Lollipop Unwrapped

Victor Eijkhout

A manual for the Lollipop TEX format,
last updated 18 October 1992

Chapter 1

Contents

1.1 Regular sections

1.2 Implementer's notes

1.3 All of the options

(You know, this section and the next look much better if you sort the `manual.oix` and `manual.cix` files before you format the document the last time. Do put a lines

```
\Writeopindex:no
\Writecsindex:on
```

somewhere in the top of the `manual.tex` file in order to prevent overwriting of this file after you've sorted it.)

1.4 All of the commands

Chapter 2

Preliminaries

2.1 What is Lollipop?

Lollipop is 'TEX made easy'. Lollipop is a macro package that functions as a toolbox for writing TEX macros. It was my intention to make macro writing so easy that implementing a fully new layout in TEX would become a matter of less than an hour for an average document, and that it would be a task that could be accomplished by someone with only a very basic training in TEX programming.

Lollipop is an attempt to make structured text formatting available for environments where previously only wysiwyg packages could be used because adapting the layout is so much more easy with them than with traditional TEX macro packages.

2.2 But is is compatible?

Lollipop, like LATEX, is not compatible with plain TEX. I don't consider this a real problem. Lollipop is meant to be used for different applications that those for which plain TEX, or LATEX for that matter, are used. The typical Lollipop user has only herself to be concerned with.

2.3 How to Use Lollipop

The following files comprise the Lollipop format:

```
        fonts.tex       lists.tex
        define.tex      heading.tex     lollipop.tex
    text.tex
        document.tex    lolplain.tex    output.tex
    tools.tex
```

and it is assumed that you have a file called `hyphen.tex` with hyphenation patters for the language you are using.

Run IniTEX on `lollipop.tex`. This gives, depending on your operating system, output that looks something like this:

```
> initex lollipop
This is TeX, C Version 3.0 (INITEX)
(lollipop.tex (lolplain.tex (hyphen.tex)) (tools.tex)
(define.tex) (fonts.tex)
(text.tex) (document.tex) (heading.tex) (output.tex)
(lists.tex))
```

```
Beginning to dump on file lollipop.fmt
 (format=lollipop 92.5.30)
3102 strings of total length 41719
27016 memory locations dumped; current usage is
142&26543
2076 multiletter control sequences
\font\nullfont=nullfont
...
2706 words of font info for 12 preloaded fonts
17 hyphenation exceptions
Hyphenation trie of length 6075 has 181 ops out of
500
  181 for language 0
No pages of output.
Transcript written on lollipop.log.
```

As a result of this, you get a file `lollipop.fmt` that contains the Lollipop format. This has to be loaded in TEX everytime you want to format a file. To process a file, say `test.tex`, with Lollipop you then type:

```
> tex test &lollipop
```

or something like that, depending on local conditions.

2.4 Lollipop Files

Any Lollipop document has to have a `\Start` and `\Stop` command. Before the `\Start` there can be style definition commands, but no text.

Implementor's Note

Before the `\Start` command, `\everypar` contains an error message. The start command installs the default value for `\everypar`.

2.5 Processing a Lollipop file

Files that you make to be processed with Lollipop contain of course the input text, but they also have to contain the design macros that determine the layout. There are two possibilities for these design macros:

- You can simply put them in the same file, either in the beginning or wherever they are first needed, or
- You can put the layout definition in a separate file and make a new format out of that. For instance, if the layout definition of a book is complicated, processing it each time will be slow, so you might put it in a file `bookstyle.tex`. IniTEX can load this definition on top of the Lollipop format:

```
> tex bookstyle &lollipop
*\dump
```

This gives you a format `bookstyle.fmt`, which you can use by typing

```
> tex book &bookstyle
```

Also in the case where the style designer and the style user are on different levels of TEXpertise it may be wise to hide the style definition from the user by making only a format available.

Implementor's Note

The file `lollipop.tex` contains a `\dump` command. This is not the way plain TEX and LATEX operate, but I like this better.

2.6 The errors of Lollipop/ known bugs

Since Lollipop is an order of magnitude more powerful (and hence complicated) than formats such as LATEX, its error messages can also be an order of magnitude more cryptic. Fortunately, Lollipop is also quite a bit better than existing formats at catching potential errors. Typos in a style definition will usually lead to warning messages, and also during run time Lollipop is able to track down ommisions.

In addition, you can switch on various trace modes to get more detailed information about Lollipop's thought processes. See chapter 12.

These are the known bugs in Lollipop at the moment.

1 Local references have been insufficiently tested, and the code definitely is buggy.
2 The 'firstpage' test in the page grids does not work.
3 Titles get written to the aux file with double spaces. This shouldn't cause any problem, but it has to be fixed.
4 Rules in page grids get white space around them.
5 There probably is a reason for all the underfull boxes in formatting the output chapter of this manual.
6 External items shouldn't declare `\FooTitle` or `\FooCounter`.

2.7 About this manual

This manual is an unashamed hodge-podge. Apart from the regular sections (the ones you are supposed to read) there are implementor's sections, which you read only at your peril. They document the internals of Lollipop in all their unscrutability.

This manual consists of a main file `manual.tex`, and the following input files:

```
titlepag.tex prelim.tex struct.tex head.tex list.tex
```

```
out.tex extern.tex opt.tex comm.tex trace.tex
appendix.tex
```

and the style definition file `mandefs.tex`.

In addition, you need `comment.tex` which is used to format this manual, but it is not really a part of Lollipop.

If you format this manual (which you'll have to do three times to get the page numbering and the table of contents straight) you'll notice something strange. The file `example.tex` is read in many, many times. This is because this manual formats its examples along the way, first writing them out, and then reading them in to show both their code and their output. This way it is guaranteed that the examples in the manual will always work.

As a result of formatting this manual you will wind up with, apart from the usual `dvi` file, with `manual` files with extensions `aux`, `toc`, and `imp`; `oix` and `cix` for indexes of options and commands, and `tct`, filetix which are for the examples.

This manual needs quite some resources: here's what TEX told me it needed.

```
Here is how much of TeX's memory you used:
 1259 strings out of 4808
 14894 string characters out of 21967
 62606 words of memory out of 65536
 3042 multiletter control sequences out of 10000
 19 hyphenation exceptions out of 307
 22i,4n,24p,225b,502s stack positions out of
 200i,60n,60p,5000b,2000s
```

This should not need a 'Big TEX', but it comes close.

2.8 The most boring section in this manual

There are a few things about Lollipop that I want to be clear about.

2.8.1 I am going to hurt you and I am not sorry

In the secret handbook for the software industry it says that the final test phase of a product consists of putting it in stores and having innocent suckers pay good money for it. (You guessed it, this is the disclaimer section.) So let me just say that Lollipop is probably good for nothing, at least, I don't claim it is.

2.8.2 Get a Lollipop, give one away

Lollipop is free software. You may copy it for your own purposes, or give away copies. However, you may not ask money for it, other than reasonable expenses such as for copying discs or manuals. If you make changes to Lollipop these should be clearly indicated as such if you give away copies.

The easiest way to get the current copy of Lollipop is to ftp it from `cs.utk.edu` from the directory `/pub/eijkhout/tex` where it is stored as `lollipop.shar.Z`. My apologies for how Unix slanted this is.

2.8.3 The status of Lollipop

Lollipop is still under development. Although I will try not to make any drastic changes in the user interface (this says nothing about the internals!) I really cannot guarantee anything. However, I do listen to complaints and suggestions.

If you have suggestions or complaints about the useability of Lollipop or the implementation, feel free to contact me at `\eijkhout@cs.utk.edu` on the Internet. Or send snail mail to:

```
Victor Eijkhout
Department of Computer Science
University of Tennessee
107 Ayres Hall
Knoxville TN 37996
USA
```

2.8.4 The wish list

Lollipop is not quite perfect. Here's a list of things that I am going to be adding in the near future. If you want to add items to this list, just mail me.

- Raggedbottom should really, really be added. Soon!
- Capitalization and initial capping of titles. If a title appears in mixed case, it should be possible to have it in all uppercase in running heads.
- A better multi-column mode.
- Interface to BibTEX.
- Inserts, in particular footnotes. At the moment floating figures are entirely lacking. (As a matter of fact, the plain TEX macros are availble, but I'm not telling that.)
- A tabular mode. Personally I always felt `\halign` to be more than sufficient, but some people seem to think otherwise.
- Maths constructs. Some things in the `\eqalign` vein would be nice.
- Adaptive list indents. Calculate the maximum needed indentation, write that to the `.aux` file, and read it in next time. Also do this globally, and have the possibility to have this spill over into the `\parindent` and such.
- A 'nomarks' option to prevent wasting two token lists. For the expert designer?

2.8.5 A bit of history

The Lollipop format was begun in late 1989 to typeset my Ph.D. thesis, '*Vectorizable and Parallelizable Preconditioners for the Conjugate Gradient Method*'. At that time I was using TEX on an Atari 1040ST. Loading the style definition for the thesis took about two minutes. Lollipop was heavily augmented in late 1991 to typeset my book '*TEX by Topic*', for which I used Sun 3 and Sun 4 computers. Writing this manual brought Lollipop to its present state. At present I am using Lightning Textures on a Macintosh Powerbook 145.

Chapter 3

The structure of Lollipop

Lollipop provides tools for realizing the style or layout of a document. Some of these tools are macros ready to be used by the end user; they concern for instance selection of fonts. Others, the 'generic constructs', are for the style designer so that she can use them to program the macros for the user.

3.1 Generic Constructs

There are five 'generic constructs': headings, lists, text blocks, page grids, and external items. For each construct type there is a defining command, for instance `\DefineHeading` which is followed a list of 'options', terminated by the word 'Stop'.

Options (possibly with values) have to be separated by a space or a line end; the keyword `Stop` has to be followed by a space or a line end. Options may have zero, one or two values; if there are values, then the first one is separated from the option by a colon, the second is separated from the first by an equals sign.

```
\DefineFoo:Bar optiona optionb optionc:value
    optiond:valuea=valueb optione
    optionf Stop
```

As a result of this definition, a command `\Bar` is created. If the `Foo` construct was a `List` or `TextBlock`, an additional command `\BarStop` is created.

This command can then be used in the ordinary way, for instance after `\DefineHeading:Foo` you can type

```
\Foo The title
```

and after `\DefineList:Foo` you can type

```
\Foo
\item One item
\item And another
\FooStop
```

Options are mostly used to specify how a construct will look. Some options, for instance `title`, indicate material that will appear on the page. Other options are interpreted as commands, for instance `IndentAfter:yes` in the definition of a heading indicates that the first paragrah after such a heading will indent.

In addition to keywords that only exist as options, commands can be used as options. Also, single characters are accepted as options. For instance a definition of a subsection heading can contain:

```
\DefineHeading:SubSection
```

```
          [...]
          SectionCounter . SubSectionCounter
          [...] Stop
```

(Here and later the `[...]` will denote arbitrary omitted text.) This definition contains the commands `\SectionCounter` and `\SubSectionCounter` and the `.` character.

If a number of options appears together in a number of constructs it is convenient to have an abbreviation for them. This can be done as follows. The options that appear together are given a common name

```
      \OptionsMacro:baz=optiona optionb:value optionc
          Stop
```

and this name is then used as

```
          macro:baz
```

in the option list wherever the options are needed. This is for instance a good way of specifying identical white space around all sorts of constructs without duplicating the typing each time.

