

KOMA-Script

a versatile \LaTeX 2 $_{\epsilon}$ bundle

The Guide

KOMA - Script

Markus Kohm

Jens-Uwe-Morawski

2007-10-12

Authors of the KOMA-Script Bundle: Frank Neukam, Markus Kohm, Axel Kielhorn

Legal Notes:

There is no warranty for any part of the documented Software. The authors have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained here.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the authors were aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

English translation of this manual by: Kevin Pfeiffer, Gernot Hassenpflug, Markus Kohm, Jens-Uwe Morawski, Karl-Heinz Zimmer, Christoph Bier, Harald Bongartz, Georg Grandke, Raimund Kohl, and Stephan Hennig.

Free screen version without any optimization of paragraph and page breaks

This guide is part of KOMA-Script, which is free under the terms and conditions of L^AT_EX Project Public License Version 1.3b. A version of this license, which is valid to KOMA-Script, is part of KOMA-Script (see `lpp1.txt`). Distribution of this manual—even if it is printed—is allowed provided that all parts of KOMA-Script are distributed. Distribution without the other parts of KOMA-Script needs a explicit, additional authorization by the authors.

To All Friends of Typography!

Contents

1	Introduction	13
1.1	Preface	13
1.2	Structure of the Guide	13
1.3	History of KOMA-Script	14
1.4	Special Thanks	14
1.5	Legal Notes	15
1.6	Installation	15
1.7	Bugreports and Other Requests	15
1.8	Additional Information	16
2	Construction of the Page Layout with typearea	17
2.1	Fundamentals of Page Layout	17
2.2	Page Layout Construction by Dividing	19
2.3	Page Layout Construction by Drawing a Circle	20
2.4	Options and Macros to Influence the Page Layout	20
2.5	Options and Macros for Paper Format Selection	31
2.6	Odd Bits without Direct Relevance to Text Layout	33
2.7	Local Defaults in the File <code>typearea.cfg</code>	34
2.8	Hints	34
3	The Main Classes <code>scrbook</code>, <code>scrreprt</code> and <code>scrartcl</code>	37
3.1	The Options	37
3.1.1	Options for Compatibility	39
3.1.2	Options for Page Layout	40
3.1.3	Options for Document Layout	41
3.1.4	Options for Font Selection	45
3.1.5	Options Affecting the Table of Contents	46
3.1.6	Options for Lists of Floats	49
3.1.7	Options Affecting the Formatting	50
3.2	General Document Characteristics	52
3.2.1	Changing Fonts	53
3.2.2	Page Style	56
3.3	Titles	63
3.4	The Table of Contents	69
3.5	Lists of Floats	72

3.6	Main Text	72
3.6.1	Separation	72
3.6.2	Structuring the Document	73
3.6.3	Footnotes	87
3.6.4	Lists	89
3.6.5	Margin Notes	100
3.6.6	Tables and Figures	101
3.6.7	Logical Markup of Text	113
3.7	Appendix	114
3.8	Obsolete Commands	117
4	Adapting Page Headers and Footers with <code>scrpage2</code>	119
4.1	Basic Functionality	119
4.1.1	Predefined Page Styles	120
4.1.2	Manual and Running Headings	124
4.1.3	Formatting of Header and Footer	125
4.1.4	Package Options	130
4.2	Defining Own Page Styles	134
4.2.1	The Interface for Beginners	134
4.2.2	The Interface for Experts	136
4.2.3	Managing Page Styles	141
5	Weekday and Time Using <code>scrdate</code> and <code>scrttime</code>	142
5.1	The Name of the Current Day of the Week Using <code>scrdate</code>	142
5.2	Getting the Time with Package <code>scrttime</code>	143
6	The New Letter Class <code>scrlettr2</code>	145
6.1	Looking Back on the Old Letter Class	145
6.2	Options	145
6.2.1	Defining Options Later	146
6.2.2	Options for Compatibility	146
6.2.3	Page Layout Options	147
6.2.4	Other Layout Options	148
6.2.5	Font Options	152
6.2.6	Options for Letterhead and Address	152
6.2.7	Formatting Options	157
6.2.8	The Letter Class Option Files	158
6.3	General Document Properties	163
6.3.1	Font Selection	163
6.3.2	Page Style	163

6.3.3	Variables	165
6.3.4	The Pseudo-Lengths	171
6.3.5	The General Structure of a Letter Document	176
6.4	The Letter Declaration	178
6.4.1	The Letterhead	178
6.4.2	The Footer	181
6.4.3	The Address	183
6.4.4	The Sender's Extensions	187
6.4.5	The Reference Fields Line	188
6.4.6	The Title and the Subject Line	190
6.4.7	Further Settings	191
6.5	The Text	193
6.5.1	The Opening	193
6.5.2	Footnotes	193
6.5.3	Lists	193
6.5.4	Margin Notes	194
6.5.5	Text Emphasis	194
6.6	The Closing Part	194
6.6.1	Closing	194
6.6.2	Postscript, Carbon Copy and Enclosures	195
6.7	Language Support	197
6.7.1	Language Selection	197
6.7.2	Language-Dependent Terms	198
6.7.3	Defining and Changing Language-dependent Terms	200
6.8	Address Files and Circular Letters	202
6.9	From scrlettr to scrlett2	206
7	Access to Address Files with scraddr	209
7.1	Overview	209
7.2	Usage	210
7.3	Package Warning Options	211
8	Creating Address Files from a Address Database	213
9	Control Package Dependencies with scrfile	214
9.1	About Package Dependencies	214
9.2	Actions Prior to and After Loading	215
	Change Log	219
	Bibliography	220

Index	223
General Index	223
Index of Commands, Environments, and Variables	225
Index of Lengths and Counters	230
Index of Elements with Capability of Font Adjustment	230
Index of Files, Classes, and Packages	231
Index of Class and Package Options	232

List of Tables

2.1	Page layout values depending on <i>DIV</i> for A4	22
2.2	<i>DIV</i> defaults for A4	23
2.3	Available symbolic <i>DIV</i> values for <code>\typearea</code>	25
2.4	Available symbolic <i>BCOR</i> values for <code>\typearea</code>	25
3.1	Class correspondence	37
3.2	Default options of the KOMA-Script classes	38
3.3	Elements, whose type style can be changed with the KOMA-Script command <code>\setkomafont</code> or <code>\addtokomafont</code>	54
3.4	Default values for the elements of a page style	58
3.5	Macros to set up page style of special pages	59
3.6	Available numbering styles of page numbers	64
3.7	Main title	67
3.8	Default font sizes for different levels of document structuring	75
3.9	Default settings for the elements of a dictum	86
3.10	Font defaults for the elements of figure or table captions ..	105
6.1	Standard values for simple switches in <code>scrلتtr2</code>	149
6.2	Possible values of option <code>cleardoublepage</code> with <code>scrلتtr2</code> ..	150
6.3	Possible values of option <code>pagenumber</code> with <code>scrلتtr2</code>	151
6.4	Possible values of option <code>parskip</code> with <code>scrلتtr2</code>	153
6.5	Possible values of option <code>fromalign</code> with <code>scrلتtr2</code>	154
6.6	Possible values of option <code>fromrule</code> with <code>scrلتtr2</code>	154
6.7	Possible values of option <code>subject</code> with <code>scrلتtr2</code>	156
6.8	Possible values of option <code>locfield</code> with <code>scrلتtr2</code>	156
6.9	Possible value of option <code>refline</code> with <code>scrلتtr2</code>	157
6.10	The predefined <code>lco</code> files	162
6.11	Alphabetical list of elements whose font can be changed in <code>scrلتtr2</code> using the commands <code>\setkomafont</code> and <code>\addtokomafont</code>	164
6.12	Alphabetical list of all supported variables in <code>scrلتtr2</code>	166
6.13	Pseudo-lengths provided by class <code>scrلتtr2</code>	171
6.14	The sender’s predefined labels for the letterhead	179
6.15	predefined labels and contents of hyphens for sender’s data in the letterhead	181

6.16 predefined labels of typical variables of the reference fields
line. The content of the macros depend on language. 189

6.17 Predefined labels of subject-related variables. 191

6.18 Language-dependent forms of the date 199

6.19 Default settings for languages **english** and **ngerman** 200

Introduction

1.1 Preface

The KOMA-Script bundle is actually several packages and classes. It provides counterparts or replacements for the standard L^AT_EX classes such as *article*, *book*, etc. (see [chapter 3](#)), but offers many additional features and its own unique look and feel.

The KOMA-Script user guide is intended to serve the advanced as well as the inexperienced L^AT_EX user and is accordingly quite large. The result is a compromise and we hope that you will keep this in mind when using it. Your suggestions for improvement are, of course, always welcome.

1.2 Structure of the Guide

The KOMA-Script user guide is not intended to be a L^AT_EX primer. Those new to L^AT_EX should look at *The Not So Short Introduction to L^AT_EX 2_ε* [[OPHS99](#)] or *L^AT_EX 2_ε for Authors* [[Tea01](#)] or a L^AT_EX reference book. You will also find useful information in the many L^AT_EX FAQs, including the *T_EX Frequently Asked Questions on the Web* [[FAQ](#)].

In this guide you will find supplemental information about L^AT_EX and KOMA-Script in (sans serif) paragraphs like this one. The information given in these explanatory sections is not essential for using KOMA-Script, but if you experience problems you should take a look at it — particularly before sending a bug report.

If you are only interested in using a single KOMA-Script class or package you can probably successfully avoid reading the entire guide. Each class and package typically has its own chapter; however, the three main classes (`scrbook`, `scrprft`, and `scrartcl`) are introduced together in chapter three. Where an example or note only applies to one or two of the three classes, it is called out in the margin.

Like
this.

The primary documentation for KOMA-Script is in German and has been translated for your convenience; like most of the L^AT_EX world, its commands, environments, options, etc., are in English. In a few cases, the name of a command may sound a little strange, but even so, we hope and believe that with the help of this guide KOMA-Script will be usable and useful to you.

1.3 History of KOMA-Script

In the early 1990s, Frank Neukam needed a method to publish an instructor’s lecture notes. At that time \LaTeX was $\text{\LaTeX}2.09$ and there was no distinction between classes and packages—there were only *styles*. Frank felt that the standard document styles were not good enough for his work; he wanted additional commands and environments. At the same time he was interested in typography and, after reading Tschichold’s *Ausgewählte Aufsätze über Fragen der Gestalt des Buches und der Typographie* (Selected Articles on the Problems of Book Design and Typography) [Tsc87], he decided to write his own document style—and not just a one-time solution to his lecture notes, but an entire style family, one specifically designed for European and German typography. Thus Script was born.

Markus Kohm, the developer of KOMA-Script, came across Script in December 1992 and added an option to use the A5 paper format. This and other changes were then incorporated into Script-2, released by Frank in December 1993.

Beginning in mid-1994, $\text{\LaTeX}2_{\epsilon}$ became available and brought with it many changes. Users of Script-2 were faced with either limiting their usage to $\text{\LaTeX}2_{\epsilon}$ ’s compatibility mode or giving up Script altogether. This situation led Markus to put together a new $\text{\LaTeX}2_{\epsilon}$ package, released on 7 July 1994 as KOMA-Script; a few months later Frank declared KOMA-Script to be the official successor to Script. KOMA-Script originally provided no *letter* class, but this deficiency was soon remedied by Axel Kielhorn, and the result became part of KOMA-Script in December 1994. Axel also wrote the first true German-language user guide, which was followed by an English-language guide by Werner Lemberg.

Since then much time has passed. \LaTeX has changed in only minor ways, but the \LaTeX landscape has changed a great deal; many new packages and classes are now available and KOMA-Script itself has grown far beyond what it was in 1994. The initial goal was to provide good \LaTeX classes for German-language authors, but today its primary purpose is to provide more-flexible alternatives to the standard classes. KOMA-Script’s success has led to e-mail from users all over the world, and this has led to many new macros—all needing documentation; hence this “small guide.”

1.4 Special Thanks

Acknowledgements in the introduction? No, the proper acknowledgements can be found in the addendum. My comments here are not intended for

the authors of this guide — and those thanks should rightly come from you, the reader, anyhow. I, the author of KOMA-Script, would like to extend my personal thanks to Frank Neukam. Without his Script family, KOMA-Script would not have come about. I am indebted to the many persons who have contributed to KOMA-Script, but with their indulgence, I would like to specifically mention Jens-Uwe Morawski and Torsten Krüger. The English translation of the guide is, among many other things, due to Jens’s untiring commitment. Torsten was the best beta-tester I ever had. His work has particularly enhanced the usability of `scrlltr2` und `scrpage2`. Many thanks to all who encouraged me to go on, to make things better and less error-prone, or to implement additional features.

Thanks go as well to DANTE, Deutschsprachige Anwendervereinigung T_EX e.V, (the German-Language T_EX User Group). Without the DANTE server, KOMA-Script could not have been released and distributed. Thanks as well to everybody in the T_EX newsgroups and mailing lists who answer questions and have helped me to provide support for KOMA-Script.

1.5 Legal Notes

KOMA-Script was released under the L^AT_EX Project Public License. You will find it in the file `lpp1.txt`. An unofficial German-language translation is also available in `lpp1-de.txt` and is valid for all German-speaking countries.

This document and the KOMA-Script bundle are provided “as is” and without warranty of any kind.

1.6 Installation

Installation information can be found in the file `INSTALL.txt`. You should also read the documentation that comes with the T_EX distribution you are using.

1.7 Bugreports and Other Requests

If you think you have found an error in the documentation or a bug in one of the KOMA-Script classes, one of the KOMA-Script packages, or another part of KOMA-Script, please do the following: first have a look on CTAN to see if a newer version of KOMA-Script is available; in this case install the applicable section and try again.

If you are using the most recent version of KOMA-Script and still have a bug, please provide a short \LaTeX document that demonstrates the problem. You should only use the packages and definitions needed to demonstrate the problem; do not use any unusual packages.

By preparing such an example it often becomes clear whether the problem is truly a KOMA-Script bug or something else. Please report KOMA-Script (only) bugs to the author of KOMA-Script. Please use `komabug.tex`, an interactive \LaTeX document, to generate your bug report and send it to the address you may find at `komabug.tex`.

If you want to ask your question in a newsgroup or mailing list, you should also include such an example as part of your question, but in this case, using `komabug.tex` is not necessary. To find out the version numbers of all packages in use, simply put `\listfiles` in the preamble of your example and read the end of the `log`-file.

1.8 Additional Information

Once you become an experienced KOMA-Script user you may want to look at some more advanced examples and information. These you will find on the KOMA-Script documentation web site [\[KDP\]](#). The main language of the site is German, but nevertheless English is welcome.

Construction of the Page Layout with `typearea`

2.1 Fundamentals of Page Layout

If you look at a single page of a book or other printed materials, you will see that it consists of top, foot, left and right margins, a (running) head area, the text block and a (running) foot area. Looking closer, there is space between the head area and the text block and between the text block and the foot area. The relations between these areas are called the *page layout*.

The literature offers and discusses different algorithms and heuristic approaches for constructing a good page layout. Often, mentioned is an approach which involves diagonals and their intersections. The result is a page where the text block proportions are related to the proportions of the *page*. In a single-sided document, the left and the right margin should have equal widths. The relation of the upper margin to the lower margin should be 1:2. In a double-sided document (e.g. a book) however, the complete inner margin (the margin at the spine) should be the same as each of the two outer margins.

In the previous paragraph, we mentioned and emphasized the *page*. Erroneously, it is often thought that the format of the page means the format of the paper. However, if you look at a bound document, it is obvious that part of the paper vanishes in the binding and that is not part of the visible page. It is not the format of the paper which is important in the layout of a page, it is the impression of the visible page to the reader. Therefore, it is clear that the calculation of the page layout must account for the “lost” paper in the binding and add this amount to the width of the inner margin. This is called the *binding correction*.

The binding correction depends on the process of actually producing the document and thus cannot be calculated in general. Every production process needs its own parameter. In professional binding, this parameter is not too important since the printing is done on oversized paper which is then cropped to the right size. The cropping is done in a way so that the relations for the visible double-sided page are as explained above.

Now we know about the relations of the individual parts of a page. However, we do not yet know about the width and the height of the text block. Once we know one of these values, we can calculate all the other values from the

paper format and the page format or the binding correction.

$$\begin{aligned}
 \text{textblock height} : \text{textblock width} &= \text{page height} : \text{page width} \\
 \text{page width} &= \text{paper width} - \text{binding correction} \\
 \text{top margin} + \text{foot margin} &= \text{page height} - \text{textblock height} \\
 \text{top margin} : \text{foot margin} &= 1 : 2 \\
 \text{left margin} : \text{right margin} &= 1 : 1 \\
 \text{half inner margin} &= \frac{1}{2} \text{outer margin} + \text{binding correction}
 \end{aligned}$$

The values *left margin* and *right margin* only exist in a single-sided document while *inner margin* and *outer margin* only exist in a double-sided document. In these equations, we work with *half inner margin* since the full inner margin belongs to a double-page. Thus, one page has half of the inner margin.

The question of the width of the textblock is also discussed in the literature. The optimum width depends on several factors:

- size, width, type of the used font
- line spacing
- word length
- available room

The importance of the font becomes clear once you think about serifs. Serifs are fine lines finishing off the letters. Letters whose main strokes run orthogonal to the text line disturb the flow more than they lead the eye along the line. Those letters have serifs at the ends of the vertical strokes, however, so the horizontal serifs lead the eye horizontally too. In addition, it helps the eye to find the beginning of the next line. Thus, the line length for a serif font can be slightly longer than for a sans serif font.

In L^AT_EX, the line spacing is about 20% of the font size. With commands like `\linespread` or, better, packages like `setspace`, the line spacing can be changed. A wider line spacing helps the eye to follow the line. A very wide line spacing, on the other hand, disturbs reading because the eye has to move a wide distance between lines. Also, the reader gets uncomfortable because of the visible stripe effect. The uniform gray value of the page gets spoiled. Still, with a wider line spacing, the lines can be longer.

The literature gives different values for good line lengths, depending on the author. To some extent, this is due to the native language of the author.

Since the eye jumps from word to word, short words make this task easier. Not considering language and font, a line length of 60 to 70 letters, including spaces and punctuation, is a usable compromise. This requires well-chosen line spacing, but \LaTeX 's default is usually good enough.

Before looking at the actual construction of the page layout, there are some minor things to know. \LaTeX doesn't start the first line of the text block at the upper edge of the text block, but with a defined distance from it. Also, \LaTeX knows the commands `\raggedbottom` and `\flushbottom`. `\raggedbottom` specifies that the last line of a page should be positioned wherever it was calculated. This means that the position of this line can be different on each page, up to the height of one line. In double-sided documents this is usually undesirable. `\flushbottom` makes sure that the last line is always at the lower edge of the text block. To achieve this, \LaTeX sometimes needs to stretch vertical glue more than allowed. Paragraph skip is such a stretchable, vertical glue, even when set to zero. In order to not stretch the paragraph skip on normal pages where it is the only stretchable glue, the height of the text block should be a multiple of the height of the text line, including the distance from the upper edge of the text block to the first line.

This concludes the introduction to page layout as handled by KOMA-Script. Now, we can begin with the actual construction.

2.2 Page Layout Construction by Dividing

The easiest way to make sure that the text area has the same ratios as the page is as follows: First, you subtract the binding correction *BCOR* from the inner edge of the paper. Then you divide the rest of the page vertically into *DIV* rows of equal height. Next, you divide the page horizontally into the same number (*DIV*) of columns. Then you take the uppermost row as the upper margin and the two lowermost rows as the lower margin. If you print double-sided, you also take the innermost column as the inner margin and the two outermost columns as the outer margin. Then, you add the binding correction *BCOR* to the inner margin. The remainder of the page is the text area. The width and the height of the text area result automatically from the number of rows and columns *DIV*. Since the margins always need three rows/columns, *DIV* must be necessarily greater than three. In order that the text area occupy at least twice as much space as the margins, *DIV* should really be equal to or greater than 9. With this value, the construction is known as the *classical division factor of 9*.

In KOMA-Script, this kind of construction is implemented in the `typearea` package. For A4 paper, *DIV* is predefined according to the font size (see

[table 2.2](#)). If there is no binding correction ($BCOR = 0\text{pt}$), the results roughly match the values of [table 2.1](#).

In addition to the predefined values, you can specify *BCOR* and *DIV* as options when loading the package (see [section 2.4](#)). There is also a command to explicitly calculate the type area by providing these values as parameters (also see [section 2.4](#)).

The `typearea` package can determine the optimal value of *DIV* for the font used automatically. Again, see [section 2.4](#).

2.3 Page Layout Construction by Drawing a Circle

In addition to the construction method previously described, a somewhat more classical method can be found in the literature. The aim of this method is not only identical ratios in the page proportions, but it is considered optimal when the height of the text block is the same the width of the page. The exact method is described in [\[Tsc87\]](#).

A disadvantage of this late Dark Ages method is that the width of the text area is no longer dependent on the font. Thus, one doesn't choose the text area to match the font, but the author or typesetter has to choose the font according to the text area. This can be considered a “must”.

In the `typearea` package this construction is changed slightly. By using a special (normally meaningless) *DIV* value or a special package option, a *DIV* value is chosen to match the perfect values as closely as possible. See also [section 2.4](#).

2.4 Options and Macros to Influence the Page Layout

The package `typearea` offers two different user interfaces to influence type area construction. The first method is to load the package with options. For information on how to load packages and to give package options, please refer to the L^AT_EX literature, e.g. [\[OPHS99\]](#) and [\[Tea01\]](#), or the examples given here. Since the `typearea` package is loaded automatically when using the KOMA-Script main classes, the package options can be given as class options (see [section 3.1](#)).

`BCORCorrection`

With the `BCORCorrection` option you specify the absolute value of the binding correction, i.e., the width of the area that is used for the binding, thus “lost” from the paper width.

This value will be used in the layout calculation automatically and will be added to the inner or left margin respectively. You can use any valid \TeX unit for *Correction*.

Example: Assume you want to produce a financial report, which is to be printed on A4 paper and bound in a folder. The rim of the folder covers 7.5 mm. Since the report is thin, only an additional 0.75 mm are lost by folding when leafing through the pages. You would use the following commands:

```
\documentclass[a4paper]{report}
\usepackage[BCOR8.25mm]{typearea}
```

or, using a KOMA-Script class:

```
\documentclass[a4paper,BCOR8.25mm]{scrreprt}
```

Please note: if you use one of the KOMA-Script classes, this option must be given as a class option. If you use another class, this only works if the class has explicit support for `typearea`. So when using the standard classes, you need to give the option when you load `typearea`. You can also use `\PassOptionsToPackage` (see [Tea99]) before you are loading `typearea`, this always works.

DIVFactor

DIVFactor defines the number of stripes the page is split into when the page layout is constructed. The exact method can be found in [section 2.2](#), but the most important thing is this: the higher *Factor*, the bigger the resulting text area, and the smaller the margins. For *Factor*, you can use any integer value larger than 4. Please note that depending on your other options a very high value for *Factor* can result in problems: for instance, in extreme cases, the running title might be outside the actual page area. So if you use **DIVFactor**, it is your own responsibility to choose a typographically acceptable line length and to pay attention to the other parameters.

In [table 2.1](#) you'll find some page layout values for the page format A4 without binding correction, with varying *DIV* factors. Font size is not taken into account.

Example: Imagine you are writing meeting minutes with the protocol¹

¹The class `protocol` is hypothetical. This manual considers the ideal case where you have a special class for every use.

Table 2.1: Page layout values depending on *DIV* for A4

<i>DIV</i>	Text area		Margins	
	Width [mm]	Height [mm]	upper [mm]	inner [mm]
6	105.00	148.50	49.50	35.00
7	120.00	169.71	42.43	30.00
8	131.25	185.63	37.13	26.25
9	140.00	198.00	33.00	23.33
10	147.00	207.90	29.70	21.00
11	152.73	216.00	27.00	19.09
12	157.50	222.75	24.75	17.50
13	161.54	228.46	22.85	16.15
14	165.00	233.36	21.21	15.00
15	168.00	237.60	19.80	14.00

class. The whole thing is supposed to be double-sided. In your company, the Bookman font in 12pt is used. This standard PostScript font is activated in L^AT_EX with the command `\usepackage{bookman}`. Bookman runs very wide, that means, the characters are wide in relation to its height. Because of that, the default for the *DIV* value in `typearea` is too small for you. Instead of 12, you want 15. The minutes will not be bound but punched and filed into a folder, so you don't need any binding correction. Thus, you write:

```
\documentclass[a4paper,twoside]{protocol}
\usepackage{bookman}
\usepackage[DIV15]{typearea}
```

After you are done you are told that the minutes will be collected and bound as a book at the end of the year. The binding will be a simple glue binding carried out in a copy shop, since it is done just for ISO 9000 anyway and nobody will ever bother to look at the minutes again. For binding you need 12 mm in average. So you change the options for `typearea` accordingly and use the ISO 9000 document class:

```
\documentclass[a4paper,twoside]{iso9000p}
\usepackage{bookman}
```

Table 2.2: *DIV* defaults for A4

Base font size:	10 pt	11 pt	12 pt
<i>DIV</i> :	8	10	12

```
\usepackage[DIV15,BCOR12mm]{typearea}
```

Of course, you can also use a KOMA-Script class here:

```
\documentclass[twoside,DIV15,BCOR12mm]{scrartcl}
}
\usepackage{bookman}
```

The option `a4paper` was omitted using class `scrartcl`, because it is the default for all KOMA-Script classes.

