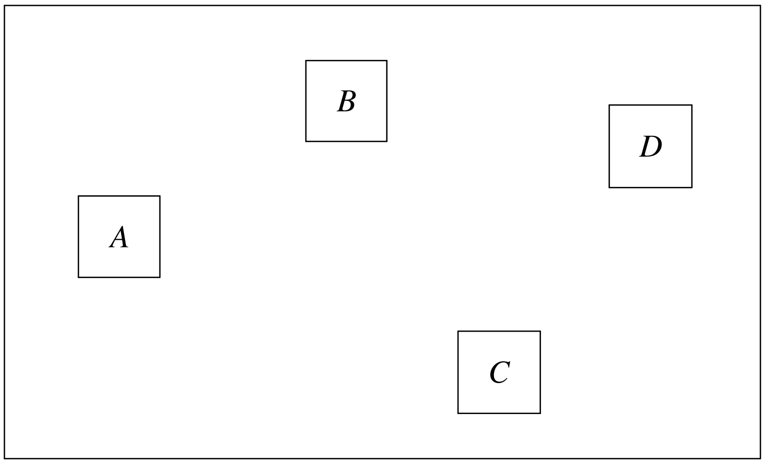
Once the frame is set up, we can specify points by their X and Y coordinates, as fractions of the total width and height:

```
X ** 0.5  ++  Y ** 0.8
```

To place the squares in the diagram above, we can use

```
// X**0.1 ++ Y**0.4  @BaseOf  A:: @Square { @I A }
// X**0.4 ++ Y**0.7  @BaseOf  B:: @Square { @I B }
// X**0.6 ++ Y**0.1  @BaseOf  C:: @Square { @I C }
// X**0.8 ++ Y**0.6  @BaseOf  D:: @Square { @I D }
```

The symbols' precedences are chosen so that this very common situation requires no braces. The result of this is




where we have drawn a box with margin 0 around the frame to make its extent clear.

Now the arrow from *A* to *B* starts on the boundary of *A* at the angle of a line drawn between the centres of *A* and *B*:

```
A@CTR ++ { {A@CTR @Angle B@CTR} A@CIRCUM }
```

and a similar expression will yield the endpoint of the arrow. Such expressions should be placed into definitions if they are to be used often:

```
import @Fig
def @JoinFigures
  left A
  named linestyle { solid }
  named dashlength { 0.15 cm }
  named arrow { noarrow }
  named linewidth { 0.5 pt }
  right B
{ @Arrow
    from { A"@CTR" ++ {{A"@CTR" @Angle B"@CTR"} A"@CIRCUM"} }
    to   { B"@CTR" ++ {{B"@CTR" @Angle A"@CTR"} B"@CIRCUM"} }
    linestyle { linestyle }
    dashlength { dashlength }
    arrow { arrow }
    linewidth { linewidth }
  {}
}
```

Definitions are the subject of Section 2.12, although for named parameters, which define options, you have to consult the Lout expert's guide [5]. Now, to the figure above we can add

```
// A @JoinFigures arrow { forward } B
// A @JoinFigures arrow { forward } C
// B @JoinFigures arrow { forward } C
// B @JoinFigures arrow { forward } D
// C @JoinFigures arrow { forward } D
```

to obtain the diagram as it appears at the beginning of this section. Definitions are the best means of managing complexity in diagrams, and serious users of advanced graphics will need to keep a stock of them in a mydefs file.

## 9.6. Errors

Lout normally produces an output file that will print without mishap on any PostScript device. However, some of the options of @Fig's symbols are passed through Lout to the output file without checking, including anything containing @Fig lengths, angles, points, and labels. Any errors in these options will not be detected until the file is printed.

The most likely errors are *syntax errors*, as in shape { 0 0 [ 0 xsize } for example, in which a ] is missing; *type errors*, as in 0 0 @Distance 45 where the right parameter is not a point; and *undefined errors*, arising from labels misspelt or used before being defined. Less commonly, the options may all be correct but the figure is too large in some way: too many labels, too deeply nested, etc.

When an error is detected, @Fig arranges for the offending page to be printed up to the point where the error occurred, with a message nearby describing the error. Printing of the document is then aborted. It is often quite easy to find the problem, because it lies in whatever should have been printed next.

## 9.7. Summary

The options to the @Fig symbol, and their default values, are as follows. The complete range of options is shown at right:

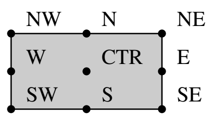
@Fig

| | | |
|---|---|---|
| maxlabels | { 200 } | any whole number |
| linestyle | { solid } | solid dashed cdashed dotted noline |
| linewidth | { 0.5 pt } | any Fig length (see below) |
| linecap | { round } | butt round project |
| dashlength | { 0.15 cm } | any Fig length |
| paint | { nopaint } | nopaint or any colour from Section 8.1 |
| margin | { 0.4c } | any Lout length |
| arrow | { noarrow } | noarrow forward back both |
| headstyle | { open } | open halfopen closed |
| headwidth | { 0.05 cm } | any Fig length |
| headlength | { 0.15 cm } | any Fig length |

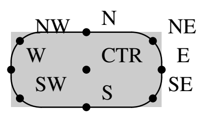
The linecap option determines the shape of the ends of lines: round puts a round cap on them, which is the most useful with @Fig; butt is a square end; and project is a square end projecting half a line width past the end of the line; it is useful for getting sharp corners on rectangles and square dots in dotted lines.[1]

The following standard shapes take the same options as @Fig, except that they do not have maxlabels or the last four (arrow-drawing) options, and occasionally they have other options. In most cases the default values of these options are taken from the enclosing @Fig. Where there are extra options, or where a different default value is used, the option and its default value are shown. The list also shows the shape's labels, and how it is superimposed on its right parameter (shown as a grey rectangle). A larger margin will enlarge the right parameter and hence the shape as well (with @CurveBox, the margin also determines the curvature at the corners). Squares, polygons and circles have a diameter equal to the larger of xsize and ysize.

@Box

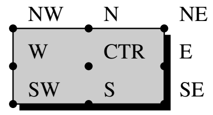



@CurveBox

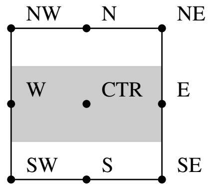



[1]The line joining options of PostScript are not reflected in @Fig options because @Fig strokes paths segment by segment, not all at once, and so there are no line joins in the PostScript sense. This was done to improve the appearance of dashed and dotted lines.

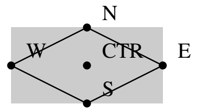
```
@ShadowBox
  shadow { 0.2f }
```



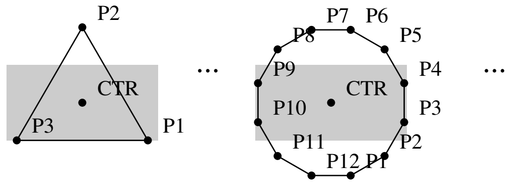

```
@Square
```



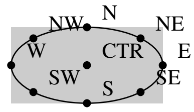

```
@Diamond
```



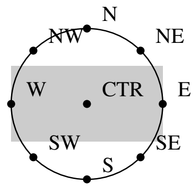

```
@Polygon
  sides { 3 }
  angle { 180/sides }
```




```
@Ellipse
```



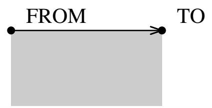

```
@Circle
```




The following standard shapes have the same options as @Fig, including the four arrow-drawing options, and occasionally they have others. In each case the only difference between the line and arrow symbols is the default value of arrow, which lines take from @Fig and arrows set to forward. The first four draw a line along the mark of the right parameter, which is not necessarily the same as its left or top edge.

```
@HLine
  margin { 0c }
```

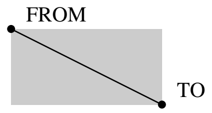

```
@HArrow
  margin { 0c }
  arrow { forward }
```



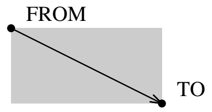
```
@VLine
  margin { 0c }
```


```
@VArrow
  margin { 0c }
  arrow { forward }
```

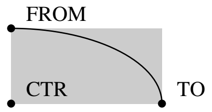

```
@Line
  from { 0 ysize }
  to { xsize 0 }
  margin { 0c }
```




```
@Arrow
  from { 0 ysize }
  to { xsize 0 }
  margin { 0c }
  arrow { forward }
```




```
@Arc
  from { 0 ysize }
  to { xsize 0 }
  ctr { 0 0 }
  direction { clockwise }
  margin { 0c }
```




More generally, the @Figure symbol takes all the options of @Fig except maxlabels, together with a shape option containing a sequence of points, and it connects each pair of points by a line or curve as specified by the following:

| | |
|---|---|
| *point point* | Straight line |
| *point* [] *point* | Draw nothing |
| *point* [ *point* ] *point* | Anticlockwise circular or elliptical arc |
| *point* [ *point* clockwise ] *point* | Clockwise circular or elliptical arc |
| *point* [ *point point* ] *point* | Bezier curve with two control points |

The remaining symbols do not draw shapes. They are

*point* @Label *label*

Within a shape, makes *label* stand for the point on the page denoted by *point*, which is not made part of the shape. The label applies from here onwards until some other point is given this label, a relabelling occurs, or the figure ends.

*label* :: *object*

Relabel every labelled point in the right parameter (which may be an arbitrary Lout object), by adding *label*@ to the front of each label.

@Frame *object*

Equivalent to @Figure shape {xsize 0 @Label X 0 ysize @Label Y} *object*.

*point* @BaseOf *object*

Translate *object* so that its bottom left-hand corner appears at *point*. Lout thinks that the result is an empty object.

*point* @MarkOf *object*

Translate *object* so that the point where its marks cross appears at *point*. Lout thinks that the result is an empty object.

@ShowLabels

Display all the labels of the figure created up to this point.