Implementor's Note

I-*1 Defining Generic Constructs*

The `\Define...` commands are not defined explicitly, instead they are generated by a call such as

```
          \@GenericConstruct{Heading}
```

Full definition:

```
      \def\@GenericConstruct#1{
```

to be used as `\@GenericConstruct{Foo}`;

```
          \append@to@list{@gencons}{\\#1;}
```

book keeping of existing generic constructs;

```
          \csarg\newtoks{#1@defaults}
          \csn #1@defaults\ecs{}
```

default commands to be executed whenever an instance of this construct is defined;

```
          \csarg\def{add@#1@default}##1%
                   {\append@to@list{#1@defaults}{##1}}
```

the command `\add@Foo@default` to add defaults for this construct;

```
          \Install@Noops{#1}
```

possibility to generate error msgs for the use of an option that is not allowed for this type of construct;

```
          \csarg\def{Define#1}:##1 {
```

instances of this construct will be defined by a statement like `\DefineFoo:Bar`;

```
              \def\@name{##1}\def\@class{#1}
              \Tmessage[def]{Defining a #1: ##1}
```

`DefineFoo:Bar` leads to `\@name` begin `Bar`, `\@class` being `Foo`;

```
              \the\generic@defaults
              \csarg\the{#1@defaults}
```

unpack token lists of generic and specific default actions;

```
              \Get@Items}
```

start recursive processing of list of options. This ends the definition of `\DefineFoo`; the definition of `\@GenericConstruct` ends with

```
          \csarg\def{@#1Option}##1%
             {\csarg\def{#1@##1}####1####2}
```

which defines the `\@FooOption` macro; see I-3.

```
          }
```

I-*2 Items Processing*

Processing the list of items is recursive; at the end some concluding actions have to be taken, mostly the actual definition of the construct.

First we have to filter out empty arguments, which can be caused by the use of option macros (3.1, I-3).

```
      \def\Get@Items#1
      {\if\EmptyList{#1}\let\get@next@item\Get@Items
          \else\def\get@next@item{\@Get@Items#1 }\fi
          \get@next@item}
```

Next, check if the argument is `Stop` (defined by `\NewDummy{Stop}`, I-15), in which case you have reached the end of a generic definition, and can start performing the final rites. Otherwise, dissect this option item and go on with the rest of the options.

```
      \def\@Get@Items#1 {\let\get@next@item=\Get@Items
          \csarg\ifx{#1}\Stop
                \the\generic@stop@defaults
                \let\get@next@item=\relax
          \else \Item@or@Macro#1::. \fi \get@next@item}
```

The `::.` concluding `\Item@or@Macro` accomodates one, possibly empty, argument.

I-*3 Options*

Options can either be specific, defined as

```
      \@FooOption{Bar}{ [...] }
```

in which case the option `Bar` can only be used inside a call to `\DefineFoo`, or they can be generic, defined as

```
      \@GenericOption{Bar}{ [...] }
```

For both definitions, the definition text can use up to two parameters. Parameters are given to the options as

```
          optiona:par1 optionb:par1=par2
```

Specific options are defined by the line

```
          \csarg\def{@#1Option}##1%
                {\csarg\def{#1@##1}####1####2}
```

in `\@DefineGenericConstruct`; a call

```
      \@FooOption{Bar}{ ... }
```

expands to

```
      \def\Foo@Bar#1#2{ ... }
```

Generic options are defined by the following command:

```
\def\@GenericOption#1{
    \append@to@list{@GenericOptions}{\\#1;}
    \csarg\def{Option@#1}##1##2}
```

A call

```
\@GenericOption{Bar}{ ... }
```

expands to

```
\def\Option@Bar#1#2{ ... }
```

Chapter 4

Headings

Headings for sections, chapters, and such, are an essential part of any TeX macro package. In Lollipop they are maybe a bit less special: all options for headings are general options, meaning that they also apply to text blocks and lists. There are only two things that distinguish headings:

1 there will be no page break after a heading;
2 there is no closing command for a heading.

4.1 Examples

Headings are defined by `\DefineHeading`. The most obvious element in a heading is the title, marked by the option `title`. The title is anything that follows the heading command, upto the first empty line.

```
\SomeHeading Some title

And some text following it.
```

Implementor's Note

Titles can also be terminated by `\par`, but knowledge of this is not encouraged. See further I-8.

The title has to be included in a line or a textcolumn for proper handling (see also section 10.3.4). For titles that do not exceed one line, the `line` option suffices (section 10.3.2); if a title is possibly more than one line long, the `textcolumn` option has to be used (section 10.3.3.

Example 4.1

```
\DefineHeading:TestHead Style:bold
    line:start TestHeadCounter Spaces:2 title line:stop
    Stop
\TestHead The Title

The text after the heading.
```

1 The Title

The text after the heading.

By default, the text after a heading is indented. Overriding this default behaviour is done with the option `indentafter`.

Example 4.2

```
\AlwaysIndent:no % as a default, don't indent paragraphs
\DefineHeading:TestHead Style:bold
    line:start TestHeadCounter Spaces:2 title line:stop
    indentafter:yes Stop
\TestHead The Title

The text after the heading.\par
The second paragraph after the heading
```

1 The Title

The text after the heading.
The second paragraph after the heading

Usually headings come in a hierarchy, where the counter of one type, for instance a subsection, is reset everytime the counter of a higer level is stepped. In Lollipop, this subordinating of headings is done by declaring one counter to be governed by another (counters are explained in full detail in section 11.1).

Example 4.3

```
\DefineHeading:OneHead Style:bold
    line:start OneHeadCounter Spaces:1 title line:stop
    Stop
\DefineHeading:TwoHead Style:italic
    line:start OneHeadCounter . TwoHeadCounter Spaces:1
        title line:stop Stop
\GoverningCounter:TwoHead=OneHead

\OneHead Level One Heading\par
\TwoHead Level Two Heading\par
Some text.
\TwoHead Level Two again\par
More text.
\OneHead Level One is Stepped\par
\TwoHead Level Two\par
Again text.
```

1 Level One Heading

1.1 Level Two Heading

Some text.

1.2 Level Two again

More text.

2 Level One is Stepped

2.1 Level Two

Again text.

Headings will often wind up in a table of contents. For this, the table of contents will have to be declared:

```
      \DefineExternalFile:contents=toc
```

and its formatting will have to be specified, but also every construct that writes to this file has to be declared as such.

```
      \DefineHeading:TestHead
          [...]
          external:contents title external:stop
          Stop
```

Usually, the title is all that has to be written out (the counter value is written by default), but the possibility exists for writing out other information as well. See section 9.2.

Implementor's Note

I-*4 Page break prevention after titles*

Simple prevention of page breaks is done by

```
      \add@Heading@default{\def\@afterpenalty{\penalty\@M}}
```

The `\@afterpenalty` is used in `\gen@open` (see I-5).

A conditional `\if@headed` is defined, which is only true after headings:

```
      \newif\if@headed
      \add@generic@default{\add@after@command{\@headedno}}
      \add@Heading@default{\add@after@command{\@headedyes}}
```

This is used in `\outer@start@commands` to prevent a page break between a heading and a subsequent generic construct.

```
              \nxp\if@headed\nxp\else
                  \ifforced@break@before\@beforepenalty
                  \else\nxp\ifnum\lastpenalty=\z@
                                \@beforepenalty
                       \nxp\fi
                  \fi
              \nxp\fi
```

If `\if@headed` is true, then no penalty at all is placed, so that the trailing infinite penalty of the heading will dominate page breaking at this point.

Chapter 5

Lists

Lists in Lollipop are defined by `\DefineList`:

```
      \DefineList:Foo [...]
          item:start [...] item:stop
          [...] Stop
```

and the resulting list is used as

```
      \Foo
      \item [ ..text.. ]
      \item [ ... ]
      \FooStop
```

where the closing command can be abbreviated as `\>`.

5.1 Label alignment

In general there is a default position for labels; either aligning with the left or the right side of the margin over which the list is indented. The two ways are indicated with the option `item`:

```
          item:left [...] item:stop
```

and

```
          item:right [...] item:stop
```

respectively. Specifying `item:start` gives the default left aligning position.

Example 5.1

```
\DefineList:enumerate
    item:start itemCounter ) item:stop Stop
\DefineList:enumerateright
    item:right ( itemCounter ) Spaces:1 item:stop Stop
\enumerate\item Some item
\item And another
\enumerateright\item First nested item
\item Next nested item\>
\item And back to the original list.\>
```

1) Some item
2) And another

   (A) First nested item
   (B) Next nested item

3) And back to the original list.

5.2 List indentation

The amount over which the text of a list (excluding the item labels) is indented is controled by a list of indentations. This is explained in section 11.4. The indentation amount is most of the time also equal to the value of the paragraph indentation outside that list.

In the rare case where the indentation of a list has to be controlled explicitly, there is an option `indentation`with one value.

```
\DefineList:SomeList indentation=30pt [...] Stop
```

5.3 Label style

Every list that uses the `itemsign`option is an 'itemize' list, no matter what it's name, and there is a counter in Lollipop that keeps track of how deep you are in itemize lists. Similarly, every list that uses `itemCounter`is an 'enumerate' lists, and these are counted too.

On every next level a new style of item sign or counter is used. For item signs this is in sequence: •, ∘, –, and · for all higher levels. The style of sign can be changed by `\SetItemSign`:

```
\SetItemSign:6=m
```

where the letter indicating the sign is interpreted as: `b` • (bullet), `c` ∘ (circle), `d` ⋄ (diamond), `m` — (em-dash), `n` – (en-dash), `.` ·.

Similarly, the counter style can be set by `\SetItemCounterRepresentation`:

```
\SetItemCounterRepresentation:2=i
```

where the letter representing the style is interpreted as: 1 Arabic, I uppercase roman, i lowercase roman, A uppercase characters, a lowercase characters.

5.4 Label width

The default width for a label is at most the width of the margin over which the list is indented. Using `item:left` or `item:right` will have the label pushed to the left or right side of this margin respectively. Now what if the label material is wider than this margin? Usually you want the label then to expand to the right, and that is indeed what happens, unless you specify `labeloverflow`with value `left`, in which case the right boundary of the label will not budge, and the label will start protruding into the outer margin.

5.5 Description lists

A common type of list is the type where each item label consists of a piece of text. Such a list is called a 'description' list in Lollipop, and it recognized by the occurrence of the option `description`in its definition. A description list can also use the item sign or the item counter, of course.

Using a description list, the description text is everything that follows the command `\item`, up to the end of the line.

Example 5.2

```
\DefineList:DescribeIt
 item:left Style:bold itemCounter . Spaces:1 description
      Spaces:2 item:stop Stop
\DescribeIt\item Do
A deer, a female deer.\item Re
According to mr. Fowler only a legal term.
\item Mimi Lafrenz-Jett
The owner/founder of ETP\>
```

1. Do   A deer, a female deer.
2. Re   According to mr. Fowler only a legal term.
3. Mimi Lafrenz-Jett   The owner/founder of ETP

As you can see, the problem of label overflow can easily occur with description lists. Thus it is a good idea to end the item material with some white space, as in the above example.

Exceptional situation: if you use an empty description text, you should write `\item{}`.

5.6 Suspended lists

Occasionally the is a need to resume an enumerate list, that is, after a piece of text that is not part of the list an enumerate list should start counting from the previous value on. In Lollipop this phenomenon can be realized by never ending the enumerate list, and simply moving the text one indentation level back with `\PopIndentLevel`.

Example 5.3

```
\DefineList:enumerate item:left itemCounter item:stop Stop
\enumerate\item First some item\par
{\PopIndentLevel \Indent:no
This text seems to be outside the list. Don't you believe
it.\par}
\item And another item\>
```

1   First some item
This text seems to be outside the list. Don't you believe it.

2 And another item

Note that the ‘popped’ text has to be in a group (otherwise the subsequent items will also be popped back), and it has to be separated from the preceding and following text by `\par`; the trailing `\par` has to be in the group.

5.7 Item counter manipulation

The item counter can be manipulated explicitly. This is necessary for instance for starting a list at another value than one. What you need to realize here is that the command `\item` starts by incrementing the counter. Furthermore, the only way to access the item counter is through the commands for counters; see section 11.1.

Example 5.4

```
\DefineList:enumerate item:left itemCounter item:stop Stop
\enumerate \SetCounter:item=-1
\item Escape: usually the backslash.
\item Begin Group.\>
```

0 Escape: usually the backslash.
1 Begin Group.

5.8 List titles and list tails

Lists can have titles. The title follows the command that invokes the list, in the usual manner. Material to follow the list can also be specified: anything following the option `text`is considered to be trailing material.