Please note: if you use one of the KOMA-Script classes, `BCOR` must be given as a class option. If you use another class, this only works if the class has explicit support for `typearea`. So when using the standard classes, you need to give `BCOR` when you load `typearea`. You can use `\PassOptionsToPackage` (see [Tea99]) too before you are loading `typearea`, this always works.

`DIVcalc`
`DIVclassic`

As mentioned in [section 2.2](#), only paper format A4 has fixed defaults for the *DIV* value. These are listed in [table 2.2](#). If you choose a different paper format, `typearea` calculates a good *DIV* value itself. Of course, you can also have it calculate that for A4: use `DIVcalc` instead of `DIVFactor`. This works for all other paper formats as well. If you want to use the automatic calculation, this is even very useful, since you can then override the defaults that are given in a configuration file (see [section 2.7](#)) with this option.

The classic construction method as described in [section 2.3](#) can also be selected (with the difference that a good *DIV* value is chosen). In this case, instead of `DIVFactor` or `DIVcalc`, use the option `DIVclassic`.

Example: In the example for `DIVFactor` which used the Bookman font, there was the problem that we needed a *DIV* value which suited the font better. As a modification of the first example, this calculation can be left to `typearea`:

```
\documentclass[a4paper,twoside]{protocol}
\usepackage{bookman}
\usepackage[DIVcalc]{typearea}
```

```
\typearea[BCOR]{DIV}
```

If you followed the examples till here, you'll ask yourself how one can make the calculation of *DIV* depend on the selected font when one uses one of the KOMA-Script classes. In that case the options to `typearea` would have to be made before loading the e. g. `bookman` package. So `typearea` could only calculate the page layout for the standard font, but not for the Bookman font which is really to be used. Therefore, after evaluating the options, `typearea` calculates the page layout by using the `\typearea[BCOR]{DIV}` command. Here, the chosen *BCOR* value is given as an optional parameter and *DIV* as a parameter. With the option `DIVcalc`, the (normally invalid) value 1 is given; with the option `DIVclassic` the (normally invalid) value 3. You can also call `\typearea` explicitly in the preamble.

Example: Let us assume again that we want to calculate a good page layout for the Bookman font. We also want to use a KOMA-Script class. This is possible using the `\typearea` command with `DIVcalc = 1` as *DIV* parameter:

```
\documentclass[BCOR12mm,DIVcalc,twoside]{←
  scrartcl}
\usepackage{bookman}
\typearea[12mm]{1}% same as class options above
```

Again option `a4paper` was not used explicitly, because it's the default of the KOMA-Script class `scrartcl`.

It would be ridiculous if one had to use the `\typearea` command with some pseudo-values, while the *DIV* option allows the use of `DIVcalc` and `DIVclassic`. Thus the `\typearea` also accepts symbolic values for the parameter *DIV* which are listed in [table 2.3](#).

The `\typearea` also understands the symbolic values for the parameter *BCOR* shown in [table 2.4](#). Thus it is not necessary to re-enter the current value.

Example: Thus calculating a good page layout for the Bookman font and a KOMA-Script class is easy when we use symbolic parameter values for *BCOR* and *DIV*:

Table 2.3: Available symbolic *DIV* values for `\typearea[BCOR]{DIV}`

<code>calc</code>	re-calculate page layout and determine <i>DIV</i> .
<code>classic</code>	re-calculate page layout using the classical method (circle).
<code>current</code>	re-calculate page layout with current value of <i>DIV</i> .
<code>default</code>	re-calculate page layout with default values for the current page and font size. If no default values exist, apply <code>calc</code> .
<code>last</code>	re-calculate page layout using the same <i>DIV</i> argument, which was set last time.

```
\documentclass[BCOR12mm,DIVcalc,twoside]{←
  scrartcl}
\usepackage{bookman}
\typearea[current]{calc}
```

If we want to use a fixed value for *DIV* we can use either:

```
\documentclass[BCOR12mm,DIV11,twoside]{scrartcl←
  }
\usepackage{bookman}
\typearea[current]{last}
```

or the old method:

```
\documentclass[a4paper,twoside]{scrartcl}
\usepackage{bookman}
\typearea[12mm]{11}
```

Table 2.4: Available symbolic *BCOR* values for `\typearea[BCOR]{DIV}`

<code>current</code>	Re-calculate page layout using the current value for <i>BCOR</i> .
----------------------	--------------------------------------------------------------------

In the end it is a matter of personal taste which of these solutions you want to use.

Frequently the re-calculation of the page layout is necessary because the line spacing was changed. Since it is essential that an integer number of lines fit into the text area, any change in line spacing requires a re-calculation of page layout.

Example: Assume you want to write a thesis and university regulations require a font size of 10pt with one and a half line spacing. L^AT_EX uses by default a line spacing of 2pt at font size 10pt (a line spacing of 1.2). Thus a stretch-factor of 1.25 is required. Let us also assume that binding correction needs 12mm. Then you might use:

```
\documentclass[10pt,twoside,%
                BCOR12mm,DIVcalc]{scrreprt}
\linespread{1.25}
\typearea[current]{last}
```

`\typearea` automatically calls `\normalsize`. So it is not necessary to use `\selectfont` after `\linespread` to activate the changed line spacing before re-calculation of the page layout.

The same example again, using the `setspace` package (see [Tob00]):

```
\documentclass[10pt,twoside,%
                BCOR12mm,DIVcalc]{scrreprt}
\usepackage{setspace}
\onehalfspacing
\typearea[current]{last}
```

Using the `setspace` package simplifies things, because you no longer need to calculate the correct stretch-factor, and you no longer need the `\selectfont` macro.

In this context it is appropriate to point out that the line spacing should be reset for the title page. A complete example therefore would look like this:

```
\documentclass[10pt,twoside,%
                BCOR12mm,DIVcalc]{scrreprt}
\usepackage{setspace}
```

```

\onehalfspacing
\typearea[current]{last}
\begin{document}
\title{Title}
\author{Markus Kohm}
\begin{spacing}{1}
\maketitle
\tableofcontents
\end{spacing}
\chapter{0k}
\end{document}

```

See also the notes in [section 2.8](#).

The command `\typearea` is currently defined in such a way that it is possible to change the page layout in the middle of a text. This however makes assumptions about the inner workings of the \LaTeX kernel and changes some internal values and definitions of that kernel. There is some probability, but no guarantee that this will also work in future versions of \LaTeX . It must be assumed that this method will not give correct results in $\text{\LaTeX}3$. However, as author of KOMA-Script I expect considerable incompatibilities when we change to $\text{\LaTeX}3$.

<pre> headinclude headexclude footinclude footexclude </pre>

So far we have discussed how the page layout is calculated and what the ratios are between the borders and between borders and text area. However, one important question has not been answered: What constitutes the borders? This question appears trivial: Borders are those parts on the right, left, top and bottom which remain empty. But this is only half of it. Borders are not always empty. There could be marginals, for example (for the `\marginpar` command refer to [\[OPHS99\]](#) or [section 3.6.5](#)).

One could also ask, whether headers and footers belong to the upper and lower borders or to the text. This can not be answered unambiguously. Of course an empty footer or header belong to the borders, since they can not be distinguished from the rest of the border. A header or footer, that contains only a page number, will optically appear more like border. For the optical appearance it is not important whether headers or footers are easily recognised as such during reading. Important is only, how a well filled page appears when

viewed out of focus. You could use the glasses of your far-sighted grandparents, or, lacking those, adjust your vision to infinity and look at the page with one eye only. Those wearing spectacles will find this much easier, of course. If the footer contains not only the page number, but other material like a copyright notice, it will optically appear more like a part of the text body. This needs to be taken into account when calculating text layout.

For the header this is even more complicated. The header frequently contains running headings. In case of running headings with long chapter and section titles the header lines will be very long and appear to be part of the text body. This effect becomes even more significant when the header contains not only the chapter or section title but also the page number. With material on the right and left side, the header will no longer appear as empty border. If the length of the titles varies, the header may appear as border on one page and as text on another. However, this pages should not be treated differently under any circumstances, as this would lead to jumping headers. In this case it is probably best to count the header with the text.

The decision is easy when text and header or footer are separated from the text body by a line. This will give a “closed” appearance and header or footer become part of the text body. Remember: It is irrelevant that the line improves the optical separation of text and header or footer, important is only the appearance when viewed out of focus.

The `typearea` package can not make the decision whether or not to count headers and footers to the text body or the border. Options `headinclude` and `footinclude` cause the header or footer to be counted as text, options `headexclude` and `footexclude` cause them to be counted as border. If you are unsure about the correct setting, re-read above explanations. Default is usually `headexclude` and `footexclude`., but this can change depending on KOMA-Script class and KOMA-Script packages used (see [section 3.1](#) and [chapter 4](#)).

`mpinclude`
`mpexclude`

v2.8q

Besides documents where the head and foot is part of the text area, there are also documents where the margin-note area must be counted to the text body as well. The option `mpinclude` does exactly this. The effect is that one width-unit of the text body is taken for the margin-note area. Using option `mpexclude`, the default setting, then the normal margin is used for the margin-note area. The width of that area is one or one and a half width-unit, depending on whether one-sided or double-sided page layout has been chosen. The option `mpinclude` is mainly for experts and

so not recommended.

In the cases where the option `mpinclude` is used often a wider margin-note area is required. In many cases not the whole margin-note width should be part of the text area, for example if the margin is used for quotations. Such quotations are typeset as ragged text with the flushed side where the text body is. Since ragged text gives no homogeneous optical impression the long lines can reach right into the normal margin. This can be done using option `mpinclude` and by an enlargement of length `\marginparwidth` after the type area has been setup. The length can be easily enlarged with the command `\addtolength`. How much the the length has to be enlarged depends on the special situation and it requires some flair. Therefore the option is primarily for experts. Of course one can setup the margin-width to reach a third right into the normal margin, for example using

```
\setlength{\marginparwidth}{1.5\marginparwidth}
```

gives the desired result.

Currently there is no option to enlarge the margin by a given amount. The only solution is that the option `mpinclude` is not used, but after the type area has been calculated one reduces the width of the text body `\textwidth` and enlarges the margin width `\marginparwidth` by the same amount. Unfortunately, this can not be attended when automatic calculation of the *DIV* value is used. In contrast `DIVcalc` heeds `mpinclude`.

Valueheadlines

We have seen how to calculate the text layout and how to specify whether header and footer are part of the text body or the borders. However, we still have to specify the height in particular of the header. This is achieved with the option `headlines`, which is preceded by the number of lines in the header. `typearea` uses a default of 1.25. This is a compromise, large enough for underlined headers (see [section 3.1](#)) and small enough that the relative weight of the top border is not affected too much when the header is not underlined. Thus in most cases you may leave `headlines` at its default value and adapt it only in special cases.

Example: Assume that you want to use a header with two lines. Normally this would result in a “`overfull \vbox`” warning for each page. To prevent this from happening, the `typearea` package is told to calculate an appropriate page layout:

```
\documentclass[a4paper]{article}
\usepackage[2.1headlines]{typearea}
```

If you use a KOMA-Script class this must be given as a class option:

```
\documentclass[a4paper,2.1headlines]{scrartcl}
```

A tool that can be used to define the contents of a header with two lines is described in [chapter 4](#).

If you use a KOMA-Script class, this option must be given as class option. With other classes this works only, if these classes explicitly supports `typearea`. If you use the standard classes, the option must be given when loading `typearea`. `\PassOptionsToPackage` will work in both cases (see also [\[Tea99\]](#)).

```
\areaset[BCOR]{Width}{Height}
```

So far we have seen how a good or even very good page layout is calculated and how the `typearea` package can support these calculations, giving you at the same time the freedom to adapt the layout to your needs. However, there are cases where the text body has to fit exactly into specified dimensions. At the same time the borders should be well spaced and a binding correction should be possible. The `typearea` package offers the command `\areaset` for this purpose. As parameters this command accepts the binding correction and the width and height of the text body. Width and position of the borders will then be calculated automatically, taking account of the options `headinclude`, `headexclude`, `footinclude` and `footexclude` where needed.

Example: Assume a text, printed on A4 paper, should have a width of exactly 60 characters of typewriter font and a height of exactly 30 lines. This could be achieved as follows:

```
\documentclass[a4paper,11pt]{article}
\usepackage{typearea}
\newlength{\CharsLX}% Width of 60 characters
\newlength{\LinesXXX}% Height of 30 lines
\settowidth{\CharsLX}{\texttt{1234567890}}
\setlength{\CharsLX}{6\CharsLX}
\setlength{\LinesXXX}{\topskip}
\addtolength{\LinesXXX}{29\baselineskip}
\areaset{\CharsLX}{\LinesXXX}
```

You need only 29 instead of 30, because the base line of the topmost text line is `\topskip` below the top margin of the type area.

A poetry book with a square text body with a page length of 15 cm and a binding correction of 1 cm could be achieved like this:

```
\documentclass{gedichte}
\usepackage{typearea}
\areaset[1cm]{15cm}{15cm}
```

The `typearea` package was not made to set up predefined margin values. If you have to do so you may use package `geometry` (see [Ume00]).

2.5 Options and Macros for Paper Format Selection

The L^AT_EX standard classes support the options `a4paper`, `a5paper`, `b5paper`, `letterpaper`, `legalpaper` and `executivepaper`.

```
letterpaper
legalpaper
executivepaper
aXpaper
bXpaper
cXpaper
dXpaper
landscape
\isopaper[series]{number}
```

The three American formats are supported by `typearea` in the same way. In addition, all ISO A-, ISO B-, ISO C- and ISO D-formats are supported and derived from their basic sizes A0, B0, C0 and D0. They may be selected directly with options `a0paper`, `a1paper` and so on. Landscape orientation is selected with the `landscape` option just as in the standard classes.

Alternatively the paper size can be adjusted with the macro `\isopaper`. This however required re-calculation of the text layout with `\typearea` or `\areaset`. I do not recommend the use of `\isopaper`.

Example: Assume you want to print on ISO A8 file cards in landscape orientation. Borders should be very small, no header or footer will be used.

```

\documentclass{article}
\usepackage[headexclude,footexclude,
            a8paper,landscape]{typearea}
\areaset{7cm}{5cm}
\pagestyle{empty}
\begin{document}
\section*{Paper Size Options}
letterpaper, legalpaper, executivepaper, ←
a0paper,
a1paper \dots\ b0paper, b1paper \dots\ c0paper,
c1paper \dots\ d0paper, d1paper \dots
\end{document}

```

All `aXpaper`, `bXpaper`, `cXpaper` and `dXpaper` options need to be given as class options when KOMA-Script classes are used. For other classes this works only if they support `typearea`. For the standard L^AT_EX classes these options need to be declared when `typearea` is loaded. `\PassOptionsToPackage` (see [Tea99]) works in both cases.

`\paperwidth`
`\paperheight`

Particularly exotic paper sizes can be defined using the lengths `\paperwidth` and `\paperheight`. This requires the re-calculation of the text layout using the commands `\typearea` or `\areaset`.

Example: Assume you want to print on endless paper with the dimensions $8\frac{1}{4}$ inch \times 12 inch. This format is not directly supported by `typearea`. Thus you have to define it before calculating the text layout:

```

\documentclass{article}
\usepackage{typearea}
\setlength{\paperwidth}{8.25in}
\setlength{\paperheight}{12in}
\typearea{1}

```

`dvips`
`pdftex`
`pagesize`

These mechanisms will set internal L^AT_EX dimensions to values that header, text body and footer can be printed on paper of the given size. However, the

specifications of the DVI format do not allow the paper format to be specified. If DVI is translated directly into a low level printer language like PCL (Hewlett-Packard printers) or Esc-P (Epson), this is usually not important, because in all these cases the origin is the upper left corner. If however the DVI source is translated into languages like PostScript or PDF, that have an origin in a different position and also contain the paper size explicitly, the required information is not available in the DVI file. To solve this problem the DVI driver will use the default paper size, which the user may set per option or in the \TeX source. In case of the DVI driver `dvips` this can be done with a `\special` command. For `pdf \TeX` two dimensions are set instead.

The option `dvips` writes the paper size as a `\special` into the DVI file. This macro is then evaluated by `dvips`. `pdftex` on the other hand writes the paper size into the `pdf \TeX` page register at the beginning of the document, so that the correct paper size is used when the resulting PDF file is viewed or printed. The option `pagesize` is more flexible and uses the correct mechanism if either a PDF or DVI file is produced.

Example: Assume you want to create a DVI file from a document and an online version in PDF. Then the preamble could look like this:

```
\documentclass{article}
\usepackage[a4paper,pagesize]{typearea}
```

If the file is run through `pdf \TeX` then the lengths `\pdfpagewidth` and `\pdfpageheight` will be set to appropriate values. If on the other hand you create a DVI file — either with \LaTeX or `pdf \LaTeX` — a `\special` will be written to the beginning of the file.

2.6 Odd Bits without Direct Relevance to Text Layout

`\ifpdfoutput{then}{else}`

Sometimes it would be nice if certain things would be done differently in a file, depending on output format. \TeX normally uses DVI as output format. With `pdf \TeX` however we now have the option to create PDF files directly. The command `\ifpdfoutput` is a branching command. If PDF output is active, the *then* branch will be executed, if PDF output is inactive or `pdf \TeX` is not used at all, the *else* branch.

Example: As you may know pdf \LaTeX will produce a DVI file instead of a PDF file, if the counter `\pdfoutput` is assigned the value 0. Only if the counter is assigned a value different from 0 output is switched to PDF. Since `\pdfoutput` is unknown when \LaTeX is used instead of pdf \LaTeX , `\pdfoutput` can not be set to 0 generally, if you want DVI output. A simple solution to this problem is to execute following command:

```
\ifpdfoutput{\pdfoutput=0}{}
```

This only works after loading `typearea` package. If you want the line above to be executed after a package, which sets `\pdfoutput` to 1 whenever the counter exists, you may combine it with the `\AfterPackage` command from `scrfile` package (see [chapter 9](#)).

2.7 Local Defaults in the File `typearea.cfg`

Even before the packet options are used, `typearea` will check for the presence of the file `typearea.cfg` and, if found, load it. Thus it is possible to define in this file the parameters for additional paper sizes.

```
\SetDIVList{List}
```

The `\SetDIVList` parameter was also intended for use in this file. Before the option `DIVcalc` was introduced this was the only possibility to define *DIV* values for different paper and font sizes. This list consists of a number of values in curly parentheses. The leftmost value is the font size, 10pt, the next for 11pt, the third for 12pt and so on. If you don't use `\SetDIVList` the predefined `\SetDIVList{{8}{10}{12}}` will be used. If no default value is given for a particular font size, 10 will be used.

This command should no longer be used, automatic calculation of text layout is recommended instead (see [section 2.4](#)).

2.8 Hints

In particular for theses many rules exist that violate even the most elementary rules of typography. The reasons for such rules include typographical incompetence of those making them, but also the fact that they were originally meant for mechanical typewriters. With a typewriter or a primitive text processor

dating back to the early '80s it is not possible to produce typographically correct output without extreme effort. Thus rules were created that appeared to be achievable and still allowed easy correction. To avoid short lines made worse by ragged margins the borders were kept narrow, and the line spacing increased to 1.5 for corrections. In a single spaced document even correction signs would have been difficult to add. When computers became widely available for text processing, some students tried to use a particularly "nice" font to make their work look better than it really was. They forgot however that such fonts are often more difficult to read and therefore unsuitable for this purpose. Thus two bread-and-butter fonts became widely used which neither fit together nor are particularly suitable for the job. In particular Times is a relatively narrow font which was developed at the beginning of the 20th century for the narrow columns of British newspapers. Modern versions usually are somewhat improved. But still the Times font required in many rules does not really fit to the border sizes prescribed.

L^AT_EX already uses sufficient line spacing, and the borders are wide enough for corrections. Thus a page will look generous, even when quite full of text. With `typearea` this is even more true, especially if the calculation of line length is left to `typearea` too. For fonts that are sensitive to long lines the line length can easily be reduced.

To some extent the questionable rules are difficult to implement in L^AT_EX. A fixed number of characters per line can be kept only when a non-proportional font is used. There are very few good non-proportional fonts around. Hardly a text typeset in this way looks really good. In many cases font designers try to increase the serifs on the 'i' or 'l' to compensate for the different character width. This can not work and results in a fragmented and agitated-looking text. If you use L^AT_EX for your paper, some of these rules have to be either ignored or at least interpreted generously. For example you may interpret "60 characters per line" not as a fixed, but average or maximal value.

As executed, record regulations are usually intended to obtain a useable result even if the author does not know what needs to be considered. Useable means frequently: readable and correctable. In my opinion the type area of a text set with L^AT_EX and the `typearea` package meets these criteria well right from the beginning. Thus if you are confronted with regulations which deviate obviously substantially from it, then I recommend to submit a text single dump to the responsible person and inquire whether it is permitted to supply the work despite the deviations in this form. If necessary the type area can be moderately adapted by modification of option `DIV`. I advise against use of `\areaset` for this purpose however. At worst you may use geometry package (see [Ume00]), which is not part of

KOMA-Script, or change the type area parameters of L^AT_EX. You may find the values determined by `typearea` in the log file of your document. Thus moderate adjustments should be possible. However, make absolutely sure that the proportions of the text area correspond approximately to those of the page including consideration of the binding correction.

If it should be absolutely necessary to set the text one-and-a-half-lined then you should not redefine `\baselinestretch` under any circumstances. Although this procedure was recommended very frequently, it is obsolete since the introduction of L^AT_EX 2_ε in 1994. Use at least the instruction `\linespread`. I recommend package `setspace` (see [Tob00]), which is not part of KOMA-Script. Also you should use `typearea` to calculate a new type area after the conversion of the line spacing. However, you should switch back to the normal line spacing for the title, preferably also for the table contents and various listings—as well as the bibliography and the index. The `setspace` package offers for this a special environment and its own instructions.

The `typearea` package even with option `DIVcalc` calculates a very generous text area. Many conservative typographers will state that the resulting line length is still excessive. The calculated *DIV* value may be found in the log file to the respective document. Thus you can select a smaller value easily after the first L^AT_EX run.

The question is put to me not infrequently, why I actually talk section after section about type area calculations, when it would be very much simpler to merely give you a package with which one can adjust the edges like during a text processing. Often also it is stated that such a package would be anyway the better solution, since everyone can judge how good edges are to be selected, and that the edges calculated by KOMA-Script are anyway not that great. I take the liberty of translating a suitable quotation from [WF00]. You may find the original German words in the German `scrguide`.

The making by oneself is quite widespread, the results are often doubtful, because layman typographers do not see what is incorrect and cannot not know what it important. Thus one gets accustomed to wrong and poor typography. [...] Now the objection could be made that typography is nevertheless a taste thing. If it concerned decoration, perhaps one could let that argument apply, however, since it concerns primarily information with typography, errors cannot only disturb, but may even cause damage.

The Main Classes `scrbook`, `scrreprt` and `scrartcl`

The main classes of the KOMA-Script bundle are designed as counterparts to the standard \LaTeX classes. This means that the KOMA-Script bundle contains replacements for the three standard classes `book`, `report` and `article`. There is also a replacement for the standard class `letter`. The document class for letters is described in a separate chapter, because it is fundamentally different from the three main classes (see [chapter 6](#)). The names of the KOMA-Script classes are composed of the prefix “`scr`” and the abbreviated name of the corresponding standard class. In order to restrict the length of the names to eight letters, the vowels, starting with the last one, are left off as necessary. The [table 3.1](#) shows an overview of the correspondence between the standard classes and the KOMA-Script classes.

The simplest way to use a KOMA-Script class instead of a standard one is to substitute the class name in the `\documentclass` command according to [table 3.1](#). Normally, the document should be processed without errors by \LaTeX , just like before the substitution. The look however should be different. Additionally, the KOMA-Script classes provide new possibilities and options that are described in the following sections.

Table 3.1: Correspondence between standard classes, KOMA-Script classes and Script styles.

standard class	KOMA-Script class
<code>article</code>	<code>scrartcl</code>
<code>report</code>	<code>scrreprt</code>
<code>book</code>	<code>scrbook</code>
<code>letter</code>	<code>scrlttr2</code>

3.1 The Options

This section describes the global options of the three main classes. The majority of the options can also be found in the standard classes. Since experience shows that many options of the standard classes are unknown, their description is included here. This is a departure from the rule that the `scrguide` should only describe those aspects whose implementation differs from the standard one.

Table 3.2 lists those options that are set by default in at least one of the KOMA-Script classes. The table shows for each KOMA-Script main class if the option is set by default and if it is even defined for that class. An undefined option cannot be set, either by default or by the user.