The following lists define all the ways to specify lengths, angles and points.[1] Brief explanations appear to the right, with the symbols' precedences in parentheses where appropriate.

[1]A length is represented in PostScript by a single number on the operand stack; so is an angle. A point is represented by two numbers on the stack. Those familiar with PostScript and willing to sacrifice portability and increase their risk of error can therefore write, for example, *length* sqrt within a shape, to obtain a length which is the square root of another length, or *point* exch to obtain the reflection of a point about the main diagonal, and so on.

*length*

| | |
|---|---|
| 0 | zero |
| xmark | distance to column mark |
| ymark | distance to row mark |
| xsize | distance to right boundary |
| ysize | distance to top boundary |
| *number* in | *number* inches (39) |
| *number* cm | *number* centimetres (39) |
| *number* pt | *number* points (39) |
| *number* em | *number* ems (39) |
| *number* sp | 1 sp is the current width of a space (39) |
| *number* vs | 1 vs is the current inter-line space (39) |
| *number* ft | 1 ft is the size of the current font (39) |
| *point* @Distance *point* | distance between two points (35) |
| *point* @XDistance *point* | horizontal distance between two points (35) |
| *point* @YDistance *point* | vertical distance between two points (35) |

*angle*

| | |
|---|---|
| *number* dg | *number* degrees (39) |
| *number* | *number* degrees (dg is optional) |
| *point* @Angle *point* | angle from first point to second (35) |

*point*

| | |
|---|---|
| *length length* | *x* and *y* distance from origin (5) |
| *length* << *angle* | distance and angle from origin (38) |
| *point* ++ *point* | vector sum of two points (36) |
| *point* -- *point* | vector difference of two points (36) |
| *point* @Max *point* | vector maximum of two points (36) |
| *point* @Min *point* | vector minimum of two points (36) |
| *point* ** *number* | multiplication of point by number (37) |
| *label* | a previously defined label |
| @Prev | the previous point in a shape |

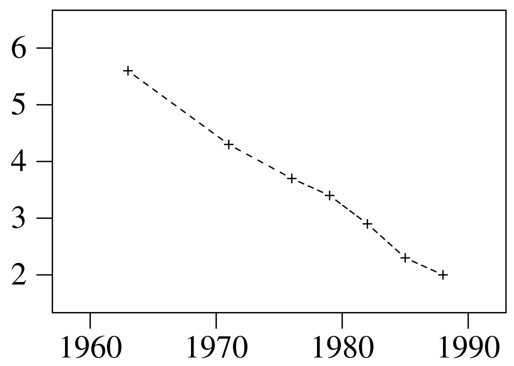
# Chapter 10. Graphs

This chapter describes how to draw graphs, using the @Graph symbol. For example,

```
@Graph
   abovecaption { New South Wales road deaths, 1960--1990
(fatalities per 100 million vehicle km) }
{
   @Data  points { plus }  pairs { dashed }
   { 1963 5.6  1971 4.3  1976 3.7  1979 3.4  1982 2.9  1985 2.3  1988 2.0 }
}
```

produces the graph

New South Wales road deaths, 1960–1990
(fatalities per 100 million vehicle km)




The features of @Graph include captions, automatic and manual ticks and labels, logarithmic axes, histograms, and plotting of mathematical functions.

### 10.1. Introduction

The Lout definitions for graph formatting are kept in a file called graph, which you must include at the start of your document if you want graphs, like this:

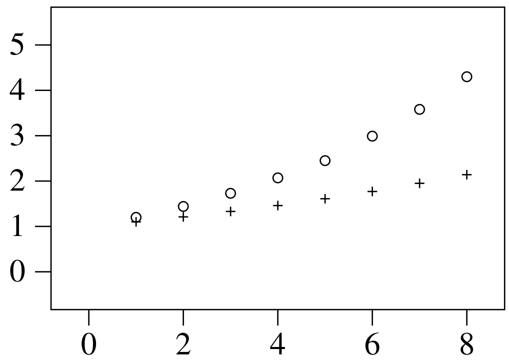
```
@SysInclude { graph }
@SysInclude { doc }
@Doc @Text @Begin
...
@End @Text
```

Files of definitions, like graph, must be included before the setup file. Alternatively, if you are using your own setup file, you may place the include commands within it; the place to put them is clearly marked in all the standard setup files. With this inclusion, the @Graph symbol used below will then be available for use anywhere within your document.

@Graph distinguishes between the overall graph, produced by the @Graph symbol itself, and the data sets to be placed within it, each of which is enclosed by a @Data symbol:

```
@CentredDisplay @Graph
{
   @Data  points { plus }
   { 1 1.10  2 1.21  3 1.33  4 1.46  5 1.61  6 1.77  7 1.95  8 2.14 }

   @Data  points { circle }
   { 1 1.20  2 1.44  3 1.73  4 2.07  5 2.45  6 2.99  7 3.58  8 4.30 }
}
```

Although it is good practice to lay the input data out neatly, layout has no effect on the result. It is not necessary to have one data point per line, for example. The result of this example is




We have used the @CentredDisplay symbol from Section 2.1 to produce a centred display, but the @Graph symbol produces an object which may appear anywhere at all – in a figure, for example, or as an entry in a table.

### 10.2. Changing the overall appearance of the graph

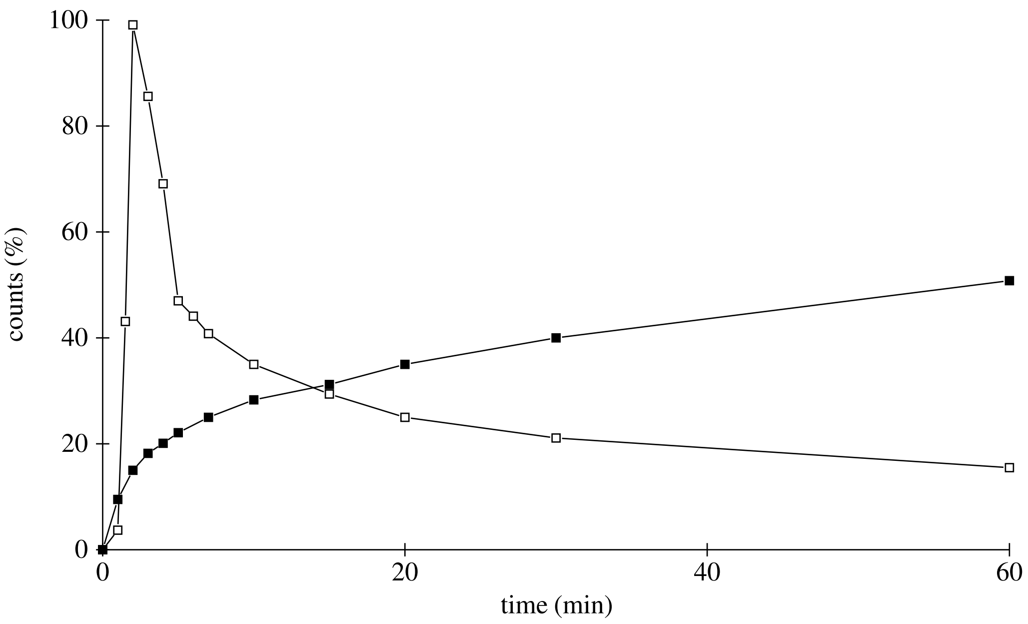
The overall appearance of the graph is controlled by options to the @Graph symbol. As usual, these options follow the @Graph symbol, with their values enclosed in braces; they may appear in any order, and if omitted are assigned some sensible default value.

There is a style option for controlling the overall style of the graph, whose value may be either frame, none, or axes. The default value is frame, and it produces a frame around the graph with ticks and labels along its left and bottom edges, as in previous examples. The none style prints nothing (no frame, no ticks, no labels), which is useful for producing graphs that don't look like graphs, as it were.

If the other value, axes, is chosen, two other options called xorigin and yorigin become compulsory:

```
-2p @Font @Graph
    style { axes }
    xorigin { 0 }
    yorigin { 0 }
    width { 12 cm }
    height { 7 cm }
    leftcaption { 90d @Rotate { counts (%) } }
    leftgap { 1.0 cm }
    belowcaption { time (min) }
    belowgap { 0 cm }
{
    @Data
        points { filledsquare }
        pairs { solid }
    { 0 0.0 1 4.8 2 7.0 3 15.2 4 19.8 5 20.0 6 21.0 7 25.0
      10 29.5 15 31.2 20 35.0 30 40.0 60 50.8
    }

    @Data
        points { square }
        pairs { solid }
    {
      0 0.0 1 3.7 1.5 43.1 2 99.1 3 85.6 4 69.1 5 47.0 6 44.1 7 40.8
      10 35.0 15 29.4 20 25.0 30 21.1 60 15.5
    }
}
```

We have requested a smaller font size for this graph as a whole by preceding it with -2p @Font, meaning two points smaller, and we have used some other options which will be explained shortly. The resulting graph has an x axis and a y axis instead of a frame, like this:



The point where the axes cross is (xorigin, yorigin).

Although @Graph does not provide explicit support for multiple axes, you can simulate them by overstriking two separate graphs of equal size. There is an @OverStrike symbol which overstrikes two objects, so

```
@Graph { ... } @OverStrike @Graph { ... }
```

will do the job.

There are xlog and ylog options which produce logarithmic x and y axes:

```
@Graph
   xlog { 10 }
   ylog { 10 }
{
   ...
}
```

The value is the base of the logarithm, usually 10 or 2. The default is the special value none, meaning not logarithmic. Logarithms to different bases differ only by a constant factor, so the main effect of different bases is on the choice of ticks and labels. An xlog option will be ignored if there are any negative or zero x data points, x ticks, or xorigin or xmin options; and similarly for ylog.

There are width and height options for setting the size of the total area enclosed:

```
@Graph
   width { 6.0 cm }
   height { 4.0 cm }
{
   ...
}
```

This shows the default width and height, six centimetres and four centimetres. These distances and others discussed below can be specified using a variety of units of measurement (see Section 10.10 for the details).

