Lollipop Unwrapped

Victor Eijkhout and Vafa Khalighi

A manual for the Lollipop TEX format,
last updated April 2014
Pages: ix+50=59.

Chapter 1

Contents

1.1 Regular sections

1.2 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 lines
“Writeopindex:no
“Writecsindex:no
somewhere in the top of the manual.tex file in order to prevent overwriting of these files after you’ve sorted them.)

1.3 All of the commands

1.4 Bibliography

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 it compatible?

Lollipop, like LATEX, is not compatible with plain TEX. I don't consider this a real problem. Most plain TEX commands will work in Lollipop with the exception of anything output routine related. (See also below.)

Lollipop is also not compatible with LATEX, although it has a lot of the same functionality. There are two reasons why Lollipop still has a reason for existing, even though LATEX is used pretty much all over the scientific world.

For one, Lollipop is targeted in part to different users than the typical plain TEX or LATEX user. For another, I have a vain hope that I can capture some of the LATEX market share. Since developing styles in Lollipop is so much more easier than in LATEX, I may stand a fighting chance.

2.2.1 Lollipop and plain TEX

Having said the above, I'll conceded that Lollipop is more compatible with plain TEX than with LATEX. You can use quite some plain TEX commands in Lollipop. However, stay away from the following:

"everypar This one is heavily used by Lollipop. You may use "EveryPar instead, which functions pretty much like "everypar; see section 11.10.3.

"output Page output is done so very differently from plain TEX that all commands pertaining to page numbers and head/footlines have been eradicated. (Well, "pageno still gives the page number.) See chapter 7.

The current version of Lollipop is based on the plain TEX file that comes with TEX version 3.0.

2.2.2 Lollipop and TEX programming

The tools in Lollipop allow you to program in a simple manner quite complicated macros. Still you may want to have some knowledge of ordinary TEX macro programming. If you are just starting in TEX you can pick up the basics from the book by Seroul and Levy [**??**], and after that there is the book by the original author of TEX [**??**] and my own TEX reference guide [**??**].

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. On some systems IniTEX is really called that, on others you may have to type tex/i or something like that. With Textures on the Macintosh you typeset using the 'VirTeX' format (you can load Lollipop on top of plain TEX if you have an old version, but you then have to ignore the error message about the "patterns command).

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 &lollipop test

or something like that, depending on local conditions (for Unix you will have to escape the ampersand).

2.4 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 &lollipopbookstyle
  *"dump

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

¿ tex &bookstyle book

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.

If you have used TEX before, you will notice that the page numbers get reported slightly differently from the usual way. See section 7.4 for the explanation.

2.5 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 (see section 3.4 for the possible origin of some of the more obscure error messages).

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 The table of contents example is slightly wrong.
4 Titles get written to the aux file with double spaces. This shouldn't cause any problem, but it has to be fixed.
5 Rules in page grids get white space around them.
6 External items shouldn't declare "FooTitle or "FooCounter.
7 "ToExternalFile doesn't work.

2.6 About this manual

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, and btxmac.tex for the BibTEX interface, but these are 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 and log 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. For the bibliography there are the BibTEX input file manual.bib and output file manual.bbl.

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.

Because of all the examples this manual takes quite some time to process. A factor of four over the time for a regular document of similar length should be expected. Ordinary Lollipop documents will proceed far faster.

2.7 The most boring section in this manual

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

2.7.1 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. And if you hurt yourself by using it, don't blame me. I warned you.

2.7.2 Get a Lollipop, give one away

Lollipop is free of charge. 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, the changed files should be given a different name, and the fact that they are changed should be clearly indicated. Although I retain the copyright, the rights for non-profit use of Lollipop are granted.

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.tar.Z. Starting with version 0.93 there is also a self-extracting Macintosh archive.

2.7.3 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

2.7.4 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.

1 Raggedbottom should really, really be added. Soon!
2 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. Some code has been disabled now.
3 A better multi-column mode.

4 Interface to BibTEX seems to largely in place; what happens if you don't load btxmac?

5 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.)

6 A 'nomarks' option to prevent wasting two token lists. Maybe other recourse saving optio for the expert designer?

7 More sophisticated white space right before and after page breaks. (Use at least so and so much.)

8 Dynamic topskip.

The following points are debatable: maybe I should just steal a few components from LATEX. Maybe this sort of stuff does not belong in Lollipop.

9 A tabular mode. Personally I always felt "halign to be more than sufficient, but some people seem to think otherwise.

10 Maths constructs. Some things in the "eqalign vein would be nice.

2.7.5 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; the first public release (version 0.9) was announced on the internet in October 1992. At present I am using Lightning Textures on a Macintosh Powerbook 145.

The name '"Lollipop' refers to a quote by Alan Perlis [**??**], quoted on page 365 of the TEXbook [**??**]. In a way it's rather pretentious. The philosophy of the "Lollipop format is described [**??**, **??**].

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 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. For a number of reasons it is advisable to put as much of the style definition before the "Start command as possible. You can do that easily by loading the style as an input file, or by first dumping it as a format (see section 2.4).

Both the start and the stop file load the .aux auxiliary file. None of this should concern you, really. Expert users who want to have certain actions performed at the start of the document may want to use "StartCommand to specify what they wish done. See section 11.7.2 for an example.