Table 3.2: Default options of the KOMA-Script classes

Option	scrbook	scrreprt	scrartcl
11pt	default	default	default
a4paper	default	default	default
abstractoff	<i>undefined</i>	default	default
bigheadings	default	default	default
chapteratlists=	10pt	10pt	<i>undefined</i>
final	default	default	default
footnosepline	default	default	default
headnosepline	default	default	default
listsindent	default	default	default
nochapterprefix	default	default	<i>undefined</i>
onelinecaption	default	default	default
notitlepage			default
onecolumn	default	default	default
oneside		default	default
openany		default	default
openright	default		
parindent	default	default	default
tablecaptionbelow	default	default	default
titlepage	default	default	
tocindent	default	default	default
twoside	default		
version=	first	first	first

Allow me an observation before proceeding with the descriptions of the options. It is often the case that at the beginning of a document one is often unsure which options to choose for that specific document. Some options, for instance the choice of paper size, may be fixed from the beginning. But already the question of which *DIV* value to use could be difficult to answer initially. On the other hand, this kind of information should be initially irrelevant for the main tasks of an author: design of the document structure, text writing, preparation of figures, tables and index. As an author you should concentrate initially on the contents. When that is done, you can concentrate on the fine

points of presentation. Besides the choice of options, this means correcting things like hyphenation, page breaks, and the distribution of tables and figures. As an example consider [table 3.2](#), which I moved repeatedly between the beginning and the end of this section. The choice of the actual position will only be made during the final production of the document.

3.1.1 Options for Compatibility

Users who archive their documents as source code generally place great value on obtaining exactly the same output in future L^AT_EX runs. However, in some cases, improvements and corrections to a class can lead to changes in behaviour, particularly as regards line and page breaks.

```
version
version=value
```

v2.96a Since version 2.96a KOMA-Script offers the choice of whether a source file should output as far as possible identical results in future L^AT_EX runs, or whether output should be determined according to the latest changes in the class. The option `version` determines with which version compatibility is to be maintained. The default setting is version 2.9t. The same result can be achieved by setting

```
version=first
```

or

```
version=2.9
```

or

```
version=2.9t.
```

If an unknown version number is given as *value* a warning is output and for safety's sake the option is set to `version=first`. With

```
version=last
```

the current latest version can be selected. In this case future compatibility is switched off. If the option is used without a value, then once again the value of `last` is assumed.

The question of compatibility is first of all a question of line and page breaking. New capabilities, which do not affect page breaks, are also available if the option of compatibility to an older version is selected. The option has no effect on changes in the page breaking when using a newer

version, which result purely through the correction of errors. If absolute compatibility including errors is required, then the requisite KOMA-Script version should be archived along with the document source.

It should be noted that the option `version` cannot be changed after the loading of the class.

3.1.2 Options for Page Layout

With the standard classes the page layout is established by the option files `size10.clo`, `size11.clo`, `size12.clo` (or `bk10.clo`, `bk11.clo`, `bk12.clo` for the book class) and by fixed values in the class definitions. The KOMA-Script classes, however, do not use a fixed page layout, but one that depends on the paper format and font size. For this task all three main classes use the `typearea` package (see [chapter 2](#)). The package is automatically loaded by the KOMA-Script main classes. Therefore it is not necessary to load the package using `\usepackage{typearea}`. If a L^AT_EX run results in an error “Option clash for package `typearea`”, then this is most likely owing to the use of an explicit command `\usepackage[package options]{typearea}`.

```
letterpaper
legalpaper
executivepaper
aXpaper
bXpaper
cXpaper
dXpaper
landscape
```

The basic options for the choice of paper format are not processed directly by the classes. Instead, they are automatically processed by the `typearea` package as global options (see [section 2.4, page 31](#)). The options `a5paper`, `a4paper`, `letterpaper`, `legalpaper` and `executivepaper` correspond to the likewise-named options of the standard classes and define the same paper format. The page layout calculated for each is different, however.

The reason that the options for the A, B, C or D format are not processed by the `typearea` is not because they are global options, but because the KOMA-Script classes explicitly pass them to the `typearea` package. This is caused by the way option processing is implemented in the `typearea` package and by the operation of the underlying option passing and processing mechanism of L^AT_EX.

This is also valid for the options, described subsequently, that set the binding correction, the divisor and the number of header lines.

BCOR*correction*
 DIV*factor*
 DIVcalc
 DIVclassic
 Valueheadlines

The options for the divisor and the binding correction are passed directly to the `typearea` package (see [section 2.4, page 20](#) till [page 29](#)). This differs from the standard classes, where there is no such transfer. This is also valid for the option that sets the number of header lines.

3.1.3 Options for Document Layout

This subsection deals with all the options that affect the document layout in general and not only the page layout. Strictly speaking, of course, all page layout options (see [section 3.1.2](#)) are also document layout options. The reverse is also partially true.

oneside
 twoside

These two options have the same effect as in the standard classes. The option `oneside` defines a one-sided document layout with a one-sided page layout. This means in particular that normally a ragged page bottom is used.

The option `twoside` defines a two-sided document layout with a two-sided page layout. This means that the \LaTeX command `\flushbottom` is used to ensure that page breaks don't leave a variable empty space at the bottom of a page. A ragged page bottom can be obtained with the \LaTeX command `\raggedbottom`.

onecolumn
 twocolumn

These options have the same effect as the corresponding standard options. They are used to switch between a one-column and a two-column layout. The standard \LaTeX capabilities for multi-column layout are only useful for very simple uses. The standard package `multicol` is much more versatile (see [\[Mit00\]](#)).

`openany`
`openright`

scrbook, These options have the same effect as the corresponding standard options. They affect the choice of the page where a chapter can begin, so they are not available with the `scrartcl` class, since the next largest unit below “part” is “section”. The chapter level is not available in `scrartcl`.

A chapter always begins on a new page. When the option `openany` is active, any page can be used. The option `openright` causes the chapter to begin on a new right page. An empty left page may be inserted automatically in this case. The empty pages are created by the implicit execution of the \LaTeX command `\cleardoublepage`.

The option `openright` has no effect with a one-sided layout, because only the two-sided layout differentiates between left and right pages. For this reason it should only be used together with the `twoside` option.

`cleardoublestandard`
`cleardoubleplain`
`cleardoubleempty`

If one wishes the empty pages created by the `\cleardoublepage` command to have no headers but only a page number, or neither headers nor page number while using the standard classes, the only possibility is to redefine the command appropriately. KOMA-Script provides options that avoid this necessity. The option `cleardoublestandard` enables the default `\cleardoublepage` behaviour. If the option `cleardoubleplain` is used, then the `plain` page style is applied to the empty left page. The option `cleardoubleempty` causes the `empty` page style to be used. The page styles are described in [section 3.2.2](#).

`titlepage`
`notitlepage`

Both options have the same effect as the corresponding standard ones. The `titlepage` option makes \LaTeX use separate pages for the titles. These pages are set inside a `titlepage` environment and normally have neither header nor footer. In comparison with standard \LaTeX , KOMA-Script expands the handling of the titles significantly (see [section 3.3](#)).

The option `notitlepage` specifies that an *in-page* title is used. This means that the title is specially emphasized, but it may be followed by more material on the same page, for instance by an abstract or a section.

```

parskip
parskip*
parskip+
parskip-
halfparskip
halfparskip*
halfparskip+
halfparskip-
parindent

```

The standard classes normally set paragraphs indented and without any vertical inter-paragraph space. This is the best solution when using a regular page layout, like the ones produced with the `typearea` package. If neither indentation nor vertical space is used, only the length of the last line would give the reader a reference point. In extreme cases, it is very difficult to detect whether a line is full or not. Furthermore, it is found that a marker at the paragraph's end tends to be easily forgotten by the start of the next line. A marker at the paragraph's beginning is more easily remembered. Inter-paragraph spacing has the drawback of disappearing in some contexts. For instance, after a displayed formula it would be impossible to detect if the previous paragraph continues or if a new one begins. Also, when starting to read at the top of a new page it might be necessary to look at the previous page in order determine if a new paragraph has been started or not. All these problems disappear when using indentation. A combination of indentation and vertical inter-paragraph spacing is redundant and therefore should be avoided. The indentation is perfectly sufficient by itself. The only drawback of indentation is the reduction of the line length. The use of inter-paragraph spacing is therefore justified when using short lines, for instance in a newspaper.

Independently of the explanation above, there are often requests for a document layout with vertical inter-paragraph spacing instead of indentation. KOMA-Script provides a large number of related options: `parskip`, `parskip-`, `parskip*`, `parskip+` and `halfparskip`, `halfparskip-`, `halfparskip*` and `halfparskip+`.

The four `parskip` options each define an inter-paragraph spacing of one line. The four `halfparskip` options use just a spacing of half a line. In order to avoid a change of paragraph going unnoticed, for instance after a page break, three of the options of each set ensure that the last line of a paragraph is not completely filled. The variants without plus or star sign ensure a free space of 1 em. The plus variant ensures that at least a third of the line is free and the star variant ensures that at least a fourth of the line is free. The minus variants make no special provision for the last line of a paragraph.

All eight options also change the spacing before, after and inside list environments. This avoids the problem of these environments or the paragraphs inside them having a larger separation than the separation between the paragraphs of normal text. Additionally, these options ensure that the table of contents and the lists of figures and tables are set without any additional spacing.

The default behaviour of KOMA-Script follows the `parindent` option. In this case, there is no spacing between paragraphs, only an indentation of the first line by 1 em.

`headsepline`
`headnosepline`
`footsepline`
`footnosepline`

In order to have a line separating the header from the text body use the option `headsepline`. The option `headnosepline` has the reverse effect. These options have no effect with the page styles `empty` and `plain`, because there is no header in this case. Such a line always has the effect of visually bringing header and text body closer together. That doesn't mean that the header must now be moved farther from the text body. Instead, the header should be considered as belonging to the text body for the purpose of page layout calculations. KOMA-Script takes this into account by automatically passing the option `headinclude` to the `typearea` package whenever the `headsepline` option is used.

The presence of a line between text body and footer is controlled by the options `footsepline` and `footnosepline`, that behave like the corresponding header functions. Whenever a line is requested by the `footsepline` option, the `footinclude` option is automatically passed to the `typearea` package. In contrast to `headsepline`, `footsepline` takes effect when used together with the page style `plain`, because the `plain` style produces a page number in the footer.

`chapterprefix`
`nochapterprefix`

scrbook, With the standard classes `book` and `report` a chapter title consists of a line `scrrept` with the word “Chapter”¹ followed by the chapter number. The title itself is set left-justified on the following lines. The same effect is obtained in KOMA-Script with the class option `chapterprefix`. The default however

¹When using another language the word “Chapter” is naturally translated to the appropriate language.

is `nchapterprefix`. These options also affect the automatic running titles in the headers (see [section 3.2.2](#)).

`appendixprefix`
`noappendixprefix`

Sometimes one wishes to have the chapter titles in simplified form according to `nchapterprefix`. But at the same time, one wishes a title of an appendix to be preceded by a line with “Appendix” followed by the appendix letter. This is achieved by using the `appendixprefix` option. Since this results in an inconsistent document layout, I advise against using this option.

The reverse option `noappendixprefix` exists only for completeness’ sake. I don’t know of any sensible use for it.

`onelinecaption`
`noonelinecaption`

The standard classes differentiate between one-line and multi-line table or figure captions. One-line captions are centered while multi-line captions are left-justified. This behavior, which is also the default with KOMA-Script, corresponds to the option `onelinecaption`. There is no special handling of one-line captions when the `noonelinecaption` option is given.

The avoidance of a special treatment for the caption has an additional effect that is sometimes highly desirable. Footnotes that appear inside a `\caption` command often have a wrong number assigned to them. This happens because the footnote counter is incremented once as soon as the text is measured to determine if it will be one line or more. When the `noonelinecaption` option is used no such measurement is made. The footnote numbers are therefore correct.

But since KOMA-Script version 2.9 you don’t need the option `noonelinecaption` to avoid the above described effect. KOMA-Script classes contain a workaround, so you can have footnotes inside captions. It should be mentioned though that when using footnotes inside floating environments, the contents of the floating environment should be encapsulated inside a `minipage`. That way it is guaranteed that floating environment and footnote are inseparable.

3.1.4 Options for Font Selection

Font options are those options that affect the font size of the document or the fonts of individual elements. Options that affect the font style are also

theoretically font options. However KOMA-Script currently has no such options.

10pt
11pt
12pt
Xpt

The options `10pt`, `11pt` and `12pt` have the same effect as the corresponding standard options. In contrast to the standard classes, KOMA-Script can be used to choose other font sizes. However, L^AT_EX provides the necessary class option files only for 10 pt, 11 pt und 12 pt, and KOMA-Script does not provide any class option files, so the user must provide any other class option files. The package `extsizes` (see [Kil99]), for example, can be used to provide a `size14.clo` class file. Very big font sizes may lead to arithmetic overflow inside the page layout calculations of the `typearea` package.

smallheadings
normalheadings
bigheadings

The font size used for the titles is relatively big, both with the standard classes and with KOMA-Script. Not everyone likes this choice; moreover it is specially problematic for small paper sizes. Consequently, KOMA-Script provides, besides the large title font size defined by the `bigheadings` option, the two options `normalheadings` and `smallheadings`, that allow for smaller title font sizes. The font sizes for headings resulting from these options for `scrbook` and `scrreprt` are shown in [table 3.8](#), [page 75](#). For `scartcl`

`scrbook`, smaller font sizes are generally used.

`scrreprt` The spacing before and after chapter titles is also influenced by these options. Chapter titles are also influenced by the options `chapterprefix` and `nochapterprefix`, and appendix titles by the options `appendixprefix` and `noappendixprefix`, all of which are described in [section 3.1.3](#), [page 44](#).

3.1.5 Options Affecting the Table of Contents

KOMA-Script has several options that affect the entries in the table of contents. The form of the table of contents is fixed but several variations can be obtained with the options provided.

```
liststotoc  
idxtotoc  
bibtotoc  
bibtotocnumbered  
liststotocnumbered
```

Normally, lists of tables and figures, index and bibliography are not included in the table of contents. These entries are purposely omitted in classical typography because, among other things, a very particular placement of these items is silently assumed, if they are present at all:

- table of contents after the title pages,
- lists of tables and figures after the table of contents,
- index right at the end,
- bibliography before the index.

Books, in which all these items are present, often include ribbons that can be used to mark the location of these items in the book, so that the reader only has to look for them once.

It is becoming increasingly common to find entries in the table of contents for the lists of tables and figures, for the bibliography, and, sometimes, even for the index. This is surely also related to the recent trend of putting lists of figures and tables at the end of the document. Both lists are similar to the table of contents in structure and intention. I'm therefore sceptical of this evolution. Since it makes no sense to include only one of the lists of tables and figures in the table of contents, there exists only one option `liststotoc` that causes entries for both types of lists to be included. This also includes any lists produced with version 1.2e or later of the `float` package (see [Lin01]). All these lists are unnumbered, since they contain entries that reference other sections of the document.

The option `idxtotoc` causes an entry for the index to be included in the table of contents. The index is unnumbered since it too only includes references to the contents of the other sectional units.

The bibliography is a different kind of listing. It does not list the contents of the present document but refers instead to external documents. For that reason, it could be argued that it qualifies as a chapter (or section) and, as such, should be numbered. The option `bibtotocnumbered` has this effect, including the generation of the corresponding entry in the table of contents. I personally think that this reasoning would lead us to consider a classical list of sources also to be a separate chapter. On the other hand, the

bibliography is finally not something that was written by the document's author. In view of this, the bibliography merits nothing more than an unnumbered entry in the table of contents, and that can be achieved with the `bibtotoc` option.

v2.8q

As the author of KOMA-Script already views the option `liststotoc` with open skepticism, and frankly detests option `bibtotocnumbered`, it should come as no surprise that he implemented option `liststotocnumbered` only under extreme duress. He fears that as a next step someone will want the table of contents numbered and entered in the table of contents. Therefore, those looking in this documentation for a detailed description of option `liststotoc` will search in vain. A similar option for the index would be just as silly, so its implementation has been determinedly refused so far.

<code>tocindent</code> <code>tocleft</code>

v2.8q

The table of contents is normally set up so that different sectional units have different indentations. The section number is set left-justified in a fixed-width field. This setup is selected with the option `tocindent`.

When there are many sections, the corresponding numbering tends to become very wide, so that the reserved field overflows. The FAQ [RNH02] suggests that the table of contents should be redefined in such a case. KOMA-Script offers an alternative format that avoids the problem completely. If the option `tocleft` is selected, then no variable indentation is applied to the titles of the sectional units. Instead, a table-like organisation is used, where all unit numbers and titles, respectively, are set in a left-justified column. The space necessary for the unit numbers is thus determined automatically.

In order to calculate automatically the space taken by the unit numbers when using the option `tocleft` it is necessary to redefine some macros. It is improbable but not impossible that this leads to problems when using other packages. If you think this may be causing problems, you should try the alternative option `tocindent`, since it does not make any redefinitions. When using packages that affect the format of the table of contents, it is possible that the use of options `tocleft` and `tocindent` too may lead to problems. When using such packages then, for safety's sake, one should refrain from using either of these options as global (class) options.

If the `tocleft` option is active, the width of the field for unit numbering is determined when outputting the table of contents. After a change that affects the table of contents, at most three L^AT_EX runs are necessary to obtain

a correctly set table of contents.

3.1.6 Options for Lists of Floats

The best known lists of floats are the list of figures and the list of tables. Additionally, with help from the `float` package, for instance, it is possible to produce new float environments with corresponding lists.

Whether KOMA-Script options have any effect on lists of floats produced by other packages depends mainly on those packages. This is generally the case with the lists of floats produced by the `float` package.

Besides the options described here, there are others that affect the lists of floats though not their formatting or contents. Instead they affect what is included in the table of contents. The corresponding descriptions can therefore be found in [section 3.1.5](#).

`listsindent`
`listsleft`

v2.8q

Lists of figures and tables are generally set up so that their numbering uses a fixed space. This corresponds to the use of option `listsindent`.

If the numbers become too large, for instance because many tables are used, it may happen that the available space is exceeded. Therefore KOMA-Script supplies an option called `listsleft` that is similar to the `tocleft` option. The width of the numbers is automatically determined and the space for them correspondingly adjusted. Concerning the mode of operation and the side effects, the observations made in [section 3.1.5](#), [page 48](#) for the `tocleft` option are equally valid in this case. Please note that when using the `listsleft` option several L^AT_EX runs are necessary before the lists of floats achieve their final form.

`chapteratlists`
`chapteratlists=value`

scrbook, Normally, every chapter entry generated with `\chapter` introduces vertical spacing into the lists of floats. Since version 2.96a this applies also for the `command \addchap`, if no compatibility option to an earlier version was chosen (see option `version` in [section 3.1.1](#), [page 39](#)).

v2.96a

Furthermore, now the option `chapteratlists` can be used to change the spacing, by passing the desired distance as *value*. The default setting is 10 pt. If the value is set to `entry` or no value is specified, then instead of a vertical distance the chapter entry itself will be entered into the lists.

This option can be changed with `\KOMAoptions{chapteratlists}` or `\KOMAoptions{chapteratlists=value}` even inside the document. It

takes effect from the next heading onwards. However, changes to the option will only become effective in the lists following two more L^AT_EX runs.

3.1.7 Options Affecting the Formatting

Formatting options are all those options that affect the form or formatting of the document and cannot be assigned to other sections. They are therefore the *remaining options*.

abstracton
abstractoff

`scrreprt`, In the standard classes the **abstract** environment sets the text “Abstract” centered before the summary text. This was normal practice in the past. `scrartcl` In the meantime, newspaper reading has trained readers to recognize a displayed text at the beginning of an article or report as the abstract. This is even more true when the text comes before the table of contents. It is also surprising when precisely this title appears small and centered. KOMA-Script provides the possibility of including or excluding the abstract’s title with the options **abstracton** and **abstractoff**.

Books typically use another type of summary. In that case there is usually a dedicated summary chapter at the beginning or end of the book. This chapter is often combined with the introduction or a description of wider prospects. Therefore, the class `scrbook` has no **abstract** environment. A summary chapter is also recommended for reports in a wider sense, like a Master’s or Ph.D. thesis.

pointednumbers
pointlessnumbers

In German, according to DUDEN, the numbering of sectional units should have no dot at the end if only arabic numbers are used (see [DUD96, R3]). On the other hand, if roman numerals or letters are appear in the numbering, then a dot should appear at the end of the numbering (see [DUD96, R4]). KOMA-Script has an internal mechanisms that tries to implement this somewhat complex rule. The resulting effect is that, normally, after the sectional commands `\part` and `\appendix` a switch is made to numbering with an ending dot. The information is saved in the aux file and takes effect on the next L^AT_EX run.

In some cases the mechanism for placing or leaving off the ending dot may fail, or other languages may have different rules. Therefore it is

possible to activate the use of the ending dot manually with the option `pointednumbers` or to deactivate it with `pointlessnumbers`.

Please note that the mechanism only takes effect on the next \LaTeX run. Therefore, before trying to use these options to forcibly control the numbering format, a further run without changing any options should be made.

Calling these options `dottednumbers` and `dotlessnumbers` or similar would be more correct. It so happened that the meaning of the chosen names was not clear to me a few years ago when the options were implemented. Some people asked me not to fix this “funny little mistake” so I didn’t.

`leqno`

Equations are normally numbered on the right. The standard option `leqno` causes the standard option file `leqno.clo` to be loaded. The equations are then numbered on the left.

`fleqn`

Displayed equations are normally centered. The standard option `fleqn` causes the standard option file `fleqn.clo` to be loaded. Displayed equations are then left-justified.

`tablecaptionbelow` `tablecaptionabove`

As described in [section 3.6.6, page 101](#), the `\caption` command acts with figures like the `\captionbelow` command. The behaviour with tables, however, depends on these two options. In the default setting, `tablecaptionbelow`, the `\caption` macro acts also with tables like the `\captionbelow` command. With the `tablecaptionabove` option, `\caption` acts like the `\captionabove` command.

Note that using any of these options does not change the position of the caption from above the top of the table to below the bottom of the table or vica versa. It only affects whether the text is formatted as a caption for use above or below a table. Whether the text is in fact placed above or below a table is set through the position of the `\caption` command inside the `table` environment.

float

Note that when using the `float` package, the options `tablecaptionbelow` and `tablecaptionabove` cease to act correctly when `\restylefloat` is applied to tables. More details of the `float` package and `\restylefloat` can be found in [\[Lin01\]](#). Additional support in KOMA-Script for the float

package may be found at the explanation of `komaabove` in [section 3.6.6](#), [page 106](#).

`origlongtable`

The package `longtable` (see [\[Car98\]](#)) sets table captions internally by calling the command `\LT@makecaption`. In order to ensure that these table captions match the ones used with normal tables, the KOMA-Script classes normally redefine that command. See [section 3.6.6](#), [page 103](#) for more details. The redefinition is performed with help of the command `\AfterPackage` immediately after the loading of package `longtable`. If the package `caption2` (see [\[Som04\]](#)) has been previously loaded, the redefinition is not made in order not to interfere with the `caption2` package.

If the table captions produced by the `longtable` package should not be redefined by the KOMA-Script classes, activate the `origlongtable` option.

`openbib`

The standard option `openbib` switches to an alternative bibliography format. The effects are twofold: The first line of a bibliography entry, normally containing the author's name, receives a smaller indentation; and the command `\newblock` is redefined to produce a paragraph. Without this option, `\newblock` introduces only a stretchable horizontal space.

`draft`
`final`

The two standard options `draft` and `final` are normally used to distinguish between the draft and final versions of a document. In particular, the option `draft` activates small black boxes that are set at the end of overly long lines. The boxes help the untrained eye to find paragraphs that have to be treated manually. With the `final` option no such boxes are shown.

The two options are also processed by other packages and affect their operation. For instance, the `graphics` and the `graphicx` packages don't actually output the graphics when the option `draft` is specified. Instead they output a framed box of the appropriate size containing only the graphic's filename (see [\[Car99b\]](#)).

3.2 General Document Characteristics

Some document characteristics do not apply to a particular section of the document like the titling, the text body or the bibliography, but do affect

the entire document. Some of these characteristics were already described in [section 3.1](#).

3.2.1 Changing Fonts

KOMA-Script does not use fixed fonts and attributes to emphasize different elements of the text. Instead there are variables that contain the commands used for changing fonts and other text attributes. In previous versions of KOMA-Script the user had to use `\renewcommand` to redefine those variables. It was also not easy to determine the name of the variable affecting an element given the element's name. Besides, it was also often necessary to determine the original definition before proceeding to redefine it.