Within the frame or axes, a small margin is kept free of data points. The size of this margin is controlled by xextra and yextra options:

```
@Graph
   xextra { 0.5 cm }
   yextra { 0.5 cm }
{
   ...
}
```

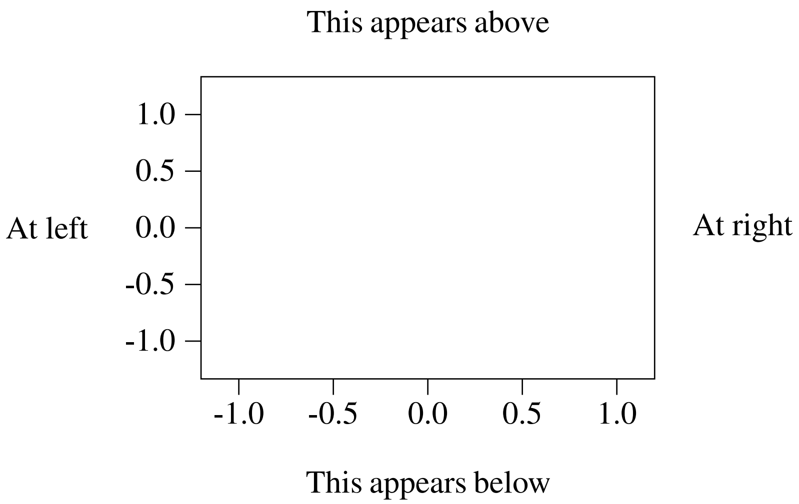
Setting xextra to 0.5 cm (the default value if the style option is frame) means that the smallest x value will be placed 0.5 centimetres to the right of the left boundary, and the largest will be placed 0.5 centimetres to the left of the right boundary. It is quite safe to set xextra to 0 cm if desired, and indeed this is the default value when style is axes or none. The yextra option works in exactly the same way for y values.

## 10.3. Captions

There are options for placing captions above, below, left, and right of the frame:

```
@Graph
   abovecaption { This appears above }
   belowcaption { This appears below }
   leftcaption { At left }
   rightcaption { At right }
{
}
```

produces




The captions may be arbitrary Lout objects, so may include equations, @Rotate, and so on. Each caption except rightcaption is printed in the clines @Break style, which means that multiple lines in one caption will be centred beneath each other. The rightcaption option uses the lines @Break style, in which the lines are left justified beneath each other. Incidentally, this example shows what happens if there is no data.

There are options for controlling the amount of space between each caption (when non-empty) and the frame. Here they are with their default values:

```
@Graph
   abovegap { 0.5 cm }
   belowgap { 0.5 cm }
   leftgap { 1.5 cm }
   rightgap { 0.5 cm }
{
   ...
}
```

This is particularly important in the case of leftgap, because Lout has no idea how wide the ticks and labels attached to the y axis are; 1.5 cm is just a wild guess and often needs adjustment. On the other hand, Lout does know how high the ticks and labels on the x axis are; it allows 1.7 times the current font size for them, and belowgap is additional to this.

When a graph is to be presented as a centred display, it is generally best if the centring is done with respect to the frame alone, not the captions, ticks, and labels. A hidecaptions option is provided which accomplishes this by making the left and right captions and gaps seem to Lout to have zero width:

```
@Graph
   hidecaptions { yes }
{
   ...
}
```

Actually yes has been made the default value, since the vast majority of graphs are centred displays. In the rare cases where this feature is not wanted (for example, if a graph appears as an entry in a table), use hidecaptions { no }. The y ticks and labels seem to Lout to have zero width already, so do not need to be hidden.

## 10.4. Ticks and labels

*Ticks* are the short lines that mark off intervals along the axes, and *labels* are the numbers appearing near the ticks (not to be confused with captions). @Graph produces ticks and labels automatically with some care, so it is probably best not to worry about them unless the result is not pleasing, in which case there are options for controlling them.

One simple way to control the production of x ticks is with the xmin, xmax, and xticksep options to @Graph. For example,

```
@Graph
   xmin { 0 }
   xmax { 5 }
   xticksep { 0.5 }
```

specifies that x values in the range 0 to 5 are to be expected, and that a tick and label is to appear every 0.5 units along the x axis. One or both of xmin and xmax may be omitted, in which case suitable values will be inferred from the data as usual.

Alternatively, complete control over the appearance of x ticks and labels is provided by the xticks option. For example,

```
@Graph
   xticks { 0@  5  10@  15  20@ }
```

specifies that x ticks are to be drawn at 0, 5, 10, 15, and 20. An @ following a number indicates that a label is to be printed as well, so the above example will produce labels at 0, 10, and 20. For even finer control, @ may be replaced by a label enclosed in parentheses:

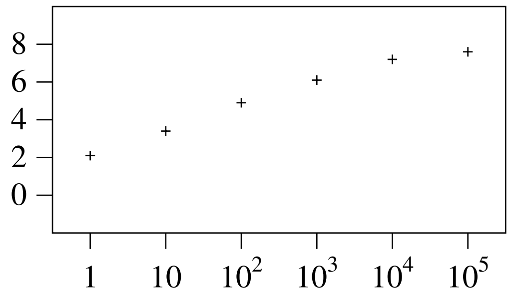
```
@Graph
    xticks { 1 (Democrat)  2 (Republican)  3 (Other) }
```

As this example shows, a label does not have to be a number; it can be any string of characters, including spaces and balanced parentheses; but it may not be an arbitrary Lout object.

The character ^ in a label is special and indicates that the remainder of the label is to be treated as an exponent:

```
@Graph
    xlog { 10 }
    xticks { 1 (1)  10 (10)  100 (10^2)  1000 (10^3)  10000 (10^4)  100000 (10^5) }
{
    @Data points { plus }
    { 1 2.1  10 3.4  100 4.9  1000 6.1  10000 7.2  100000 7.6 }
}
```

In fact, the labels inserted automatically when xticks is omitted have exponents when the axis is logarithmic, so xticks is hardly necessary in this example. Anyway the result is




Setting xticks to empty produces no x ticks (this is not the same as omitting xticks).

Similar options control ticks and labels on the y axis: ymin, ymax, yticksep, and yticks. There are also xticklength and yticklength options which set the length of ticks:

```
@Graph
    xticklength { 0.5 ft }
    yticklength { 0.5 ft }
```

shows the default values, half the current font size in both cases.

Lout has only a hazy idea of how much space is occupied by ticks and labels. Unless xticks is empty, Lout allows 1.7 times the current font size below the graph for x ticks and labels, which is usually about right; but it does not allow any space for y ticks and labels since it has no idea how wide the labels will be. The discussion of captions in Section 10.3 explains how to use the leftgap option to work around this deficiency.

## 10.5. Changing the appearance of the data

The @Data symbol has several options for controlling the appearance of the data it encloses. We have already seen the points option, which controls what is printed at each data point. The choices for it are

| | | | |
|---|---|---|---|
| cross | × | plus | + |
| square | □ | filledsquare | ■ |
| diamond | ◇ | filleddiamond | ◆ |
| circle | ○ | filledcircle | ● |
| triangle | △ | filledtriangle | ▲ |

If the points option is omitted or empty, nothing is printed. The symbols are centred over the data point. There is a symbolsize option which controls the size (radius) of all these symbols:

```
@Data
   points { triangle }
   symbolsize { 0.15 ft }
```

shows the default size, 0.15 times the current font size. More precisely, the default value of this option is taken from an option to the @Graph symbol, also called symbolsize. By setting this option you can therefore set the symbol size of all data points in the graph at once; its default value is 0.15 ft.

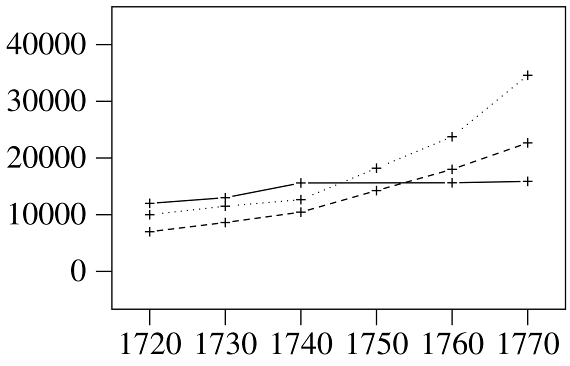
The @Data symbol also has a pairs option which determines how each pair of points is connected. The choices are none (not connected, the default), solid (a solid line), dashed (a dashed line), or dotted (a dotted line). For example,

```
@Graph
   abovecaption { Estimated population of Boston, New York, and Philadelphia
1720--1770 }
{
   @Data  points { plus }  pairs { solid }
   { 1720 12000  1730 13000  1740 15601  1760 15631  1770 15877 }

   @Data  points { plus }  pairs { dashed }
   { 1720 7000  1730 8622  1740 10451  1750 14255  1760 18000  1770 22667 }

   @Data  points { plus }  pairs { dotted }
   { 1720 10000  1730 11500  1740 12654  1750 18202  1760 23750  1770 34583 }
}
```

produces[1]

[1]Source: R. C. Simmons, *The American Colonies*, W. W. Norton, New York, 1981.

Estimated population of Boston, New York, and Philadelphia
1720–1770




We will see in Section 10.8 how to add an explanatory key to this graph. If the points have symbols, these connecting lines will stop 1.5 symbolsizes away from the data points, so as not to overstrike them. If the points have no symbols and pairs is dashed, the first and last dash in each segment will have half the length of the others.