3.2 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

3.3 Options

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 with the command “OptionsMacro as follows. The options that appear together are given a common name

“OptionsMacro:baz=optiona optionb:value optionc
Stop

(be sure to leave no spaces around the equals sign) 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. However, it is only for your convenience: it doesn't save any TEX resources or processing time.

3.4 Popular error messages. Not!

Lollipop is a macro package on top of an existing program, TEX. Therefore it is inevitable that you will get TEX error messages every once in a while. Some of these may confuse you.

Here are a few of the errors that I keep making.

3.4.1 Missing “endcsname *inserted*

If you forget the second parameter in a “Distance or “SetCounter command, writing for instance

“Distance:TheWidth

instead of

“Distance:TheWidth=15pt

TEX will scan forward, and it can easily bump into something that is highly unexpected given the context. If this is a “def or “Define... command, a ‘missing “endcsname’ results. If a blank line follows the incomplete declaration, the following section applies.

3.4.2 Paragraph ended before something was complete

TEX has found a blank line (or a “par command) where this was not expected. See for instance the previous section.

3.4.3 Missing number

You have used something that you thought was the name of a control sequence, but it wasn't. Example:

“Distance:parskip=parundent

Since “parundent is undefined, Lollipop thought you were writing something like

“Distance:parskip=5pt

And yes, the message refers to ‘number’ even though what is missing is a distance.

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.
The title has to be included in a line or a textcolumn for proper handling (see also section 10. 3.5). For titles that do not exceed one line, the line option suffices (section 10.3.3); if a title is possibly more than one line long, the textcolumn option has to be used (section 10.3.4.

Example 4.1

```
“DefineHeading:TestSection Style:bold
   line:start TestSectionCounter Spaces:2 title line:stop
   Stop
“TestSection 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:TestSection Style:bold
   line:start TestSectionCounter Spaces:2 title line:stop
   indentafter:yes Stop
“TestSection 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:TestChapter Style:bold
   line:start TestChapterCounter Spaces:1 title line:stop
   Stop
“DefineHeading:TestSection Style:italic
   line:start TestChapterCounter : TestSectionCounter . Spaces:1
      title line:stop Stop
“GoverningCounter:TestSection=TestChapter

“TestChapter Level One Heading“par
“TestSection Level Two Heading“par
Some text.
“TestSection Level Two again“par
More text.
“TestChapter Level One is Stepped“par
“TestSection 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:TestSection
   [...]
   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.

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

C) 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:TestList
 item:left Style:bold itemCounter . Spaces:1 description
     Spaces:2 item:stop Stop
"TestList"item Do
A deer, a female deer."item Re
According to mr. Fowler only a legal term.
"item Mimi 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 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:TestList hrule line:start Style:bold title line:stop
 item:left Style:italic itemCounter item:stop
 text vwhite:3pt hrule Stop
“TestList 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.

Material before and after the text should usually not be broken. Hence "nobreak is automatically inserted. See the "Example macro for this manual: pages should not be broken after rules, or around the text 'example xyz'.

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.
```

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

```
"OptionsMacro:ManPageSize=raggedbottom:10pt topskip:12pt
 height:page=5cm width:page=6cm Stop
"DefinePageGrid:TestPage macro:ManPageSize
 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.

This page is hardly better.

11 12

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 macro:ManPageSize
 pagerule textband:start text hwhite:10pt text textband:stop
 pagerule band:start PageCounter band:stop Stop
“FlushRight:no “sometext
```

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

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

12

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 macro:ManPageSize
 pagerule textband:start text hwhite:10pt 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 words, words. Just a bit of words, words. Just a bit of words, words. Just a bit of words, words. Just a bit of words, words. Just a bit of words, words. Just a bit of words, words. Just a bit of words, words. Just a bit of words, words. 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 macro:ManPageSize 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.

13

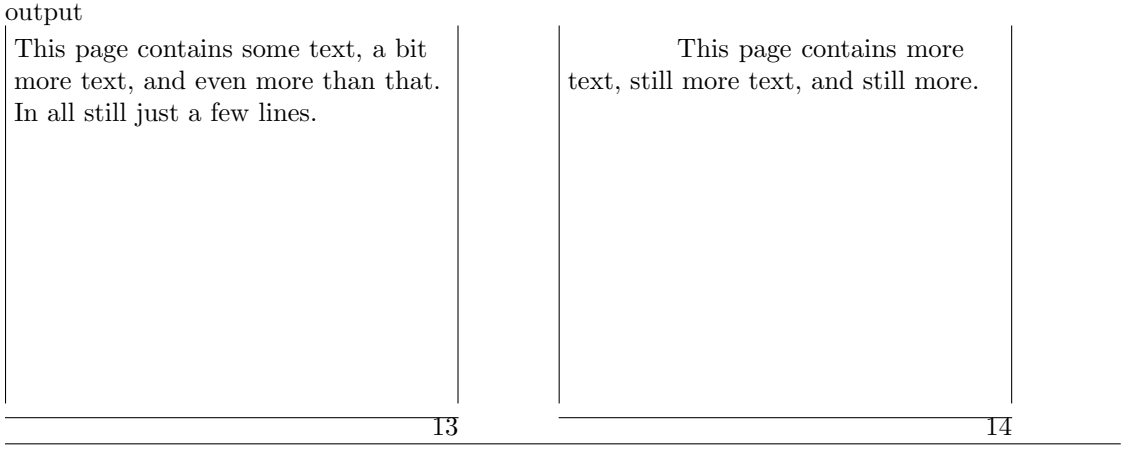
This page contains more text, still more text, and still more.