These difficulties were actually intended, since the interface was not for users, but only for package authors building their packages on top of KOMA-Script. The years have shown, however, that the interface was in fact mainly used by document authors. So a new, simpler interface was created. However, the author explicitly advises the typographically inexperienced user against changing font sizes and other graphical characteristics according to his taste. Knowledge and feeling are basic conditions for the selection and mixture of different font sizes, attributes and families.

```
\setkomafont{element}{commands}
\addtokomafont{element}{commands}
\usekomafont{element}
```

v2.8p

With the help of the two commands `\setkomafont` and `\addtokomafont` it is possible to define the *commands* that change the characteristics of a given *element*. Theoretically all possible statements including literal text could be used as *commands*. You should however absolutely limit yourself to those statements that really switch only one font attribute. This will usually be the commands `\normalfont`, `\rmfamily`, `\sffamily`, `\ttfamily`, `\mdseries`, `\bfseries`, `\upshape`, `\itshape`, `\slshape`, `\scshape` and the font size commands `\Huge`, `\huge`, `\LARGE`, etc. The description of these commands can be found in [OPHS99], [Tea01] or [Tea00]. Color switching commands like `\normalcolor` (see [Car99b]) are also acceptable. The behavior when using other commands, specially those that make re-definitions or generate output, is not defined. Strange behavior is possible and does not represent a bug.

The command `\setkomafont` provides a font switching command with a completely new definition. In contrast to this the `\addtokomafont` command merely extends an existing definition. It is recommended to not use

both commands in the same document. Usage examples can be found in the paragraphs on the corresponding element. Names and meanings of the individual items are listed in [table 3.3](#). The default values are shown in the corresponding paragraphs.

The command `\usekomafont` can change the current font specification to the one currently used with the specified *element*.

Example: Assume that you want to use for the element `captionlabel` the same font specification that is used with `descriptionlabel`. This can be easily done with:

```
\setkomafont{captionlabel}{\usekomafont{↔
descriptionlabel}}
```

You can find other examples in the paragraphs on each element.

Table 3.3: Elements, whose type style can be changed with the KOMA-Script command `\setkomafont` or `\addtokomafont`

<code>caption</code>	Text of a table or figure caption
<code>captionlabel</code>	Label of a table or figure caption; used according to the element <code>caption</code>
<code>chapter</code>	Title of the sectional unit <code>\chapter</code>
<code>descriptionlabel</code>	Labels, i.e., the optional argument of <code>\item</code> in the <code>description</code> environment
<code>dictum</code>	wise saying (see command <code>\dictum</code>)
<code>dictumauthor</code>	Author of a wise saying; used according to the element <code>dictumtext</code>
<code>dictumtext</code>	Another name for <code>dictum</code>

Table 3.3: Elements, whose type style can be changed (*continuation*)

<code>disposition</code>	All sectional unit titles, i.e., the arguments of <code>\part</code> down to <code>\subparagraph</code> and <code>\minisec</code> , including the title of the abstract; used before the element of the corresponding unit
<code>footnote</code>	Footnote text and marker
<code>footnotelabel</code>	Mark of a footnote; used according to the element <code>footnote</code>
<code>footnotereference</code>	Footnote reference in the text
<code>minisec</code>	Title of <code>\minisec</code>
<code>pagefoot</code>	The foot of a page, but also the head of a page
<code>pagehead</code>	The head of a page, but also the foot of a page
<code>pagenumber</code>	Page number in the header or footer
<code>pagination</code>	Another name for <code>pagenumber</code>
<code>paragraph</code>	Title of the sectional unit <code>\paragraph</code>
<code>part</code>	Title of the <code>\part</code> sectional unit, without the line containing the part number
<code>partnumber</code>	Line containing the part number in a title of the sectional unit <code>\part</code>
<code>section</code>	Title of the sectional unit <code>\section</code>

Table 3.3: Elements, whose type style can be changed (*continuation*)

<code>sectioning</code>	Another name for <code>disposition</code>
<code>subparagraph</code>	Title of the sectional unit <code>\subparagraph</code>
<code>subsection</code>	Title of the sectional unit <code>\subsection</code>
<code>subsubsection</code>	Title of the sectional unit <code>\subsubsection</code>
<code>title</code>	Main title of the document, i.e., the argument of <code>\title</code> (for details about the title size see the additional note in the text from page 66)

3.2.2 Page Style

One of the general characteristics of a document is the page style. In \LaTeX this means mostly the contents of headers and footers.

```

\pagestyle{empty}
\pagestyle{plain}
\pagestyle{headings}
\pagestyle{myheadings}
\thispagestyle{local page style}

```

Usually one distinguishes four different page styles.

empty is the page style with entirely empty headers and footers. In KOMA-Script this is completely identical to the standard classes.

plain is the page style with empty header and only a page number in the footer. With the standard classes this page number is always centered in the footer. With KOMA-Script the page number appears on double-sided layout on the outer side of the footer. The one-sided page style behaves like the standard setup.

headings is the page style with running headings in the header. These are headings for which titles are automatically inserted into the header. With the classes `scrbook` and `scrreprt` the titles of chapters

and sections are repeated in the header for double-sided layout — with KOMA-Script on the outer side, with the standard classes on the inner side. The page number is set on the outer side of the footer with KOMA-Script, with the standard classes it is set on the inner side of the header. In one-sided layouts only the titles of the chapters are used and are, with KOMA-Script, centered in the header. The page numbers are set centered in the footer with KOMA-Script. `scrartcl` behaves similarly, but starting a level deeper in the section hierarchy with sections and subsections, because the chapter level does not exist in this case.

While the standard classes automatically set running headings always in capitals, KOMA-Script applies the style of the title. This has several typographic reasons. Capitals as a decoration are actually far too strong. If one applies them nevertheless, they should be set in a one point smaller type size and with tighter spacing. The standard classes do not take these points in consideration.

myheadings corresponds mostly to the page style **headings**, but the running headings are not automatically produced, but have to be defined by the user. The commands `\markboth` and `\markright` can be used for that purpose.

Besides, the form of the page styles **headings** and **myheadings** is affected by each of the four class options `headsepline`, `headnosepline`, `footsepline` and `footnosepline` (see [section 3.1.3](#), [page 44](#)). The page style starting with the current page is changed by the command `\pagestyle`. On the other hand `\thispagestyle` changes only the style of the current page.

The page style can be set at any time with the help of the `\pagestyle` command and takes effect with the next page that is output. Usually one sets the page style only once at the beginning of the document or in the preamble. To change the page style of the current page only, one uses the `\thispagestyle` command. This also happens automatically at some places in the document. For example, the instruction `\thispagestyle{plain}` is issued implicitly on the first page of a chapter.

Please note that the change between automatic and manual running headings is no longer performed by page style changes when using the `scrpage2` package, but instead via special instructions. The page styles **headings** and **myheadings** should not be used together with this package (see [chapter 4](#), [page 124](#)).

Table 3.4: Default values for the elements of a page style

Element	Default value
pagefoot	<code>\normalfont\normalcolor\slshape</code>
pagehead	<code>\normalfont\normalcolor\slshape</code>
pagenumber	<code>\normalfont\normalcolor</code>

v2.8p

In order to change the type style used in the header, footer or for the page number, please use the interface described in [section 3.2.1](#). The same element is used for header and footer, which you can designate equivalently with `pagehead` or `pagefoot`. The element for the page number within the header or footer is called `pagenumber`. The default settings can be found in [table 3.4](#).

Example: Assume that you want to set header and footer in a smaller type size and in italics. However, the page number should not be set in italics but bold. Apart from the fact that the result will look horrible, you can obtain this as follows:

```
\setkomafont{pagehead}{%
  \normalfont\normalcolor\itshape\small
}
\setkomafont{pagenumber}{\normalfont\bfseries}
```

If you want only that in addition to the default slanted variant a smaller type size is used, it is sufficient to use the following:

```
\addtokomafont{pagefoot}{\small}
```

As you can see, the last example uses the element `pagefoot`. You can achieve the same result using `pagehead` instead (see [table 3.3](#) on [page 54](#)).

It is not possible to use these methods to force capitals to be used automatically for the running headings. For that, please use the `scrpage2` package (see [chapter 4](#), [page 133](#)).

If you define your own page styles, the commands `\usekomafont{pagehead}` and `\usekomafont{pagenumber}` can be useful. If you do not use the KOMA-Script package `scrpage2` (see [chapter 4](#)) for that, but, for example, the package `fancyhdr` (see [\[vO00\]](#)), you can use these commands in your definitions. Thereby you can remain

Table 3.5: Macros to set up page style of special pages

`\titlepagestyle`

Page style for a title page when using *in-page* titles.

`\partpagestyle`

Page style for the pages with `\part` titles.

`\chapterpagestyle`

Page style for the first page of a chapter.

`\indexpagestyle`

Page style for the first page of the index.

compatible with KOMA-Script as much as possible. If you do not use these commands in your own definitions, changes like those shown in the previous examples have no effect. The packages `scrpage` and `scrpage2` take care to keep the maximum possible compatibility with other packages.

```
\titlepagestyle
\partpagestyle
\chapterpagestyle
\indexpagestyle
```

For some pages a different page style is chosen with the help of the command `\thispagestyle`. Which page style this actually is, is defined by these four macros, of which `\partpagestyle` and `\chapterpagestyle` are found only with classes `scrbook` and `scrreprt`, but not in `scrartcl`. The default value for all four cases is `plain`. The meaning of these macros can be taken from [table 3.5](#). The page styles can be redefined with the `\renewcommand` macro.

Example: Assume that you want the pages with a `\part` heading to have no number. Then you can use the following command, for example in the preamble of your document:

```
\renewcommand*{\partpagestyle}{empty}
```

As mentioned previously on [page 56](#), the page style `empty` is exactly what is required in this example. Naturally you can also use a user-defined page style.

Assume you have defined your own page style for initial chapter pages with the package `scrpage2` (see [chapter 4](#)).

You have given to this page style the fitting name `chapter`. To actually use this style, you must redefine the macro `\chapterpagestyle` accordingly:

```
\renewcommand*{\chapterpagestyle}{chapter}
```

Assume that you want that the table of contents of a book to have no page numbers. However, everything after the table of contents should work again with the page style `headings`, as well as with `plain` on every first page of a chapter. You can use the following commands:

```
\clearpage
\pagestyle{empty}
\renewcommand*{\chapterpagestyle}{empty}
\tableofcontents
\clearpage
\pagestyle{headings}
\renewcommand*{\chapterpagestyle}{plain}
```

Instead of the above you may do a local redefinition using a group. The advantage will be that you don't need to know the current page style before the change to switch back at the end.

```
\clearpage
\begingroup
  \pagestyle{empty}
  \renewcommand*{\chapterpagestyle}{empty}
  \tableofcontents
  \clearpage
\endgroup
```

But notice that you never should put a numbered head into a group. Otherwise you may get funny results with commands like `\label`.

Whoever thinks that it is possible to put running headings on the first page of a chapter by using the command

```
\renewcommand*{\chapterpagestyle}{headings}
```

will be surprised at the results. For sure, the page style `headings` is thereby applied to the initial page of a chapter. But nevertheless no running headings

appear when using the `openright` option. The reason for this behaviour can be found in the \LaTeX core. There, the command `\rightmark`, that generates the marks for right-hand pages, is defined with;

```
\let\@rightmark\@secondoftwo
\def\rightmark{\expandafter\@rightmark
\firstmark\@empty\@empty}
```

The right-hand mark is set with `\firstmark`. `\firstmark` contains the left-hand and right-hand marks that were first set for a page. Within `\chapter`, `\markboth` is used to set the left mark to the chapter header and the right mark to empty. Hence, the first right mark on a chapter beginning with a right-hand page is empty. Therefore, the running heading is also empty on those pages.

You could redefine `\rightmark` in the preamble so that the last mark on the page is used instead of the first:

```
\makeatletter
\renewcommand*{\rightmark}{\%
\expandafter\@rightmark\botmark\@empty\@empty}
\makeatother
```

This would however cause the running heading of the first page of a chapter to use the title of the last section in the page. This is confusing and should be avoided.

It is also confusing (and hence should be avoided) to have as running heading of the first page of a chapter the chapter title instead of the the section title. Therefore, the current behavior should be considered to be correct.

```
\clearpage
\cleardoublepage
\cleardoublestandardpage
\cleardoubleplainpage
\cleardoubleemptypage
```

The \LaTeX core contains the `\clearpage` command, which takes care that all not yet output floats are output, and then starts a new page. There exists the instruction `\cleardoublepage` which works like `\clearpage` but which, in the double-sided layouts (see layout option `twoside` in [section 3.1.3, page 41](#)) starts a new right-hand page. An empty left page in the current page style is output if necessary.

With `\cleardoublestandardpage` KOMA-Script works as described above. The `\cleardoubleplainpage` command changes the page style

of the empty left page to `plain` in order to suppress the running heading. Analogously, the page style `empty` is applied to the empty page with `\cleardoubleemptypage`, suppressing the page number as well as the running heading. The page is thus entirely empty. However, the approach used by `\cleardoublepage` is dependent on the layout options `cleardoublestandard`, `cleardoubleplain` and `cleardoubleempty` described in [section 3.1.3, page 42](#) and acts according to the active option.

```
\ifthispageodd{true}{false} \ifthispagewasoddtrue\elsefalse\fi
```

A peculiarity of \LaTeX consists of the fact that it is not possible to determine on which page the current text will fall. It is also difficult to say whether the current page has an odd or an even page number. Now some will argue that there is, nevertheless, the \TeX test macro `\ifodd` which one needs only to apply to the current page counter. However, this is an error. At the time of the evaluation of such a test \LaTeX does not know at all whether the text just processed will be typeset on the current page or only on the next. The page breaks take place not while reading the paragraph, but only in the *output* routine of \LaTeX . However, at that moment a command of the form `\ifodd\value{page}` would already have been completely evaluated.

To find out reliably whether a text falls on an even or odd page, one must usually work with a label and a page reference to this label. One must also take special precautionary measures during the first \LaTeX run, when the label is not yet known.

If one wants to find out with KOMA-Script whether a text falls on an even or odd page, one can use the `\ifthispageodd` command. The *true* argument is executed only if the command falls on an odd page. Otherwise the *false* argument is executed.

More precisely stated, the question is not where the text is, but whether a page reference to a label placed in this location would refer to an odd or an even page.

Example: Assume that you want to indicate if an odd or even page is output. This could be achieved with the command:

```
This is a page with an \ifthispageodd{odd}{even←
    }
page number.
```

The output would then be:

This is a page with an odd page number.

Because the `\ifthispageodd` command uses a mechanism that is very similar to a label and a reference to it, at least two \LaTeX runs are required after every text modification. Only then the decision is correct. In the first run a heuristic is used to make the first choice.

There are situations where the `\ifthispageodd` command never leads to the correct result. Suppose that the command is used within a box. A box is set by \LaTeX always as a whole. No page breaks take place inside. Assume further that the *true* part is very big, but the *false* part is empty. If we suppose further that the box with the *false* part still fits on the current, even page, but that with the *true* part it does not. Further assume that KOMA-Script heuristically decides for the first run that the *true* part applies. The decision is wrong and is revised in the next run. The *false* part is thereby processed, instead of the *true* part. The decision must again be revised in the next run and so on.

These cases are rare. Nevertheless it should not be said that I have not pointed out that they are possible.

Sometimes you need to know the state of the last decision. This may be done using the expert command `\ifthispagewasodd`. This is either same like `\iftrue` or `\iffalse` and may be used like those.

`\pagenumbering{numbering style}`

This command works the same way in KOMA-Script as in the standard classes. More precisely it is a command from the \LaTeX kernel. You can specify with this command the *numbering style* of page numbers. The changes take effect immediately, hence starting with the page that contains the command. The possible settings can be found in [table 3.6](#). Using the command `\pagenumbering` also resets the page counter. Thus the page number of the next page which \TeX outputs will have the number 1 in the style *numbering style*.

3.3 Titles

After having described the options and some general issues, we begin the document where it usually begins: with the titles. The titles comprise everything that belongs in the widest sense to the title of a document. Like already mentioned in [section 3.1.3, page 42](#), we can distinguish between title pages and *in-page* titles. Article classes like `article` or `scrartcl` have by default *in-page* titles, while classes like `report`, `book`, `scrreprt` and `scrbook` have title pages

Table 3.6: Available numbering styles of page numbers

numbering style	example	description
arabic	8	Arabic numbers
roman	viii	lower-case Roman numbers
Roman	VIII	upper-case Roman numbers
alph	h	letters
Alph	H	capital letters

as default. The defaults can be changed with the class options `titlepage` and `notitlepage`.

`titlepage`

With the standard classes and with KOMA-Script all title pages are defined in a special environment, the `titlepage` environment. This environment always starts a new page — in the two-sided layout a new right page. For this page, the style is changed by `\thispagestyle{empty}`, so that neither page number nor running heading are output. At the end of the environment the page is automatically shipped out. Should you not be able to use the automatic layout of the title page, it is advisable to design a new one with the help of this environment.

Example: Assume you want a title page on which only the word “Me” stands at the top on the left, as large as possible and in bold — no author, no date, nothing else. The following document creates just that:

```
\documentclass{scrbook}
\begin{document}
\begin{titlepage}
  \textbf{\Huge Me}
\end{titlepage}
\end{document}
```

Simple? Right.

`\maketitle[page number]`

While the the standard classes produce a title page that may have the three items title, author and date, with KOMA-Script the `\maketitle` command can produce up to six pages.

In contrast to the standard classes, the `\maketitle` macro in KOMA-Script accepts an optional numeric argument. If it is used, this number is made the page number of the first title page. However, this page number is not output, but affects only the numbering. You should choose an odd number, because otherwise the whole counting gets mixed up. In my opinion there are only two meaningful applications for the optional argument. On the one hand, one could give to the half-title the logical page number -1 in order to give the full title page the number 1. On the other hand, it could be used to start at a higher page number, for instance, 3, 5, or 7 to accommodate other title pages added by the publishing house. The optional argument is ignored for *in-page* titles. However, the page style of such a title page can be changed by redefining the `\titlepagestyle` macro. For that see [section 3.2.2, page 59](#).

The following commands do not lead necessarily to the production of the titles. The typesetting of the title pages is always done by `\maketitle`. The commands explained below only define the contents of the title pages. It is however not necessary, and when using the `babel` package not recommended, to use these in the preamble before `\begin{document}` (see [\[Bra01\]](#)). Examples can be found at the end of this section.

`\extratitle{half-title}`

In earlier times the inner book was often not protected from dirt by a cover. This task was then taken over by the first page of the book which carried mostly a shortened title called the *half-title*. Nowadays the extra page is often applied before the real full title and contains information about the publisher, series number and similar information.

With KOMA-Script it is possible to include a page before the real title page. The *half-title* can be arbitrary text—even several paragraphs. The contents of the *half-title* are output by KOMA-Script without additional formatting. Their organisation is completely left to the user. The back of the half-title remains empty. The half-title has its own title page even when *in-page* titles are used. The output of the half-title defined with `\extratitle` takes place as part of the titles produced by `\maketitle`.

Example: Let's go back to the previous example and assume that the spartan “Me” is the half-title. The full title should still follow the half-title. One can proceed as follows:

```
\documentclass{scrbook}
\begin{document}
```

```

\extratitle{\textbf{\Huge Me}}
\title{It's me}
\maketitle
\end{document}

```

You can center the half-title and put it a little lower down the page:

```

\documentclass{scrbook}
\begin{document}
\extratitle{\vspace*{4\baselineskip}
\begin{center}\textbf{\Huge Me}\end{center}}
\title{It's me}
\maketitle
\end{document}

```

The command `\title` is necessary in order to make the examples above work correctly. It is explained next.

```

\titlehead{Titlehead}
\subject{Subject}
\title{Title}
\author{Author}
\date{Date}
\publishers{Publisher}
\and
\thanks{Footnote}

```

The contents of the full title page are defined by six elements. The *title head* is defined with the command `\titlehead`. It is typeset in regular paragraph style and full width at the top of the page. It can be freely designed by the user.

The *Subject* is output immediately above the *Title*. A slightly larger font size than the regular one is used.

v2.8p

The *Title* is output with a very large font size. Besides the change of size, the settings for the element `title` also take effect. By default these settings are identical to the settings for the element `disposition` (see [table 3.3, page 54](#)). The font size is however not affected (see [table 3.3, page 67](#)). The default settings can be changed with the commands of [section 3.2.1](#).

Below the *Title* appears the *Author*. Several authors can be specified in the argument of `\author`. They should be separated by `\and`.

Table 3.7: Font size and horizontal positioning of the elements in the main title page in the order of their vertical position from top to bottom when typeset with `\maketitle`

Element	Command	Font	Justification
Title head	<code>\titlehead</code>	<code>\normalsize</code>	Regular paragraph
Subject	<code>\subject</code>	<code>\Large</code>	centered
Title	<code>\title</code>	<code>\huge</code>	centered
Authors	<code>\author</code>	<code>\Large</code>	centered
Date	<code>\date</code>	<code>\Large</code>	centered
Publishers	<code>\publishers</code>	<code>\Large</code>	centered

Below the author or authors appears the date. The default value is the present date, as produced by `\today`. The `\date` command accepts arbitrary information or even an empty argument.

Finally comes the *Publisher*. Of course this command can also be used for any other information of little importance. If necessary, the `\parbox` command can be used to typeset this information over the full page width like a regular paragraph. Then it is to be considered equivalent to the title head. However, note that this field is put above any existing footnotes.

Footnotes on the title page are produced not with `\footnote`, but with `\thanks`. They serve typically for notes associated with the authors. Symbols are used as footnote markers instead of numbers.

With the exception of *titlehead* and possible footnotes, all the items are centered horizontally. The information is summarised in [table 3.7](#).

Example: Assume you are writing a dissertation. The title page should have the university’s name and address at the top, flush left, and the semester flush right. As usual a title is to be used, including author and delivery date. The adviser must also be indicated, together with the fact that the document is a dissertation. This can be obtained as follows:

```
\documentclass{scrbook}
\begin{document}
\titlehead{{\Large Unseen University
\hfill SS~2002\\}
Higher Analytical Institute\\
Mythological Rd\\
34567 Etherworld}
```

```

\subject{Dissertation}
\title{Digital space simulation with the DSP←
      \,56004}
\author{Fuzzy George}
\date{30. February 2002}
\publishers{Adviser Prof. John Eccentric Doe}
\maketitle
\end{document}

```

A frequent misunderstanding concerns the role of the full title page. It is often erroneously assumed that the cover (or dust cover) is meant. Therefore, it is frequently expected that the title page does not follow the normal page layout, but has equally large left and right margins.

However if one takes a book and opens it, one notices very quickly at least one title page under the cover within the so-called inner book. Precisely these title pages are produced by `\maketitle`. As is the case with the half-title, the full title page belongs to the inner book, and therefore should have the same page layout as the rest of the document. A cover is actually something that should be created in a separate document. The cover often has a very individual format. It can also be designed with the help of a graphics or DTP program. A separate document should also be used because the cover will be printed on a different medium, possibly cardboard, and possibly with another printer.

```

\uppertitleback{titlebackhead}
\lowertitleback{titlebackfoot}

```

With the standard classes, the back of the title page is left empty. However, with KOMA-Script the back of the full title page can be used for other information. Exactly two elements which the user can freely format are recognized: *titlebackhead* and *titlebackfoot*. The head can reach up to the foot and vice versa. If one takes this manual as an example, the exclusion of liability was set with the help of the `\uppertitleback` command.

```
\dedication{dedication}
```

KOMA-Script provides a page for dedications. The dedication is centered and uses a slightly larger type size. The back is empty like the back page of the half-title. The dedication page is produced by `\maketitle` and must therefore be defined before this command is issued.

Example: This time assume that you have written a poetry book and you want to dedicate it to your wife. A solution would look like this:

```
\documentclass{scrbook}
\begin{document}
\extratitle{\textbf{\Huge In Love}}
\title{In Love}
\author{Prince Ironheart}
\date{1412}
\lowertitleback{This poem book was set with%
    the help of {\KOMAScript} and {\LaTeX}}
\uppertitleback{Selfmockery Publishers}
\dedication{To my treasure hazel-hen\\
    in eternal love\\
    from your dormouse.}
\maketitle
\end{document}
```

Please use your own favorite pet names.

abstract

Particularly with articles, more rarely with reports, there is a summary directly under the title and before the table of contents. Therefore, this is often considered a part of the titles. Some L^AT_EX classes offer a special environment for this summary, the **abstract** environment. This is output directly, at it is not a component of the titles set by `\maketitle`. Please note that **abstract** is an environment, not a command. Whether the summary has a heading or not is determined by the options **abstracton** and **abstractoff** (see [section 3.1.7, page 50](#))

With books (`scrbook`) the summary is frequently a component of the introduction or a separate chapter at the end of the document. Therefore no **abstract** environment is provided. When using the class `scrreprt` it is surely worth considering whether one should not proceed likewise.

3.4 The Table of Contents

The titles are normally followed by the table of contents. Often the table of contents is followed by lists of floats, e. g., lists of tables and figures (see [section 3.6.6](#)).