A dashlength option controls the length of dashes and also the separation between dots, and a linewidth option controls the width (thickness) of the lines and dots:

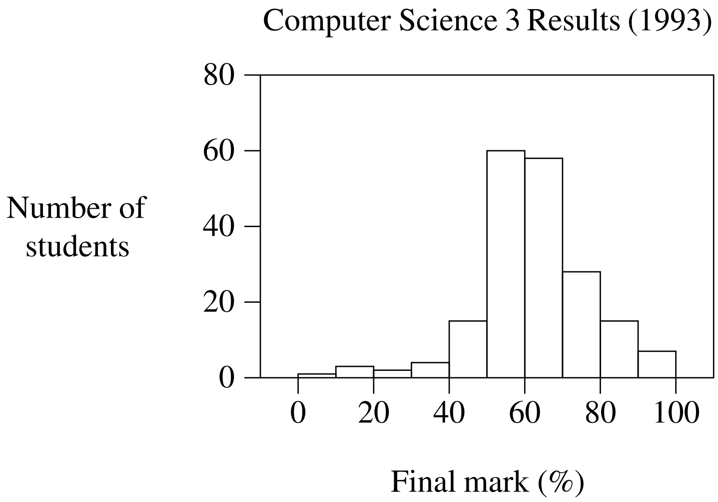
```
@Data
   pairs { dashed }
   dashlength { 0.2 ft }
   linewidth { 0.5 pt }
{
   ...
}
```

This shows the default values, 0.2 ft for dashlength and 0.5 pt (half a point) for linewidth. Actually the default value for linewidth is whatever happens to be already in use, but Lout sets line widths to half a point initially.

The pairs option is also used for producing histograms, like this:

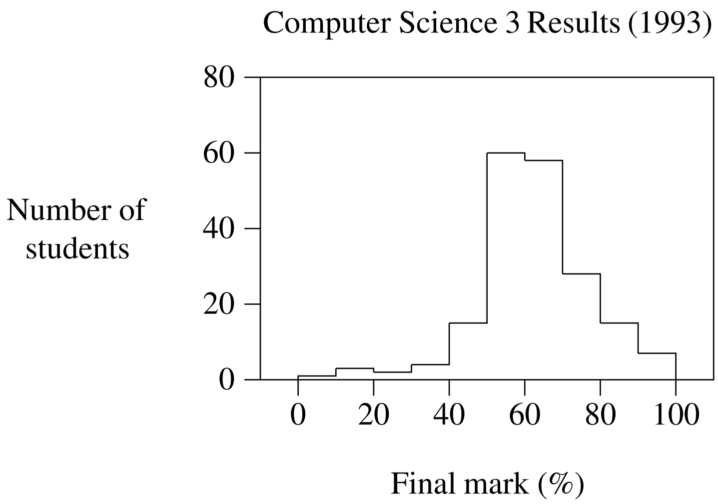
```
@Graph
   hidecaptions { yes }
   abovecaption { Computer Science 3 Results (1993) }
   leftcaption { Number of
students }
   belowcaption { Final mark (%) }
   yextra { 0 cm }
   ymax { 80 }
{
   @Data
      pairs { yhisto }
   { 0 1 10 3 20 2 30 4 40 15 50 60 60 58 70 28 80 15 90 7 100 0 }
}
```

which has result




Note carefully that one y histogram rectangle occupies the space from one x value to the next, with height equal to the y value lying between these two x values. This means that the very last y value has no effect on the result (however, there must be a last y value anyway).

There is a variant of yhisto called surfaceyhisto. Here is the previous example again, this time using surfaceyhisto:



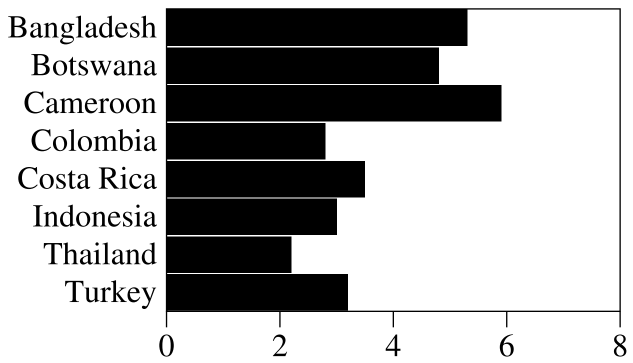

As you can see, surfaceyhisto draws just the surface of the histogram, not the descending lines.

There are xhisto and surfacexhisto values of pairs which produce a histogram whose bars are parallel to the x axis. There are also filledyhisto and filledxhisto values which produce filled rectangles rather than outlined ones:

```
@Graph
   abovecaption { Fertility rates in some developing countries }
   xextra { 0 cm }
   yextra { 0 cm }
   xmax { 8 }
   yticks {
      1.5 (Turkey) 2.5 (Thailand)
      3.5 (Indonesia) 4.5 (Costa Rica)
      5.5 (Colombia) 6.5 (Cameroon)
      7.5 (Botswana) 8.5 (Bangladesh)
   }
   yticklength { 0 cm }
{
   @Data
      pairs { filledxhisto }
   { 0 1 3.2 2 2.2 3 3.0 4 3.5 5 2.8 6 5.9 7 4.8 8 5.3 9 }
}
```

produces[1]

Fertility rates in some developing countries




Once again each bar goes from one y value to the next, with its x value equal to the x value lying between the two y values; this time the very first x value has no effect on the result.

The colour of one set of data can be changed with a colour option:

```
@Data
   colour { blue }
```

For the complete list of acceptable colour names, see Section 8.1. The colour option's name may also be spelt color.

It is also possible to paint the area between the data points and the x axis (or frame if style is not axes), using

[1] Source: Bryant Robey, Shea O. Rutstein, and Leo Morros: The fertility decline in developing countries, *Scientific American*, December 1993.

```
@Data
   paint { yes }
```

The paint colour is determined by the colour option just introduced; it will be black if no colour is specified. Paint (including white paint) hides paint, points, and lines drawn by previous data sets. However the points and lines of each data set are drawn after painting that set, so they cannot be hidden under their own paint; and axes and frames are drawn last so that they too are never hidden.

A dataformat option is provided for changing the interpretation of the data. Ordinarily, as we know, the numbers are taken to be pairs of x and y coordinates, like this:

```
@Data
{
   x y  x y  ...  x y
}
```

However, by setting dataformat to yonly, the interpretation is changed to a sequence of y coordinates only:

```
@Data
   dataformat { yonly }
{
   y y  ...  y
}
```

and x values 1, 2, and so on are inserted automatically, just as though the original input had been

```
@Data
{
   1 y  2 y  ...
}
```

There is also xonly, which inserts y values 1, 2, and so on. The default value, xandy, gives the usual interpretation. The layout of data on lines has no effect on the interpretation.

## 10.6. Placing arbitrary objects on the graph

As we have just seen, the repertoire of symbols that @Data is able to place on the graph is quite limited. However, there is a way to place any number of arbitrary Lout objects anywhere on the graph, using the objects option to the @Graph symbol:

```
@Graph
   objects {
      @CTR at {2.5  6.0}  @Eq { y = x sup 2 }
      @CTR at {4.5  7.0}  @Eq { y = x sup 3 }
   }
```

where we have used the @Eq symbol from Chapter 7 twice to place two equations onto the graph at the points 2.5 6.0 and 4.5 7.0 respectively. An example result appears in the next section.

In addition to @CTR, there are eight other symbols which may be used within the objects option in the same way: @NW, @SW, @SE, @NE, @N, @W, @S, and @E. These place the object just to the northwest of the point, to the southwest, and so on instead of centring it over the point. By 'to the northwest' we mean that the object's bottom right corner coincides with the point, and similarly for the other symbols.

Each of these symbols has a margin option which enlarges the object by adding a margin around it before placing it:

```
@NW at {2.5  6.0} margin { 0.3 ft }  @Eq { y = x sup 2 }
```

shows the default value, 0.3 times the current font size. As the margin is increased, the object moves further away from the point.

The major advantage of the objects option over the @Data symbol is that arbitrary Lout objects may be used. The @Data symbol however is able to place many copies of its symbols onto the graph, and also allow for them when connecting points together with lines. Also, the points within the objects option are not taken into account when deciding on the permissible range of x and y values, whereas the points within the @Data symbol are. Altogether it seems best to use the @Data symbol for the bulk of the data points, and to use the objects option for adding a small number of labels or other decorations.

## 10.7. Mathematical functions, loops, and tests

@Graph offers quite a large selection of mathematical functions, available everywhere that x and y coordinates are required: within the xticks and yticks options, within the points within the objects option, and within the right parameter of the @Data symbol. For example,

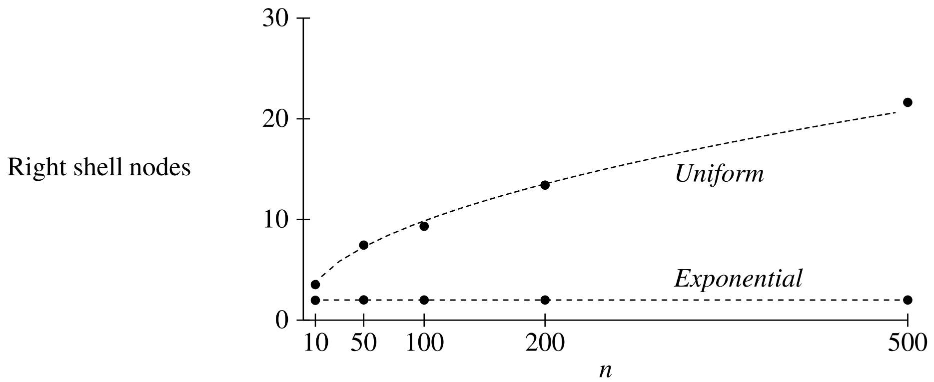
```
@Data
   pairs { solid }
{
   0 0  pi sin { pi/2 }
}
```

draws a solid line from $(0, 0)$ to $(\pi, \sin(\pi/2))$. Section 10.10 lists all the functions; they include the four arithmetical operators +, −, ∗, and /, as well as sin, cos, sqrt, and many others. Braces are used for grouping, never parentheses.