14

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 macro:ManPageSize
 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.
```



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.

If you process a Lollipop document you see that everytime a page is generated, an item such as [8,7] is written on the log file or the screen. Most of the time the two numbers will be the same, as in [8,8], but they will differ if you have tinkered with the page number. The first number is the 'sheet counter': it counts how many pages you have produced so far. The second number is the value of "PageCounter for the page that was written out. Take a look at the log file for this manual for an example.

7.5 Page tests

The page grid definition can set/query several properties of the page. The following tests have been provided (see section 11.9 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 macro:ManPageSize
 pagerule textband:start text textband:stop pagerule
 band:start ifIsLeftPage else hwhite:fillup fi PageCounter
    band:stop Stop "SetCounter:Page=12
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.

12 13

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 macro:ManPageSize
 pagerule textband:start text textband:stop pagerule
 band:start Style:italic FirstPlaced:HeadTitle
        white:fillup PageCounter band:stop Stop
"DefineHeading:TestHeading Style:bold
 line:start TestHeadingCounter Spaces:2 title line:stop Stop
"TestHeading A first section"par And some text."EjectPage
```

This page contains text. “TestHeading A second Section“par
And more text.

1 A first section

And some text.

This page contains text.

2 A second Section

And more text.

16 *17*

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 macro:ManPageSize
 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 “SetCounter:Page=10
“DefineHeading:TestHeading Style:bold
 line:start TestHeadingCounter Spaces:2 title line:stop Stop
“TestHeading A first section“par And some text.
“TestHeading Second section“par More text.“EjectPage
“TestHeading Third section“par Is on the right page.
“TestHeading Fourth section“par Concludes this page.
```

1 A first section

And some text.

2 Second section

More text.

3 Third section

Is on the right page.

4 Fourth section

Concludes this page.

10 *11*

7.7 Alternating page grids

In Lollipop it is very easy to switch page grids with the option NextPageGrid: you simply specify
16 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 macro:ManPageSize
 pagerule textband:start text textband:stop pagerule
 band:start Style:italic
    PageCounter white:fillup FirstPlaced:HeadTitle
    band:stop NextPageGrid:RTestPage Stop
“DefinePageGrid:RTestPage macro:ManPageSize
 pagerule textband:start text textband:stop pagerule
 band:start Style:italic
    LastPlaced:HeadTitle white:fillup PageCounter
    band:stop NextPageGrid:LTestPage Stop
“SetCounter:Page=42
“DefineHeading:TestHeading Style:bold
 line:start TestHeadingCounter Spaces:2 title line:stop Stop
“LTestPage
“TestHeading A first section“par And some text.
“TestHeading Second section“par More text.“EjectPage
“TestHeading Third section“par Is on the right page.
“TestHeading Fourth section“par Concludes this page.
```

1 A first section

And some text.

2 Second section

More text.

42

3 Third section

Is on the right page.

4 Fourth section

Concludes this page.

43

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 reverts the effect of previous command. Note that “NoPages does not incur any savings in time: full processing of the document is performed.

When a page is finished it rests in box “WholePage. Then a call is made to “CurrentShipout, which is by default “shipout“box“WholePage. However, you are free to define it otherwise. 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, including the current definition of “CurrentOutput. (This is necessary because a number of parameters concerned are changed by global assignments.) See the definition of “OutputExample in the appendix of this manual for an elaborate example.

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:TestSection
 line:start Style:italic TestSectionCounter Spaces:2 title
 line:stop Stop
"TestSection[one:section?] First section"par
After this section will come section~"ref[other:section!].

"TestSection[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
```

You can then reference the key by "ref, or the page where it appeared by "pgref:

```
Section "ref[this:section] on page "pgref[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. There are two ways in which you can change this, one systematic, and one on-the-fly.

8.2.1 Defining the shape of the 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:TestSection
 line:start Style:italic TestSectionCounter Spaces:2 title line:stop
 label:start ( TestSectionCounter ) label:stop Stop
“TestSection[one:section?2] First section“par
After this section will come section~“ref[other:section!2].

“TestSection[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:TestChapter
 line:start Style:bold TestChapterCounter / title line:stop Stop
“DefineHeading:TestSection
 line:start Style:italic TestSectionCounter Spaces:2 title line:stop
 label:start TestChapterCounter .
          TestSectionCounter label:stop Stop
“TestChapter First chapter“par
Pretty short chapter
“TestChapter Second chapter“par
“TestSection[one:section?3] First section“par
After this section will come section~“ref[other:section!3].

“TestSection[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:TestSection
 line:start Style:italic TestSectionCounter Spaces:2 title line:stop
 label:start TestSectionCounter literal: Spaces:1
          Style:italic title label:stop Stop
“TestSection[one:section?4] First section“par
After this section will come section~“ref[other:section!4].

“TestSection[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*.

8.2.2 Explicit specification of the reference

For every specific object referenced you can specify the reference by using an optional argument before the label key. Have a look at the next example.