`\tableofcontents`
`\contentsname`

The production of the table of contents is done by the `\tableofcontents` command. To get a correct table of contents, at least two \LaTeX runs are necessary after every change. The option `liststotoc` causes the lists of figures and tables to be included in the table of contents. `idxotoc` is the corresponding option for the index. This is rather uncommon in classical typography. One does find the bibliography included in the table of contents slightly more frequently. This can be obtained with the options `bibtotoc` and `bibtotocnumbered`. These options are explained in [section 3.1.5, page 47](#).

The table of contents is set as an unnumbered chapter and is therefore subject to the side effects of the standard `\chapter*` command, which are described in [section 3.6.2, page 77](#). However, the running headings for left and right pages are correctly filled with the heading of the table of contents.

The text of the heading is given by the macro `\contentsname`. If you make use of a language package such as `babel`, please read the documentation of that package before redefining this macro.

There are two variants for the construction of the table of contents. With the standard variant, the titles of the sectional units are indented so that the unit number is flush left to the edge of the text of the next upper sectional unit. However, the space for the numbers is thereby limited and is only sufficient for a little more than 1.5 places per unit level. Should this become a problem, the option `tocleft` can be used to set the behaviour such that all entries in the table of contents are set flush left under one another. As explained in [section 3.1.5, page 48](#), several \LaTeX runs are needed.

The entry for the highest sectional unit below `\part`, i. e., `\chapter` with `scrbook` and `scrreprt` or `\section` with `scrartcl` is not indented. The font style is however affected by the settings of the element disposition (see [table 3.3, page 54](#)). There are no dots between the text of the sectional unit heading and the page number. The typographic reasons for this are that the font is usually different, and the desire for appropriate emphasis. The table of contents of this manual is a good example of these considerations.

`tocdepth`

Normally, the units included in the table of contents are all the units from `\part` to `\subsection` (for the classes `scrbook` and `scrreprt`) or from `\part` to `\subsubsection` (for the class `scrartcl`). The inclusion of a sectional unit in the table of contents is controlled by the counter `tocdepth`. This has the

value `-1` for `\part`, `0` for `\chapter`, and so on. By setting, incrementing or decrementing the counter, one can choose the lowest sectional unit level to be included in the table of contents. The same happens with the standard classes.

The user of the `scrpage2` package (see [chapter 4](#)) does not need to remember the numerical values of each sectional unit. They are given by the values of the macros `\chapterlevel`, `\sectionlevel` and so on down to `\subparagraphlevel`.

Example: Assume that you are preparing an article that uses the sectional unit `\subsubsection`. However, you don't want this sectional unit to appear in the table of contents. The preamble of your document might contain the following:

```
\documentclass{scrartcl}
\setcounter{tocdepth}{2}
```

You set the counter `tocdepth` to `2` because you know that this is the value for `\subsection`. If you know that `scrartcl` normally includes all levels down to `\subsubsection` in the table of contents, you can simply decrement the counter `tocdepth` by one:

```
\documentclass{scrartcl}
\addtocounter{tocdepth}{-1}
```

How much you should add to or subtract from the `tocdepth` counter can also be found by looking at the table of contents after the first \LaTeX run.

A small hint in order that you do not need to remember which sectional unit has which number: in the table of contents count the number of units required extra or less and then, as in the above example, use `\addtocounter` to add or subtract that number to or from `tocdepth`.

KOMA-Script has always attempted to avoid page breaking directly between a sectional unit and the adjacent next lower unit, for example, between a chapter title and its first section title. However, the mechanism worked poorly or not at all until version 2.96. In version 2.96a the mechanism was much improved and should now always work correctly. There can be changes in the page breaking in the table of contents as a result though. Thus, the new mechanism is only active, if the compatibility option is not set to version 2.96 or less (see option `version`, [section 3.1.1](#),

page 39). The mechanism also does not work if the commands to generate the table of contents are redefined, for example, by the use of the package `tocloft`.

3.5 Lists of Floats

As a rule, the lists of floats, e. g., list of tables and list of figures, can be found directly after the table of contents. In some documents, they can even be found in the appendix. However, the author of this manual prefers their location after the table of contents, therefore the explanation is given here.

```
\listoftables
\listoffigures
\listtablename
\listfigurename
```

These commands generate a list of tables or figures. Changes in the document that modify these lists will require two \LaTeX runs in order to take effect. The layout of the lists can be influenced by the options `listsindent` and `listsleft` (see [section 3.1.6, page 49](#)). Moreover, the options `liststotoc` and `liststotocnumbered` have indirect influence (see [section 3.1.5, page 47](#)).

The text of the titles of this tables are stored in the macros `\listtablename` and `\listfigurename`. If you use a language package like `babel` and want to redefine these macros, you should read the documentation of the language package.

3.6 Main Text

This section explains everything provided by KOMA-Script in order to write the main text. The main text is the part that the author should focus on first. Of course this includes tables, figures and comparable information as well.

scrbook 3.6.1 Separation

Before getting to the main text we will have a short look at three commands which exist both in the standard class `book` and the KOMA-Script class `scrbook`. They are used for separation of the *front matter*, the *main matter* and the *back matter* of a book.


```
\frontmatter
\mainmatter
\backmatter
```

The macro `\frontmatter` introduces the front matter in which roman numerals are used for the page numbers. Chapter headings in a front matter are not numbered. The section titles would be numbered, start at chapter 0, and would be consecutively numbered across chapter boundaries. However, this is of no import, as the front matter is used only for the title pages, table of contents, lists of figures and tables, and a foreword. The foreword can thus be set as a normal chapter. A foreword should never be divided into sections but kept as short as possible. Therefore in the foreword there is no need for a deeper structuring than the chapter level.

`\mainmatter` introduces the main matter with the main text. If there is no front matter then this command can be omitted. The default page numbering in the main matter uses Arabic numerals (re)starting in the main matter at 1.

The back matter is introduced with `\backmatter`. Opinions differ in what should be part of the back matter. So in some cases you will find only the bibliography, in some cases only the index, and in other cases both of these as well as the appendices. The chapters in the back matter are similar to the chapters in the front matter, but page numbering is not reset. If you do require separate page numbering you may use the command `\pagenumbering` from [section 3.2.2, page 63](#).

3.6.2 Structuring the Document

There are several commands to structure a document into parts, chapters, sections and so on.

```
\part[Short version]{Heading}
\chapter[Short version]{Heading}
\section[Short version]{Heading}
\subsection[Short version]{Heading}
\subsubsection[Short version]{Heading}
\paragraph[Short version]{Heading}
\subparagraph[Short version]{Heading}
```

The standard sectioning commands in KOMA-Script work in a similar fashion to those of the standard classes. Thus, an alternative entry for the table of contents and running headings can be specified as an optional argument to the sectioning commands.

The title of the level part (`\part`) is distinguished from other sectioning levels by being numbered independently from the other parts. This

means that the chapter level (in `scrbook` or `scrreprt`), or the section level (in `scartcl`) is numbered consecutively over all parts. Furthermore, for classes `scrbook` and `scrreprt` the title of the part level together with the corresponding preamble (see `\setpartpreamble`, page 84) is set on a separate page.

`scartcl` `\chapter` only exists in book or report classes, that is, in classes `book`, `scrbook`, `report` and `scrreprt`, but not in the article classes `article` and `scartcl`. In addition to this, the command `\chapter` in KOMA-Script differs substantially from the version in the standard class. In the standard classes the chapter number is used together with the prefix “Chapter”, or the corresponding word in the appropriate language, on a separate line above the actual chapter title text. This overpowering style is replaced in KOMA-Script by a simple chapter number before the chapter heading text, can however be reverted by the option `chapterprefix` (see section 3.1.3, page 44).

Please note that `\part` and `\chapter` in classes `scrbook` and `scrreprt` change the page style for one page. The applied page style in KOMA-Script is defined in the macros `\partpagestyle` and `\chapterpagestyle` (see section 3.2.2, page 59).

v2.8p

The font of all headings can be changed with the commands `\setkomafont` and `\addtokomafont` described in section 3.2.1. In doing this, generally the element `disposition` is used, followed by a specific element for every section level (see table 3.3, page 54). The font for the element `disposition` is predefined as `\normalfont\normalcolor\sffamily\bfseries`. The default font size for the specific elements depends on the options `bigheadings`, `normalheadings` and `smallheadings` (see section 3.1.4, page 46). The defaults are listed in table 3.8

Example: Suppose you are using the class option `bigheadings` and notice that the very big headings of document parts are too bold. You could change this as follows:

```
\setkomafont{disposition}{\normalcolor\sffamily←
}
\part{Appendices}
\addtokomafont{disposition}{\bfseries}
```

Using the command above you only switch off the font attribute **bold** for a heading “Appendices”. A much more comfortable and elegant solution is to change all `\part` headings at once.

Table 3.8: Default font sizes for different levels of document structuring in `scrbook` and `scrreprt`

class option	element	default
bigheadings	part	<code>\Huge</code>
	partnumber	<code>\huge</code>
	chapter	<code>\huge</code>
	section	<code>\Large</code>
	subsection	<code>\large</code>
	subsubsection	<code>\normalsize</code>
	paragraph	<code>\normalsize</code>
	subparagraph	<code>\normalsize</code>
normalheadings	part	<code>\huge</code>
	partnumber	<code>\huge</code>
	chapter	<code>\LARGE</code>
	section	<code>\Large</code>
	subsection	<code>\large</code>
	subsubsection	<code>\normalsize</code>
	paragraph	<code>\normalsize</code>
	subparagraph	<code>\normalsize</code>
smallheadings	part	<code>\LARGE</code>
	partnumber	<code>\LARGE</code>
	chapter	<code>\Large</code>
	section	<code>\large</code>
	subsection	<code>\normalsize</code>
	subsubsection	<code>\normalsize</code>
	paragraph	<code>\normalsize</code>
	subparagraph	<code>\normalsize</code>

This is done either by:

```
\addtokomafont{part}{\normalfont\sffamily}
\addtokomafont{partnumber}{\normalfont\sffamily}
}
```

or simply using:

```
\addtokomafont{part}{\mdseries}
\addtokomafont{partnumber}{\mdseries}
```

The last version is to be preferred because it gives you the correct result even when you make changes to the `disposition` element, for instance:

```
\setkomafont{disposition}{\normalcolor\bfseries←
}
```

With this change it is possible to set all section levels at once to no longer use sans serif fonts.

Please be warned of misusing the possibilities of font switching to mix fonts, font sizes and font attributes excessively. Picking the most suitable font for a given task is a hard task even for professionals and has almost nothing to do with the personal tastes of non-experts. Please refer to the citation at the end of [section 2.8, page 36](#) and to the following explanation.

It is possible to use different font types for different section levels in KOMA-Script. Non-experts in typography should for very good typographical reasons refrain absolutely from using these possibilities.

There is a rule in typography which states that one should mix as few fonts as possible. Using sans serif for headings already seems to be a breach of this rule. However, one should know that bold, large serif letters are much too heavy for headings. Strictly speaking, one would then have to at least use a normal instead of a bold or semi bold font. However, in deeper levels of the structuring a normal font may then appear too lightly weighted. On the other hand, sans serif fonts in headings have a very pleasant appearance and in fact find acceptance almost solely for headings. That is why sans serif is the carefully chosen default in KOMA-Script.

More variety should however be avoided. Font mixing is only for professionals. In case you want to use other fonts than the standard T_EX fonts—regardless of whether these are CM, EC or LM fonts—you should consult an expert, or for safety's sake redefine the font for the element `disposition` as seen in the example above. The author of this documentation considers the commonly encountered combinations Times and Helvetica or Palatino with Helvetica as unfavourable.

```
\part*{Heading}
\chapter*{Heading}
\section*{Heading}
\subsection*{Heading}
\subsubsection*{Heading}
\paragraph*{Heading}
\subparagraph*{Heading}
```

All disposition commands have starred versions, which are unnumbered, and produce section headings which do not show up in the table of contents or in the running heading. The absence of a running heading often has an unwanted side effect. For example, if a chapter which is set using `\chapter*` spans several pages, then the running heading of the previous chapter suddenly reappears. KOMA-Script offers a solution for this which `scrbook`, is described below. `\chapter*` only exists in book and report classes, that `scrreprt` is, `book`, `scrbook`, `report` and `scrreport`, but not the article classes `article` and `scrtartcl`.

Please note that `\part` and `\chapter` change the page style for one page. The applied style is defined in the macros `\partpagestyle` and `\chapterpagestyle` in KOMA-Script (see [section 3.2.2, page 59](#)).

v2.8p

As for the possibilities of font switching, the same explanations apply as were given above for the unstarred variants. The structuring elements are named the same since they do not indicate variants but structuring levels.

```
\addpart[Short version]{Heading}
\addpart*{Heading}
\addchap[Short version]{Heading}
\addchap*{Heading}
\addsec[Short version]{Heading}
\addsec*{Heading}
```

In addition to the commands of the standard classes KOMA-Script offers the new commands `\addsec` and `\addchap`. They are similar to the standard commands `\chapter` and `\section` except that they are unnumbered. They thus produce both a running heading and an entry in the table of contents. The starred variants `\addchap*` and `\addsec*` are similar to the standard commands `\chapter*` and `\section*` except for a tiny but important difference: The running headings are deleted. This eliminates the side effect of obsolete headers mentioned above. Instead, the running headings on following pages remain empty. `\addchap` and `\addchap*` of course only exist in book and report classes, namely `book`, `scrbook`, `report` and `scrreport`, but not in the article classes `article` and `scrtartcl`.

Similarly, the command `\addpart` produces an unnumbered document

part with an entry in the table of contents. Since the running headings are already deleted by `\part` and `\part*` the problem of obsolete headers does not exist. The starred version `\addpart*` is thus identical to `\part*` and is only defined for consistency reasons.

Please note that `\addpart` and `\addchap` and their starred versions change the page style for one page. The particular page style is defined in the macros `\partpagestyle` and `\chapterpagestyle` (see [section 3.2.2, page 59](#)).

v2.8p

As for the possibilities of font switching, the same explanations apply as given above for the normal structuring commands. The elements are named the same since they describe not variants but structuring levels.

`\minisec{Heading}`

Sometimes a heading is wanted which is highlighted but also closely linked to the following text. Such a heading should not be separated by a large vertical skip.

The command `\minisec` is designed for this situation. This heading is not associated with any structuring level. Such a *mini section* does not produce an entry in the table of contents nor does it receive any numbering.

Example: You have developed a kit for building a mouse trap and want the documentation separated into a list of necessary items and an assembly description. You could write the following:

```
\minisec{Items needed}

\begin{flushleft}
  1 plank ($100\times 50 \times 12$)\
  1 spring-plug of a beer-bottle\
  1 spring of a ball-point pen\
  1 drawing pin\
  2 screws\
  1 hammer\
  1 knife
\end{flushleft}
```

```
\minisec{Assembly}
```

At first one searches the mouse-hole and puts ←
 the drawing pin
 directly behind the hole. Thus the mouse cannot←
 escape during the

following actions.

Then one knocks the spring-plug with the hammer↵
into the mouse-hole.

If the spring-plug's size is not big enough in ↵
order to shut the

mouse-hole entirely, then one can utilize the ↵
plank instead and

fasten it against the front of the mouse-hole ↵
utilizing the two

screws and the knife. Instead of the knife one ↵
can use a

screw-driver instead.

Which gives:

Items needed

1 plank (100 × 50 × 12)
1 spring-plug of a beer-bottle
1 spring of a ball-point pen
1 drawing pin
2 screws
1 hammer
1 knife

Assembly

At first one searches the mouse-hole and puts the drawing pin directly behind the hole. Thus the mouse cannot escape during the following actions.

Then one knocks the spring-plug with the hammer into the mouse-hole. If the spring-plug's size is not big enough in order to shut the mouse-hole entirely, then one can utilize the plank instead and fasten it against the front of the mouse-hole utilizing the two screws and the knife. Instead of the knife one can use a screw-driver instead.

2.96a

The font type of the structuring command `\minisec` be changed using the element `disposition` (see [table 3.3, page 54](#)) and `minisec`. Default setting of element `minisec` is empty, so the default of the element `disposition` is active.

`\raggedsection`

In the standard classes headings are set as justified text. That means that hyphenated words can occur and headings with more than one line are

stretched up to the text border. This is a rather uncommon approach in typography. KOMA-Script therefore formats the headings left aligned with hanging indentation using `\raggedsection` with the definition:

```
\newcommand*{\raggedsection}{\raggedright}
```

This command can be redefined with `\renewcommand`.

Example: You prefer justified headings, so you write in the preamble of your document:

```
\renewcommand*{\raggedsection}{}

```

or more compactly:

```
\let\raggedsection\relax

```

You will get a formatting of the headings which is very close to that of the standard classes. It will become even closer when you combine this change with the change of the element disposition mentioned above.

```
\partformat
\chapterformat
\othersectionlevelsformat{section name}
\autodot

```

As you might know, for every counter in \LaTeX there is a command `\thecountername`, which outputs the value of the counter. Depending on the class the counter for a particular level starting from `\section` (book, scrbook, report, scrreprt) or `\subsection` (article, scartcl) is composed of the counter for the next higher level followed by a dot and the Arabic number of the *countername* of the respective level.

KOMA-Script has added a further logical level to the output of the section number. The counters for the respective heading are not merely output. They are formatted using the commands `\partformat`, `\chapterformat` and `\othersectionlevelsformat`. Of course the command `\chapterformat` like `\thechapter` does not exist in the class `scartcl` but only in the classes `scrbook` and `scrreprt`.

As described in [section 3.1.7](#), [page 50](#), dots in section numbers should be handled for the German-speaking region according to the rules given in [\[DUD96\]](#). The command `\autodot` in KOMA-Script ensures that these rules are being followed. In all levels except for `\part` a dot is followed by a further `\enskip`. This corresponds to a horizontal skip of 0.5em.

The command `\othersectionlevelsformat` takes as a parameter the name of the section level, such as “*section*”, “*subsection*” ... Per default therefore, only the levels `\part` and `\chapter` have formatting commands of their own, while all other section levels are covered by one general formatting command. This has historical reasons. At the time that Werner Lemberg suggested a suitable extension of KOMA-Script for his CJK package, only this differentiation was needed.

The formatting commands can be redefined using `\renewcommand` to fit them to your personal needs. The following original definitions are used by the KOMA-Script classes:

```
\newcommand*{\partformat}{\partname~\thepart\autodot}
\newcommand*{\chapterformat}{%
  \chapappifchapterprefix{\ }~\thechapter\autodot\enskip}
\newcommand*{\othersectionlevelsformat}[1]{%
  \csname the#1\endcsname\autodot\enskip}
```

Example: Assume that when using `\part` you do not want the word “Part” written in front of the part number. You could use the following command in the preamble of your document:

```
\renewcommand*{\partformat}{\thepart\autodot}
```

Strictly speaking, you could do without `\autodot` at this point and insert a fixed dot instead. As `\part` is numbered with roman numerals, according to [DUD96] a dot has to be applied. However, you thereby give up the possibility to use one of the options `pointednumbers` and `pointlessnumbers` and optionally depart from the rules. More details concerning class options can be found in [section 3.1.7, page 50](#).

An additional possibility could be to place the section numbers in the left margin in such a way that the heading text is left aligned with the surrounding text. This can be accomplished with:

```
\renewcommand*{\othersectionlevelsformat}[1]{%
  \llap{\csname the#1\endcsname\autodot\enskip}
}
```

The little known T_EX command `\llap` in the definition above puts its argument left of the current position without changing

the position thereby. A much better L^AT_EX solution would be:

```
\renewcommand*{\othersectionlevelsformat}[1]{%
  \makebox[0pt][r]{%
    \csname the#1\endcsname\autodot\enskip}}
```

See [Tea01] for more information about the optional arguments of `\makebox`.

```
\chapappifchapterprefix{additional text}
\chapapp
```

These two commands are not only used internally by KOMA-Script but `scrbook`, `scrreprt` are also provided to the user. Later it will be shown how they can be used for example to redefine other commands. Using the layout option `chapterprefix` (see section 3.1.3, page 44) `\chapappifchapterprefix` outputs the word “Chapter” in the main part of the document in the current language, followed by *additional text*. In the appendix, the word “Appendix” in the current language is output instead, followed by *additional text*. If the option `nochapterprefix` is set, then nothing is output.

The command `\chapapp` always outputs the word “Chapter” or “Appendix”. In this case the options `chapterprefix` and `nochapterprefix` have no effect.

Since chapters only exist in the classes `scrbook` and `scrreprt` these commands only exist in these classes.

```
\chaptermark{Running heading}
\sectionmark{Running heading}
\subsectionmark{Running heading}
\chaptermarkformat
\sectionmarkformat
\subsectionmarkformat
```

As mentioned in section 3.2.2 the page style headings works with automatic running headings. For this, the commands `\chaptermark` and `\sectionmark`, or `\sectionmark` and `\subsectionmark`, respectively, are defined. Every structuring command (`\chapter`, `\section` ...) automatically carries out the respective `\...mark` command. The parameter passed contains the text of the section heading. The respective section number is added automatically in the `\...mark` command. The formatting is done according to the section level with one of the three commands `\chaptermarkformat`, `\sectionmarkformat` or `\subsectionmarkformat`.

Of course there is no command `\chaptermark` or `\chaptermarkformat` in `scrbook`, `scrreprt` or `scrartcl`. Accordingly, `\subsectionmark` and `\subsectionmarkformat` exist only in `scrartcl`. This changes when you use the `scrpage2` package (see [chapter 4](#)).

Similar to `\chapterformat` and `\othersectionlevelsformat`, the commands `\chaptermarkformat` (not in `scrartcl`), `\sectionmarkformat` and `\subsectionmarkformat` (only in `scrartcl`) define the formatting of the sectioning numbers in the automatic running headings. They can be adapted to your personal needs with `\renewcommand`. The original definitions for the KOMA-Script classes are:

```
\newcommand*{\chaptermarkformat}{%
  \chapappifchapterprefix{\ } \thechapter \autodot \enskip}
\newcommand*{\sectionmarkformat}{\thesection \autodot \enskip}
\newcommand*{\subsectionmarkformat}{%
  \thesubsection \autodot \enskip}
```

Example: Suppose you want to prepend the word “Chapter” to the chapter number in the running heading. For example you could insert the following definition in the preamble of your document :

```
\renewcommand*{\chaptermarkformat}{%
  \chapapp~ \thechapter \autodot \enskip}
```

As you can see, both the commands `\chapappifchapterprefix` and `\chapapp` explained above are used here.

secnumdepth

Per default, in the classes `scrbook` and `scrreprt` the section levels from `\part` down to `\subsection` and in the class `scrartcl` the levels from `\part` down to `\subsubsection` are numbered. This is controlled by the L^AT_EX counter `secnumdepth`. The value `-1` represents `\part`, `0` the level `\chapter`, and so on. By defining, incrementing or decrementing this counter you can determine down to which level the headings are numbered. The same applies in the standard classes. Please refer also to the explanation concerning the counter `tocdepth` in [section 3.4](#), [page 70](#).

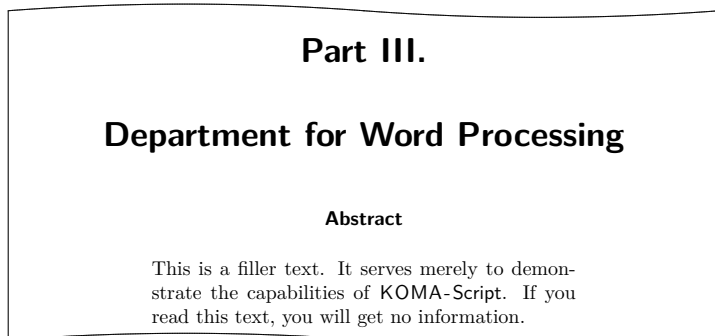
```
\setpartpreamble[position][width]{preamble}
\setchapterpreamble[position][width]{preamble}
```

Parts and chapters in KOMA-Script can be started with a *preamble*. This is particularly useful when you are using a two column layout with the class option `twocolumn`, since the heading together with the *preamble* is always set in a one column layout. The *preamble* can comprise more than one paragraph. The command to output the *preamble* has to be placed before the respective `\part`, `\addpart`, `\chapter` or `\addchap` command.