For plotting functions there are three looping symbols, xloop, yloop, and zloop. For example, the following plots the two functions $y = 2$ and $y = \sqrt{\pi x/4} + 1$ for $x$ from 10 to 500:

```
-2p @Font @Graph
    style { axes }
    xorigin { 0 }
    yorigin { 0 }
    width { 8 cm }
    xticks { 10@ 50@ 100@ 200@ 500@ }
    objects {  @NE at { 300 2 } @I { Exponential }
            @SE at { 300 sqrt { pi*300/4 } + 1 } @I { Uniform } }
    belowcaption { @I n }
    belowgap { 0 cm }
    leftcaption { Right shell nodes }
{
    @Data points { filledcircle }
    { 10 1.97  50 2.01  100 2.00  200 2.0  500 2.00 }

    @Data points { filledcircle }
    { 10 3.53  50 7.45  100 9.32  200 13.41  500 21.63 }

    @Data pairs { dashed }
    { 10 2  500 2 }

    @Data pairs { dashed }
    {
        xloop from { 10 } to { 500 } by { 20 } do
        {
            x  sqrt { pi*x / 4 } + 1
        }
    }
}
```

The do option of xloop is replicated repeatedly with each occurrence of x replaced by 10, 30, 50, ... up to 490. The result is[1]




The points are connected by straight line segments as usual, but a smallish by option of about

[1]Source: Jeffrey H. Kingston, Analysis of tree algorithms for the simulation event list. *Acta Informatica* **22**, pp. 15–33 (1985).

one-twentieth of the range creates the illusion of a smooth curve quite well.

There is also an if symbol which produces alternative results, depending on whether a condition evaluates to true or false:

```
xloop from { -5 } to { +5 } by { 0.2 } do
{
   if cond { abs { x } > 0.1 } then { x 1/x } else {}
}
```

This plots the function $y = 1/x$, skipping points near zero. Actually the else part could be omitted since its default value is empty.

Adventurous users might enjoy nesting a yloop or zloop within an xloop, or using loops to generate ticks, like this:

```
xticks {
   xloop from { 0 } to { 20 } do
   {
     x if cond { x mod 5 = 0 } then { @ }
   }
}
```

The missing by option defaults to 1, so this produces x ticks at 0, 1, 2, ..., 20, with labels at 0, 5, 10, 15, and 20. It is quite all right to mix @ and even labels in with numbers, as long as the final result obeys the rules of Section 10.4.

You can define your own functions using Lout definitions, placed in your mydefs file as explained in Section 2.12. Here is an example of a function definition:

```
import @Graph @Data
def @Tan
   precedence 40
   right x
{
   sin x / cos x
}
```

This defines a function called @Tan which implements the trigonometric tangent function. It may then be used in expressions just like any other function:

```
@Data {
   yloop from { 0 } to { 0.95 } by { 0.05 } do
   {
      y   @Tan { y / pi }
   }
}
```

Following is a detailed explanation.

The first line, import @Graph @Data, is the import clause. Its function is to grant the definition access to the three previously defined functions (symbols) that it uses, namely sin, cos, and /. These are found within the @Data symbol within @Graph.

After the import clause comes the def keyword, meaning 'define,' and then the name of the symbol being defined, in this case @Tan. We have chosen @Tan rather than tan because symbols defined by the user in this way are visible throughout the document, and we do not want the literal word tan to be taken as a symbol.

Next comes the symbol's precedence, in this case the same as sin and cos (see Section 9.7 for the precedence of each symbol). Next is a list of the formal parameters, in this case just one, called x, that is to be passed on the right.

Finally comes the body of the definition enclosed in braces. When @Tan is invoked, its value will be this body with each occurrence of the formal parameter x replaced by the object following the @Tan symbol. For example, the do option of the yloop above becomes

```
y   sin { y / pi } / cos { y / pi }
```

as you would expect.

## 10.8. Adding a key to the graph

A *key* to a graph is an explanation of what each data set represents. To assist you in constructing a key, some extra symbols are provided in addition to @Graph. Here are their names and values:

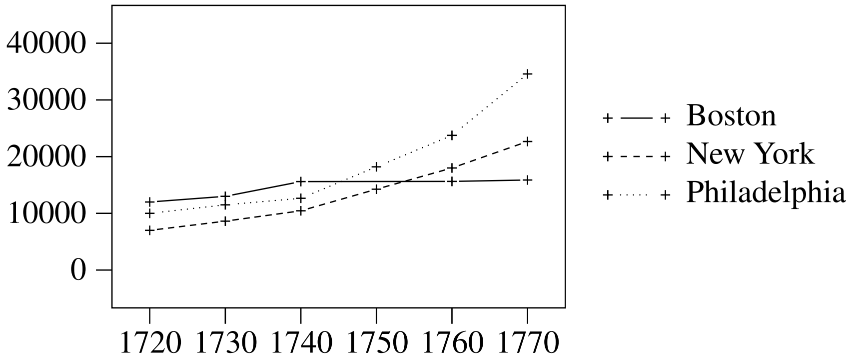
| | | | |
|---|---|---|---|
| @GraphCross | × | @GraphPlus | + |
| @GraphSquare | □ | @GraphFilledSquare | ■ |
| @GraphDiamond | ◇ | @GraphFilledDiamond | ◆ |
| @GraphCircle | ○ | @GraphFilledCircle | ● |
| @GraphTriangle | △ | @GraphFilledTriangle | ▲ |

| | |
|---|---|
| @GraphNoLine | |
| @GraphSolid | — |
| @GraphDashed | - - - |
| @GraphDotted | ..... |

These extra symbols may be used anywhere in your document except within the right parameter of @Graph; they are commonly used within the caption options of @Graph:

```
@Graph
   rightcaption {
@GraphPlus @GraphSolid @GraphPlus  Boston
@GraphPlus @GraphDashed @GraphPlus  New York
@GraphPlus @GraphDotted @GraphPlus  Philadelphia
}
```

Recall that unlike the other captions, rightcaption is set using the lines @Break style rather than clines @Break (Section 10.3). Adding this caption to the graph from Section 10.5, the complete result is




The first eight symbols have a symbolsize option with the usual meaning and the usual default value (0.15 ft). The last four symbols have dashlength and linewidth options with the usual default values, 0.2 ft and 0.5 pt respectively, and a length option, which determines the length of the line drawn by each symbol; its default value is 1.0 ft.

## 10.9. Errors

Lout normally produces an output file that will print without mishap on any PostScript device. However, some of the options of @Graph and all of the data and labels are passed through Lout to the output file without checking. Any errors in this material will not be detected until the file is printed.

The most likely errors are *rangecheck errors*, for example if an attempt is made to divide by zero or take the square root of a negative number, and *undefined errors*, arising from symbols misspelt, use of x outside an xloop, etc. Less commonly, everything may be correct but the graph is too large in some way: too much data, expression too deeply nested, etc. All the data is stored in the printer while the graph is being printed, after which the data is discarded and all memory consumed by the graph is reclaimed.

When an error is detected, @Graph arranges for the offending page to be printed up to the point where the error occurred, with a message nearby describing the error. Printing of the document is then aborted. The problem is usually easy to locate since it lies in whatever should have been printed next.

Older versions of the popular GhostView previewer have a bug which causes exponents in labels to be printed in diminishing sizes. Recent versions no longer have this problem.

## 10.10. Summary

The options to the @Graph symbol, their default values, and their possible values are:

@Graph

| | | | | |
|---|---|---|---|---|
| style | { | frame | } | frame, axes, or none |
| width | { | 6.0 cm | } | any *distance* |
| height | { | 4.0 cm | } | any *distance* |
| xextra | { | 0.5 cm | } | any *distance* (axes and none default is 0 cm) |
| yextra | { | 0.5 cm | } | any *distance* (axes and none default is 0 cm) |
| leftcaption | { | | } | any Lout object |
| rightcaption | { | | } | any Lout object |
| abovecaption | { | | } | any Lout object |
| belowcaption | { | | } | any Lout object |
| leftgap | { | 1.5 cm | } | any *distance* |
| rightgap | { | 0.5 cm | } | any *distance* |
| abovegap | { | 0.5 cm | } | any *distance* |
| belowgap | { | 0.5 cm | } | any *distance* |
| hidecaptions | { | yes | } | yes or no |
| xorigin | { | none | } | none or any *number* |
| yorigin | { | none | } | none or any *number* |
| xlog | { | none | } | none or any *number* greater than 1 |
| ylog | { | none | } | none or any *number* greater than 1 |
| xmin | { | none | } | none or any *number* |
| xmax | { | none | } | none or any *number* |
| ymin | { | none | } | none or any *number* |
| ymax | { | none | } | none or any *number* |
| xticksep | { | none | } | none or any *number* greater than 0 |
| yticksep | { | none | } | none or any *number* greater than 0 |
| xticks | { | none | } | *sequence* (of numbers and strings), or none meaning inserted automatically |
| yticks | { | none | } | *sequence* (of numbers and strings), or none meaning inserted automatically |
| xticklength | { | 0.5 ft | } | any *distance* |
| yticklength | { | 0.5 ft | } | any *distance* |
| objects | { | | } | sequence of @CTR, @NW, @SW, @SE, @NE, @N, @W, @S, @E symbols |
| points | { | none | } | none, plus, cross, square, filledsquare, diamond, filleddiamond, circle, filledcircle, triangle, filledtriangle |
| pairs | { | none | } | none, solid, dashed, dotted, yhisto, xhisto, filledyhisto, filledxhisto, surfaceyhisto, surfacexhisto |
| colour/color | { | none | } | none or any colour name from Section 8.1 |
| paint | { | no | } | no or yes |
| dataformat | { | xandy | } | xandy, yonly, xonly |
| dashlength | { | 0.2 ft | } | any *distance* |
| linewidth | { | 0.5 pt | } | any *distance* |
| symbolsize | { | 0.15 ft | } | any *distance* |

*Number* means an ordinary decimal number; *distance* means a number followed by at least one space followed by any one of the following units of measurement:

| | |
|---|---|
| cm | centimetres |
| in | inches |
| em | Ems (12 ems = 1 inch) |
| pt | Points (72 points = 1 inch) |
| ft | 1 ft is the size of the current font |
| sp | 1 sp is the width of the space character in the current font |
| vs | 1 vs is the current inter-line spacing |

In general, numbers denote x or y values while distances denote lengths on the printed result.