Example 8.5

```
“DefineHeading:TestSection counter:A
 line:start Style:bold TestSectionCounter . Spaces:2 title line:stop
 Stop
“TestSection[ref:one] Lalala“par
This is before section “ref[ref:two].
“TestSection¡‘the Didi section’¿[ref:two] Dididi“par
This is after section “ref[ref:one].
```

A. Lalala

This is before section ‘the Didi section’.

B. Dididi

This is after section A.

This mechanism is also used in lists, where it’s mostly useful for bibliographies generated by BibTEX; see section 8.4.2.

8.3 Local references

Some documents are collated out of parts that were documents in themselves. The Lollipop command “InputFile facilitates in a number of ways working with such a segmented file (see section 11.8). One of the problems with input files is that in such a case it may happen that the

same reference key is used in more than one part of the document. This phenomenon is not at all unknown in multiple authored documents. 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 for each input file.

8.4 Bibliography citations

Lollipop has an interface to BibTEX. However, since a bibliography is just a list, referencing items in it is quite easy, even if you don't use BibTEX.

8.4.1 Bibliographies without BibTEX

This section doesn't tell you anything that cannot be found elsewhere in this manual. The following two examples define a bibliography as just a list, and by giving labels to the items you can refer to them.

Example 8.6

```
“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 8.7

```
“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.

8.4.2 Bibliographies with BibTEX

Lollipophas an interface to the popular BibTEX bibliography database program, based on the 'BtxMac' macros by Karl Berry and Oren Patashnik. Lollipop is set up for them, you only have to "input the file btxmac.tex. (The version of btxmac used to test this is 0.99h.) You can find global information about BibTEX in the LATEX book [**??**], since BibTEX was originally written for LATEX.

Since there is some redefining going on between btxmac and Lollipop you have to load the btxmac file before the "Start command (see section 3.1).

Since the btxmac file already has a default way of formatting the bibliography you can get away with just putting the lines
"bibliography– ¡bfile¿ ″
"bibliographystyle– ¡bstyle¿ ″
in your file wherever you want the bibliography.

If you want to define your own bibliography, you have to use "DefineBBL which is practically a synonym for "DefineList:BBL, so you can see in the 'lists' chapter of this manual what options apply.

For example:
"DefineBBL line:start Style:italic literal:Literature line:stop
item:left [begingroup Style:bold itemCounter
endgroup] item:stop
Stop
You refer to a bibliography item by "bibref, as in "bibref[Knuth80]. This command has a very simple definition
"def"bibref[#1]–["ref[#1]]"nocite–#1″″
so you can easily redefine it. For instance
"def"supref[#1]–$ˆ–"rm "ref[#1]″$"nocite–#1″″
will make your references into superscripts. Make sure that the call to "nocite appears, because that generates the request for BibTEX.

The Lollipop command "WriteExtern:no (see section 9.1) defines "noauxfile to prevent regeneration of the bib entries in the .aux file. You don't have to do that anymore.

8.5 Obscure details

For the sake of efficiency, not all macros in Lollipop automatically accept labels for referencing, only the ones that use the label option, or that have a counter (remember, the default form of the reference is just the counter). If you want a macro that has no counter and no label specification to accept a label, use the option haslabel. One reason for doing this is that you have access to the label itself through the control sequence "RefLabel.

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.

You can have at most 15 external files per document.

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. (The declaration of an external file has to come before any calls to "ExternalItem or any options external that write to this file.)

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.

If you write more than just title to an external file, know that any control sequence you specify is automatically protected from expansion. See section **??** for an exception to this.

Other titles can be included by specifying title:OtherThing. Using OtherThingTitle would not work, because of the protection of control sequences mentioned above.

You can write arbitrary information to an external file, if you see a reason to do so, by “ToExternalFile, which takes a file name and an text argument. The example below has an instance of this command. In order to format this information you have to define an external construct of type “anon.

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.

Chapter 10

Options

This chapter discusses the various options that are common to all Lollipop constructs.

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. For instance, you define
“DefineBar:Foo ...
HasTitle
ComplicatedTitle
... Stop
where “ComplicatedTitle is a macro that formats the title.

10.1.1 Short titles

The option HasTitle can have a parameter short, denoting that the title is not delimited by an empty line (or “par) but by the line end. For an example see section 13.2.

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, the option HasCounter will cause the counter to be created anyhow. This is analogous to the HasTitle option.

10.2.1 Counter synonyms

Every once in a while you may want different constructs to use the same counter. For instance, if your book has definitions, theorems, lemmas, corollaries and notations, it may confuse the reader if they all have their own counter. The numbering would seem to jump all over the place.

A 'counter synonym' can be declared in Lollipop by a slight abuse of the counter option.

Example 10.1

```
“DefineTextBlock:Theorem counter:1 begingroup Style:bold
 literal:Theorem Spaces:1 TheoremCounter Spaces:2 endgroup
 text Stop
“DefineTextBlock:Corollary counter:Theorem begingroup Style:italic
 literal:Corollary Spaces:1 CorollaryCounter Spaces:2 endgroup
 text Stop
“DefineTextBlock:Definition counter:Theorem begingroup Style:roman
 literal:Definition Spaces:1 DefinitionCounter Spaces:2 endgroup
 text Stop
“Definition We define a –“it Foo” to be an arbitrary object“¿
“Theorem Foos have arbitrary properties“¿
“Corollary Foos are extremely valuable“¿
“Corollary Foos are extremely worthless“¿
“Theorem Foos don’t exist“¿
```

Definition 1 We define a *Foo* to be an arbitrary object

Theorem 2 Foos have arbitrary properties

Corollary 3 Foos are extremely valuable

Corollary 4 Foos are extremely worthless

Theorem 5 Foos don’t exist

You can only declare a counter to be synonym for something that has already been created. In the above example you cannot define the “Theorem after the “Corollary.