Example: You are writing a report about the condition of a company. You organize the report in such a way that every department gets its own partial report. Every one of these parts should be introduced by an abstract on the corresponding title page. You could write the following:

```
\setpartpreamble{%
  \begin{abstract}
    This is a filler text. It serves merely to
    demonstrate the
    capabilities of {\KOMAScript}. If you read
    this text, you will
    get no information.
  \end{abstract}
}
\part{Department for Word Processing}
```

Depending on the settings for the heading font size (see [section 3.1.4, page 46](#)) and the options for the `abstract` environment (see [section 3.1.7, page 50](#)), the result would look similar to:



Please note that it is *you* who is responsible for the spaces between the

heading, preamble and the following text. Please note also that there is no **abstract** environment in the class `scrbook` (see [section 3.3, page 69](#)).

v2.8p

The first optional argument *position* determines the position at which the preamble is placed with the help of one or two letters. For the vertical placement there are two possibilities at present:

o: above the heading

u: below the heading

You can insert one preamble above and another below a heading. For the horizontal placement you have the choice between three alignments:

l: left-aligned

r: right-aligned

c: centered

However, this does not output the text of the *preamble* in such a manner, but inserts a box whose width is determined by the second optional argument *width*. If you leave out this second argument the whole text width is used. In that case the option for horizontal positioning will have no effect. You can combine exactly one letter from the vertical with one letter from the horizontal positioning.

```
\dictum[author]{dictum}
\dictumwidth
\dictumauthorformat{author}
\raggeddictum
\raggeddictumtext
\raggeddictumauthor
```

`scrbook`, Apart from an introductory paragraph you can use `\setpartpreamble` or `\setchapterpreamble` for a kind of *aphorism* (also known as “dictum”) at the beginning of a chapter or section. The command `\dictum` inserts such an aphorism. This macro can be used as obligatory argument of either the command `\setchapterpreamble` or `\setpartpreamble`. However, this is not obligatory.

v2.8q

The dictum together with an optional *author* is inserted in a `\parbox` (see [\[Tea01\]](#)) of width `\dictumwidth`. Yet `\dictumwidth` is not a length which can be set with `\setlength`. It is a macro that can be redefined using `\renewcommand`. Default setting is `0.3333\textwidth`, which is a third of the `textwidth`. The box itself is positioned with the command

Table 3.9: Default settings for the elements of a dictum

Element	Default
<code>dictumtext</code>	<code>\normalfont\normalcolor\sffamily\small</code>
<code>dictumauthor</code>	<code>\itshape</code>

`\raggeddictum`. Default here is `\raggedleft`, that is, right justified. The command `\raggeddictum` can be redefined using `\renewcommand`.

Within the box the *dictum* is set using `\raggeddictumtext`. Default setting is `\raggedright`, that is, left justified. Similarly to `\raggeddictum` this can be redefined with `\renewcommand`. The output uses the default font setting for the element `dictumtext`, which can be changed with the commands from [section 3.2.1](#). Default settings are listed in [table 3.9](#).

If there is an *author* name, it is separated from the *dictum* by a line the full width of the `\parbox`. This is defined by the macro `\raggeddictumauthor`. Default is `\raggedleft`. This command can also be redefined using `\renewcommand`. The format of the output is defined with `\dictumauthorformat`. This macro expects the `\author` as argument. As default `\dictumauthorformat` is defined as:

```
\newcommand*{\dictumauthorformat}[1]{(#1)}
```

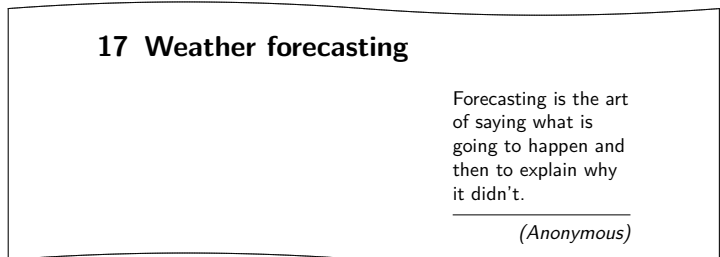
Thus the *author* is set enclosed in rounded parenthesis. For the element `dictumauthor` a different font than for the element `dictumtext` can be defined. Default settings are listed in [table 3.9](#). Changes can be made using the commands from [section 3.2.1](#). If `\dictum` is used within the macro `\setchapterpreamble` or `\setpartpreamble` you have to take care of the following: the horizontal positioning is always done with `\raggeddictum`. Therefore, the optional argument for horizontal positioning which is implemented for these two commands has no effect. `\textwidth` is not the width of the whole text corpus but the actually used text width. If `\dictumwidth` is set to `.5\textwidth` and `\setchapterpreamble` has an optional width of `.5\textwidth` too, you will get a box with a width one quarter of the text width. Therefore, if you use `\dictum` it is recommended to refrain from setting the optional width for `\setchapterpreamble` or `\setpartpreamble`.

If you have more than one dictum one under another, you should separate them by an additional vertical space, easily accomplished using the command `\bigskip`.

Example: You are writing a chapter on an aspect of weather forecasting. You have come across an aphorism which you would like to place at the beginning of the chapter beneath the heading. You could write:

```
\setchapterpreamble[u]{%
  \dictum[Anonymous]{Forecasting is the art of ↵
    saying
      what is going to happen and then to ↵
        explain
          why it didn't.}}
\chapter{Weather forecasting}
```

The output would look as follows:



If you would rather the dictum span only a quarter of the text width rather than one third you can redefine `\dictumwidth`:

```
\renewcommand*{\dictumwidth}{.25\textwidth}
```

For a somewhat more sophisticated formatting of left- or right-aligned paragraphs including hyphenation you can use the package `ragged2e` [Sch03].

3.6.3 Footnotes

Footnotes are not limited to the main part of the document. However, since footnotes are mainly used in the main text they are covered in this section.

```
\footnote[number]{text}
\footnotemark[number]
\footnotetext[number]{text}
```

Similarly to the standard classes, footnotes in KOMA-Script are produced with the `\footnote` command, or alternatively the pairwise usage

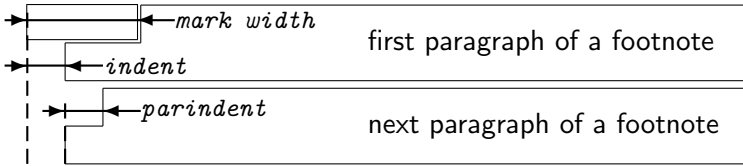


Figure 3.1: Parameters that control the footnote layout

of the commands `\footnotemark` and `\footnotetext`. As in the standard classes it is possible that a page break occurs within a footnote. Normally this happens if the footnote mark is placed so near the bottom of a page as to leave \LaTeX no choice but to break the footnote onto the next page.

```
\deffootnote[mark width]{indent}{parindent}{definition}
\deffootnotemark{definition}
\thefootnotemark
\textsuperscript{text}
```

Footnotes are formatted slightly differently in KOMA-Script to in the standard classes. As in the standard classes the footnote mark in the text is depicted using a small superscripted number. The same formatting is used in the footnote itself. The mark in the footnote is type-set right-aligned in a box with width *mark width*. The first line of the footnote follows directly.

All following lines will be indented by the length of *indent*. If the optional parameter *mark width* is not specified, it defaults to *indent*. If the footnote consists of more than one paragraph, then the first line of a paragraph is indented, in addition to *indent*, by the value of *parindent*.

Figure 3.1 illustrates the layout parameters once more. The default configuration of KOMA-Script is:

```
\deffootnote[1em]{1.5em}{1em}
{\textsuperscript{\thefootnotemark}}
```

`\textsuperscript` controls both the superscript and the smaller font size. `\thefootnotemark` is the current footnote mark without any formatting.

v2.8q

The font element `footnote` determines the font of the footnote including the footnote mark. Using the element `footnotelabel` the font of the footnote mark can be changed separately with the commands mentioned in [section 3.2.1](#) Please refer also to [table 3.3](#), [page 54](#). Default setting is no change in the font.

The footnote mark in the text is defined separately from the mark in

front of the actual footnote. This is done with `\deffootnotemark`. Default setting is:

```
\deffootnotemark{%
  \textsuperscript{\thefootnotemark}}
```

v2.8q

In the above the font for the element `footnotereference` is applied (see [table 3.3, page 54](#)). Thus the footnote marks in the text and in the footnote itself are identical. The font can be changed with the commands described in [section 3.2.1](#).

Example: A feature often asked for is footnote marks which are neither in superscript nor in a smaller font size. They should not touch the footnote text but be separated by a small space. This can be accomplished as follows:

```
\deffootnote{1em}{1em}{\thefootnotemark\ }
```

The footnote mark and the following space are therefore set right-aligned into a box of width 1em. The following lines of the footnote text are also indented by 1em from the left margin.

Another often requested footnote layout is left-aligned footnote marks. These can be obtained with:

```
\deffootnote{1.5em}{1em}{%
  \makebox[1.5em][l]{\thefootnotemark}}
```

If you want however only to change the font for all footnotes, for example to sans serif, you can solve this problem simply by using the commands from [section 3.1.4](#):

```
\setkomafont{footnote}{\sffamily}
```

As demonstrated with the examples above, the simple user interface of KOMA-Script provides a great variety of different footnote formattings.

3.6.4 Lists

Both L^AT_EX and the standard classes offer different environments for lists. Though slightly changed or extended all these list are of course offered in KOMA-Script as well. In general all lists — even of different kind — can be nested up to four levels. From a typographical view, anything more would

make no sense, as more than three levels can no longer be easily perceived. The recommended procedure in such a case is to split the large list into several smaller ones.

```

itemize
\item
\labelitemi
\labelitemii
\labelitemiii
\labelitemiv

```

The simplest form of a list is an `itemize` list. The users of a certain disliked word processing package often refer to this form of a list as *bulletpoints*. Presumably, these users are unable to envisage that, depending on the level, a different symbol from a large dot could be used to introduce each point. Depending on the level, KOMA-Script uses the following marks: “•”, “—”, “*” and “.”. The definition of these symbols is specified in the macros `\labelitemi`, `\labelitemii`, `\labelitemiii` and `\labelitemiv`, all of which can be redefined using `\renewcommand`. Every item is introduced with `\item`.

Example: You have a simple list which is nested in several levels. You write for example:

```

\minisec{Vehicles}
\begin{itemize}
\item aeroplanes
\begin{itemize}
\item biplane
\item jets
\item transport planes
\begin{itemize}
\item single-engined
\begin{itemize}
\item jet-driven
\item propeller-driven
\end{itemize}
\end{itemize}
\item multi-engined
\end{itemize}
\item helicopters
\end{itemize}
%% wahrscheinlich Platzfiller in Deutscher Ausgabe←
?
```

```

% \item motorcycles
% \begin{itemize}
% \item historically accurate
% \item futuristic, not real
% \end{itemize}
  \item automobiles
  \begin{itemize}
    \item racing cars
    \item private cars
    \item lorries
  \end{itemize}
  \item bicycles
\end{itemize}

```

As output you get:

Vehicles

- aeroplanes
 - biplanes
 - jets
 - transport planes
 - * single-engined
 - jet-driven
 - propeller-driven
 - * multi-engined
 - helicopters
- automobiles
 - racing cars
 - private cars
 - lorries
- bicycles

```

enumerate
\item
\theenumi
\theenumii
\theenumiii
\theenumiv
\labelenumi
\labelenumii
\labelenumiii
\labelenumiv

```

Another form of a list often used is a numbered list which is already implemented by the \LaTeX kernel. Depending on the level, the numbering uses the following characters: Arabic numbers, small letters, small roman numerals and capital letters. The kind of numbering is defined with the macros `\theenumi` down to `\theenumiv`. The output format is determined by the macros `\labelenumi` to `\labelenumiv`. While the small letter of the second level is followed by a round parenthesis, the values of all other levels are followed by a dot. Every item is introduced with `\item`.

Example: Replacing every occurrence of an `itemize` environment with an `enumerate` environment in the example above we get the following result:

Vehicles

1. aeroplanes
 - a) biplanes
 - b) jets
 - c) transport planes
 - i. single-engined
 - A. jet-driven
 - B. propeller-driven
 - ii. multi-engined
 - d) helicopters
2. automobiles
 - a) racing cars
 - b) private cars
 - c) lorries
3. bicycles

Using `\label` within a list you can set labels which are referenced with `\ref`. In the example above, a label was set after the jet-driven, single-engined transport planes with `\label{xmp:jets}`. The `\ref` value is then **1(c)iA**.

description
`\item[item]`

A further list form is the description list. Its main use is the description of several items. The item itself is an optional parameter in `\item`. The font which is responsible for emphasizing the item can be changed with the commands for the element `descriptionlabel` (see [table 3.3, page 54](#)) described in [section 3.2.1](#). Default setting is `\sffamily\bfseries`.

v2.8p

Example: Instead of items in sans serif and bold you want them printed in the standard font in bold. Using

```
\setkomafont{descriptionlabel}{\normalfont\bfseries}
```

you redefine the font accordingly.

An example for a description list is the output of the page styles listed in [section 3.2.2](#). The heavily abbreviated source code is:

```
\begin{description}
\item[empty] is the page style without any header or footer.
\item[plain] is the page style without headings.
\item[headings] is the page style with running headings.
\item[myheadings] is the page style for manual headings.
\end{description}
```

This abbreviated version gives:

empty is the page style without any header or footer.

plain is the page style without headings.

headings is the page style with running headings.

myheadings is the page style for manual headings.

`labeling[delimiter]{widest pattern}`
`\item[keyword]`

An additional form of a description list is only available in the KOMA-Script classes: the `labeling` environment. Unlike the `description` en-

vironment, you can provide a pattern whose length determines the indentation of all items. Furthermore, you can put an optional *delimiter* between the item and its description.

Example: Slightly changing the example from the `description` environment, we could write:

```
\begin{labeling}[~--]{%
  \usekomafont{descriptionlabel}myheadings}
\item[\usekomafont{descriptionlabel}empty]
  Page style without header and footer
\item[\usekomafont{descriptionlabel}plain]
  Page style for chapter beginnings without ↵
  headings
\item[\usekomafont{descriptionlabel}headings]
  Page style for running headings
\item[\usekomafont{descriptionlabel}↵
  myheadings]
  Page style for manual headings
\end{labeling}
```

As result we get:

empty	– Page style without header and footer
plain	– Page style for chapter beginnings without headings
headings	– Page style for running headings
myheadings	– Page style for manual headings

As can be seen in this example, a font changing command has to be repeated both in the pattern and in the optional parameter of every `\item` command in this environment.

Originally this environment was implemented for things like “Precondition, Assertion, Proof”, or “Given, Required, Solution” that are often used in lecture hand-outs. By now this environment has found many different applications. For example, the environment for examples in this guide was defined with the `labeling` environment.

verse

Usually the `verse` environment is not perceived as a list environment because you do not work with `\item` commands. Instead, fixed line breaks

are used within the `flushleft` environment. Yet internally in both the standard classes as well as KOMA-Script it is indeed a list environment.

In general the `verse` environment is used for poems. Lines are indented both left and right. Individual lines of verse are ended by a fixed line break `\\`. Verses are set as paragraphs, separated by an empty line. Often also `\medskip` or `\bigskip` is used instead. To avoid a page break at the end of a line of verse you as usual insert `*` instead of `\\`.

Example: As an example, the first lines of “Little Red Riding Hood and the Wolf” by Roald Dahl:

```
\begin{verse}
  As soon as Wolf began to feel\\*
  that he would like a decent meal,\\*
  He went and knocked on Grandma's door.\\*
  When Grandma opened it, she saw\\*
  The sharp white teeth, the horrid grin,\\*
  And Wolfie said, 'May I come in?'
\end{verse}
```

The result is as follows:

As soon as Wolf began to feel
 That he would like a decent meal,
 He went and knocked on Grandma's door.
 When Grandma opened it, she saw
 The sharp white teeth, the horrid grin,
 And Wolfie said, 'May I come in?'

However, if you have very long lines of verse, for instance:

```
\begin{verse}
  Both the philosopher and the house-owner
  have always something to repair.\\
  \bigskip
  Don't trust a man, my son, who tells you
  that he has never lied.
\end{verse}
```

where a line break occurs within a line of verse:

Both the philosopher and the house-owner
have always something to repair.

Don't trust a man, my son, who tells you that
he has never lied.

there `*` can not prevent a page break occurring within a verse at such a line break. To prevent such a page break, a `\nopagebreak` would have to be inserted somewhere in the first line:

```
\begin{verse}
  Both the philosopher and the house-owner\↵
    nopagebreak
  have always something to repair.\
\bigskip
  Don't trust a man, my son, who tells you\↵
    nopagebreak
  that he has never lied.
\end{verse}
```

In the above example, `\bigskip` was used to separate the lines of verse.

quote
quotation

These two environments are also list environments and can be found both in the standard and the KOMA-Script classes. Both environments use justified text which is indented both on the left and right side. Usually they are used to separate long citations from the main text. The difference between these two lies in the manner in which paragraphs are typeset. While `quote` paragraphs are highlighted by vertical space, in `quotation` paragraphs the first line is indented. This is also true for the first line of a `quotation` environment. To prevent indentation you have to insert a `\noindent` command before the text.

Example: You want to highlight a short anecdote. You write the following `quotation` environment for this:

```
A small example for a short anecdote:
\begin{quotation}
  The old year was turning brown; the West Wind↵
    was
```


calling;

Tom caught the beechen leaf in the forest ←
falling.

“‘I’ve caught the happy day blown me by the ←
breezes!

Why wait till morrow-year? I’ll take it when ←
me pleases.

This I’ll mend my boat and journey as it ←
chances

west down the withy-stream, following my ←
fancies!’’

Little Bird sat on twig. “‘Whillo, Tom! I ←
heed you.

I’ve a guess, I’ve a guess where your fancies←
lead you.

Shall I go, shall I go, bring him word to ←
meet you?’’

\end{quotation}

The result is:

A small example for a short anecdote:

The old year was turning brown; the West
Wind was calling;

Tom caught the beechen leaf in the forest
falling. “I’ve caught the happy day blown me
by the breezes! Why wait till morrow-year?
I’ll take it when me pleases. This I’ll mend
my boat and journey as it chances west down
the withy-stream, following my fancies!”

Little Bird sat on twig. “Whillo, Tom! I
heed you. I’ve a guess, I’ve a guess where
your fancies lead you. Shall I go, shall I go,
bring him word to meet you?”

Using a quote environment instead you get:

A small example for a short anecdote:

The old year was turning brown; the West
Wind was calling;

Tom caught the beechen leaf in the forest
falling. "I've caught the happy day blown me
by the breezes! Why wait till morrow-year?
I'll take it when me pleases. This I'll mend
my boat and journey as it chances west down
the withy-stream, following my fancies!"

Little Bird sat on twig. "Whillo, Tom! I
heed you. I've a guess, I've a guess where
your fancies lead you. Shall I go, shall I go,
bring him word to meet you?"

```
addmargin[left indentation]{indentation}
addmargin*[inner indentation]{indentation}
```

Similar to `quote` and `quotation`, the `addmargin` environment changes the margin. In contrast to the first two environments, with `addmargin` the user can set the width of the indentation. Besides this, this environment does not change the indentation of the first line nor the vertical spacing between paragraphs.

If only the obligatory argument *indentation* is given, both the left and right margin are expanded by this value. If the optional argument *left indentation* is given as well, then at the left margin the value *left indentation* is used instead of *indentation*.

The starred `addmargin*` only differs from the normal version in a two-sided layout. Furthermore, the difference only occurs if the optional argument *inner indentation* is used. In this case this value *inner indentation* is added to the normal inner indentation. For right-hand pages this is the left margin, for left-hand pages the right margin. Then the value of *indentation* determines the width of the opposite margin.

Both versions of this environment take also negative values for all parameters. This has the effect of expanding the environment into the margin.

Example: Suppose you write a documentation which includes short source code examples. To highlight these you want them separated from the text by a horizontal line and shifted slightly into the outer margin. First you define the environment:

```
\newenvironment{SourceCodeFrame}{%
  \begin{addmargin*}[1em]{-1em}%
  \begin{minipage}{\linewidth}%
```

```

\rule{\linewidth}{2pt}%
}{%
\rule[.25\baselineskip]{\linewidth}{2pt}%
\end{minipage}%
\end{addmargin*}%
}

```

If you now put your source code in such an environment it will show up as:

You define yourself the following environment:

```

\newenvironment{\SourceCodeFrame}{%
\begin{addmargin*}[1em]{-1em}%
\begin{minipage}{\linewidth}%
\rule{\linewidth}{2pt}%
}{%
\rule[.25\baselineskip]{\linewidth}{2pt}←
}%
\end{minipage}%
\end{addmargin*}%
}

```

This may be feasible or not. In any case it shows the usage of this environment.

The optional argument of the `addmargin*` environment makes sure that the inner margin is extended by 1 em. In turn the outer margin is decreased by 1 em. The result is a shift by 1 em to the outside. Instead of 1em you can of course use a length, for example, `2\parindent`.

There is one problem with the `addmargin*` which you should be aware of. If a page break occurs within an `addmargin*` environment, the indentation on the following page will be on the wrong side. This means that suddenly the *inner indentation* is applied on the outside of the page. Therefore it is recommended to prevent page breaks within this environment. This can be achieved by using an additional `\parbox` or, as in the example above, a `minipage`. This makes use of the fact that neither the argument of a `\parbox` nor the content of a `minipage` breaks at the end of a page. Unfortunately this is not without another disadvantage: in some cases pages can no longer be filled correctly, which has the effect of generating several warnings.

Incidentally, whether a page is going to be on the left or right side of the book can not be determined for certain in the first L^AT_EX run. For

details please refer to the explanation of the commands `\ifthispageodd` and `\ifthispagewasodd` in [section 3.2.2, page 62](#).

One concluding remark on list environments: on the internet and during support it is often asked why such an environment is followed by a indented paragraph. In fact, this is not the case but is the result of the user demanding a new paragraph. In \LaTeX empty lines are interpreted as a new paragraph. This is also the case before and after list environments. Thus, if you want a list environment to be set within a paragraph you have to omit empty lines before and after. To nevertheless separate the environment from the rest of your text in the \LaTeX source file, you can insert a comment line before and after, that is, lines which begin with a percent character and contain nothing more.

3.6.5 Margin Notes

```
\marginpar[margin note left]{margin note}
\marginline{margin note}
```

Usually margin notes in \LaTeX are inserted with the command `\marginpar`. They are placed in the outer margin. In documents with one-sided layout the right border is used. Though `\marginpar` can take an optional different margin note argument in case the output is in the left margin, margin notes are always set in justified layout. However, experience has shown that many users prefer left- or right-aligned margin notes instead. To facilitate this, KOMA-Script offers the command `\marginline`.

Example: In the introduction, the class name `scrartcl` can be found in the margin. This can be produced² with:

```
\marginline{\texttt{scrartcl}}
```

Instead of `\marginline` you could have used `\marginpar`. In fact the first command is implemented internally as:

```
\marginpar[\raggedleft\texttt{scrartcl}]
{\raggedright\texttt{scrartcl}}
```

Thus `\marginline` is really only an abbreviated writing of the code above.

²In fact, instead of `\texttt`, a semantic highlighting was used. To avoid confusion this was replaced in the example.

Unfortunately `\marginpar` does not always work correctly in two-sided layout. Whether a margin note will end up in the left or right margin is already decided while evaluating the command `\marginpar`. If the output routine now shifts the margin note onto the next page the formatting is no longer correct. This behaviour is deeply rooted within \LaTeX and was therefore declared a feature by the $\text{\LaTeX}3$ team. `\marginline` suffers from this “feature” too. The package `mparhack` (see [SU03]) offers a standard solution for this problem which naturally benefits also `\marginpar` and `\marginline`.

Note that you may not use `\marginpar` or `\marginline` within float environments such as tables or figures. Also, these commands will not function in displayed math formulas.

3.6.6 Tables and Figures

With the floating environments \LaTeX offers a very capable and comfortable mechanism for automatic placement of figures and tables. But often these floating environments are slightly misunderstood by beginners. They often ask for a fixed position of a table or figure within the text. However, since these floating environments are being referenced in the text this is not necessary in most cases. It is also not sensible because such an object can only be set on the page if there is enough space left for it. If this is not the case the object would have to be shifted onto the next page, thereby possibly leaving a huge blank space on the page before.

Often one finds in a document for every floating object the same optional argument for positioning the object. This also makes no sense. In such cases one should rather change the standard parameter globally. For more details refer to [RNH02].

One last important note before starting this section: most mechanisms described here which extend the capabilities of the standard classes no longer work correctly when used together with packages which modify the typesetting of captions of figures and tables. This should be self evident, but it is often not understood.

```
\caption[entry]{title}
\captionbelow[entry]{title}
\captionabove[entry]{title}
```

In the standard classes caption text *title* of tables and figures is inserted with the `\caption` command below the table or figure. In general this is correct for figures. Opinions differ as to whether captions of tables are to be placed above or, consistent with captions of figures, below the table. That is the reason why KOMA-Script, unlike the standard classes, offers

`\captionbelow` for captions below and `\captionabove` for captions above tables or figures. Using `\caption` for figures always produces captions below the figure, whereas with tables the behaviour of `\caption` can be modified using the options `tablecaptionabove` and `tablecaptionbelow` (see [section 3.1.7, page 51](#)). For compatibility reasons the default behaviour of `\caption` used with tables is similar to `\captionbelow`.

Example: Instead of using captions below a table you want to place your captions above it, because you have tables which span more than one page. In the standard classes you could only write:

```
\begin{table}
  \caption{This is an example table}
  \begin{tabular}{llll}
    This & is & an & example.\\\hline
    This & is & an & example.\\
    This & is & an & example.
  \end{tabular}
\end{table}
```

Then you would get the unsatisfying result:

Table 30.2: This is an example table.				
This	is	an	example.	
This	is	an	example.	
This	is	an	example.	