Example 5.5

```
\DefineList:TitledList hrule line:start Style:bold title
line:stop
 item:left Style:italic itemCounter item:stop
 text vwhite:3pt hrule Stop
\TitledList In the last fiscal year, have you:\par
\item Eaten peanuts? \item Walked the dog?
\item Bought a Frank Zappa record?\>
```

In the last fiscal year, have you:
1 Eaten peanuts?
2 Walked the dog?

3 Bought a Frank Zappa record?

In case you wonder what happens with textual material after `item:stop` and before any `text`, well, that is taken to be inserted immediately after each item label.

5.9 Between the items

There are special list options controlling what happens in between items. Lollipop has an option `breakbetween`, analogous to `breakbefore` and `breakafter`; see section 10.5. This item be default has a value of −`50`, implying that breaks in between items should be preferred slightly over breaks in between the lines of an item.

Similarly, there is an option `whitebetween`controlling the amount of white space in between items that is analogous to `whitebefore` and `whiteafter`. Like these two options, it can also be set by the `\Distance` command (section 11.7).

5.10 Indentation in lists

An item can be considered to be consisting of at least one paragraph. That paragraph is never indented. For the behaviour of any next paragraph within the same item, the option `indentinside`can be used. This option has values yes/no. In case paragraphs inside an item indent, the indentation amount is level-controlled; see section 11.4.

Chapter 6

Text Blocks

The 'text block' is a way of treating a moderate sized chunk of text in a different way from the surrounding text. Text blocks are created by `\DefineTextBlock`. Here is a small example.

Example 6.1

```
\DefineTextBlock:Quote
 PushIndentLevel PointSize:9 SetFont text Stop
\Indent:no In some context it has been written that
\Quote No man is an island.\QuoteStop
In another:
\Quote Run don't walk to the nearest island.\>
Sometimes one would wish women weren't so logical.
```

In some context it has been written that

> No man is an island.

In another:

> Run don't walk to the nearest island.

Sometimes one would wish women weren't so logical.

Note that the text block has an explicit closing command, consisting of the name of the block followed by `Stop`, and that implicit closing by `\>` is possible.

6.1 The `text` *option*

Text blocks have only one specific option: `text`. This option is used to separate material heading the block from material trailing the block. Example:

Example 6.2

```
\DefineTextBlock:DisplayEq
 whitebefore:abovedisplayskip whiteafter:belowdisplayskip
 line:start white:parindent $ displaystyle text $ line:stop
Stop
The formula
\DisplayEq e^{\pi i}+1=0\>
contains nature's five most interesting constants.
```

The formula

$$e^{\pi i}+1=0$$

contains nature's five most interesting constants.

Here one dollar comes before the text, and one after, so the first is inserted by `\DisplayEq` and the second by the corresponding closing command.

6.2 More examples

A text block can encompass more than one paragraph, so the options `indentinside` and `indentfirst` are particularly useful here.

Example 6.3

```
\AlwaysIndent:no
\DefineTextBlock:TestBlock PushIndentLevel
 indentafter:yes indentfirst:no indentinside:yes
 text unskip hfill $ bullet $ par Stop
One paragraph.\par The next paragraph
\TestBlock Inside the block one paragraph.\par
Inside the block the next paragraph.\>
Outside the following paragraph.\par And the last
paragraph.
```

One paragraph.
The next paragraph

Inside the block one paragraph.
Inside the block the next paragraph. ●

Outside the following paragraph.
And the last paragraph.

Implementor's Note

I-*5 The environment in generic constructs*

The text block is just about the purest use of the Lollipop environment mechanism. Here is how a text block is defined:

```
\def\@DefineTextBlock{
    \csarg\edef{\@name}{\@gen@open
        \the\before@coms
        }
    \@DefineStopCommand{\the\after@coms \@gen@close}
    }
```

The `\before@coms` and `\after@coms` are two token lists with the heading and trailing commands.

It is important to note that the definition of the control sequence of the block is defined by `\edef`. This first of all unwraps the token lists, but it also has an important effect on `\@gen@open/close`. These control sequences contain a lot of conditionals which, in combination with the `\edef` give a really dynamic definition of generic constructs in Lollipop.

First of all, the opening and closing commands induce a group so that various quantities can be set locally.

```
\def\@gen@open{\outer@start@commands
    \begingroup \inner@start@commands}
\def\@gen@close{\inner@end@commands
    \endgroup \outer@end@commands}
```

The outer start commands concern global actions such as backspacing previous skips, incrementing counters and setting references.

```
\def\outer@start@commands{%
    \iftext@construct
        \ifleft@embedded@construct
              \nxp\bsp@hack
        \else \nxp\leavehmode
\nxp\bvwit{\the\@whitebefore}\fi
```

The 'embedded construct' tests are only true if the construct can be embedded in a paragraph. A rare occurence most of the time.

```
        % backspace previous white space while it's
visible
        \nxp\if@headed\nxp\else
            \ifforced@break@before\@beforepenalty
            \else\nxp\ifnum\lastpenalty=\z@
                         \@beforepenalty\nxp\fi
            \fi
        \nxp\fi
```

A subtle point: a preceding heading will have placed `\nobreak` followed by a skip. It is dangerous to place any sort of penalty after this because it might induce a page break.

Now the counter, title, and stuff connected to that.

```
    \fi
    \ifhas@counter
\nxp\StepCounter:\expandafter\@name\@space
        % This sets the \current@label by default
```

Since this is used inside an `\edef` we can use some trickery to get the space token after the argument to `\StepCounter`.

```
        \ifhas@marks \edef\nxp\cs@e
               {\nxp\nxp\nxp\refresh@mark@item
                {\@name Counter}{\CSname{\@name
Counter}}}%
            \nxp\cs@e
        \fi
        \fi
    \iflabel@defined
        \global\current@label={\the\@labelcoms}\fi
    \ifhas@title \install@title@code
```

This title business is explained in I-11. This piece of code also refreshes the title in the mark structure. This has to be done after any page break for the benefit of headers/footers.

```
        \fi
        \ifhas@marks\nxp\ifnin{\nxp\place@mark}\fi
            %otherwise IniTeX'ing Lollipop will output a
    page
        \nxp\xx@label\the\extern@toks\penalty\@M
            % also subtle: if this white space would be
    higher, it would
            % be invisible because of marks et cetera.
            % insert nobreak after marks/writes to
    prevent page breaks.
        \iftext@construct
            \ifleft@embedded@construct
                  \else
    \nxp\@vwhite{\the\@whitebefore}\fi
        \fi
        }
```

Inner start commands are concerned with setting local values.

```
    \def\inner@start@commands{%
        \nxp\Open@Group\CSname{\@class}\CSname{\@name}%
```

The `\Open@Group` call makes it possible to track down groups that have inadvertendly been left open. Since we now know the name we can give helpful error msgs.

```
        \install@stop
```

Install the right implicit closing, see 10.8 and I-6.

```
        \ifleft@embedded@construct
        \else \nxp\hold@parskip
            \nxp\@defaulteverypar
            \ifwhiteleft@defined \advance\leftskip
    \the\@whiteleft \relax \fi
            \ifwhiteright@defined \advance\rightskip
    \the\@whiteright \relax \fi
            \nxp\let\nxp\par=\nxp\@par %explain to me
    again why this is necessary...
            \inside@indent \first@indent
        \fi
        \advance\nest@depth\@ne
```

The nest depth is used for determining indentation levels.

```
        }
```

End commands set up some conditions, most of which will be tested by the start of any next construct.

```
    \def\inner@end@commands{%
        \nxp\Close@Group\CSname{\@class}\CSname{\@name}%
        \ifright@embedded@construct \else \nxp\leavehmode
    \fi
        \@afterpenalty
        \ifright@embedded@construct \else
    \nxp\@vwhite{\the\@whiteafter}\fi
        }
    \def\outer@end@commands{%
        \the\after@toks
        \ifright@embedded@construct
            \nxp\@headedno \nxp\esp@hack
        \else
            \after@indent \nxp\dono@parskip
        \fi}
```

I-*6 Implicit closing*

Constructs with an explicit closing command, lists and text blocks, can be closed by `\>`, which simply closes the current construct. A more drastic version, `\>]`, closes all currently open constructs.

```
\def\outer@stop@command{\Emessage{Vacuous group
closing}}
\let\default@stop@command\outer@stop@command
\def\>{\default@stop@command%[fool the editor
    \ifNextChar]{%
        \ifx\default@stop@command\outer@stop@command
            \xp\take@one
        \else \xp\>\fi}{}}
```

The `\outer@stop@command` is meant to give an error msg if the user attempts to close a group while none is open.

The current meaning of `\>` is installed in `\inner@start@commands`:

```
\def\install@stop{\if@implicitclose
        \def\nxp\default@stop@command
               {\CSname{\stop@command}}%
    \else \let\nxp\default@stop@command
              \nxp\outer@stop@command
    \fi}
```

By default, constructs can be closed implicitly, but there is an option `noimplicitclose`to disable this.

```
\newif\if@implicitclose
\add@generic@default{\@implicitcloseyes}
\@GenericOption{noimplicitclose}{\@implicitcloseno}
```

This option is for instance used in the examples in this manual. Otherwise closing a construct in the example would also close the example itself.

Chapter 7

output

Every page is formatted according to a 'page grid' consisting of three elements:

1 the page head, this is everything that's over the running text;
2 the page foot, this is everything that is below the running text;
3 the running text. TEX acts as if text is on a long scroll, and the running text part of a page is simply a portion cut off from this scroll.

Either or both of the head and foot of the page can be empty, but usually one of the two contains a page number.

Example 7.1

```
\DefinePageGrid:TestPage height:page=3cm width:page=5cm
 pagerule textband:start text textband:stop
 pagerule band:start PageCounter band:stop Stop
\TestPage This page does not contain much
special.\EjectPage
This page is hardly better.
```

This page does not contain much special.

1

This page is hardly better.

2

This example illustrates how you first define a page grid by `\DefinePageGrid`, and then activate it by calling its name. That last action is in fact not necessary: each definition of a page grid automatically installs that grid as the current one.

7.1 Page dimensions

Most of the time it is easiest to specify the total height of a page, that is, including head and bottom, but sometimes it is more convenient to specify the height of the text, and let the head and foot simply go over and under that.

In the first case you can give the command `\Height`with two parameters:

```
\Height:Page=23.5cm
```

or inside a page grid definition the option `height:page=...`.

In the second case you can give the command

```
\Height:Text=19.55cm
```

or inside a page grid definition the option `height:text=...`.

In page grid definitions there is the additional option `height:lines=23`.

The `\Height` command cannot be used in a page grid definition.

7.2 Positioning the page on the paper

If your printer driver is up to specs (and you have not done any creative macro writing) it should have the upper left corner of the text landing at 2.54cm from the top and left side of the paper. If the result is not to your liking, you can shift the page by

```
\Distance:hoffset= ...
\Distance:voffset= ...
```

These offset parameters are zero ordinarily, and they indicate the extra shift added to the customary 2.54cm in horizontal and vertical direction.

7.3 Page head, foot, text

Somewhere in the page grid the option `text`has to appear. This option has to be inside a `textband`:

```
textband:start text textband:stop
```

This is not a case of overspecification, because inside a textband the text option can appear more than once. In this manner a multicolumn page grid can be specified.

Example 7.2

```
\DefinePageGrid:TestPage height:page=3cm width:page=5.6cm
 pagerule textband:start text hwhite:3mm text textband:stop
 pagerule band:start PageCounter band:stop Stop
\FlushRight:no \sometext
```

```
______________________________________
                    words, words.
Just a bit of       Just a bit of
words, words.       words, words.
Just a bit of       Just a bit of
words, words.       words, words.
Just a bit of       Just a bit of
______________________________________
1
```

7 output

```
                            Just a bit of
words, words.               words, words.
Just a bit of               Just a bit of
words, words.               words, words.
Just a bit of               Just a bit of
words, words.               words, words.

2

Just a bit of
words, words.




3
```

Next to the option `textband` there is `band`. Both are ways of creating a page wide band. The option `band` is used for all material that is not a text column, for instance footers, as in the above examples.

The option `band` can have one unusual parameter: `invisible`. This makes the band act as if it has zero height or width, depending on whether it is below or above the text, respectively.