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 10.3.4.

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 10.2

```
“DefineHeading:TestHeading
 line:start block:hang PointSize:8 SetFont
                TestHeadingCounter fillupto:20pt
            block:hang PointSize:14 SetFont title block:stop
 line:stop Stop
“TestHeading Top Aligning the Title
```

1 Top Aligning the Title

The block is usually in between the margins of the text, but it can be made to stick out into the margin, by closing it with the option stickout. 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 Block Measuring

All blocks are immediately measured when they are placed. This makes it possible for instance to underline a title exactly. After a block has been placed, its width is available as “BlockWidth.

Example 10.3

```
“def“rulespecs–height 1pt width “BlockWidth“relax″
“DefineHeading:TestHeading Style:bold
 line:start block:start TestHeadingCounter . Spaces:2 stickout:left
          block:start title block:stop line:stop
 vwhite:2pt hrule rulespecs vwhite:10pt Stop
“TestHeading The Title Is Underlined
```

The text follows.

1. The Title Is Underlined

The text follows.

Observe how a control sequence “rulespecs is used to sneak the height and width of the rule into the definition. This is necessary because control sequences are not allowed in a construct definition.

10.3.3 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 10.4

```
“DefineHeading:TestHeading
   line:start block:start TestHeadingCounter Spaces:1.5 stickout:left
           title line:stop Stop
“TestHeading A Title
```

1 A Title

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

Example 10.5

```
“DefineHeading:TestHeading
   line:start TestHeadingCounter fillup title line:stop Stop
“TestHeading The title
```

1 The title

10.3.4 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 10.6

```
“DefineHeading:TestHeading
   line:start block:start TestHeadingCounter Spaces:2 block:stop
           textcolumn:topline title textcolumn:stop
   line:stop Stop
“TestHeading 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 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.)

10.3.5 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 10.7

```
“DefineHeading:TestHeading
   block:start TestHeadingCounter Spaces:2 block:stop
   title Stop
“TestHeading 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:TestSection
 line:start Style:italic TestSectionCounter Spaces:2 title line:stop
 label:start ( TestSectionCounter ) 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 Rules

There is an option hrule. You should not write

```
“def“rulemacro–“hrule height [...] ″
“DefineHeading [...] rulemacro [...] Stop
```

because then Lollipop cannot prevent page breaks around the rule. Instead write

```
“def“rulespecs– height [...] ″
```

“DefineHeading [...] hrule rulespecs [...] Stop
so that the option hrule is used. See section 10.3.2 for an example.

10.8 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 10.8

```
“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?

A word of warning though. Headings and such generate TEX ‘marks’ which take up main memory. You only need these if you are going to be referring to that object with “LastPlaced or such; see section 7.6. Embedded headings used as above usually don’t need these marks, so you can prevent TEX overflow by putting the option nomarks in their definition.

10.9 Obscure options

10.9.1 Arguments

In case you want to interface to other macro packages, you may want to let a construct generate a call to the other package. For this, the Lollipop macro should be able to produce sequences such as “begin–itemize″. The problem here is the braces. The option “arg produces a braced expression.

For instance

```
“DefineTextBlock:LaTeXlist begin arg:–itemize″ text
   end arg:–itemize″ Stop
```

makes it possible to use call LaTeX macros from Lollipop.

10.9.2 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. Using “¿] inside such a construct will close all enclosed constructs.

See the definition of “EExample in this manual for an example.

10.10 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:

With “StartCounter reset the counter to one:
“StartCounter:things
With “StepCounter increase the counter by one:
“StepCounter:things
With “BackStepCounter decrease the counter by one:
“BackStepCounter:things
With “SetCounter set the counter to some specified value:
“SetCounter:things=5

With “AddToCounter increase the counter by some specified value:
“AddToCounter:things=7
The last two commands accept names of numerical parameters, for instance the value of a counter:
“NewCounter:Favourite
“AddToCounter:things=FavouriteValue
More about this ‘value’ thing in the next section.

11.1.3 What do you get when you define a counter?

After you define a counter as
“NewCounter:MyThings
there are two commands that it is important to distinguish between. First of all, “MyThingsCounter gives the *printed* value of the counter. This is the command that you will use mostly. It uses whatever representation you have specified for the counter, or plain Arabic numerals if haven't specified anything.

Secondly, “MyThingsValue gives the value of the counter. You can not use this command on its own: you will get a ‘number expected’ error.

11.1.4 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.

11.1.5 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

11.1.6 Adaptive counters

It may happen that you want to compute a value during one run of TEX, and use it in the next. An example is the fact that this manual states the total number of pages on the title page. For this you can use “AdaptiveCounter
“AdaptiveCounter:LastPage
which is like “NewCounter, except that the value of this counter gets written to the .aux file.

There are two ways of setting an adaptive counter: you can simply use it in some construct (for instance through a counter synonym; section ref[sec:counter:repr]), or you can set it explicitly, for instance as
“SetCounter:LastPage=PageValue
See how this manual does it.

11.1.7 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 ...

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 “Fsize, 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“Fsize¡7 ti7“fi“fi
“ifStyle:tty “ifnum“Fsize¡8 tt8“fi“fi]
You may have noticed that this scheme is not all-powerful. Thus I found it easier to move all Computer Modern sans serif fonts into a new typeface: ComputerSans.

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 “Fsize pt;″
“makeatother

11.2.3 Math fonts

Switching styles in math mode should be possible:

11.2.4 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
This has the same effect as
“def“name–“TypeFace:face “PointSize:size “Style:style ″
but it takes considerable less processing inside TEX.

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 Basic indent

There is a quantity “basicindent that is used on the first indentation level (see the next section for an explanation of these levels). At the start of a document it is set to the then current value of “parindent. You can override that by “BasicIndentIsSet: give
“BasicIndentIsSet:no
before the “Start command.

This way, setting “parindent in the style definition controls the indentation in the whole document.

11.4.3 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.

By default the indentation on different levels is a fraction of the “basicindent. Thus you can regulate the indentation on all levels simultaneously by resetting the “basicindent.

Example 11.3