Using KOMA-Script you write instead:

```
\begin{table}
  \captionabove{This is just an example table}
  \begin{tabular}{llll}
    This & is & an & example.\\\hline
    This & is & an & example.\\
    This & is & an & example.
  \end{tabular}
\end{table}
```

Then you get:

Table 30.2: This is just an example table				
This	is	an	example.	
This	is	an	example.	
This	is	an	example.	

Since you want all your tables typeset with captions above, you could of course use the option `tablecaptionabove` instead (see [section 3.1.7, page 51](#)). Then you can use `\caption` as you would in the standard classes. You will get the same result as with `\captionabove`.

Some would argue that you could achieve the same result using the `\topcaption` command from the `topcapt` package (see [\[Fai99\]](#)). However, that is not the case. The command `\topcaption` is ignored by packages which directly redefine the `\caption` macro. The `hyperref` package (see [\[Rah01\]](#)) is one such example. In KOMA-Script, `\captionabove` and `\captionbelow` are so implemented that changes have an effect on both of these commands as well.

If the `longtable` package is used, KOMA-Script ensures that captions above tables which are placed within a `longtable` environment have the same appearance as those in a normal `table` environment. This also means that you can apply the same settings as in a `table` environment. Please note that in the `longtable` package the maximum width of a table caption can be limited and the default is set to 4 in (see [\[Car98\]](#)). Used together with KOMA-Script this mechanism in `longtable` works only if the class option `origlongtable` is set (see [section 3.1.7, page 52](#)). If the `caption2` or `caption` package (see [\[Som04\]](#)) is loaded, table captions are handled by this package.

Please note that `\captionabove` and `\captionbelow`, if placed within a float environment which was defined using the `float` package, have the exact same behaviour described in [\[Lin01\]](#) for the `\caption` command. In this case, the float style determines whether the caption will be set below or above the figure or table.

```
captionbeside[entry]{title}[placement][width][offset]
captionbeside[entry]{title}[placement][width][offset]*
```

v2.8q

Apart from captions above and below the figure, one often finds captions, in particular with small figures, which are placed beside the figure. In general in this case both the baseline of the figure and of the caption are aligned at the bottom. With some fiddling and the use of two `\parbox` commands this could also be achieved in the standard classes. However, KOMA-Script offers a special environment for this which can be used within the floating environment. The first optional parameter *entry* and the obligatory parameter *title* mean the same as the corresponding parameters of `\caption`, `\captionabove` or `\captionbelow`. The caption text *title* is placed beside the content of the environment in this case.

Whether the caption text *title* is placed on the left or the right can be determined by the parameter *placement*. Exactly one of the following letters is allowed:

- l – left
- r – right
- i – inner margin in two-sided layout
- o – outer margin in two-sided layout

Default setting is to the right of the content of the environment. If either o or i are used you may need to run L^AT_EX twice to obtain the correct placement.

Per default the content of the environment and the caption text *title* fill the entire available text width. However, using the optional parameter *width*, it is possible to adjust the width used. This width could even be larger than the current text width.

When supplying a *width* the used width is usually centered with respect to the text width. Using the optional parameter *offset*, you can shift the environment relative to the left margin. A positive value corresponds to a shift to the right, whereas a negative value corresponds to a shift to the left. An *offset* of 0pt gives you a left-aligned output.

Adding a star to the optional parameter *offset* makes the the value mean a shift relative to the right margin on left hand pages in two-sided layout. A positive value corresponds to a shift towards the outer margin, whereas a negative value corresponds to a shift towards the inner margin. An *offset* of 0pt means alignment with the inner margin. As mentioned before, in some cases it takes two L^AT_EX runs for this to work correctly.

Example: An example for the usage of the `captionbeside` environment can be found in [figure 3.2](#). This figure was typeset with:

```
\begin{figure}
  \begin{captionbeside}[Example for a figure ↵
    description]%
    {A figure description which is neither above↵
      nor
      below, but beside the figure}[i][\linewidth↵
        ] [2em]*
  \fbox{%
```


Figure 3.2: A figure description which is neither above nor below, but beside the figure

```

\parbox[b][5\baselineskip][c]{.25\leftarrow
    textwidth}{%
\hspace*{\fill}\KOMAScript\hspace*{\fill\leftarrow
    }\par}}
\end{captionbeside}
\label{fig:maincls.captionbeside}
\end{figure}

```

The total width is thus the currently available width `\linewidth`. However, this width is shifted `2em` to the outside. The caption text or description is placed on the inner side beside the figure. The figure itself is shifted `2em` into the outer margin.

v2.8p

The font style for the description and the label — “Figure” or “Table”, followed by the number and the delimiter — can be changed with the commands described in [section 3.2.1](#). The respective elements for this are `caption` and `captionlabel` (see [table 3.3, page 54](#)). First the font style for the element `caption` is applied to the element `captionlabel` too. After this the font style of `captionlabel` is applied on the respective element. The default settings are listed in [table 3.10](#).

Example: You want the table and figure descriptions typeset in a smaller font size. Thus you could write the following in the preamble of your document:

```
\addtokomafont{caption}{\small}
```

Table 3.10: Font defaults for the elements of figure or table captions

element	default
<code>caption</code>	<code>\normalfont</code>
<code>captionlabel</code>	<code>\normalfont</code>

Furthermore, you would like the labels to be printed in sans serif and bold. You add:

```
\setkomafont{captionlabel}{\sffamily\bfseries}
```

As you can see, simple extensions of the default definitions are possible.

komaabove
komabelow

float If you use the float package the appearance of the float environments is solely defined by the *float* style. This includes whether captions above or below are used. In the float package there is no predefined style which gives you the same output and offers the same setting options (see below) as KOMA-Script. Therefore KOMA-Script defines the two additional styles komaabove and komabelow. When using the float package these styles can be activated just like the styles plain, boxed or ruled defined in float. For details refer to [Lin01]. The style komaabove inserts \caption, \captionabove and \captionbelow above, whereas komabelow inserts them below the float content.

\captionformat

In KOMA-Script there are different ways to change the formatting of the caption text. The definition of different font styles was already explained above. This or the caption delimiter between the label and the label text itself is specified in the macro \captionformat. In contrast to all other \dots.format commands, in this case it does not contain the counter but only the items which follow it. The original definition is:

```
\newcommand*{\captionformat}{:\ }
```

This too can be changed with \renewcommand.

Example: For some inexplicable reasons you want a dash with spaces before and after instead of a colon followed by a space as label delimiter. You define:

```
\renewcommand*{\captionformat}{~--~}
```

This definition should be put in the preamble of your document.

```
\figureformat
\tableformat
```

It was already mentioned that `\captionformat` does not contain formatting for the label itself. This situation should under no circumstances be changed using redefinitions of the commands for the output of counters, `\thefigure` or `\thetable`. Such a redefinition would have unwanted side effects on the output of `\ref` or the table of contents, list of figures and list of tables. To deal with the situation, KOMA-Script offers two `\...format` commands instead. These are predefined as follows:

```
\newcommand*{\figureformat}{\figurename~\thefigure\↵
    autodot}
\newcommand*{\tableformat}{\tablename~\thetable\autodot}
```

They also can be adapted to your personal preferences with `\renewcommand`.

Example: From time to time captions without any label and of course without delimiter are desired. In KOMA-Script it takes only the following definitions to achieve this:

```
\renewcommand*{\figureformat}{}
\renewcommand*{\tableformat}{}
\renewcommand*{\captionformat}{}

```

It should be noted, however, that although no numbering is output, the internal counters are nevertheless incremented. This becomes important especially if this redefinition is applied only to selected `figure` or `table` environments.

```
\setcapindent{indent}
\setcapindent*{xindent}
\setcaphanging
```

As mentioned previously, in the standard classes the captions are set in a non-hanging style, that is, in multi-line captions the second and subsequent lines start directly beneath the label. The standard classes provide no direct mechanism to change this behaviour. In KOMA-Script, on the contrary, beginning at the second line all lines are indented by the width of the label so that the caption text is aligned.

This behaviour, which corresponds to the usage of `\setcaphanging`, can easily be changed by using the command `\setcapindent` or `\setcapindent*`. Here the parameter *indent* determines the indentation of the second and subsequent lines.

If you want a line break after the label and before the caption text, then you can define the indentation *xindent* of the caption text with the starred version of the command instead: `\setcapindent*`.

Using a negative value of *indent* instead, a line break is also inserted before the caption text and only the first line of the caption text but not subsequent lines are indented by the absolute value of *indent*.

Whether one-line captions are set as captions with more than one line or are treated separately is specified with the class options `onelinecaption` and `noonelinecaption`. For details please refer to the explanations of these options in [section 3.1.3, page 45](#).

Example: For the examples please refer to figures [3.3](#) to [3.6](#). As you can see the usage of a fully hanging indentation is not advantageous when combined with narrow column width. To illustrate, the source code for the second figure is given here with a modified caption text:

```
\begin{figure}
  \setcapindent{1em}
  \fbox{\parbox{.95\linewidth}{\centering{\←
    KOMAScript}}}}
  \caption{Example with slightly indented ←
    caption
    starting at the second line}
\end{figure}
```

As can be seen the formatting can also be changed locally within the `figure` environment. The change then affects only the current figure. Following figures once again use the default settings or global settings set, for example, in the preamble of the document. This also of course applies to tables.

```
\setcapwidth[justification]{width}
\setcapmargin[margin left]{margin}
\setcapmargin*[margin inside]{margin}
```

v2.8q

Using these three commands you can specify the width and justification of the caption text. In general the whole text width or column width is available for the caption.

With the command `\setcapwidth` you can decrease this *width*. The obligatory argument determines the maximum *width* of the caption. As an optional argument you can supply exactly one letter which specifies

KOMA-Script

Figure 3.3: Equivalent to the standard setting, similar to the usage of `\setcaphanging`

KOMA-Script

Figure 3.4: With slightly hanging indentation starting at the second line using `\setcapindent{1em}`

KOMA-Script

Figure 3.5:
With hanging indentation starting at the second line and line break before the description using `\setcapindent*{1em}`

KOMA-Script

Figure 3.6:
With indentation in the second line only and line break before the description using `\setcapindent{-1em}`

the horizontal justification. The possible justifications are given in the following list.

- l – left-aligned
- c – centered
- r – right-aligned
- i – alignment at the inner margin in double-sided output
- o – alignment at the outer margin in double-sided output

The justification inside and outside corresponds to left-aligned and right-aligned, respectively, in single-sided output. Within `longtable` tables the justification inside or outside does not work correctly. In particular, the captions on subsequent pages of such tables are aligned according to the format of the caption on the first page of the table. This is a conceptual problem in the implementation of `longtable`.

With the command `\setcapmargin` you can specify a *margin* which is to be left free next to the description in addition to the normal text margin. If you want margins with different widths at the left and right side you can specify these using the optional argument *margin left*. The starred version `\setcapmargin*` defines instead of a *margin left* a *margin inside* in a double-sided layout. In case of `longtable` tables you have to deal with the same problem with justification inside or outside as mentioned with the macro `\setcapwidth`. Furthermore, the usage of `\setcapmargin`

or `\setcapmargin*` switches on the option `noonelinecaption` (see [section 3.1.3, page 45](#)) for the captions which are typeset with this margin setting.

`longtable` places the caption in a box, which is issued again on subsequent pages as needed. When outputting a box, the macros needed for its creation are not reevaluated. That is the reason why it is not possible for KOMA-Script to swap margin settings for even pages in double-sided layout. This is what would be necessary in order to produce a justification which is shifted towards the outside or inside.

You can also submit negative values for *margin* and *margin left* or *margin inside*. This has the effect of the caption expanding into the margin.

Example: A rather odd problem is that of a figure caption which is required to be both centered and of the same width as the figure itself. If the width of the figure is known in advance, the solution with KOMA-Script is quite easy. Supposing the figure has a width of 8 cm, it only takes:

```
\setcapwidth[c]{8cm}
```

directly in front of `\caption` or `\captionbelow`. If the width is unknown then you first have to define a length in the preamble of your document:

```
\newlength{\FigureWidth}
```

Having done this you can calculate the width directly with the \LaTeX command `\settowidth` (see [\[Tea01\]](#)) in many cases. A possible solution would look as follows:

```
\begin{figure}
  \centering%
  \settowidth{\FigureWidth}{%
    \fbox{\quad\KOMAScript\quad}%
  }%
  \fbox{\quad\KOMAScript\quad}%
  \setcapwidth[c]{\FigureWidth}
  \caption{Example of a centered caption below ↔
    the figure}
\end{figure}
```

However, it is awkward to write the content twice and to call `\setcapwidth` for every figure. Yet nothing is easier than defining a new command in the preamble of your document which hides the three steps of:

1. defining the width of the argument
2. specifying the width of the caption
3. outputting the argument

in:

```
\newcommand{\Figure}[1]{%
  \settowidth{\FigureWidth}{#1}%
  \setcapwidth[c]{\FigureWidth}%
  #1}
```

Using this command the example abbreviates to:

```
\begin{figure}
  \centering%
  \Figure{\fbox{\quad\KOMAScript\quad}}%
  \caption{Example of a centered caption below ↔
           the figure}
\end{figure}
```

However, commands have the disadvantage that errors in the macros of the argument in case of arguments with more than one line are not reported with the very accurate line numbers by L^AT_EX. Thus in some cases the use of an environment has advantages. Then, however, the question arises of how the width of the content of the environment can be determined. The solution involves the `lrbox` environment, described in [Tea01]:

```
\newsavebox{\FigureBox}
\newenvironment{FigureDefinesCaptionWidth}{%
  \begin{lrbox}{\FigureBox}%
}{%
  \end{lrbox}%
  \global\setbox\FigureBox=\box\FigureBox%
  \aftergroup\SetFigureBox%
}
\newcommand{\SetFigureBox}{%
  \Figure{\usebox{\FigureBox}}}
```

This definition uses the macro `\Figure` defined above. In the main text you write:

```
\begin{figure}
  \centering%
  \begin{FigureDefinesCaptionWidth}
    \fbox{\hspace{1em}\KOMAScript\hspace{1em}}
  \end{FigureDefinesCaptionWidth}
  \caption{Example of a centered caption below ↵
    the figure}
\end{figure}
```

Admittedly, the environment in this example is not necessary. However, its definition using `\global` is quite tricky. Most users would probably not be able to define such an environment without help. Thus, as this definition can be very useful, it was introduced in the above example.

Even if the `captionbeside` environment did not exist you could nevertheless place the figure caption beside the figure in a quite simple way. For this `\SetFigureBox` from the example above would have to be redefined first:

```
\renewcommand{\SetFigureBox}{%
  \settowidth{\captionwidth}{\usebox{\FigureBox↵
    }}%
  \parbox[b]{\captionwidth}{\usebox{\FigureBox↵
    }}%
  \hfill%
  \addtolength{\captionwidth}{1em}%
  \addtolength{\captionwidth}{-\hspace}%
  \setlength{\captionwidth}{-\captionwidth}%
  \setcapwidth[c]{\captionwidth}%
}
```

Finally you only have to put the `\caption` command in a `\parbox` too:

```
\begin{figure}
  \centering%
  \begin{FigureSetsCaptionWidth}
    \fbox{\rule{0pt}{5\baselineskip}}%
```



```

\hspace{1em}\KOMAScript\hspace{1em}}
\end{FigureSetsCaptionWidth}
\parbox[b]{\FigureWidth}{%
  \caption{Example of a centered caption
           below the figure}
}
\end{figure}

```

The `\rule` command in this example only serves as an invisible support to produce an example figure with a greater vertical height.

3.6.7 Logical Markup of Text

L^AT_EX offers different possibilities for logical markup of text. Strictly speaking, a heading is a kind of markup too. However, in this section we are only concerned with direct markup, i. e., markup which does not have an additional semantic meaning and which can be used for different purposes. More details on the normally defined possibilities can be found in [OPHS99], [Tea01] and [Tea00].

`\textsubscript{text}`

In [section 3.6.3, page 88](#), the command `\textsuperscript` was already introduced as an integral part of the L^AT_EX kernel. Unfortunately, L^AT_EX itself does not offer a command to produce text in subscript instead of superscript. KOMA-Script defines `\textsubscript` for this purpose.

Example: You are writing a text on human metabolism. From time to time you have to give some simple chemical formulas in which the numbers are in subscript. For enabling logical markup you first define in the document preamble or in a separate package:

```
\newcommand*{\molec}[2]{#1\textsubscript{#2}}
```

Using this you then write:

```

The cell produces its energy partly from ←
  reaction of \molec C6\molec
H{12}\molec O6 and \molec O2 to produce \molec ←
  H2\Molec O{ } and
\molec C{ }\molec O2. However, arsenic (\molec{←
  As}{ }) has a quite

```

detrimental effect on the metabolism.

The output looks as follows:

The cell produces its energy from reaction of $\text{C}_6\text{H}_{12}\text{O}_6$ and O_2 to produce H_2O and CO_2 . However, arsenic (As) has a quite detrimental effect on the metabolism.

Some time later you decide that the chemical formulas should be typeset in sans serif. Now you can see the advantages of using logical markup. You only have to redefine the `\molec` command:

```
\newcommand*{\molec}[2]{\textsf{\textsubscript{#1}\textsubscript{#2}}}
```

Now the output in the whole document changes to:

The cell produces its energy partly from reaction of $\text{C}_6\text{H}_{12}\text{O}_6$ and O_2 to produce H_2O and CO_2 . However, arsenic (As) has a quite detrimental effect on the metabolism.

In the example above, the notation “`\molec C6`” is used. This makes use of the fact that arguments consisting of only one character do not have to be enclosed in parentheses. That is why “`\molec C6`” is similar to “`\molec{C}{6}`”. You might already know this from indices or powers in mathematical environments, such as “ x^2 ” instead of “ $x^{\{2\}}$ ” for “ x^2 ”.

3.7 Appendix

The last part of a document usually contains the appendix, the bibliography and, if necessary, the index.

`\appendix`

The appendix in the standard as well as the KOMA-Script classes is introduced with `\appendix`. This command switches, among other things, the chapter numbering to upper case letters, also ensuring that the rules according to [DUD96] are followed (for German-speaking regions). These rules are explained in more detail in the description of the class options `pointednumbers` and `pointlessnumbers` in [section 3.1.7, page 50](#).

Please note that `\appendix` is a command, *not* an environment! This command does not expect any argument. Sectioning in the appendix uses `\chapter` and `\section` just as does the main text.

`\appendixmore`

There is a peculiarity within the `\appendix` command in the KOMA-Script classes. If the command `\appendixmore` is defined, this command is executed also by the `\appendix` command. Internally the KOMA-Script classes `scrbook` and `scrreprt` take advantage of this behaviour to implement the options `appendixprefix` and `noappendixprefix` (see [section 3.1.3, page 45](#)). You should take note of this in case you decide to define or redefine the `\appendixmore`. In case one of these options is set, you will receive an error message when using `\newcommand{\appendixmore}{...}`. This behaviour is intended to prevent you from disabling options without noticing it.

Example: You do not want the chapters in the main part of the classes `scrbook` or `scrreprt` to be introduced by a prefix line (see layout options `chapterprefix` and `nochapterprefix` in [section 3.1.3, page 44](#)). For consistency you also do not want such a line in the appendix either. Instead, you would like to see the word “Chapter” in the language of your choice written in front of the chapter letter and, simultaneously, in the page headings. Instead of using the either layout option `appendixprefix` or `noappendixprefix`, you would define in the document preamble:

```
\newcommand*{\appendixmore}{%
  \renewcommand*{\chapterformat}{%
    \appendixname~\thechapter\autodot\enskip}
  \renewcommand*{\chaptermarkformat}{%
    \appendixname~\thechapter\autodot\enskip}
}
```

In case you subsequently change your mind and decide to use the option `appendixprefix` at a later stage, you will get an error message because of the already defined `\appendixmore` command. This behaviour prevents the definition made above from invisibly changing the settings intended with the option.

It is also possible to get a similar behaviour of the appendix

for the class `scrartcl`. You would write in the preamble of your document:

```
\newcommand*{\appendixmore}{%
  \renewcommand*{\othersectionlevelsformat←
    }[1]{%
      \ifthenelse{\equal{##1}{section}}{\←
        appendixname~}{}%
      \csname the##1\endcsname\autodot\enskip}
  \renewcommand*{\sectionmarkformat}{%
    \appendixname~\thesection\autodot\enskip}
}
```

In addition, the package `ifthen` (see [Car99a]) is required.

Redefined commands are explained in more detail in [section 3.6.2, page 80](#) and [page 82](#).

`\setbibpreamble{preamble}`

The command `\setbibpreamble` can be used to set a preamble for the bibliography. This can be achieved by placing the preamble before the command for issuing the bibliography. However, it does not have to be directly in front of it. For example, it could be placed at the beginning of the document. Similar to the class options `bibtotoc` and `bibtotocnumbered`, this command can only be successful if you have not loaded a package which prevents this by redefining the `thebibliography` environment. Even though the `natbib` package makes unauthorized use of internal macros of KOMA-Script it could be achieved that `\setbibpreamble` works with the current version of `natbib` (see [Dal99]).

Example: You want to point out that the sorting of the references in the bibliography is not according to their occurrence in the text, but in alphabetical order. You use the following command:

```
\setbibpreamble{References are in alphabetical ←
order.
References with more than one author are ←
sorted
according to the first author.\par\bigskip}
```

The `\bigskip` command makes sure that the preamble and the first reference are separated by a large vertical space.

`\setindexpreamble{preamble}`

Similarly to the bibliography you can use a preamble to the index. This is often the case if you have more than one index or if you use different kinds of referencing by highlighting the page numbers in different ways.

Example: You have a document in which terms are both defined and used. The page numbers of definitions are in bold. Of course you want to make your reader aware of this fact. Thus you insert a preamble for the index:

```
\setindexpreamble{In \textbf{bold} printed page↵
    numbers are
    references to the definition of terms. Other ↵
    page numbers indicate
    the use of a term.\par\bigskip}
```

Please note that the page style of the first page of the index is changed. The applied page style is defined in the macro `\indexpagestyle` (see [section 3.2.2, page 59](#)).

The production, sorting and output of the index is done by the standard L^AT_EX packages and additional programs. Similar to the standard classes KOMA-Script only provides the basic macros and environments.

3.8 Obsolete Commands

In this section you will find commands which should not be used any longer. They are part of older KOMA-Script versions and their use was documented. For compatibility reasons they can still be used in the current KOMA-Script release. There are however new mechanisms and user interfaces which you should use instead. The reason for listing the obsolete macros in this documentation is only to aid users in understanding old documents. Furthermore, package authors are free to use these macros in the future.

`\sectfont`

This macro sets the font which is used for all section headings and the abstract, the main title and the highest level below `\part` in the table of contents. Instead, use the commands for the element disposition, described in [section 3.2.1](#).

`\capfont`
`\caplabelfont`

The macro `\capfont` sets the font which is used for captions in tables and figures. The macro `\caplabelfont` sets the font which is used for the label and numbering of tables and pictures. Instead, use the commands for the elements `caption` and `captionlabel`, described in [section 3.2.1](#).

`\descfont`

This macro sets the font for the optional item arguments of a `description` environment. Instead, use the commands for the element `descriptionlabel`, described in [section 3.2.1](#).

Adapting Page Headers and Footers with `scrpage2`

As already mentioned in the two previous chapters, KOMA-Script includes a package to customise the document page header and footer. As of 2001, this package is no longer `scrpage` but the much improved and enhanced successor `scrpage2`. Therefore, this documentation describes only `scrpage2`. The package `scrpage` is obsolete.

In place of `scrpage2` you can of course make use of `fancyhdr` (see [v000]). However, `scrpage2` integrated markedly better with the KOMA-Script bundle. For this reason, and because at the time the forerunner to `fancyhdr` was missing many features, `scrpage2` was developed. Naturally, `scrpage2` is not limited to use only with the KOMA-Script classes, but can just as easily be used with other document classes.

Included as part of the basic functionality of `scrpage2` are various pre-defined and configurable page styles.

4.1 Basic Functionality

To understand the following description, an overview of \LaTeX 's fairly involved header and footer mechanism is needed. The \LaTeX kernel defines the page styles `empty`, which produces a completely empty header and footer, and `plain`, which produces usually only a page number in the footer and an empty header. Apart from these, many document classes provide the style `headings`, which allows more complex style settings and running headings. The `headings` style often has a related variant, `myheadings`, which is similar except for switching off the running headings and reverting them to manual control by the user. A more detailed description is given in [section 3.2.2](#) where it is also noted that some \LaTeX commands automatically switch to the page style `plain` for the current page, independent of what page style was chosen by the author, and consequently a document needs an appropriate `plain` page style.

Therefore `scrpage2` defines its own `plain` and `headings` page styles, named `scrplain` and `scrheadings`. The manual activation of `scrplain` is not necessary, since the activation of `scrheadings` takes care of it automatically. Only if one wants to use one's own page style in combination with `scrplain` must the page style `scrplain` be activated first, i.e., with `\pagestyle{scrplain}`.