Example 7.3

```
\DefinePageGrid:TestPage height:page=3cm width:page=5.6cm
 pagerule textband:start text hwhite:3mm text
textband:stop
 pagerule
 band:invisible block:start Style:bold PageCounter
Spaces:2
     stickout:left band:stop Stop
\FlushRight:no \sometext
```

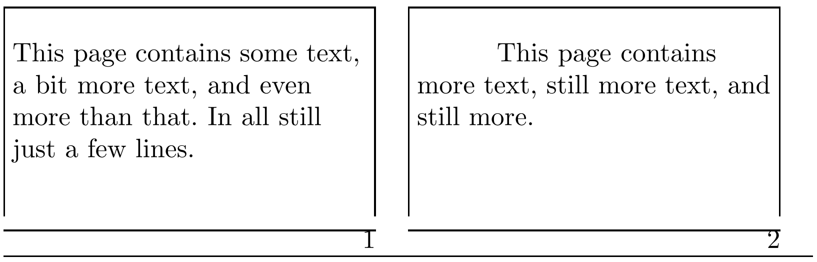
```
                               Just a bit of
   Just a bit of               words, words.
   words, words.               Just a bit of
   Just a bit of               words, words.
   words, words.               Just a bit of
   Just a bit of               words, words.
1  words, words.               Just a bit of
                               words, words.
   words, words.               Just a bit of
   Just a bit of               words, words.
   words, words.               Just a bit of
   Just a bit of               words, words.
   words, words.
2  Just a bit of
```

7.3.1 More about text bands

The text band is that part of the page that has the text in it. You can also put other material in it, such as rules or white space.

Example 7.4

```
\DefinePageGrid:TestPage height:page=3cm width:page=140pt
pagerule
 textband:start vrule white:3pt text white:3pt vrule
textband:stop
 pagerule band:start white:fillup PageCounter band:stop
Stop
\TestPage This page contains some text, a bit more text,
and even more than that. In all still just a few
lines.\EjectPage
This page contains more text, still more text, and still
more.
```


This page contains some text, a bit more text, and even more than that. In all still just a few lines.

1

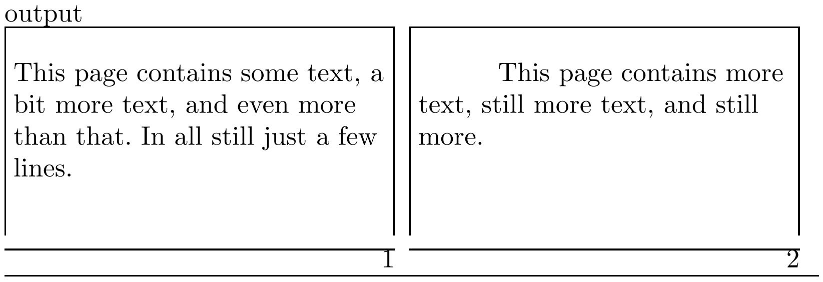
This page contains more text, still more text, and still more.

2

In the previous example the width of the page was specified. If we only give the width of the text, the page width is calculated dynamically.

Example 7.5

```
\DefinePageGrid:TestPage height:page=3cm width:text=140pt
pagerule
 textband:start vrule white:3pt text white:3pt vrule
textband:stop
 pagerule band:start white:fillup PageCounter band:stop
Stop
\noindent This page contains some text, a bit more text,
and even more than that. In all still just a few
lines.\EjectPage
This page contains more text, still more text, and still
more.
```


This page contains some text, a bit more text, and even more than that. In all still just a few lines.
This page contains more text, still more text, and still more.
1
2

Note how the `pagerule`and `band` objects stretch with the page.

7.3.2 Topskip

In between the page head and the text is some white space, the topskip, with special properties. The topskip is defined from the bottom of the head to the bottom of the first line of the text. If the height of this first line varies from page to page the topskip acts as a buffer, keeping the bottom-to-bottom distance constant.

Topskip is set by the option `topskip`, for example

```
topskip:25pt
```

but if this option is left out, the page grid uses the value of `\topskip` that was current at the time of the definition. Unfortunately there is no way to change this value after the definition.

7.4 The page number

The page number behaves as if it had been defined by

```
\NewCounter:Page
\CounterRepresentation:Page=1
```

Thus you can use any command from section 11.1 on it. For instance, you can have page numbers in roman numerals by specifying

```
\CounterRepresentation:Page=I
```

The page number is typically used as the option `PageCounter`, but for some applications the corresponding command `\PageCounter`can be used.

7.5 Page tests

The page grid definition can set/query several properties of the page. The following tests have been provided (see section 11.8 for tests):

```
\DefineTest:IsRightPage
\DefineTest:IsLeftPage
\DefineTest:FirstPage
\DefineTest:LastPage
\DefineTest:FlushBottom
```

- The tests for left/right pages are done by testing whether the page number of odd or even.
- The first/last page tests can be used either for the whole document, or for a file that's loaded as an `\InputFile`.
- The first page test doesn't work at present.

Example 7.6

```
\DefinePageGrid:TestPage height:page=3cm width:page=5cm
 pagerule textband:start text textband:stop pagerule
 band:start ifIsLeftPage else hwhite:fillup fi PageCounter
     band:stop Stop
This is a left hand page. \EjectPage
This page is on the right side of a spread.
```

This is a left hand page.

This page is on the right side of a spread.

1 2

7.6 *Running heads / footers*

Above it was explained how pages can be given a head and foot part. Quite often you want changing information in such parts, for instance the head of a left page often contains the number or title of section that was current when that page started; the head of a right page often contains the number or title of the section that was current when that page ended.

In Lollipop all constructs that have a title or a counter can have that information referenced in page grids.

`\FirstPlaced`:SectionTitle Take the title of the first section that started on this page, or the last one that started before this page if no section started on this page.

`\LastPlaced`:SubSectionCounter Take the title of the last subsection that started on this page, or the last one that started before this page if no subsection started on this pgae.

`\PreviousPlaced`:SectionCounter Take the counter value of the last section that started before this page.

Example 7.7

```
\DefinePageGrid:TestPage height:page=3cm width:page=5cm
```

```
 pagerule textband:start text textband:stop pagerule
 band:start Style:italic FirstPlaced:HeadTitle
            white:fillup PageCounter band:stop Stop
\DefineHeading:Head Style:bold
 line:start HeadCounter Spaces:2 title line:stop Stop
\Head A first section\par And some text.\EjectPage
This page contains text. \Head A second Section\par
And more text.
```

1 A first section

And some text.

A first section *1*

This page contains text.

2 A second Section

And more text.

A second Section *2*

The commands `\FirstPlaced` and `\PreviousPlaced` are typically used on left pages; `\LastPlaced` is more common on right pages. You can test on what sort of page you are; see section 7.5.

Example 7.8

```
\DefinePageGrid:TestPage height:page=3cm width:page=5cm
 pagerule textband:start text textband:stop pagerule
 band:start Style:italic
     ifIsLeftPage FirstPlaced:HeadTitle white:fillup fi
     PageCounter
     ifIsRightPage white:fillup LastPlaced:HeadTitle fi
     band:stop Stop
\DefineHeading:Head Style:bold
 line:start HeadCounter Spaces:2 title line:stop Stop
\Head A first section\par And some text.
\Head Second section\par More text.\EjectPage
\Head Third section\par Is on the right page.
\Head Fourth section\par Concludes this page.
```

1 A first section

And some text.

2 Second section

More text.

1 *Second section*

3 Third section

Is on the right page.

4 Fourth section

Concludes this page.

Third section *2*

I-7 *Marks*

All constructs that can have marks add their mark items to a globally maintained list:

```
\newtoks\mark@items
```

Only headings and page grids can have marks. Hence by default mark generation is switched off.

```
\newif\ifhas@marks
\add@generic@default{\has@marksno}
```

At the moment all counters and titles wind up in the mark structure:

```
\def\install@counter#1{ ...
    \xp\add@mark@item\xp{\@name Counter} ... }
\add@generic@stop@default
   {\ifhas@title\xp\add@mark@item\xp{\@name
Title}\fi}
```

The test whether the object currently being defined has marks is performed by `\add@mark@item`:

```
\def\add@mark@item#1{\ifhas@marks
    \csarg\newtoks{mark@toks@#1}%
    \Tmessage[mark]{Adding mark item #1}%
    \global\mark@items\xp{\the\mark@items{#1}}\fi}
```

This routine allocates a token list per mark item. For instance, for a `\SectionTitle` a `\mark@toks@SectionTitle` will be allocated. Every time a `\Section` occurs, a call

```
\refresh@mark@item{SectionTitle}{the title}
```

will then be made. This merely refreshes the contents of the specific token list:

```
\def\refresh@mark@item#1#2%
   {\csarg\global{mark@toks@#1}{#2}}
```

Also, a call to `\place@mark` will be made, which puts a mark on the current page, containing a list of item name / item value pairs. This proceeds fully by expansion.

```
\def\get@mark@items#1{\if\EqualStringX{#1}{SM}%
    \else{#1}{\csarg\the{mark@toks@#1}}%
         \xp\get@mark@items
    \fi}
\def\place@mark
   {\mark{\xp\get@mark@items\the\mark@items{SM}}}
```

Retrieving information out of a mark consists of traversing the list of pairs, and taking the value of the keyword requested. Thus you can call `\FirstPlaced:SectionTitle`.

```
\def\FirstPlaced:#1
   {\Tmessage[mark]{First Placed #1 from \firstmark}%
    \xp\get@placed\xp{\firstmark}{#1}}
```

The `\PreviousPlaced` and `\LastPlaced` commands are analogous, but based on the `\topmark` and `\botmark`.

Again, everything here proceeds by expansion only, so the string tester will consume some processor power.

```
\def\get@placed#1#2{\get@@placed{#2}#1{SM}{}$}% SM:
StopMarker
```

```
\def\get@@placed#1#2#3{\if\EqualStringX{#2}{SM}%
        \xp\take@to@dollar
    \else\if\EqualStringX{#1}{#2}%
             \maybe@uppercase{#3}%
             \xp\xp\xp\take@to@dollar
        \else\xp\xp\xp\get@@placed\fi
    \fi{#1}}
```

7.7 Alternating page grids

In Lollipop it is very easy to switch page grids: you simply specify

`NextPageGrid:otherpage`

as one of the options in the definition. If no next grid is indicated, the same page grid keeps being used continuously until another page grid is activated explicitly.

Example 7.9

```
\DefinePageGrid:LTestPage height:page=3cm width:page=5cm
 pagerule textband:start text textband:stop pagerule
 band:start Style:italic
     FirstPlaced:HeadTitle white:fillup PageCounter
     band:stop NextPageGrid:RTestPage Stop
\DefinePageGrid:RTestPage height:page=3cm width:page=5cm
 pagerule textband:start text textband:stop pagerule
 band:start Style:italic
     PageCounter white:fillup LastPlaced:HeadTitle
     band:stop NextPageGrid:LTestPage Stop
\DefineHeading:Head Style:bold
 line:start HeadCounter Spaces:2 title line:stop Stop
\LTestPage
\Head A first section\par And some text.
\Head Second section\par More text.\EjectPage
\Head Third section\par Is on the right page.
\Head Fourth section\par Concludes this page.
```

1 A first section

And some text.

2 Second section

More text.

1 *1*

3 Third section

Is on the right page.

4 Fourth section

Concludes this page.

2 *Fourth section*

Another very useful application of this mechanism is to have a special definition for the opening page of a chapter. This manual uses a one-shot page grid `\EmptyPage` to remove the header and footer on the title page. It installs `\LeftPage` as the next grid.

7.8 Additional User Control

7.8.1 Elementary manipulation

There are a few commands for simple page manipulation:

`\EjectPage` The current page is filled up with white space, and a new page is started.

`\ToRecto` As `\EjectPage` but if the next page is a left page (meaning that the page number is even) then the page number is increased by one, so that the next page is a right hand page.

`\ToVerso` As `\ToRecto`, except that the next page is a left page.