```
“Distance:basicindent=15pt
“DefineList:TestList item:left itemCounter item:stop Stop
“TestList“item Level one “TestList“item Level two
“TestList“item Level three“¿]
“Distance:basicindent=25pt
“TestList“item Level one “TestList“item Level two
“TestList“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:TestList item:left itemCounter item:stop Stop
“TestList“item Level one “TestList“item Level two
“TestList“item Level three“¿]
```

1 Level one

A Level two

I Level three

In fact, sometimes you may want to know the name of the indentation on a certain level. This is a control sequence such as “levelindentii for the second level. You get the idea.

11.4.4 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 and “FlushLeft which are tests:
“FlushRight:no “FlushLeft:no
If the margins are not flush, the stretchable white space used is “rightmarginstretch and “leftmarginstretch, which can be set by “Distance.

You have to set the amount of stretch *before* specifying that the margins will not be flush. The “FlushRight/Left commands take the current value whenever they are called.

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.

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″
The effect of “Distance is global. Let me know if you don’t like it.

11.7.1 Distance synonyms

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.

11.7.2 Adaptive distances

Suppose you want to declare a section heading as
“DefineHeading:Section ...
  block:start [...] fillupto:widelabel title
where “widelabel is the width of the widest label that occurs in your document. This requires just a tad of TEX programming. Just copy the details from the example below, which is the definition of “Section in this manual.

By declaring something a “AdaptiveDistance instead of just “Distance its value gets written to the .aux file at the end of the run, and restored in the next run. The second argument is simply the default value, in case you don’t have an auxiliary file yet.

```
“AdaptiveDistance:WidestLabel=15pt
“def“MeasureLabel–“ifdim“BlockWidth¿“WidestLabel
   “global“WidestLabel“BlockWidth“fi″
“DefineHeading:Section
   whitebefore:–20pt plus 2pt″ whiteafter:14pt
   line:start PointSize:14 Style:italic
      block:start block:start ChapterCounter . SectionCounter
           Spaces:1 block:stop MeasureLabel
           fillupto:WidestLabel
      title line:stop
   external:contents title external:stop
   label:start ChapterCounter . SectionCounter label:stop
   Stop
```

Note how two nested blocks are used: the first is to measure the label, and the width is written to the adaptive distance by means of a small macro; the second block is to fill out the white space.

If you want the paragraph indentation to depend on this adaptive width, you can give
“StartCommand–“Distance:parindent=WidestLabel ″
to set “parindent at the start of the document. See section 3.1 and 11.4.2.

11.8 Input Files

Parts of a document can be loaded by
“InputFile:parta
“InputFile:partb
et cetera. A document part loaded by “InputFile always starts on a new page. In section 8.3 it was already explained how local references for such files can be created.

Perhaps most importantly, 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.

11.9 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

11.10 Goodies

11.10.1 List commands

Lollipop does a lot of list processing internally, and a few of the commands have been made available to the user.

“NewList creates a list, and sets it to empty:
“NewList:mylist

“EmptyList just empty a list.

“TheList inserts the list. This will typeset the contents of it.
“TheList:mylist

“AppendToList adds data to a list.
“AppendToList:mylist=–my data″
The data are terminated by a space or line end, hence the braces.

11.10.2 Programming Tools

A few commands are useful for the Lollipop style designer who wants to write more sophisticated macros (see for instance the address book macros in the last chapter).

“UndefinedCS is a test on control sequences.
“if“UndefinedCS–testcs″ ... “else ... “fi

“EqualString tests equality of strings.
“if“EqualString–one″–two″ ... “else ... “fi

“EqualStringX tests equality of strings, using only expansion.
“if“EqualStringX–one″–two″ ... “else ... “fi

“StringBefore tests lexicographic ordering of strings. The string are not supposed to contain characters with char codes 0, 127, 255.
“if“StringBefore–one″–two″ ... “else ... “fi

“NextChar chooses between two actions, based on the next character.
“if“NextChar[–“macro″–“macro[default]″ ...

“IsEmptyList can test whether an argument is empty.
“if“IsEmptyList–#1″ ...
is true for calls such as “macro–″.

“loop can be used for repeated execution of statements. (Users of plaing TEX may recognize this macro; it is slightly extended here to include the “else case.) It is used as:
“loop ... “if ... “repeat
of
“loop ... “if ... “else ... “repeat

11.10.3 “everypar

The TEX primitive “everypar should note be used any more. Instead use the command “EveryParagraph as if you are setting a token list:
“EveryParagraph– ... ″

11.10.4 Allocation

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

92/11/18 Adaptive distances explained 92/11/20 Adaptive counters explained 93/05/13 changes in font macros

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

Chapter 13

Example styles

To show you the strength of Lollipop, this chapter collects a few example style definitions. The first one is that of this manual.

13.1 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.

Of course 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! On second thought, do. It disables most of LaTeX. Just kidding.)