The minimum plottable x value is the minimum of all the x data, xticks, xorigin, xmin, and xmax whenever these are not none. If xticks is none, this minimum may be reduced further to a 'round' number. The maximum plottable x value is the maximum of the same values, and it may be increased further if xticks is none. Similar remarks apply to the minimum and maximum plottable y values.

The value of the objects option is a sequence of zero or more of the following:

| | | |
|---|---|---|
| @CTR | at { *expression expression* } | *object* |
| @NW | at { *expression expression* } | *object* |
| @SW | at { *expression expression* } | *object* |
| @SE | at { *expression expression* } | *object* |
| @NE | at { *expression expression* } | *object* |
| @N | at { *expression expression* } | *object* |
| @W | at { *expression expression* } | *object* |
| @S | at { *expression expression* } | *object* |
| @E | at { *expression expression* } | *object* |

where *object* is an arbitrary Lout object. Each of these nine symbols also has a margin option whose value may be any non-negative distance, with default value 0.3 ft.

The options to the @Data symbol, their default values, and their possible values are:

| | | |
|---|---|---|
| @Data | | |
| points | { *inherited* } | none, plus, cross, square, filledsquare, diamond, filleddiamond, circle, filledcircle, triangle, filledtriangle |
| pairs | { *inherited* } | none, solid, dashed, dotted, yhisto, xhisto, filledyhisto, filledxhisto, surfaceyhisto, surfacexhisto |
| colour/color | { *inherited* } | none, or any colour name from Section 8.1 |
| paint | { *inherited* } | no or yes |
| dataformat | { *inherited* } | xandy, yonly, xonly |
| dashlength | { *inherited* } | any *distance* |
| linewidth | { *inherited* } | any *distance* |
| symbolsize | { *inherited* } | any *distance* |
| { *sequence* } | | |

*Inherited* means that the default value is taken from the @Graph option with the same name.

The right parameter of @Data contains a *sequence* of zero or more *expressions*. The xticks and yticks options also are sequences, which may contain @ and labels as well as expressions. An *expression* is any of the following (operators are shown in decreasing precedence order, with the precedence, if relevant, at right):

| | |
|---|---|
| *number* | |
| x (within xloop only) | |
| y (within yloop only) | |
| z (within zloop only) | |
| pi | |
| e | |
| { *expression* } | |
| sqrt *expression* | 40 |
| abs *expression* | 40 |
| ceiling *expression* | 40 |
| floor *expression* | 40 |
| truncate *expression* | 40 |
| round *expression* | 40 |
| cos *expression* | 40 |
| sin *expression* | 40 |
| *expression* atan *expression* | 39 |
| *expression* exp *expression* | 38 |
| *expression* log *expression* | 37 |
| *expression* rand *expression* | 36 |
| *expression* * *expression* | 35 |
| *expression* / *expression* | 34 |
| *expression* idiv *expression* | 34 |
| *expression* mod *expression* | 34 |
| *expression* − *expression* | 33 |
| − *expression* | 33 |
| *expression* + *expression* | 32 |
| + *expression* | 32 |
| if cond { *boolean* } then { *expression* } else { *expression* } | |

A − immediately followed by a digit or decimal point is always taken to be a minus sign, never a subtraction. The left parameter of exp and log is the base of the exponentiation and logarithm respectively; idiv is integer division; and rand returns a uniform random integer lying between its two parameters (inclusive). Now a *sequence* is zero or more of the following:

| | |
|---|---|
| @ | (within xticks and yticks only) |
| (*label*) | (within xticks and yticks only) |
| *expression* | |
| xloop from { *expression* } to { *expression* } by { *expression* } do { *sequence* } | |
| yloop from { *expression* } to { *expression* } by { *expression* } do { *sequence* } | |
| zloop from { *expression* } to { *expression* } by { *expression* } do { *sequence* } | |
| if cond { *boolean* } then { *sequence* } else { *sequence* } | |

The by part of the loop symbols is optional with default value 1; the else part of if is optional with default value equal to the empty sequence. A *boolean* is any one of the following things, again shown in decreasing precedence order, with the precedence at right:

| | |
|---|---|
| true | |
| false | |
| { *boolean* } | |
| *expression* = *expression* | 30 |
| *expression* != *expression* | 30 |
| *expression* < *expression* | 30 |
| *expression* <= *expression* | 30 |
| *expression* > *expression* | 30 |
| *expression* >= *expression* | 30 |
| not *boolean* | 25 |
| *boolean* and *boolean* | 24 |
| *boolean* xor *boolean* | 23 |
| *boolean* or *boolean* | 22 |
| if cond { *boolean* } then { *boolean* } else { *boolean* } | |

# Chapter 11. C and C++ Programs

This chapter describes how to typeset C and C++ program text using the @CP symbol in conjunction with the c2lout filter. The @CP symbol looks after printing keywords in bold, variables in italic, and so on, depending on a style you choose. It does not lay out programs in the sense of choosing indenting, it preserves the layout you give to the program. From now on, 'C' means 'C or C++' wherever it occurs.

It is possible to simply print out one or more C files; we call this *stand-alone mode*. Alternatively, the C program text may be printed as part of a larger Lout document; we call this *embedded mode*.

## 11.1. Stand-alone mode

Printing of C files in stand-alone mode is accomplished by the following Unix pipeline:

```
c2lout options C-files | lout -s > out.ps
```

As usual with Lout, the output will be a PostScript file. Each input file will begin on a new page of the output, starting with its name in bold type. The options provide control over the final appearance, as follows:

-p*style* Select a printing style. Your choices are -pfixed, -pvarying, and -psymbol, with the default being -pfixed. Consult Section 11.2 for examples of these styles.

-n Do not print file names.

-f*font* Select a Lout font family. The default is -fCourier for -pfixed, and -fTimes for -pvarying and -psymbol.

-v*vsize* Select an inter-line spacing size in Lout units. The default is -v1.1fx meaning 1.1 times the font size measured from baseline to baseline.

There are also -t and -T options for dealing with tab characters (Section 11.5).

## 11.2. Embedded mode

When the C program texts are to be embedded in a larger Lout document, the procedure is somewhat different. For each setup file (doc, report, book, etc.), there is a corresponding alternative setup file whose name begins with c (cdoc, creport, cbook, etc.). So your document might look like this:

```
@SysInclude { cdoc }
@Doc @Text @Begin
...
@End @Text
```

These alternative setup files are identical to the original ones except that they include the wherewithal to do C program formatting.

The C parts of the document are enclosed in @CP { ... } like this:

```
@IndentedDisplay @CP {
#include <stdio.h>

treeprint(p)        /* print tree p recursively */
struct tnode *p;
{
	if (p != NULL) {
		treeprint(p->left);
		printf("%4d %s\n", p->count, p->word);
		treeprint(p->right);
	}
}
}
```

Although C programs violate the rules of legal Lout input in many ways, these rules are suspended by the @CP symbol, allowing the C text to be incorporated with absolutely no modifications. The result is

```
#include <stdio.h>

treeprint(p)        /* print tree p recursively */
struct tnode *p;
{
	if (p != NULL) {
		treeprint(p->left);
		printf("%4d %s\n", p->count, p->word);
		treeprint(p->right);
	}
}
```

We have chosen to use the @IndentedDisplay symbol from Section 2.1 to obtain an indented display, but in fact @CP may appear anywhere at all. When including a C text within a paragraph, use @OneCol @CP { ... } to prevent it being broken across two lines, if desired.

In cases where the C text has unbalanced braces, it is necessary to use the alternative form @CP @Begin ... @End @CP so that Lout does not confuse C braces with Lout braces.

The @CP symbol has a style option for changing the printing style. The default value of style is fixed, which produces the style shown above. To obtain a varying-width font style, use style { varying } like this:

```
@CP
   style { varying }
{
#include <stdio.h>

treeprint(p)      /* print tree p recursively */
struct tnode *p;
{
      if (p != NULL) {
            treeprint(p->left);
            printf("%4d %s\n", p->count, p->word);
            treeprint(p->right);
      }
}
}
```

The result in this case will be

```
#include <stdio.h>

treeprint(p)      /* print tree p recursively */
struct tnode *p;
{
      if (p != NULL) {
            treeprint(p->left);
            printf("%4d %s\n", p->count, p->word);
            treeprint(p->right);
      }
}
```

There is also a third style called style { symbol } which is similar to varying except that it uses characters from the Adobe Symbol font to produce a more mathematical-looking result:

```
#include <stdio.h>

treeprint(p)      /* print tree p recursively */
struct tnode *p;
{
      if (p ≠ NULL) {
            treeprint(p→left);
            printf("%4d %s\n", p→count, p→word);
            treeprint(p→right);
      }
}
```

The @CP symbol has additional options which allow a finer control over the style. Here they all are, with their default values:

```
@CP
   style { fixed }
   font { Courier }
   strings { Base }
   identifiers { Base }
   comments { Base }
   keywords { Base }
   numbers { Base }
   operators { Base }
   size { -1.0p }
   line { 1.0vx }
   tabin { 8 }
   tabout { 8s }
{
   ...
}
```

We are already familiar with style. After that comes font, which determines the font family to use, followed by six options giving the particular faces within that family in which to print C strings, identifiers, comments, keywords, numbers, and operators. Base means the basic face; other commonly available choices are Slope and Bold. These options may all be set to different faces if desired. The default values shown are correct for style { fixed } only; the other styles have other defaults (Section 11.3).