Additionally, `\NoPages`lets all formatting and updating of values be performed, but no pages are written to the dvi file; `\PagesOut`revert the effect of previous command

Implementor's Note

The test `\ifsink@pages` determines whether pages will be output to the dvi file (if the test is false) or silently dropped (if true). The sheet counter is only updated for pages written to the dvi file so that it will takes consecutive values no matter if pages are sunk. If a pages is dropped, `\deadcycles` is set to zero, otherwise it would not be possible to drop more than `\maxdeadcycles` pages.

When a page is finished, the whole box is given to `\CurrentShipout`, which is by default `\shipout`. However, you are free to define it otherwise. See the definition of `\OutputExample` in the appendix for an example. If your `\CurrentShipout` does not actually ship out pages, you may want to set `\CountSheetsno`to prevent the effective page counter from being updated.

Redefining `\CurrentShipout` usually goes together with `\SuspendOutput`and `\ResumeOutput`. These commands temporarily save the page number and the current state of the page.

If you want to see te output routines in action, specify

```
\Trace:out
```

In addition

```
\Trace:mark
```

tells you what information is being saved for running head and foot lines.

Chapter 8

Referencing

In manuals and scientific documents you often want to write something like ‘see Chapter 4’. But what if you shuffle the chapters a bit? It would be nice if the number would be updated automatically. With Lollipop, as with many other TEX macro packages, this is easily done.

Here is an example to set the mood for the rest of this chapter. The sort of thing that is referred to most is a heading. So suppose you want to refer to a section number.

Example 8.1

```
\DefineHeading:ASection
 line:start Style:italic ASectionCounter Spaces:2 title
 line:stop Stop
\ASection[one:section?] First section\par
After this section will come section~\ref[other:section!].

\ASection[other:section!] Another section\par
This is the section that came after
section~\ref[one:section?].
```

1 First section

After this section will come section 2.

2 Another section

This is the section that came after section 1.

8.1 What and how do you reference?

You can reference not only headings but everything that has a counter. Thus all generic constructs can be referenced, and in addition you can reference item numbers in a list (there are examples of this latter possibility in section 8.4). The simplest way of referencing something is to put the key in square brackets behind it:

```
\Section[this:section] The title of This Section
```

The key is used by typing

```
\ref[this:section]
```

As you may have guessed from the above examples, keys can contain all sorts of characters. Only brackets, braces, and the hash sign are excepted. You get an error message if you try to use the same key twice.

Another way of declaring a key is to use the command `\label`carrying the key

```
      \label[the:key]
```

This can be useful if you want to declare two keys for a single reference. Make sure that the `\label` command is not part of the title. Unexplained phenomena occur if you do that. Instead put the label after the construct you want to reference:

```
      \Section Precautions and remedies

      \label[sec:precautions]\label[sec:remedies]
      In this section ...
```

8.2 The shape of the reference

By default, a reference consists of just the number of the thing you reference. You can customize the way an object is referenced by using the option `label`in its definition. For instance, often you want things like parentheses around references. Putting this information in the label definition saves you a lot of work in case you change your mind later.

Example 8.2

```
\DefineHeading:ASection
 line:start Style:italic ASectionCounter Spaces:2 title
line:stop
 label:start ( ASectionCounter ) label:stop Stop
\ASection[one:section?2] First section\par
After this section will come section~\ref[other:section!2].

\ASection[other:section!2] Another section\par
This is the section that came after
section~\ref[one:section?2].
```

1 First section

After this section will come section (2).

2 Another section

This is the section that came after section (1).

Another use of customized labels is including other counters in the reference:

Example 8.3

```
\DefineHeading:AChapter
 line:start Style:bold AChapterCounter / title line:stop
Stop
```

```
\DefineHeading:ASection
 line:start Style:italic ASectionCounter Spaces:2 title
line:stop
 label:start AChapterCounter . ASectionCounter label:stop
Stop
\AChapter First chapter\par
Pretty short chapter
\AChapter Second chapter\par
\ASection[one:section?3] First section\par
After this section will come section~\ref[other:section!3].

\ASection[other:section!3] Another section\par
This is the section that came after
section~\ref[one:section?3].
```

1/First chapter

Pretty short chapter

2/Second chapter

1 First section

After this section will come section 2.2.

2 Another section

This is the section that came after section 2.1.

A more surprising application of explicit definition of labels is inclusion of the title in the reference.

Example 8.4

```
\DefineHeading:ASection
 line:start Style:italic ASectionCounter Spaces:2 title
line:stop
 label:start ASectionCounter literal: Spaces:1
             Style:italic title label:stop Stop
\ASection[one:section?4] First section\par
After this section will come section~\ref[other:section!4].

\ASection[other:section!4] Another section\par
This is the section that came after
section~\ref[one:section?4].
```

1 First section

After this section will come section 2 *Another section.*

2 Another section

This is the section that came after section 1 *First section.*

8.3 Local references

Some documents are collated out of parts that were documents in themselves. In such a case it may happen that the same reference key is used in more than one part of the document. Ordinarily this would result in incorrect references.

To prevent such collisions Lollipop can use local references: the command `\LocalReferences`has default `no`, and specifying

```
LocalReferences:yes
```

creates local `aux` files. Furthermore, the parts of the document have to be loaded by

```
\InputFile:parta
\InputFile:partb
```

et cetera. A document part loaded by `\InputFile`always starts on a new page.

In addition, loading files this way provides a form of error checking; Lollipop checks at the end of such a file whether all used constructs are balanced properly.

8.4 Bibliography citations

Lollipop has as yet no separate facilities for bibliographies such as an interface to BibTEX. However, since a bibliography is just a list, referencing items in it is quite easy.

Example 8.5

```
\DefineList:BibList item:left [ itemCounter ] item:stop
 label:start [ itemCounter ] label:stop Stop
In this example we shall have occasion to refer to
\ref[Abee80] and~\ref[Ceede79].\par
\Indent:no Bibliography
\BibList \item[Ace55] C.D. Ace, Inscrutible title.
\item[Abee80] E.F. Abee, Worthless drivel.
\item[Ceede79] G.H. Ceede, Contractual obligation.
\>
```

In this example we shall have occasion to refer to [2] and [3].
Bibliography

[1] C.D. Ace, Inscrutible title.
[2] E.F. Abee, Worthless drivel.

[3] G.H. Ceede, Contractual obligation.

Here is a way to customize the label (if you need to refresh your memory about description lists, see section 5.5).

Example 8.6

```
\DefineList:BibList item:left [ itemCounter ] item:stop
 label:start ( description ) label:stop Stop
In this example we shall have occasion to refer to
\ref[Abe80] and~\ref[Ceedee79].\par
\Indent:no Bibliography
\BibList \item[Aace55] Aace55
C.D. Aace, Inscrutible title.
\item[Abe80] Abe80
E.F. Abe, Worthless drivel.
\item[Ceedee79] Ceedee79
G.H. Ceedee, Contractual obligation.
\>
```

In this example we shall have occasion to refer to (Abe80) and (Ceedee79).
Bibliography

[1] C.D. Aace, Inscrutible title.
[2] E.F. Abe, Worthless drivel.
[3] G.H. Ceedee, Contractual obligation.

Chapter 9

External Files

Some document require information to be collected during a run. Such information typically is a table of contents or index, and it is gathered in an external file. (The reason for gathering such information in a file at all is that often some external manipulation, for instance sorting of an index, is needed.) Since there are many possibilities for external files (mathematical monographs may have a list of definitions, or a list of notations) Lollipop does not predefine such files, but supplies all of the tools for creating them.

External files involve four actions:

1 The file should be declared.
2 It should be specified what information is to be written to the file.
3 The formatting of the contents of the file has to be specified.
4 The file has to be loaded.

9.1 Declaring and loading an external file

The first act, declaring the existence of the external file is very easy with the command `\DefineExternalFile`: an internal name and a three-character file name extension have to be given as parameters.

```
\DefineExternalFile:contents=toc
```

With this definition, if the document is called `book.tex` then the 'contents' will be gathered in a file called `book.toc`.

For each external file `Foo` there is a command to determine whether that file will be regenerated in the next run of TEX: `\WriteFoo`with values `yes`/`no` will allow or prevent the file being regenerated. The value `yes` is default. The command `\WriteExtern`(values `yes`/`no`) can be used to prevent writing out any external files (including the `.aux` file that keeps track of references).

The final act, loading an external file, is as easy as declaring it: use `\LoadExternalFile`as in

```
\LoadExternalFile:contents
```

This does not cause any page breaks or headings to be set over the loaded material, so you have to do that explicitly.

9.2 Generating external files

Next, macros that write to the table of contents have to declare this information. The `external`option is used for this. Any counter that the construct has will be written out automatically, and the style designer usually has to specify only that the title will be written out.

```
\DefineHeading:Section
    [...]
    external:contents title external:stop
```

There is no objection to a construct writing information to more than one external file.

9.3 Formatting an external file

The hardest part is declaring the formatting of an external file. For this a separate generic construct exists: the ‘external item’ with defining command `\DefineExternalItem`. For example, if `\Section` writes to `contents`, than an external item `Section` corresponding to this file has to be declared. The option `file`is use to declare to which file the external item belongs. This way the same name can be reused for more than one file.

```
\DefineExternalItem:Section file:contents
    [...] Stop
```

An external item is basically a list with just one item. Thus, the option `item`is available. The elements of an external item are the label (the counter value), the page number where the information was generated, and the title. For the label (say for a construct `\Foo`) an option `FooLabel`is created. Thus the typical formating looks like

```
\DefineExternalItem:Chapter file:contents
    item:left ChapterLabel item:stop
    title begingroup Spaces:2 Style:italic Page
endgroup
    Stop
```

In fact, a control sequence `\FooLabel`is created, which can be used in other external items.

Since an external item is a list in itself, you have to pull a certain trick to get items for subsections to indent further than those for sections. This is what the command `\PushIndentLevel` was designed for.

A typical indented item looks like:

```
\DefineExternalItem:SubSection file:contents
    PushIndentLevel PushIndentLevel
    item:left SectionLabel . SubSectionLabel
item:stop
    title begingroup Spaces:2 Style:italic Page
endgroup
    Stop
```

9.4 Example

The following example is for a typical table of contents that records sections and subsections. In good old-fashioned style, the subsections are indented with respect to the sections.

Example 9.1

```
\DefineExternalFile:TheContents=tct
\DefineHeading:LevelOne Style:bold
 line:start LevelOneCounter Spaces:2 title line:stop
 external:TheContents title external:stop Stop
\DefineExternalItem:LevelOne file:TheContents
 item:left Style:bold LevelOneLabel item:stop title
white:5pt
      Page par Stop
\DefineHeading:LevelTwo Style:italic
 line:start LevelOneCounter . LevelTwoCounter Spaces:2
title line:stop
 external:TheContents title external:stop Stop
\GoverningCounter:LevelTwo=LevelOne
\DefineExternalItem:LevelTwo file:TheContents
PushIndentLevel
 item:left Style:bold LevelOneLabel . LevelTwoLabel
item:stop
 title white:5pt Page par Stop

\LoadExternalFile:TheContents
\LevelOne First heading\par
\LevelTwo First subheading\par
Some text might be nice.
\LevelTwo Second subheading\par
Some more text.
\LevelOne Second heading\par
\LevelTwo Third subheading\par
Yet more text.
\LevelTwo Fourth subheading\par
And again: text.
```

TwoHead;

1 First heading 36

  1.1 First subheading 36

  1.2 Second subheading 36

2 Second heading 36

  2.1 Third subheading 36

  2.2 Fourth subheading 36

9 External Files

1 First heading

1.1 First subheading

Some text might be nice.

1.2 Second subheading

Some more text.

2 Second heading

2.1 Third subheading

Yet more text.

2.2 Fourth subheading

And again: text.

Chapter 10

Options

10.1 Titles

Any construct can have a title, although of course it is most useful for headings. A construct has a title if the option `title`appears. Example:

```
\DefineHeading:Section [...]
    Style:bold title
    [...] Stop
```

will define a `\Section` macro that has a title. The macro is then used as

```
\Section The title of this section

Some text in this section.
```

that is, the title is delimited by an empty line.