```
% Mandefs.tex style definition for the Lollipop manual
% copyright 1992/3 Victor Eijkhout
% copyright 2014 Vafa Khalighi
%
“def“con#1—“tt#1″″
“def“n#1—“tt#1″″
“def“file#1—“tt#1″″

“def“Lollipop–Lollipop″

“Distance:rightmarginstretch=–0cm plus 2.54cm″
“Distance:whitebefore=–6pt plus 3pt minus 2pt″
“Distance:whiteafter=whitebefore
“FlushRight:no

“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

“AdaptiveDistance:WidestLabel=15pt
“def“MeasureLabel–“ifdim“BlockWidth¿“WidestLabel
   “global“WidestLabel“BlockWidth“fi″
“StartCommand–“Distance:parindent=WidestLabel ″
“DefineHeading:Section
   whitebefore:–20pt plus 2pt″ whiteafter:14pt
   line:start PointSize:14 Style:italic
      block:start block:start ChapterCounter . SectionCounter
            Spaces:1 block:stop MeasureLabel
            fillupto:WidestLabel
      title line:stop
   external:contents title external:stop
   label:start ChapterCounter . SectionCounter label:stop
   Stop
“GoverningCounter:Section=Chapter

“DefineHeading:SubSection
```

```
    whitebefore:–14pt plus 2pt″ 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:–10pt plus 1pt″ whiteafter:8pt
    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
    whitebefore:12pt whiteafter:11pt
    line:start PointSize:12 Style:italic impnotetxt line:stop
    SansFace PointSize:9 SetFont text
    Stop
“excludecomment–ImpNote″

“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

“DefineBBL
  item:left [ - itemCounter - ] 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 rule:h vwhite:3pt
    line:start literal:Example Spaces:1.5
                  ChapterCounter . EExampleCounter
              line:stop
    vwhite:3pt rule:h vwhite:3pt text vwhite:3pt rule:h
    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
       “global“setbox“PageRow“hbox–”%
            “let“CurrentShipout“ToPageRow
       “xInputFile:example
       “endgroup
    “ResumeOutput “RestoreAlloc
    “noindent“unhbox“PageRow“hbox–”“par
    “EExampleStop”
“newbox“PageRow“newbox“RowPage
“def“ToPageRow–“setbox“RowPage“box“WholePage “xToPageRow”
“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
“OptionsMacro:PageDims=width:page=15cm height:page=23cm Stop
“DefinePageGrid:LeftPage macro:PageDims
   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 macro:PageDims
   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 macro:PageDims
   textband:start text textband:stop
   NextPageGrid:LeftPage Stop

“AdaptiveCounter:LastInPage “CounterRepresentation:LastInPage=i
“AdaptiveCounter:LastRegPage “CounterRepresentation:LastRegPage=1
%“SetCounter:LastInPage=PageValue – at the end of prelim.tex
%“SetCounter:LastRegPage=PageValue – at the end of manual.tex

“endinput

92/11/05 stretch added
92/11/18 adaptive label width
92/11/19 adaptive last page
```

13.2 Address book

The following macros generate an address book. Several noteworthy features:

- Most titles are short, that is, delimited by the line end.
- Since a page will now have several dozens of headings, the number of marks placed will become a problem, therefore the option nomarks is included everywhere. Without this you would easily have memory overflows.
- The “At heading writes its information to an external file. This is then parsed by the macro “CompNam. A slight amount of knowledge of Lollipop internals is used here for parameter parsing, but not more than can be gleaned from simply looking at the external file.

  Then a token list is created for each company, and these lists are printed somewhere down the file. This is a bit of TEX programming that is not quite elementary, but still Lollipopsaves you a lot of work.

If you want to see the output, run TEX with Lollipop twice on the address.tex file.

```
% Address book macros
% copyright 1993 Victor Eijkhout
% copyright 2014 Vafa Khalighi
%
% These macros are based on the Lollipop macro package,
% copyright 1992/3 Victor Eijkhout
% copyright 2014 Vafa Khalighi
%
```

```
% Format this file twice to get external files right.
%

% Some general options.
% Please use another typeface if you have it!
%
“Distance:whitebefore=0pt “Distance:whiteafter=0pt
“AlwaysIndent:no “FlushRight:no
%“TypeFace:macHelvetica “PointSize:7 “Style:roman
“TypeFace:ComputerSans “PointSize:8 “Style:roman
“def““–,“Spaces:1 ″

% The page is four columns, no footers etc
%
“DefinePageGrid:ThePage width:page=6.5in height:page=5in
 textband:start text white:10pt text white:10pt
   text white:10pt text textband:stop Stop