The size option is the font size to use, and line is the inter-line spacing. The default values specify that size is to be one point smaller than in the surrounding document; this was done to compensate for Courier's relatively large appearance compared to other fonts of the same nominal size. Again, these defaults are different for different values of style.

The tabin and tabout options are the subject of Section 11.5.

## 11.3. Changing the default values

We have just seen that the @CP symbol has many options for changing the appearance of the C text. However, most people would not want to have a different style for every C text in their document; they want to define the style once at the start, and have all their C texts come out in that style without laboriously setting options on every @CP symbol. This is done by copying the setup file and changing it.

For general information about how to make your own setup file, consult Section 4.1. The options that determine the default values are in the @CPrint @Use clause near the end of the setup file:

```
@Use { @CPrint
    # style                  { fixed      }

    # fixedfont              { Courier    }
    # fixedstrings           { Base       }
    # fixedidentifiers       { Base       }
    # fixedcomments          { Base       }
    # fixedkeywords          { Base       }
    # fixednumbers           { Base       }
    # fixedoperators         { Base       }
    # fixedsize              { -1.0p      }
    # fixedline              { 1.0vx      }
    # fixedtabin             { 8          }
    # fixedtabout            { 8s         }

    # varyingfont            {            }
    # varyingstrings         { Slope      }
    # varyingidentifiers     { Slope      }
    # varyingcomments        { Base       }
    # varyingkeywords        { Bold       }
    # varyingnumbers         { Base       }
    # varyingoperators       { Base       }
    # varyingsize            { 1.0f       }
    # varyingline            { 1.0vx      }
    # varyingtabin           { 8          }
    # varyingtabout          { 3f         }

    # symbolfont             {            }
    # symbolstrings          { Slope      }
    # symbolidentifiers      { Slope      }
    # symbolcomments         { Base       }
    # symbolkeywords         { Bold       }
    # symbolnumbers          { Base       }
    # symboloperators        { Base       }
    # symbolsize             { 1.0f       }
    # symbolline             { 1.0vx      }
    # symboltabin            { 8          }
    # symboltabout           { 3f         }
}
```

These show the default font families, font faces, font sizes, line spacings, and tab settings in force for the three styles, and also that the default style is fixed. Notice that the font family name for fixed style is Courier, but for the other styles is empty. This causes the fixed style to always switch to Courier, and the other styles to use the same font family as in the surrounding document.

To change a default value, delete the preceding # and change the part between braces. For example, suppose you are happy with fixed except that you want bold keywords. Then one line needs to be changed, to

```
fixedkeywords { Bold }
```

Or suppose you like varying as it stands, but would like it to be the default style rather than fixed. Again, only one line needs to be changed, to style { varying }.

### 11.4. Lout inside C comments

It is possible to embed Lout text inside C and C++ comments, by starting off the comment with an @ character. The entire comment after the @ character should be Lout text. For example, to force Lout to start a new page at some point within a C program, place

```
/*@ @NP */
```

at that point. Or you could make a heading like this:

```
/*@ @Display @Heading { treeprint() } */
```

Other possible uses for this feature include index entries and margin notes. Incredible as it may seem, you can even write

```
/*@ @CD @Heading { Function @CP { treeprint() } } */
```

with a @CP symbol and some C code inside the Lout code inside the C code. You probably can't go further, however, since that would require a C comment inside a C comment.

### 11.5. Tab characters

Tab characters provide a convenient way to indent and align parts of C programs. With care, this alignment can be preserved in the final print even with varying-width fonts.

The distance between two tab stops in the input file is by default taken to be 8 characters, which is standard for Unix. This can be changed with the tabin option. For example,

```
@CP tabin { 4 }
```

informs Lout that tab stops occur every 4 characters in the input file.

The distance between two tab stops in the output file (on the printed page) is quite a different thing, and it is determined by the value of the tabout option, which must be a Lout length. For example,

```
@CP tabout { 0.5i }
```

requests that tab stops be placed at half-inch intervals. In other words, a distance of one tab stop in the input will be equivalent to a distance of half an inch in the output. For example,

```
@CP style { varying } tabout { 3f }
```

might produce the following, where tab characters in the input file have been used for indenting and also to align the comments:

```
struct tnode {                    /* the basic node */
        char *word;               /* points to the text */
        int count;                /* number of occurrences */
        struct tnode *left;       /* left child */
        struct tnode *right;      /* right child */
};
```

The value 3f means three times the current font size, and it is the default value of tabout for the varying and symbol styles (Section 11.3). In a 12 point font this is 36 points, or half an inch.

If tabout is made too small, there is a danger that the alignment might fail. For example,

```
@CP style { varying } tabout { 0.2i }
```

produces

```
struct tnode { /* the basic node */
    char *word; /* points to the text */
    int count; /* number of occurrences */
    struct tnode *left; /* left child */
    struct tnode *right; /* right child */
};
```

given the same C text as the previous example. The problem here is that we are asking for `/*` to appear four tab stops or 0.8 inches from the left edge, and yet the material to its left on the line is wider than this. This causes `/*` to be shifted further to the right than expected, and the alignment is lost. The only solution is to increase tabout.

In stand-alone mode there are -t and -T options equivalent to tabin and tabout respectively. For example, -T0.5i produces a half-inch tab width.

### 11.6. Eiffel program printing

There is an @Eiffel symbol for typesetting Eiffel programs in conjuction with a filter called eif2lout. Apart from the change of language, everything is identical to C printing. The file and symbol names are different, of course:

| | |
|---|---|
| cdoc, creport, etc. | edoc, ereport, etc. |
| c2lout | eif2lout |
| @CP | @Eiffel |
| @CPrint | @EiffelPrint |

but everything works in an exactly analogous way: you place

```
@SysInclude { edoc }
```

at the top of your document, enclose Eiffel program texts in @Eiffel { ... }, and so on. The default style has been changed to varying, so as to conform to the style guidelines in the standard Eiffel reference [8]. Some care has gone into making this conformance strict; in particular, if you enclose identifiers within comments in ‘ and ’, as the style guidelines say you should, they will come out in italics.

The files needed for Eiffel printing are distributed separately from Basser Lout. You can get them from the author’s ftp directory (see the preface of this guide).

# Chapter 12. Pascal Programs

There is a @Pas symbol for printing Pascal programs [1]. No attempt is made to follow any particular printing standard; the design simply reflects this author's taste. To use @Pas, place @SysInclude { pas } at the start of your document in the usual way. A Pascal program or program fragment is entered like this:

```
@ID @Pas {
procedure PriDelete(x: PriEntry; var Q: PriorityQueue);
   var i: integer;
begin
   with Q^ do begin
      size := size - 1;
      if x^.back <= size then
      begin
         i := x^.back;
         A[i] := A[size + 1];
         A[i]^.back := i;
         PriAddRoot(i, Q);
         PriAddLeaf(i, Q)
      end
   end
end;
}
```

This produces

**procedure** *PriDelete*(*x*: *PriEntry*; **var** *Q*: *PriorityQueue*);
   **var** *i*: *integer*;
**begin**
   **with** *Q*↑ **do begin**
      *size* := *size* – 1;
      **if** *x*↑.*back* ≤ *size* **then**
      **begin**
         *i* := *x*↑.*back*;
         *A*[*i*] := *A*[*size* + 1];
         *A*[*i*]↑.*back* := *i*;
         *PriAddRoot*(*i*, *Q*);
         *PriAddLeaf*(*i*, *Q*)
      **end**
   **end**
**end**;

Blank lines, line breaks, indents and spaces in the input are respected, with a tab being considered equal to eight spaces. @Pas can also be used within a paragraph to produce a fragment like *A*[*i*.. *j*]. Use @OneCol @Pas { ... } to prevent the result from breaking over two lines.

@Pas does not attempt to rearrange the program in any way. Each item is simply printed according to the following plan:

| | | | |
|---|---|---|---|
| and | **and** | 0 | 0 |
| array | **array** | 1 | 1 |
| begin | **begin** | 2 | 2 |
| case | **case** | 3 | 3 |
| const | **const** | 4 | 4 |
| div | **div** | 5 | 5 |
| do | **do** | 6 | 6 |
| downto | **downto** | 7 | 7 |
| else | **else** | 8 | 8 |
| end | **end** | 9 | 9 |
| file | **file** | . | . |
| for | **for** | , | , |
| forward | **forward** | : | : |
| function | **function** | ; | ; |
| goto | **goto** | ’ | ’ |
| if | **if** | ‘ | ‘ |
| in | **in** | + | + |
| label | **label** | - | − |
| mod | **mod** | * | ∗ |
| nil | **nil** | / | ∕ |
| not | **not** | ( | ( |
| of | **of** | ) | ) |
| or | **or** | [ | [ |
| otherwise | **otherwise** | ] | ] |
| packed | **packed** | ^ | ↑ |
| procedure | **procedure** | .. | .. |
| program | **program** | = | = |
| record | **record** | < | < |
| repeat | **repeat** | > | > |
| set | **set** | <> | ≠ |
| then | **then** | <= | ≤ |
| to | **to** | >= | ≥ |
| type | **type** | := | := |
| until | **until** | | |
| var | **var** | | |
| while | **while** | | |
| with | **with** | | |

Anything not mentioned here will appear in italic font.

Unlike the @CP symbol from the previous chapter, the @Pas symbol is a quick-and-dirty production which does not offer you any options, or indeed attempt to solve every problem of Pascal formatting. In particular, Pascal strings need attention before formatting by @Pas. Their interiors are best enclosed in double quotes to prevent the above transformations from occurring inside them. Any \ or " characters inside strings will need to be replaced by \\ and \" respectively, and the opening quote should be replaced by ‘.

Similar remarks apply to Pascal comments; don't forget that { and } must be enclosed in double quotes. Alternatively, a @Com symbol can be placed in front of a comment enclosed in braces. It will add literal braces:

@Com { A Pascal comment }

has result

{ *A Pascal comment* }

It may still be necessary to enclose the interior in double quotes.