The title is actually available as a macro `\FooTitle`, so that you can write a macro, for instance

```
\def\ComplicatedTitle{ .. \hrule ...
    \vrule ... \vbox \bgroup ...
    \FooTitle ...
    }
```

and use this macro instead of the `title` option

```
\DefineBar:Foo ...
    ComplicatedTitle
    ... Stop
```

However, since the option `title` now doesn't appear anymore, it becomes necessary to specify explicitly that there is a title. This can be done with the `HasTitle`option.

```
\DefineBar:Foo ...
    HasTitle
    ComplicatedTitle
    ... Stop
```

Implementor's Note

I-*8 Delimiting the title*

The title is actually delimited by `\par`, so

```
\Section The title\par
```

is allowed. Since delimiting by an empty line delimits by a space plus `\par` some extra measures are needed to get rid of the space in exceptional cases. The title is in effect augmented by `\unskip`. Thus, every time the title is typeset any trailing space is removed. See the definition of `\@Titelize` in section I-10.

I-*9 Is there a title?*

The options `title` and `HasTitle` both set a test `\has@title` to true; this test is false by default

```
\newif\ifhas@title\add@generic@default{\has@titleno}
```

The first of the two further causes inclusion of `\FooTitle` in the current option token list.

```
\@GenericOption{title}{
    \global\has@titleyes
    \ifin@label \label@append@title
    \else \edef\cs@e{\nxp\@add@toks
                 {\CSname{\@name Title}}}\cs@e
    \fi}
\@GenericOption{HasTitle}{
    \switch {\if\EqualString{#1}}
    {yes} {\global\has@titleyes}
    {no}  {\global\has@titleno}
    {default} {\global\has@titleyes}
    \endswitch
    }
```

Titles wind up in marks: at the end of defining `\Foo` the `\FooTitle` is added to the mark items.

```
\add@generic@stop@default
   {\ifhas@title\xp\add@mark@item\xp{\@name
Title}\fi}
```

I-*10 Giving a macro a title*

Lollipop macros are first defined without titles. If necessary they are then redefined to have a title.

The redefinition depends on how many arguments the macro originally had; this is determined by a counter `\extra@args`.

```
\newcount\extra@args
\add@generic@default{\extra@args\z@}
```

At present, only external items (section 9.3) have extra arguments.

The redefinition proceeds by storing the original definition in `\tit@Foo`, the macro is then redefined as a macro with an extra argument, which is stored in `\title@toks`. Often it is convenient to have the title in a token list to prevent it from being expanded.

Additionally, an `\unhskip` is appended to the title, because delimiting with an empty lines will give a space before the delimiting `\par`.

```
\def\@Titelize#1{%
    \edef\cs@e{\let\CSname{tit@#1}=\CSname{#1}}\cs@e
    \ifcase\extra@args %0:
        \csarg\edef{#1}##1\par
             {\ti-
tle@toks{{##1\nxp\protect\nxp\unhskip}%
              \CSname{tit@#1}}
        \or %1:
            \csarg\edef{#1}##1##2\par
                 {\ti-
    tle@toks{##2\nxp\protect\nxp\unhskip}%
                  \CSname{tit@#1}{##1}}
        \or %2:
            \csarg\edef{#1}##1##2##3\par
                 {\ti-
    tle@toks{##3\nxp\protect\nxp\unhskip}%
                  \CSname{tit@#1}{##1}{##2}}
        \else \Wmessage{Sorry, too many extra arguments
                        for '\@class' : '\@name'}
        \fi}
```

I-*11* *Storing the title*

In `\outer@start@commands` the title is then put in the macro `\FooTitle`.

```
      \def\install@title@code
         {\nxp\xp\def\nxp\xp\CSname{\@name Title}\nxp\xp{%
                 \nxp\xp\nxp\maybe@uppercase\nxp\xp
                       {\nxp\the\title@toks}}%
          \ifhas@marks \edef\nxp\cs@e
                   {\nxp\nxp\nxp\refresh@mark@item
                     {\@name Title}{\nxp\the\title@toks}}%
                  \nxp\cs@e
          \fi}
```

This piece of code is inserted after any page break, because it refreshes the mark information. Furthermore, its inclusion is conditional.

```
         \ifhas@title \install@title@code
         \fi
```

10.2 Counters

There are three ways for Lollipop to figure out that a generic construct has a counter. First of all, in

```
      \DefineFoo:Bar [...]
          BarCounter [...]
```

the `\BarCounter` will be defined automatically.

Additionally there is the option `counter`, which can be used to declare the representation of the option, for instance `counter:i` allocated a counter that is printed in lowercase roman numerals. For the available representations, see 11.1.1.

Finally, if the counter is only used in a macro, then the option `HasCounter`will cause the counter to be created anyhow. This is analogous to the `HasTitle` option.

At the start of defining the construct, `\BarCounter` is defined to be an option:

```
\add@generic@default{\has@counterno
    \def\counter@repr{1}
    \csarg\def{\gen@option@name{\@name Counter}}{%
        \@add@toks{\@name
Counter}\global\has@counteryes}}
```

Then,

```
\add@generic@stop@default{\ifhas@counter
    \xp\expandafter\xp\install@counter
           \xp\counter@repr\@space\fi}
```

The counter is stepped, and the new value is stored in a mark item, in `\outer@start@commands`:

```
    \ifhas@counter
       \nxp\StepCounter:\expandafter\@name\@space
       % This sets the \current@label by default
       \ifhas@marks \edef\nxp\cs@e
              {\nxp\nxp\nxp\refresh@mark@item
                {\@name Counter}{\CSname{\@name
Counter}}}%
            \nxp\cs@e
       \fi
       \fi
```

10.3 Chunks of text

Especially in headings, short chunks of text may need a special treatment. For instance, the number may have to be filled to a certain width, or a line may have to be drawn of the exact length of the title. Lollipop have various general options (so they can also be used in other contexts than headings) for handling pieces of text.

10.3.1 The `block` *option*

The `block`option takes up a piece of text and fits it on one line. It can measure the text, or set the size. Also there are a number of ways of placing a block.

Basic usage:

```
block:start [...] block:stop
```

This takes the enclosed text, and reproduces it. This is mostly interesting in combination with `textcolumn`, see 10.3.3.

```
block:hang [...] block:stop
```

The resulting block is dropped until its top touches the baseline. For uniformity of appearance, the resulting width of the block can be specified:

```
block:start [...] fillupto:20pt
```

The name of a `\Distance` parameter can be used here.

Example 10.1

```
\DefineHeading:Test
 line:start block:hang PointSize:8 SetFont
                 TestCounter fillupto:20pt
           block:hang PointSize:14 SetFont title
block:stop
 line:stop Stop
\Test Top Aligning the Title
```

1 Top Aligning the Title

The block is usually in between the margins of the text, but it can made to stick out into the margin. For the left margin this done as

```
          block:start [...] stickout:left
```

and for the right margin

```
          block:start [...] stickout:right
```

The size of the box can be specified, for instance as

```
          block:start [...] stickout:left=20pt
```

For a left box the material in it is pushed to the left edge, for a right sticking box it is shifted to the right.

10.3.2 The `line` *option*

The option `line`is used to create a single strip of text that fits exactly in between the margins of the page. Most of the time, titles will be in a line.

Example 10.2

```
\DefineHeading:Test
    line:start block:start TestCounter Spaces:1.5
stickout:left
               title line:stop Stop
\Test A Title
```

1 A Title

Another example was above. Here is another use of a line:

Example 10.3

```
\DefineHeading:Test
    line:start TestCounter fillup title line:stop Stop
\Test The title
```

1 The title

10.3.3 The `textcolumn` *option*

In the examples above all titles fit on one line comfortably. If this is not the case, the title can be put in a `textcolumn`which can span several lines.

Example 10.4

```
\DefineHeading:Test
    line:start block:start TestCounter Spaces:2 block:stop
               textcolumn:topline title textcolumn:stop
    line:stop Stop
\Test A very very very very very very very very very very
very very
very very very very very very very very very very very very
very very very very very very very very long title
```

1 A very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very long title

This option is mostly interesting in combination with others such as `block` and `line`. As is apparent from the above example: a block placed in the same line as a text column will detract from the latter's width.

(In fact it is the other way around: Lollipop sets the line with a text column of width zero to determine the remaining space. Then the line is set again. This may give problems if you manipulate parameters inside the line, because the line is in effect typeset twice. Also make sure not to have other `\vbox`-es in the line than the text column.)

10.3.4 Traps

It is a bad idea to have material in headings and such that is not inside a block, textcolumn, or line. For instance:

Example 10.5

```
\DefineHeading:Test
    block:start TestCounter Spaces:2 block:stop
    title Stop
\Test Where does the title go?
```

1
Where does the title go?

10.4 Labels

References to any counter will always be correct, no matter if that counter has changed after retypesetting the document, if symbolic references are used. Referencing is explained in detail in chapter 8.

The way a symbolic reference is formatted can be altered from the default (just give the counter) by using the `label`option.

```
\DefineHeading:ASection
 line:start Style:italic ASectionCounter Spaces:2
title line:stop
 label:start ( ASectionCounter ) label:stop Stop
```

See further section 8.2.

10.5 Break before / after

The options `breakbefore`and `breakafter`control how eager TEX will be to break the page before or after a construct. These options take one value, a so-called ‘penalty’ value, meaning that the higher the value you specify, the higher the penalty is, and therefore the less likely it is that the page will be broken there.

Numerical values are typically in the tens or hundreds; any value of `10 000` or more means that there will never be a break at that point; a value of `-10 000` or less means a guaranteed break. If you don’t want to remember these rules, values of `yes` and `no` mean a guaranteed break, and no break respectively.

A further exceptional value is `breakbefore/after:0`, this will cause no penalty to be placed. The reason for this is highly TEXnical.

10.6 Indentation

The option `indentafter`controls the behaviour of the first paragraph after a generic construct., `indentinside`, `indentfirst`.

10.7 Embedded constructs

Most generic constructs will be vertically separated from the surrounding text. However, in rare cases (and for unusual applications) it be desired to have a construct that is part of a paragraph. For this the option `embedded`exists.

This option has the following values.

`embedded:no`
This is the default.

`embedded:left`
The construct continues an already started paragraph, but after the construct a vertical break follows.

`embedded:right`
After the construct a paragraph can continue, but the construct is separated vertically from preceding text.

`embedded:yes`
The construct is both left and right embedded.

Embedding a construct has an interesting application to generating indexes. (See chapter 9 for general information about external files.) This can be done by having embedded headings that write their title to the index file.

Example 10.6

```
\DefineExternalFile:tIndex=tix
\DefineHeading:NewWord embedded:yes
 block:start Style:bold title block:stop
 external:tIndex title external:stop Stop
\def\introword#1{\NewWord #1\par}
In this sentence two \introword{flubrious} words are
\introword{stinselsed}.
```

In this sentence two **flubrious**words are **stinselsed**.

Cute, ain't it?

10.8 Implicit closing

The control sequence `\>`closes the current group, and `\>]`closes all currently open groups. Every once in a while this is too drastic. Hence there is an option `noimplicitclose`that can be used to prevent a construct from being closed implicitly.

10.9 Testing

There is an option `test`.

Chapter 11

Commands

11.1 Counters

Counters can be declared explicitly by the user, but more often they are defined automatically in some generic construct:

The `\Foo` defined by

```
\DefineBar:Foo ...
    counter:i ...
    Stop
```

will have a counter that counts in roman lowercase, and which is accessible as `\FooCounter`. Everytime `\Foo` is used, this counter is increased by one.

The use of the `counter` option is described in 10.2. Here are the commands for explicit manipulation of counters.

11.1.1 Allocation and representation

A counter is created by for instance

```
\NewCounter:Things
```

This will create control sequence `\ThingsCounter` that will print the value of the counter. The counter will usually be printed as an Arabic numeral, but other counter representations can be specified by `\CounterRepresentation`. Here are their codes:

1	numeric
a	lowercase character
A	uppercase character
i	lowercase roman
I	lowercase roman

for instance

```
\CounterRepresentation:Things=i
```

will cause `\ThingsCounter` to print a lowercase Roman numeral.