% The main macro is the text block “Entity, representing one item
% in the address book.
%
“DefineTextBlock:Entity HasTitle:short
 whitebefore:–6pt plus 6pt minus 3pt″ whiteafter:whitebefore
 line:start rule:v=–height 7pt width7pt depth0pt″ Spaces:2
      Style:italic title line:stop Stop

% Data in an entity is formatted as a number of headings.
% We declare all headings to be embedded, ie, they form a paragraph.
% The 'nomarks' option is to prevent memory overflow.
%
“OptionsMacro:embed=embedded:yes nomarks
 whitebefore:0pt whiteafter:0pt Stop
“DefineHeading:phone HasTitle:short macro:embed
 literal:phone : Spaces:1.5 title , Spaces:2 Stop
“DefineHeading:fax HasTitle:short macro:embed
 literal:fax : Spaces:1.5 title , Spaces:2 Stop
“DefineHeading:Address HasTitle:short macro:embed
 title , Spaces:2 Stop
“DefineHeading:Note HasTitle:short macro:embed
 ( title ) Spaces:2 Stop

% Here's an example of non-standard handling of the title:
% email addresses are set in “tt, and can be broken differently
%
“def“breakemail–“hyphenchar“font=‘. ″
“DefineHeading:email HasTitle:short macro:embed
 begingroup tt breakemail title endgroup , Spaces:2 Stop

% This is invisible information; maybe later we'll do something
% with it.
“DefineHeading:Route HasTitle:short Stop

% Here is the first really cute application.
% If a person is declared to be “At a certain company, then
% that fact is written out to an external file, which can be loaded
% later.
%
“DefineExternalFile:companies=ats
“def“WorksAt–Works at: ″
“DefineHeading:At macro:embed
 HasTitle:short WorksAt title Spaces:2
 external:companies title title:Entity external:stop Stop
% When the 'companies' file gets loaded the title only gets
% parsed by the macro “CompNam, and nothing else happens.
% The title is split into company and person; #1 becomes the name
% of a list to which #2 is added.
```

```
%
“def“CompNam#1“unhskip#2“unhskip–
  “if“UndefinedCS–#1″“NewList:–#1″ “fi
  “AppendToList:#1=–#2“unhskip, ″ ″
“DefineExternalItem:At file:companies
  expandafter CompNam title Stop
% The company lists are later simply loaded; see below.

% Even more complicated: birthdays.
% All birthdays are written to an external file.
%
“DefineExternalFile:births=brt
“DefineHeading:birthday HasTitle:short macro:embed
  literal:Born : title Spaces:2
 external:births title title:Entity external:stop Stop
% Later every month becomes something like an entity;
% here is how we generate a token list for each month.
%
“def“month#1–“ifcase#1“or jan“or feb“or mar“or apr“or may“or jun“or
  jul“or aug“or sep“or oct“or nov“or dec“fi″
“tempcounta=1 “loop“ifnum“tempcounta¡13
  “xp“NewList“xp:“xp–“month“tempcounta″
  “advance“tempcounta1 “repeat
% Formatting the caboodle means:
% – loading the 'births' file
% – formatting each month separately by “OneMonth
%   which is essentially a call to the text block “Month
%
“def“AllBirths–“LoadExternalFile:births
  “tempcounta=1 “loop“ifnum“tempcounta¡13
  “OneMonth “advance“tempcounta1 “repeat″
“def“OneMonth–“xp“Month“month“tempcounta“par
  “TheList:–“month“tempcounta″ “¿″
% Month is much like “Entity, just a bit different visually.
%
“DefineTextBlock:Month
 whitebefore:–6pt plus 6pt minus 3pt″ whiteafter:whitebefore
 line:start rule:v=–height 7pt width7pt depth0pt″ Spaces:2
      Style:italic title Spaces:2
      fillup rule:v=–height 7pt width7pt depth0pt″ line:stop
 Stop
% When the external file is loaded, every birthday is processed
% by “BirNam which splits it into #1 year #2 month #3 day and
% #4 person's name. This is then written to the list for the
% appropriate month.
%
“def“BirNam#1 #2 #3“unhskip #4“unhskip
 –“tempcounta#2“relax
  “AppendToList:–“month“tempcounta″=–“JollyFellow #3 #4 (#1)“par″
  ″
“DefineExternalItem:birthday file:births
  expandafter BirNam title Stop
“DefineHeading:JollyFellow title nomarks Stop

% Phooeew! Now we can get down to business!

“WriteExtern:yes
“Start
“LoadExternalFile:companies

“Entity Adam Aardvark
“Address Page˜1, English Language Dictionary
“¿

“Entity Barbara Beeton
“At Tugbt
```

“Note Editor in chief
“¿

“Entity Jane Doe
“email doe@re.mi.sol
“¿

“Entity John Doe
“phone +1 212 555 4141
“¿

“Entity Victor Eijkhout
“Address Department of Computer Science, University of Tennessee
“phone +1 615 974 8298
“At Tugbt
“Note merely associate editor
“email eijkhout@cs.utk.edu
“birthday 1959 11 29
“¿

“Entity Elvis Presley
“birthday 1938(?) 01 08
“¿

“Entity Tugboat
“TheList:Tugbt
“¿

“AllBirths

“Stop