# Appendix A. Lout Quick Reference Guide

### 1. Running Lout

lout filename > postscript.ps

### 2. Ordinary documents (simple form)

```
@SysInclude { doc }
@Doc @Text @Begin
...
@End @Text
```

### 3. Ordinary documents (full form)

```
@SysInclude { doc }
@Document
   @InitialFont { Times Base 12p }
   @InitialBreak { adjust 1.2fx hyphen }
   @InitialLanguage { English }
   @PageHeaders { Simple }
   @FirstPageNumber { 1 }
   @ColumnNumber { 1 }
//
@Text @Begin
...
@BeginSections
@Section ... @End @Section
@EndSections
@End @Text
```

### 4. Technical reports

```
@SysInclude { report }
@Report
   @Title { ... }
   @Author { ... }
   @Institution { ... }
   @DateLine { No }
   @CoverSheet { Yes }
   @InitialFont { Times Base 12p }
   @InitialBreak { adjust 1.2fx hyphen }
   @InitialLanguage { English }
   @PageHeaders { Simple }
   @FirstPageNumber { 1 }
   @ColumnNumber { 1 }
//
@Abstract ... @End @Abstract
@Section ... @End @Section
@Appendix ... @End @Appendix
```

### 5. Large-scale structure symbols

```
@Section
   @Title { ... }
   @RunningTitle { ... }
   @Tag { ... }
@Begin
@PP
...
@End @Section
```

@Section / @SubSection / @SubSubSection
@Appendix / @SubAppendix / @SubSubAppendix
@BeginSubSections ... @EndSubSections if inner.

### 6. Cross references

| | |
|---|---|
| @Tag { foo } | @PageOf foo |
| @PageMark foo | @NumberOf foo |

### 7. Font changes

| | |
|---|---|
| @B { bold font } | @I { italic font } |
| @BI { bold-italic font } | @R { Roman font } |
| @S { small-caps font} | @F { fixed-width font } |

{ family face size } @Font { ... }

Times Helvetica Courier ...
Base Slope Bold BoldSlope ...
10p 12p +2p -2p 2.0f ...

### 8. Paragraph breaking styles

{ breakstyle linesep hyphen } @Break { ... }

adjust ragged lines clines ...
1.2fx 2vx 0.9vx ...
hyphen nohyphen

### 9. New paragraph and new page

| | |
|---|---|
| @PP | Plain paragraph |
| @LP | Left paragraph |
| @LLP | New line |
| @DP | Display paragraph |
| @NP | New page |
| @CNP | Conditional new page |

#### 10. Displays and headings

@CD @Heading { A centred heading }
@ID { An indented display }

| | |
|---|---|
| @D | @Display |
| @LD | @LeftDisplay |
| @ID | @IndentedDisplay |
| @QD | @QuotedDisplay |
| @CD | @CentredDisplay |
| | @CenteredDisplay |
| | @RightDisplay |

#### 11. Lists

@List
@ListItem { A list item }
@ListItem { Another list item }
@EndList

| | |
|---|---|
| @L | @List |
| @LL | @LeftList |
| @IL | @IndentedList |
| @QL | @QuotedList |
| @CL | @CentredList |
| | @CenteredList |
| @NL | @NumberedList |
| @RL | @RomanList |
| @UCRL | @UCRomanList |
| @AL | @AlphaList |
| @UCAL | @UCAlphaList |
| @PNL | @ParenNumberedList |
| @PRL | @ParenRomanList |
| @PUCRL | @ParenUCRomanList |
| @PAL | @ParenAlphaList |
| @PUCAL | @ParenUCAlphaList |
| @BL | @BulletList |
| @SL | @StarList |
| @DL | @DashList |

@TaggedList
@TagItem { label } { A list item }
@TagItem { label } { Another list item }
@EndList

| | |
|---|---|
| @TL | @TaggedList |
| @WTL | @WideTaggedList |
| @VWTL | @VeryWideTaggedList |

#### 12. Footnotes, endnotes, margin notes

| | |
|---|---|
| @FootNote { ... } | @EndNote { ... } |
| @LeftNote { ... } | @RightNote { ... } |
| @OuterNote { ... } | @InnerNote { ... } |

#### 13. Floating figures and tables

| | |
|---|---|
| @Figure | @Table |
| @Caption { ... } | @Caption { ... } |
| @Tag { ... } | @Tag { ... } |
| @Begin | @Begin |
| ... | ... |
| @End @Figure | @End @Table |

#### 14. Tables

```
@SysInclude { tab }
@SysInclude { doc }
...
@Tab
   @Fmta { @Col A ! @Col B }
   vmargin { 0.5vx }
{
@Rowa
   A { ... }
   B { ... }
@Rowa
   ...
}
```

#### 15. Equations

```
@SysInclude { eq }
@SysInclude { doc }
...
@Eq { sum from i=0 to n { r sup i over sqrt pi } }
```

#### 16. Basic graphics

grey @Colour { ... }
gray @Color { ... }
@Box { ... }
@CurveBox { ... }
@ShadowBox { ... }
60d @Rotate { ... }
0.71 @Scale { ... }
@QuotedDisplay @Scale { ... }
@IncludeGraphic filename.eps

#### 17. Miscellaneous

@Underline { Underline }
@Date
@Time
German @Language { ... }
# comment to end of line
"#&/@^{}|"~ (enclose these characters in quotes)

# Appendix B. Bypass Symbols

The ‘bypass’ symbols described in this appendix are intended to be used only in Lout which is generated by computer programs. Their purpose is to bypass the Lout cross reference database, and so reduce the number of passes needed to finalise a document. These symbols should not be used by people, because that would only lead back to the consistency problems that prompted the introduction of cross references in the first place.

To produce a bypass table of contents, set the @MakeContents setup file option to Bypass and use @BypassContentsEntry symbols at the outermost level just before the introduction or first chapter:

```
@BypassContentsEntry
   indent { 0f }          the indent, e.g. 0f, 2f, 4f ...
   number {}              the section (etc.) number e.g. 5.2
   title {}               the section (etc.) title e.g. Azaleas
   pagenum {}             the page number e.g. @PageOf azaleas
```

For major entries such as chapters, use @BypassMajorContentsEntry with the same options. This increases the vertical spacing and uses bold font. When @MakeContents is Bypass, no contents entries will be produced automatically.

To bypass Lout’s automatic numbering of large-scale structure symbols, use the @BypassNumber option:

```
@Section
   @Title { Azaleas }
   @Tag { azaleas }
   @BypassNumber { 5.2 }
...
```

Give the full ‘path number’ (5.2 or whatever) of the symbol. There is a @BypassNumber option for every symbol that has a @Title option and is usually assigned a number automatically by Lout, plus @Figure and @Table. No changes to the setup file are required in order to use @BypassNumber, and it is permitted for some large-scale structure symbols to have this option and others not.

To produce a bypass reference list, set the @MakeReferences setup file option to Bypass and place reference entries at the end of the document, after the last chapter or other large-scale structure symbol but before any bypass index entries (see below), like this:

```
@BypassReference
   label { [Kin94a] }
   value { @RefPrint kingston1995lout.expert }
```

The two options are objects which become the label and value of the reference entry. The value option can be any object, including an explicit reference; but @RefPrint does not introduce any cross-referencing delay if the @Reference symbols lie in a separate database file. No sorting or removal of duplicate entries will be done by Lout. When @MakeReferences is Bypass, @Cite and related symbols are ignored.

There is also @BypassChapReference with the same options for producing bypass chapter reference lists; these symbols should be placed at the outer level immediately after the preface, introduction, chapter or appendix that they refer to.

To produce bypass indexes, set the @MakeIndex setup file option to Bypass and use the @BypassRawIndex symbol repeatedly at the very end of the document, enclosed in @BypassBeginIndex and @BypassEndIndex symbols:

```
@BypassBeginIndex
@BypassRawIndex  indent { 0f }  { Azaleas, @PageOf azaleas }
...
@BypassEndIndex
```

The indent option gives the indent (0f, 1f, 2f, etc.), and the right parameter is as for @RawIndex. No @PageMark operations, sorting, merging, or attachment of page numbers will be done by Lout. When @MakeIndex is Bypass, @Index and related symbols are ignored. At present, bypass index symbols work only in books, not with ordinary documents or technical reports. There are corresponding symbols for creating bypass indexes A and B.

# References

[1] K. Jensen and N. Wirth. *Pascal User Manual and Report*. Springer-Verlag, 1975.

[2] Brian W. Kernighan and Lorinda L. Cherry. A system for typesetting mathematics. *Communications of the ACM* **18**, 182–193 (1975).

[3] Jeffrey H. Kingston. The design and implementation of the Lout document formatting language. *Software—Practice and Experience* **23**, 1001–1041 (1993).

[4] Jeffrey H. Kingston. A Practical Introduction to the Lout Document Formatting System. Overhead transparencies (1994), Basser Department of Computer Science, University of Sydney.

[5] Jeffrey H. Kingston. *An Expert's Guide to the Lout Document Formatting System (Version 3).* Basser Department of Computer Science, University of Sydney, 1995.

[6] Donald E. Knuth. *The TEXBook*. Addison-Wesley, 1984.

[7] Leslie Lamport. *LATEX User's Guide and Reference Manual*. Addison-Wesley, 1986.

[8] Bertrand Meyer. *Eiffel: The Language*. Prentice-Hall, 1992.

[9] Joseph F. Ossanna. Nroff/Troff User's Manual. Tech. Rep. 54 (1976), Bell Laboratories, Murray Hill, NJ 07974.

[10] Brian K. Reid. A High-Level Approach to Computer Document Production. In *Proceedings of the 7th Symposium on the Principles of Programming Languages (POPL), Las Vegas NV*, pages 24–31, 1980.

[11] William Strunk and E. B. White. *The Elements of Style.* Macmillan. Third Edition 1979.

[12] Mary-Claire van Leunen. *A Handbook for Scholars.* Oxford. Revised Edition, 1992.

# Index