However, a call such as

```
\CounterRepresentation:Theorem=Lemma
```

will make the `\TheoremCounter` a synonym of an earlier created `\LemmaCounter`

11.1.2 Counter manipulation

The following commands can be used to manipulate counters, both when they are created by hand using `\NewCounter`and when they were generated automatically in some generic construct:

Reset the counter to one:

```
\StartCounter:things
```

Increase the counter by one:

```
\StepCounter:things
```

Decrease the counter by one:

```
\BackStepCounter:things
```

Set the counter to some specified value

```
\SetCounter:things=5
```

11.1.3 Counter hierarchies

Often counters are related to each other. For instance, when a new section begins, the subsection counter has to be reset. The same may be true for equation counters. In Lollipop such a relation is indicated by a call to `\GoverningCounter`, for instance

```
\GoverningCounter:SubSection=Section
```

All of the counter manipulation commands applied to a governing counter will cause all governed counters to be reset. Such a reset also occurs if the counter was created in some generic construct.

For examples, see section 4.1.

11.1.4 Referencing counters

All counters that are declared as part of a generic construct, or explicitly through `\NewCounter`automatically become the current reference when they are altered. Thus `\label[bar]` will make `\ref[bar]` refer to the value of the counter most recently changed. The way the counter is referenced can be altered by the `label` option in generic constructs; see section 10.4.

For generic constructs with a counter no explicit `\label`commands need to be given; such commands take an optional argument with the label key:

```
\Section[sec:examples] Examples
```

11.1.5 Examples of counter usage

Items start at the value of one, so if a starting value of zero is necessary, the following will work

```
\Enumerate \SetCounter:item=-1
\item ...
```

46 *Implementor's Note*

I-*12 The counter name*

The \count register associated with a counter receives an internal name:

```
\def\counter@name#1{#1@C}
```

Also the following common abbreviations are provided:

```
\def\cs@counter@name#1{\csname#1@C\endcsname}
\def\counter@@name#1{\CSname{#1@C}}
```

I-*13 Allocation and representation*

The user command \NewCounterallocates a counter plus an associated 'reset list':

```
\def\NewCounter:#1 {
    \csarg\newtoks{#1@RL}
    \csn #1@RL\ecs={}
    \new@counter{#1}
    }
\def\new@counter#1{
    \new@@counter{#1}
    \CounterRepresentation:{#1}=1
    \StartCounter:{#1}
    }

\def\CounterRepresentation:#1=#2 {
    \if\UndefinedCS{\counter@name{#2}}%is deze teller
een synoniem?
          \represent@counter{#1}{#2}
    \else \@SynonymCounter{#1}{#2}
    \fi}
\def\represent@counter#1#2{
    \edef\cs@e{@\if#2iroman\else
       \if#2IRoman\else \if#2alcascii\else
       \if#2Aucascii\else arabic\fi\fi\fi\fi}
    \csarg\edef{\counter@repr{#1}}{\CSname{\cs@e}}
    \csarg\edef{#1Counter}%
      {\CSname{\counter@repr{#1}}\counter@@name{#1}}
    }
\def\@SynonymCounter#1#2{\edef\cs@b{%
   \nxp\let\counter@@name{#1}=\counter@@name{#2}
   \nxp\let\CSname{#1Counter}=\CSname{#2Counter}
   \nxp\let\CSname{#1@RL}=\CSname{#2@RL}}
    \cs@b
    }

\@GenericOption{sharecounter}
   {\CounterRepresentation:\@name=#1 }
```

I-14 Governing and resetting

A counter can be defined as being governed by another counter; whenever the other counter is manipulated, this counter is reset. The implementation is through 'reset lists': every counter is added to the reset list of its governing counter, and whenever a counter is altered, everything in its reset list is reset.

```
\def\GoverningCounter:#1=#2 {\if\UndefinedCS{#2@RL}
    \Emessage{No counter defined for '#2'}
    \else\append@to@list{#2@RL}{\\#1;}\fi}
\def\reset@subordinates#1{%
    \def\\##1;{\start@counter{##1}}%
    \the\csname #1@RL\endcsname  \let\\=\relax}
```

The command `\reset@subordinates` is executed by all user level commands:

```
\def\StartCounter:#1
{\handle@user@counter{#1}{start}{}}
\def\StepCounter:#1
{\handle@user@counter{#1}{step}{}}
\def\BackStepCounter:#1
{\handle@user@counter{#1}{back@step}{}}
\def\SetCounter:#1=#2
{\handle@user@counter{#1}{set}{#2}}
\def\handle@user@counter#1#2#3%
   {\if\UndefinedCS{\counter@name{#1}}
          \Wmessage{Unknown counter: #1}
    \else \csarg\global{#2@counter}{#1}{#3}%
          \re-
set@subordinates{#1}\define@reference{#1}%
    \fi}
```

The system level commands have no further complications.

```
\def\step@counter#1%
   {\increase@value{\counter@name{#1}}\@ne}
\def\back@step@counter#1%
   {\increase@value{\counter@name{#1}}\m@ne}
\def\start@counter#1%
   {\set@value{\counter@name{#1}}\z@}
\def\set@counter#1#2%
   {\set@value{\counter@name{#1}}{#2}\relax}
```

11.2 Font selection

In Lollipop, choosing a font is done through three parameters:

Typeface A collection of related styles and sizes. The typeface is set by the command `\Typeface`.

Style Italic, bold, roman, typewriter. You know. The style is set by the command `\Style`.

PointSize The size of a font in typographical points (72.27 per inch). The pointsize is set by the command `\PointSize`.

The most common change of font is a change in style. Therefore, issuing a command such as

```
\Style:bold
```

immediately changes the font to the bold of the current typeface in the current pointsize.

However, issuing a command such as

```
\Typeface:GoudyOldStyle
```

or

```
\PointSize:28
```

will not change the font, since such changes are usually accompanied by a change in style. In case that an immediate switch is necessary, the command `\SetFont`can be given. This evaluates the current value of the typeface, style, and pointsize commands, and sets the font accordingly.

A number of typeface names have been predefined in Lollipop, however, in order to print them your printer (software) must have them available.

Example 11.1

```
\SerifFace \PointSize:12
\Style:roman This \Style:italic sentence \PointSize:10 has
\SetFont way \SansFace \Style:roman too \SetFont many
\PointSize:12 \SetFont font \Style:bold changes.
```

This *sentence has way* too many font **changes.**

(The commands `\SerifFace` and `\SansFace` are defined in the master file of this manual, and serve to make this manual formattable on any system.)

11.2.1 Relative size changes

Apart from setting the pointsize explicitly, it is also possibly to make size changes relative to the current size. For instance, `\PointSizeLarger`and `\PointSizeSmaller`with an optional argument indicating the size of the change can be used. These commands are not cumulative.

Example 11.2

```
\SerifFace
\PointSize:9 \SetFont Every once in a while,\SaveFont
\PointSizeLarger[2] shouting \PointSizeLarger helps.
\PointSizeSmaller[2]But most of the times it doesn't.
\RestoreFont Unfortunately.
```

Every once in a while, shouting helps. But most of the times it doesn't. Unfortunately.

Similar to the changes in mathematics mode to script and scriptscript size, the same relative changes are available in text mode through the control sequences `\script`and `\scriptscript`. The control sequence `\normal`can be used to restore the default size.

Here is one application of such relative changes:

```
L\kern -.3em\raise .35ex\hbox {\script A}\kern
-.1em\TeX
```

which gives definition of the LATEX logo that is independent of typeface, size and style.

The relative sizes of script and scriptscript fonts are by default at 70% and 50%, but they can be set explicitly by

```
\PointSizeScriptSizes:10=10,7,5
```

This also gives the possibility to have the `\normal` size to be different from the surrounding pointsize.

11.2.2 Typeface definition

Defining a typeface means telling Lollipop how the external font name, that is, the name of the `tfm` file, is to be constructed from the internal parameters. The command `\DefineTypeface`takes four parameters and an optional fifth. The parameters are in sequence

1 The internal name of the typeface: the name that is given to the `\Typeface` command.
2 The root of the external file name. It is assumed that all font names of different styles and sizes are constructed by appending characters to this base.
3 Suffixes corresponding to the styles that are available.
4 Suffixes corresponding to the sizes that are available.

Here is the definition of the Computer Modern typeface:

```
\DefineTypeface{ComputerModern}{cm}
    {roman:r; slant:sl; italic:ti; mitalic:mi;
bold:bx; tty:tt;
     default:r;}
    {<6:5; <7:6; <8:7; <9:8; <10:9; <11:10;
     <12:10 \scaled\magstephalf;
     <14:10 \scaled\magstep1; <16:10
\scaled\magstep2;
     <20:10 \scaled\magstep3; >19:10
\scaled\magstep4;
     default:10;}
```

Actually, not all combinations of styles and sizes are available. That's where the optional argument comes in. This argument can be used to specify with TEX conditionals exceptional style/size combinations. Here some trickery is needed: internally the size is stored in `\F@size`, and in order to use this parameter we need to make the at-sign a letter temporarily.

```
\makeatletter
\DefineTypeface{Compu ...
    ...
    default:10;}
    [\ifStyle:italic \ifnum\F@size<7 ti7\fi\fi
     \ifStyle:tty \ifnum\F@size<8 tt8\fi\fi]
```

For other typefaces specifying the size suffix may be much easier than for Computer Modern. For instance, here is the definition of the PostScript Helvetica typeface.

```
\makeatletter
\DefineTypeface{psHelvetica}{helv}
    {roman:; italic:i; mitalic:i; bold:b; default:;}
    {default: at \F@size pt;}
\makeatother
```

11.2.3 Other font matters

The combination `\SaveFont`with a subsequent `\RestoreFont`can be used to save and restore the current font.

An abbreviation for a font can be defined by

```
\DefineFont:name=face,size,style
```

Even if you don't use Computer Modern as your main typeface, the typewriter style is not bad, so a control sequence

```
\def\tt{\Typeface:ComputerModern \Style:tty }
```

has been given that makes `\tt`always refer to the `cmtt` fonts. You're at liberty to change this, of course.

11.3 Baselineskip

Corresponding to a font size usually the baseline skip has to change. By default a fixed ratio of 1.2 for this is taken, for instance using a 12 point baseline skip for 10 point fonts. Changing the ratio can be done by

```
\BaselineSkipPointSizeRatio:1.3
```

If only for some specific size the baseline skip has to deviate from the default ratio, then this can be set by

```
\SetPointSizeBaselineSkip:9=12
```

11.4 Indentation Control

11.4.1 To indent or not to indent

In most documents there is a general rule that all paragraphs indent unless a certain condition, or that they do not indent unless certain special conditions hold. For Lollipop documents this is determined by the command `\AlwaysIndent`, with values `yes/no`.

To override this default setting a command `\Indent`(with values `yes/no`) exists, but that is mostly useful as an option in generic constructs, and even there it will not be used much. See section 10.6 for options relating to indentation.

Important: never set `\parindent` to zero. Preventing indentation globally should be done through `\AlwaysIndent:no`.

11.4.2 Indentation levels; indentation size

When Lollipop decides that text should be indented, it refers to a list of indentations for the exact amount. This list contains indentation amounts for each 'level' of indentation: initially the level is one, and if you nest constructs that indent (for instance using a list inside a list) the level goes up one step per nested construct.

There is a quantity

`\Distance:basicindent`

that is used on the first indentation level. By default on higher levels a fraction of the `\basicindent`is used. Thus you can regulate the indentation on all levels simultaneously by resetting the `\basicindent`.

Example 11.3

```
\Distance:basicindent=15pt
\DefineList:AList item:left itemCounter item:stop Stop
\AList\item Level one \AList\item Level two
\AList\item Level three\>]
\Distance:basicindent=25pt
\AList\item Level one \AList\item Level two
\AList\item Level three\>]
```

```
1       Level one
        A Level two
          I Level three

1       Level one
        A   Level two
              I   Level three
```

The amount of indentation on a certain level can be set explicitly with `\LevelIndent`.

Example 11.4

```
\Distance:basicindent=15pt
\LevelIndent:2=20pt
\DefineList:AList item:left itemCounter item:stop Stop
\AList\item Level one \AList\item Level two
\AList\item Level three\>]
```

1 Level one

A Level two

I Level three

11.4.3 Manipulating the indentation level

Every once in a while it can be useful to move to a next indentation level, or to return to a previous level. For this the two commands `\PushIndentLevel`and `\PopIndentLevel`are available. One application is for ‘interrupted lists’:

Example 11.5

```
\Itemize\item One
{\par\PopIndentLevel Interrupted text!\par}
\item Two\>
```

• One
Interrupted text!
• Two

See chapter 9 for examples of the use of `\PushIndentLevel`

11.5 Margins

By default, Lollipop tries to keep straight margins. You can change its mind about that by

```
\FlushRight:no \FlushLeft:no
```

If the margins are not flush, the stretchable white space used is `\rightmarginstretch`and `\leftmarginstretch`.

11.6 White Space

White space can be indicated by `\hwhite`and `\vwhite`. They are often useful in style definitions. Use:

```
\vwhite:15pt
```

or

```
\hwhite:{15pt minus 3pt}
```

for stretch and shrink. The command `\white`is independent of the mode, and it expands to `\hwhite` or `\vwhite` depending on the prevailing mode of TEX.

The command `\fillup`is mostly useful in style definitions: it tries to fill up as much white space as is possible. For instance

```
        line:start litteral:foo fillup litteral:bar
line:stop
```

will push `foo` and `bar` as far apart as is possibly within the margins.

Implementor's Note

All three control sequences `\white`, `\hwhite`, `\vwhite` have internal equivalents, for instance

```
\def\white:#1 {\@white{#1}}
```

11.7 Distances

The command `\Distance`can be used to declare a name for a certain distance, or in more correct TEXnical lingo, for a certain piece of glue. For instance, declaring that

```
\Distance:oneline=15pt
```

means that you can specify in some constructs

```
\DefineFoo:Bar whitebefore:oneline whiteafter:oneline
```

If you change your mind later about the value of `oneline` you only need to change one line in the style definition.

Since the second parameter of `\Distance` is bounded by a space (or the line end, whatever comes first), you can specify stretchable distances by enclosing `plus` and `minus` parts in braces:

```
\Distance:oneline={15pt plus 2pt minus 3pt}
```

Another use of `\Distance` is to define one distance as a synonym of another. This may come in handy if you use some basic distance, such as `oneline` for several purposes. Example: if you specify

```
\Distance:whitebefore=oneline
```

than the whitespace before a construct will be taken to be `oneline` if you don't use the `whitebefore` option explicitly.

11.8 Tests

Users can define tests:

```
\DefineTest:SomethingTheMatter
```
which are set like any other test:
```
\SomethingTheMatter:yes
```
or
```
\SomethingTheMatter:no
```
Tests can be used as
```
\ifSomethingTheMatter ... \else ... \fi
```
Like any other conditional, test can be used inside constructs.
```
\DefineFoo:Bar [...]
 ifSomethingTheMatter [...] fi
 [...] Stop
```

11.9 Goodies

The commands `\SaveAlloc`and subsequent `\RestoreAlloc`save and reset the internal TEX allocation counters.

Implementor's Note

Obscure goodies

I-*15 Dummy commands*

For purposes such as termination of an argument it is usefule to have control sequences that have a meaning different from any other control sequence. The command `\NewDummy`gives such control sequences.
The call `\NewDummy{some}` defines `\some`, or gives an error msg at redefinition.

Chapter 12

Tracing

12.1 Do you really want to see this?

You can get glimpses of Lollipop's internal workings by enabling some of the internal traces. The extreme positions

```
\Trace:yes
```

and

```
\Trace:no
```

cause all trace information and no trace at all respectively to be generated. You may find this trace interesting, or it may dumbfound you. Of course, if your name is Victor you find it pretty useful.

The following traces are available:

```
\NewTrace:def  % definition of user constructs
\NewTrace:ref  % cross references
\NewTrace:ext  % external files
\NewTrace:doc  % document structure
\NewTrace:font % font loading
\NewTrace:out  % output routine
\NewTrace:indent % indentation control
\NewTrace:gen  % general tools
```

Implementor's Note

Trace messages are generated by calls to

```
\Tmessage[type]{text}
```

Setting

```
\Trace:no
```

defines `\Tmessage` to discard its arguments. This is the most efficient way of generating no trace information.

Tracing is controlled by a global parameter `\trace@all`. A value of -1 disables all tracing; $+1$ corresponds to all tracing on; 0 gives selective tracing. In the third case a call

```
\Trace:xyz
```

set `\trace:xyz` positive so that only `\Tmessage[xyz]{...}` calls will produce output. Enabling a selective trace sets `\trace@all` to zero, in case it was -1.

Chapter 13

The style definition for this book

In case you were wondering how this book was typeset, here is the full style definition. By the standards of what Lollipop can do it is pretty pedestrian.

One thing that may have provide intellectual titilation is the definition of `\Example` and `\OutExample`. It allowed me to keep the examples in sync with their output.

Unfortunately that doesn't really rely on Lollipop. It does illustrate the fact that Lollipop is interfaceable to arbitrary macros. (But don't try loading Lollipop on top of LaTeX!)

```
\chardef\busje`\\
\def\cs#1{{\tt\char\busje#1}}
\def\con#1{{\tt#1}}
\def\n#1{{\tt#1}}
\def\file#1{{\tt#1}}

\def\Lollipop{Lollipop}

\Distance:rightmarginstretch={0cm plus 2cm}
\Distance:whitebefore={6pt plus 3pt minus 2pt}
\Distance:whiteafter=whitebefore

\DefineExternalFile:contents=toc

\DefineHeading:Chapter
    breakbefore:yes whiteafter:20pt
    line:start PointSize:14 Style:bold literal:Chapter
        Spaces:1 ChapterCounter line:stop
    vwhite:15pt
    line:start PointSize:16 Style:bold title line:stop
    external:contents title external:stop
    Stop

\DefineHeading:Section
    whitebefore:20pt whiteafter:14pt
    line:start PointSize:14 Style:italic
        ChapterCounter . SectionCounter
        Spaces:1 title line:stop
    external:contents title external:stop
    label:start ChapterCounter . SectionCounter label:stop
    Stop
```

```
\GoverningCounter:Section=Chapter

\DefineHeading:SubSection
    whitebefore:14pt whiteafter:8pt
    line:start PointSize:10 Style:italic
        ChapterCounter . SectionCounter . SubSectionCounter
        Spaces:1 title line:stop
    label:start ChapterCounter . Spaces:.2 SectionCounter
        . Spaces:.2 SubSectionCounter label:stop
    Stop
\GoverningCounter:SubSection=Section

\DefineExternalFile:impnotes=imp
\DefineHeading:iSection
    whitebefore:20pt whiteafter:14pt
    line:start PointSize:12 Style:bold I -
        Style:italic iSectionCounter
        Spaces:1 title line:stop
    label:start I - iSectionCounter label:stop
    external:impnotes title external:stop
    Stop
%\GoverningCounter:iSection=Chapter
\DefineExternalItem:iSection file:impnotes PushIndentLevel
    item:left I - Style:italic iSectionLabel item:stop
    title begingroup Spaces:2 Style:italic Page endgroup
    Stop

\DefineExternalItem:Chapter file:contents
    item:left ChapterLabel item:stop
    title begingroup Spaces:2 Style:italic Page endgroup
    Stop
\DefineExternalItem:Section file:contents PushIndentLevel
    item:left ChapterLabel . SectionLabel item:stop
    title begingroup Spaces:2 Style:italic Page endgroup
    Stop

\def\impnotetxt{Implementor's Note}
\DefineTextBlock:ImpNote PushIndentLevel
    line:start PointSize:12 Style:italic impnotetxt
line:stop
    SansFace PointSize:9 SetFont text
    Stop

\DefineTextBlock:WizNote
    PushIndentLevel PointSize:9 SetFont text
    Stop
```

```
\DefineList:Description
    item:left description Spaces:2 item:stop
whitebetween:6pt
    Stop

\DefineList:cDescription
    item:left tt char busje description Spaces:2 item:stop
    whitebetween:6pt
    Stop

\DefineList:Enumerate
    item:left itemCounter item:stop
    Stop

\DefineList:Itemize
    item:left itemsign item:stop
    Stop

\SerifFace \SetFont

\newwrite\exfile
\def\HereAndOut#1{\immediate\write\exfile{#1}}
\specialcomment{Example}
   {\EExample
    \immediate\openout\exfile=example.tex\relax
    \let\ThisComment\HereAndOut}
   {\immediate\closeout\exfile
    \begingroup \tt \SetFont
                \verbatimfile{example.tex}\endgroup
    \SaveAlloc \input example.tex\relax \RestoreAlloc
    \EExampleStop}
\DefineTextBlock:EExample whiteafter:{6pt plus 5pt}
    noimplicitclose hrule vwhite:3pt
    line:start literal:Example Spaces:1.5
                  ChapterCounter . EExampleCounter
                line:stop
    vwhite:3pt hrule vwhite:3pt text vwhite:3pt hrule
    Stop
\GoverningCounter:EExample=Chapter

\specialcomment{OutExample}
 {\EExample
    \immediate\openout\exfile=example.tex\relax
    \let\ThisComment\HereAndOut}
   {\immediate\closeout\exfile
```

```
    \begingroup \tt \SetFont
                \verbatimfile{example.tex}\endgroup
    \par\penalty0\relax
    \SaveAlloc \SuspendOutput \begingroup \CountSheetsno
        \SetCounter:Page=1
        \global\setbox\PageRow\hbox{}%
               \let\CurrentShipout\ToPageRow
        \xInputFile:example
        \endgroup
    \ResumeOutput \RestoreAlloc
    \noindent\unhbox\PageRow\hbox{}\par
    \EExampleStop}
\newbox\PageRow\newbox\RowPage
\def\ToPageRow{\afterassignment\xToPageRow\setbox\RowPage}
\def\xToPageRow{\global\setbox\PageRow
    \hbox{\unhbox\PageRow\box\RowPage\hfill}}

\def\opt#1{{\tt#1}}
\DefineExternalFile:optindex=oix
\def\refopt#1{\OptToIdx #1\par}
\DefineHeading:OptToIdx embedded:yes
    block:start tt title block:stop
    external:optindex title external:stop
    nomarks Stop
\DefineExternalItem:OptToIdx file:optindex
    embedded:yes
    begingroup tt title endgroup
    nobreak Spaces:1.5 Page Spaces:2.5 Stop

\DefineExternalFile:csindex=cix
\def\refcs#1{\CsToIdx #1\par}
\DefineHeading:CsToIdx embedded:yes
    block:start tt char busje title block:stop
    external:csindex title external:stop
    nomarks Stop
\DefineExternalItem:CsToIdx file:csindex
    embedded:yes
    begingroup tt char busje title endgroup
    nobreak Spaces:1.5 Page Spaces:2.5 Stop

\topskip20pt
\DefinePageGrid:LeftPage width:page=11cm height:page=20cm
    band:start block:start PointSize:9 Style:italic
               FirstPlaced:ChapterCounter Spaces:2
stickout:left
         FirstPlaced:ChapterTitle band:stop
    textband:start text textband:stop
    band:invisible block:start PointSize:9 Style:bold
         PageCounter Spaces:2 stickout:left band:stop
    NextPageGrid:RightPage Stop
\DefinePageGrid:RightPage width:page=11cm height:page=20cm
    band:start fillup PointSize:9 Style:italic
        LastPlaced:SectionTitle
        block:start Spaces:2 LastPlaced:ChapterCounter .
            LastPlaced:SectionCounter stickout:right
        band:stop
    textband:start text textband:stop
    band:invisible fillup
        block:start PointSize:9 Style:bold Spaces:2
            PageCounter stickout:right band:stop
    NextPageGrid:LeftPage Stop
\DefinePageGrid:EmptyPage width:page=11cm height:page=20cm
    textband:start text textband:stop
    NextPageGrid:LeftPage Stop

\endinput
```