The original `headings` page style of the document class is available as

`useheadings`. This re-definition is required since `scrpage2` uses a different way to deal with automatic and manual headings. This way is more flexible and allows configurations which would usually prove difficult to implement for inexperienced users. The required commands to work with the `scrpage2` implementation are introduced at the end of [section 4.1.1](#) and the beginning of [section 4.1.2](#).

4.1.1 Predefined Page Styles

`scrheadings`
`scrplain`

Package `scrpage2` delivers its own page style, named `scrheadings`, which can be activated with the `\pagestyle{scrheadings}`. When this page style is in use, an appropriate `scrplain` page style is used for the `plain` page style. In this case *appropriate* means that this new `plain` page style is also configurable by the commands introduced in [section 4.1.3](#), which, for example, configure the header and footer width. Neither the activation of `scrheadings` nor the attendant change to the `plain` page style influences the mode of manual or automatic headings (see [section 4.1.2](#)). The `scrplain` page style can also be activated directly with `\pagestyle`.

```
\lehead[scrplain-left-even]{scrheadings-left-even}
\cehead[scrplain-center-even]{scrheadings-center-even}
\rehead[scrplain-right-even]{scrheadings-right-even}
\lefoot[scrplain-left-even]{scrheadings-left-even}
\cefoot[scrplain-center-even]{scrheadings-center-even}
\refoot[scrplain-right-even]{scrheadings-right-even}
\lohead[scrplain-left-odd]{scrheadings-left-odd}
\cohead[scrplain-center-odd]{scrheadings-center-odd}
\rohead[scrplain-right-odd]{scrheadings-right-odd}
\lofoot[scrplain-left-odd]{scrheadings-left-odd}
\cofoot[scrplain-center-odd]{scrheadings-center-odd}
\rofoot[scrplain-right-odd]{scrheadings-right-odd}
\ihhead[scrplain-inside]{scrheadings-inside}
\chhead[scrplain-centered]{scrheadings-centered}
\ohhead[scrplain-outside]{scrheadings-outside}
\ifoot[scrplain-inside]{scrheadings-inside}
\cfoot[scrplain-centered]{scrheadings-centered}
\ofoot[scrplain-outside]{scrheadings-outside}
```

The page styles include three boxes in both the header and the footer. The commands modifying the content of these boxes can be seen in [figure 4.1](#).

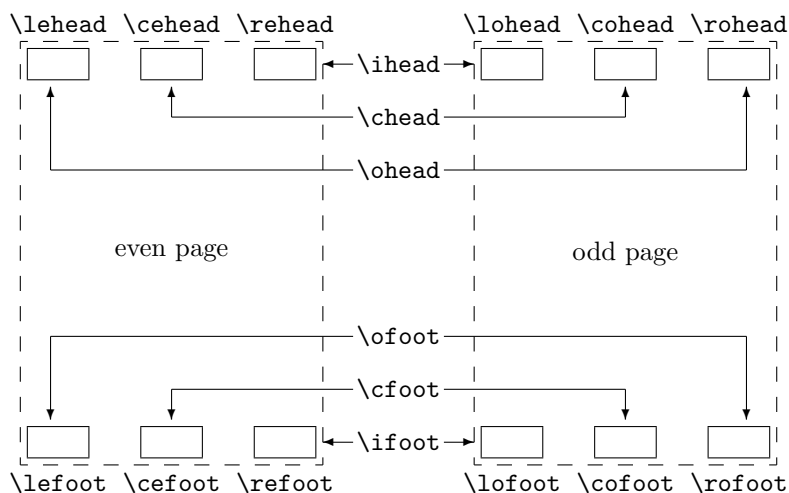


Figure 4.1: Commands for modification of page styles `scrheadings` and `scrplain` and their relationship to header and footer elements

Commands in the middle column modify the box contents on both odd and even pages.

Example: If one wants the page number be placed in the middle of the footer, then following can be used:

```
\cfoot{\pagemark}
```

The next example shows how to place both running heading and page number in the header; the running heading inside and the page number outside:

```
\ohead{\pagemark}
\ihead{\headmark}
\cfoot{}
```

The command `\cfoot{}` is only required in order to empty the item in the middle of the footer, which normally contains the page number.

The commands which are associated with only one item can be used for more advanced settings.

Example: Assuming one has the order to write an annual report for one's company, one could use commands like this:

```

\ohead{\pagemark}
\rehead{Annual Report 2001}
\lohead{\headmark}
\cefoot{TheCompanyName Inc.}
\cofoot{Department: Development}

```

In order to keep the data in the footer synchronized with the content of the document, the footer has to be updated using `\cofoot` when a new department is discussed in the report.

As mentioned above, there is a new **plain** page style which corresponds to **scrheadings**. Since it should also be possible to customize this style, the commands support an optional argument with which the contents of the appropriate fields of this **plain** page style can be modified.

Example: The position of the page number for the page style **scrheadings** can be declared as follows:

```

\cfoot[\pagemark]{ }
\ohead[]{\pagemark}

```

When the command `\chapter`, after it has started a new page, now switches to the **plain** page style, then the page number is centered in the footer.

```

\clearscrheadings
\clearscrplain
\clearscrheadfoot

```

If one wants to redefine both the page style **scrheadings** and the corresponding **plain** page style, frequently one must empty some already occupied page elements. Since one rarely fills all items with new content, in most cases several instructions with empty parameters are necessary. With the help of these three instructions the quick and thorough deletion is possible. While `\clearscrheadings` only deletes all fields of the page style **scrheadings**, and `\clearscrplain` deletes all fields of the corresponding **plain** page style, `\clearscrheadfoot` sets all fields of both page styles to empty.

Example: If one wants to reset the page style to the default KOMA-Script settings, independent of the actual configuration, only these three commands are sufficient:

```

\clearscrheadfoot
\ohead{\headmark}
\ofoot[\pagemark]{\pagemark}

```

Without the commands `\clearscrheadfoot`, `\clearscrheadings` and `\clearscrplain`, 6 commands with 9 empty arguments would be required:

```

\ihead[]{}
\chead[]{}
\ohead[]{\headmark}
\ifoot[]{}
\cfoot[]{}
\ofoot[\pagemark]{\pagemark}

```

Of course, for a specific configuration, some of them could be dropped.

In the previous examples two commands were used which have not been introduced yet. The description of these commands follows.

`\leftmark`
`\rightmark`

These two instructions make it possible to access the running headings, which are normally meant for the left or for the right page. These two instruction are not made available by `scrpage2`, but directly by the `LATEX` kernel. When in this section running headings of the left page or the right page are mentioned, this refers to the contents of `\leftmark` or `\rightmark`, respectively.

`\headmark`

This command gives access to the content of running headings. In contrast to `\leftmark` and `\rightmark`, one need not regard the proper assignment to left or right page.

`\pagemark`

This command returns the formatted page number. The formatting can be controlled by `\pnumfont`, introduced in [section 4.1.3, page 125](#), which `\pagemark` heeds automatically. Alternatively, `\setkomafont` can be used if a KOMA-Script class is used (see [section 3.2.1](#)).

useheadings

The package `scrpage2` is meant primarily for use of the supplied styles or for defining one's own styles. However, it may be necessary to shift back also to a style provided by the document class. It might appear that this should be done with `\page style{headings}`, but this has the disadvantage that commands `\automark` and `\manualmark`, to be discussed shortly, do not function as expected. For this reason one should shift back to the original styles using `\page style{useheadings}`, which chooses the correct page styles automatically for both manual and automatic running headings.

4.1.2 Manual and Running Headings

Usually there is a *my*-version of the `headings` page style. If such a page style is active, then the running headings are no longer updated no longer automatically and become manual headings. With `scrpage2` a different path is taken. Whether the headings are running or manual is determined by the instructions `\automark` and `\manualmark`, respectively. The default can be set already while loading of the package, with the options `automark` and `manualmark` (see [section 4.1.4, page 132](#)).

`\manualmark`

As the name suggests, `\manualmark` switches off the updating of the running headings and makes them manual. It is left to the user to update and provide contents for the headings. For that purpose the instructions `\markboth` and `\markright` are available.

`\automark[right page]{left page}`

The macro `\automark` activates the automatic updating, that is, running headings. For the two parameters the designations of the document sectioning level whose title is to appear in appropriate place are to be used. Valid values for the parameters are: `part`, `chapter`, `section`, `subsection`, `subsubsection`, `paragraph`, and `subparagraph`. For most of the classes use of `part` will not produce the expected result. So far only KOMA-Script classes from version 2.9s up are known to support this value.

2.2

The optional argument *right page* is understandably meant only for two-sided documents. In the one-sided case one should normally not use it. With the help of the option `autooneside` one can also set that the optional argument in one-sided mode is ignored automatically (see [section 4.1.4, page 133](#)).

Example: Assuming that the document uses a *book* class, whose topmost section level is *chapter*, then after a preceding `\manualmark`

```
\automark[section]{chapter}
```

restores the original behaviour. If one prefers lower section levels in running headings, the following can be used:

```
\automark[subsection]{section}
```

How useful the last declaration is, everybody has to decide for themselves.

For the upper section level, the data of the headings is set by the command `\markboth`, while that for the lower section level by `\markright` or `\markleft`. These commands are called indirectly by the sectioning commands. The macro `\markleft` is provided by the package `scrpage2` and is defined similarly to `\markright` in the \LaTeX kernel. Although `\markleft` is not defined as an internal command, the direct use is not recommended.

4.1.3 Formatting of Header and Footer

The previous section concerned itself mainly with the contents of the header and footer. This is of course not sufficient to satisfy formative ambitions. Therefore we devote this section exclusively to this topic.

```
\headfont
\pnumfont
```

The command `\headfont` contains the commands which determine the font of header and footer lines. The style of the page number is defined by the command `\pnumfont`.

Example: If, for example, one wants the header and footer to be typeset in bold sans serif, and the page number in a slanted serif style, then one can use the following definitions:

```
\renewcommand{\headfont}{\normalfont\sffamily\leftarrow
  bfseries}
\renewcommand{\pnumfont}{\normalfont\rmfamily\leftarrow
  slshape}
```

From version 2.8p of the KOMA-Script classes a new unified user interface scheme is implemented for font attributes. If `scrpage2` is used together

with one of these classes, then it is recommended to set up font attributes in the manner described in [section 3.2.1](#).

Example: Instead of `\renewcommand` the command `\setkomafont` should be used to configure the font attributes. The previous definitions can then be written as:

```
\setkomafont{pagehead}\normalfont\sffamily\l←
bfseries}
\setkomafont{pagenumber}{\normalfont\rmfamily\l←
slshape}
```

```
\setheadwidth[shift]{width}
\setfootwidth[shift]{width}
```

Normally the widths of header and footer lines correspond to the width of the text body. The commands `\setheadwidth` and `\setfootwidth` enable the user to adapt in a simple manner the widths to his needs. The mandatory argument *width* takes the value of the desired width of the page header or footer, while *shift* is a length parameter by which amount the appropriate item is shifted toward the outside page edge.

For the most common situations the mandatory argument *width* accepts the following symbolic values:

- `paper` – the width of the paper
- `page` – the width of the page
- `text` – the width of the text body
- `textwithmarginpar` – the width of the text body including margin
- `head` – the current header width
- `foot` – the current footer width

The difference between `paper` and `page` is that `page` means the width of the paper less the binding correction if the package `typearea` is used (see [chapter 2](#)). Without `typearea` both values are identical.

Example: Assume that one wants a layout like that of *The L^AT_EX Companion*, where the header projects into the margin. This can be obtained with:

```
\setheadwidth[0pt]{textwithmarginpar}
```

which appears like this on an odd page:

KOMA-Script	3
<hr/> This fill text is currently seized by 130 million receptors in your retina. Thereby the nerve Retina cells are put in a state of stimulation, which <u>spreads into the rear part</u> of your brain origi- nating from	

If the footer line should have the same width and alignment, then two ways to set this up are possible. The first way simply repeats the settings for the case of the footer line:

```
\setfootwidth[0pt]{textwithmarginpar}
```

In the second way the symbolic value `head` is used, since the header already has the desired settings.

```
\setfootwidth[0pt]{head}
```

If no *shift* is indicated, i. e., without the optional argument, then the header or footer appears arranged symmetrically on the page. In other words, a value for the *shift* is determined automatically to correspond to the current page shape.

Example: Continuing with the previous example, we remove the optional argument:

```
\setheadwidth{textwithmarginpar}
```

which appears like this on an odd page:

KOMA-Script	3
<hr/> This fill text is currently seized by 130 million receptors in your retina. Thereby the nerve Retina cells are put in a state of stimulation, which <u>spreads into the rear part</u> of your brain origi- nating from	

As can be seen, the header is now shifted inward, while the header width has not changed. The shift is calculated in a way that the configuration of the typearea become visible also here.

```
\setheadtopline[length]{thickness}[commands]
\setheadsepline[length]{thickness}[commands]
\setfootsepline[length]{thickness}[commands]
\setfootbotline[length]{thickness}[commands]
```

Corresponding to the size configuration parameters of header and footer there are commands to modify the lines above and below the header and footer.

`\setheadtopline` – configures the line above the header

`\setheadsepline` – configures the line below the header

`\setfootsepline` – configures the line above the footer

`\setfootbotline` – configures the line below the footer

The mandatory argument *thickness* determines how strongly the line is drawn. The optional argument *length* accepts the same symbolic values as *width* for `\setheadwidth`, as well as also a normal length expression. As long as the optional argument *length* is not assigned a value, the appropriate line length adapts automatically the width of the header or the footer.

Use `auto` in the length argument to restore this automation for the length of a line.

2.2

The optional argument *commands* may be used to specify additional commands to be executed before the respective line is drawn. For example, such commands could be used for changing the color of the line. When using a KOMA-Script class you could also use `\setkomafont` to specify commands for one of the elements `headtopline`, `headsepline`, `footsepline`, `footbottomline`, or `footbotline`. These can then be extended via `\addtokomafont`. See [section 3.2.1](#) for details on the `\setkomafont` and `\addkomafont` commands.

```
\setheadtopline[auto]{current}
\setheadtopline[auto]{}
\setheadtopline[auto]{}[]
```

The arguments shown here for the command `\setheadtopline` are of course valid for the other three configuration commands too.

If the mandatory parameter has the value `current` or has been left empty, then the line thickness is not changed. This may be used to modify the length of the line without changing its thickness.

If the optional argument `commands` is omitted, then all command settings that might have been specified before will remain active, while an empty `commands` argument will revoke any previously valid commands.

Example: If the header, for example, is to be contrasted by a strong line of 2pt above and a normal line of 0.4pt between header and body, one can achieve this with:

```
\setheadtopline{2pt}
\setheadsepline{.4pt}
```

<hr/>	
KOMA-Script	3
<hr/>	
This fill text is currently seized by 130 million receptors in your retina. Thereby the nerve cells are put in a state of stimulation, which spreads into the rear part of your brain origi- nating from	Retina

To specify that this line is to be drawn also, e.g., in red color, you would change the commands like this:

```
\setheadtopline{2pt}[\color{red}]
\setheadsepline{.4pt}[\color{red}]
```

In this example, as well as in the following one, line color is activated by applying the syntax of the `color` package, so this package must of course be loaded. Since `scrpage2` comes without built-in color handling, any package providing color support may be used.

KOMA-Script classes also support the following way of color specification:

```
\setheadtopline{2pt}
\setheadsepline{.4pt}
\setkomafont{headtopline}[\color{red}]
\setkomafont{headsepline}[\color{red}]
```

The automatic adjustment to the header and footer width is illustrated in the following example:

```
\setfootbotline{2pt}
\setfootsepline[text]{.4pt}
\setfootwidth[0pt]{textwithmarginpar}
```

This fill text is currently seized by 130 million
receptors in your retina. Thereby the nerve Retina
cells are put in a state of stimulation, which _____
KOMA-Script 3

Now not everyone will like the alignment of the line above the footer; instead, one would expect the line to be left-aligned. This can only be achieved with a global package option, which will be described together with other package options in the next [section 4.1.4](#).

4.1.4 Package Options

```
headinclude
headexclude
footinclude
footexclude
```

These options determine whether the page header or the page footer are considered as part of the page body for the calculation of the type area. The adjustments necessary by the use of these parameters are made by the package `typearea` (see [section 2.4](#)), if this package is loaded after `scrpage2`. Important here is that when using a KOMA-Script class, these options must be given for the document class and not for the package `scrpage2`, in order to be effective.

```
headtoplineand plainheadtopline
headseplineand plainheadsepline
footseplineand plainfootsepline
footbotlineand plainfootbotline
```

Basic adjustment of the lines under and over header and footer can be made with these options. These adjustments are then considered the default for

all page styles defined with `scrpage2`. If one of these options is used, then a line thickness 0.4pt is set.

Since there is a corresponding `plain` page style to the page style `scrheadings`, the corresponding line in the plain style can also be configured with the `plain...` options. These `plain` options do however work only if the corresponding options without `plain` are activated. Thus, `plainheadtopline` shows no effect without the `headtopline` option set.

With these options, it is to be noted that the appropriate page part, header or footer, is considered as a part of the text area for the calculation of the type area in case a line has been activated. This means that, if the separation line between header and text is activated with `headsepline`, then the package `typearea` calculates the type area in such a way that the page header is part of the text block automatically.

The conditions for the options of the preceding paragraph apply also to this automation. That means that the package `typearea` must be loaded after `scrpage2`, or that on use of a KOMA-Script class, the options `headinclude` and `footinclude` must be set explicitly with `\documentclass` in order to transfer header or footer line in the text area.

`ilines`
`clines`
`olines`

With the definition of the line lengths the case can arise where the lengths are set correctly, but the justification is not as desired because the line will be centered in the header or footer area. With the package options presented here, this specification can be modified for all page styles defined with `scrpage2`. The option `ilines` sets the justification in such a way that the lines align to the inside edge. The option `clines` behaves like the default justification, and `olines` aligns at the outside edge.

Example: The next example illustrates the influence of the option `ilines`. Please compare to the example for `\setfootsepline` on [page 130](#).

```
\usepackage[ilines]{scrpage2}
\setfootbotline{2pt}
\setfootsepline[text]{.4pt}
\setfootwidth[0pt]{textwithmarginpar}
```

The mere use of the option `ilines` leads to the different result shown below:

This fill text is currently seized by 130 million
receptors in your retina. Thereby the nerve Retina
cells are put in a state of stimulation, which

~~HOWA~~-Script

3

In contrast to the default configuration, the separation line between text and footer is now left-aligned, not centered.

automark
manualmark

These options set at the beginning of the document whether to use running headings or manual ones. The option **automark** switches the automatic updating on, **manualmark** deactivates it. Without the use of one of the two options, the setting which was valid when the package was loaded is preserved, .

Example: You load the package **scrpage2** directly after the document class **scrreprt** without any package options:

```
\documentclass{scrreprt}
\usepackage{scrpage2}
```

Since the default page style of **scrreprt** is **plain**, this page style is also now still active. Furthermore, **plain** means manual headings. If one now activates the page style **scrheadings** with

```
\pagestyle{scrheadings}
```

then the manual headings are nevertheless still active.

If you instead use the document class **scrbook**, then after

```
\documentclass{scrbook}
\usepackage{scrpage2}
```

the page style **headings** is active and the running headings are updated automatically. Switching to the page style **scrheadings** keeps this setting active. The marking commands of **scrbook** continue to be used.

However, the use of

```
\usepackage[automark]{scrpage2}
```

activates running headings independently of the used document class. The option does not of course affect the used page style **plain** of the class **scrreprt**. The headings are not visible until the page style is changed to **scrheadings**, **useheadings** or another user-defined page style with headings.

autooneside

This option ensures that the optional parameter of `\automark` will be ignored automatically in one-sided mode. See also the explanation of the command `\automark` in [section 4.1.2, page 124](#).

komastyle standardstyle

These options determine the look of the predefined page style **scrheadings**. The option **komastyle** configures a look like that of the KOMA-Script classes. This is the default for KOMA-Script classes and can in this way also be set for other classes.

The option **standardstyle** configures a page style as it is expected by the standard classes. Furthermore, the option **markuppercase** will be activated automatically, but only if option **markusedcase** is not given.

markuppercase markusedcase

In order to achieve the functionality of `\automark`, the package **scrpage2** modifies internal commands which are used by the document structuring commands to set the running headings. Since some classes, in contrast to the KOMA-Script classes, write the headings in uppercase letters, **scrpage2** has to know how the used document class sets the headings.

Option **markuppercase** shows **scrpage2** that the document class uses uppercase letters. If the document class does not set the headings in uppercase letters, then the option **markusedcase** should be given. These options are not suitable to force a representation; thus, unexpected effects may occur if the given option does not match the actual behaviour of the document class.

nouppercase

In the previous paragraph dealing with **markuppercase** and **markusedcase**, it has been already stated that some document classes

set the running headings in uppercase letters using the commands `\MakeUppercase` or `\uppercase`. Setting the option `nouppercase` allows disabling both these commands in the headers and footers. However, this is valid only for page styles defined by `scrpage2`, including `scrheadings` and its corresponding `plain` page style.

The applied method is very brutal and can cause that desired changes of normal letters to uppercase letters do not occur. Since these cases do not occur frequently, the option `nouppercase` usually affords a useful solution.

Example: Your document uses the standard class `book`, but you do not want the uppercase headings but mixed case headings. Then the preamble of your document could start with:

```
\documentclass{book}
\usepackage[nouppercase]{scrpage2}
\pagestyle{scrheadings}
```

The selection of the page style `scrheadings` is necessary, since otherwise the page style `headings` is active, which does not respect the settings made by option `nouppercase`.

In some cases not only classes but also packages set the running headings in uppercase letters. Also in these cases the option `nouppercase` should be able to switch back to the normal mixed case headings.

4.2 Defining Own Page Styles

4.2.1 The Interface for Beginners

Now one would not like to remain bound to only the provided page styles, but may wish to define one's own page styles. Sometimes there will be a special need, since a specific *Corporate Identity* may require the declaration of its own page styles. The easiest way to deal with this is:

```
\deftripstyle{name}[LO][LI]{HI}{HC}{HO}{FI}{FC}{FO}
```

The individual parameters have the following meanings:

- name* – the name of the page style, in order to activate it using the command `\pagestyle{name}`
- LO* – the thickness of the outside lines, i.e., the line above the header and the line below the footer (optional)

- LI* – the thickness of the separation lines, i. e., the line below the header and the line above the foot (optional)
- HI* – contents of the inside box in the page header for two-sided layout or left for one-sided layout
- HC* – contents of the centered box in the page header
- HO* – contents of the outside box in the page header for two-sided layout or right for one-sided layout
- FI* – contents of the inside box in the page footer for two-sided layout or left for one-sided layout
- FC* – contents of the centered box in the page footer
- FO* – contents of the outside box in the page footer for two-sided layout or right for one-sided layout

The command `\deftripstyle` definitely represents the simplest possibility of defining page styles. Unfortunately, there are also restrictions connected with this, since in a page range using a page style defined via `deftripstyle`, no modification of the lines above and below header and footer can take place.

Example: Assume a two-sided layout, where the running headings are placed on the inside. Furthermore, the document title, here “Report”, shall be placed outside in the header, the page number shall be centered in the footer.

```
\deftripstyle{TheReport}%
      {\headmark}{}{Report}%
      {}{\pagemark}{}{}
```

If moreover the lines above the header and below the footer shall be drawn with a thickness of 2 pt, and the text body be separated from header and footer with 0.4 pt lines, then the definition has to be extended:

```
\deftripstyle{TheReport}[2pt][.4pt]%
      {\headmark}{}{Report}%
      {}{\pagemark}{}{}
```

Report	2. The Eye
2.1 Retina This fill text is currently seized by 130 million receptors in your retina. Thereby the nerve cells are put in a state of stimulation, which spreads into the rear part of your brain originating from the optic nerve. From there the stimulation is transmitted in a split second also in other parts of your cerebrum. Your frontal lobe becomes stimulated. Intention impulses spread from there, which your central nervous	

2.1 Retina	Report
system transforms in actual deeds. Head and eyes already react. They follow the text, taking the information present there and transmit them via the optic nerve.	
<hr/> 15 <hr/>	

4.2.2 The Interface for Experts

Simple page styles, as they can be defined with `\deftripstyle`, are fairly rare according to experience. Either a professor requires that the thesis looks like his or her own — and who seriously wants to argue against such a wish? — or a company would like that half the financial accounting emerges in the page footer. No problem, the solution is:

```
\defpagestyle{name}{header definition}{footer definition}
\newpagestyle{name}{header definition}{footer definition}
\renewpagestyle{name}{header definition}{footer definition}
\providepagestyle{name}{header definition}{footer definition}
```

These four commands give full access to the capabilities of `scrpage2` to define page styles. Their structure is indential, they differ only in the manner of working.

- `\defpagestyle` – defines a new page style. If a page style with this name already exists it will be overwritten.
- `\newpagestyle` – defines a new page style. If a page style with this name already exists a error message will be given.
- `\renewpagestyle` – redefines a page style. If a page style with this name does not exist a error message will be given.
- `\providepagestyle` – defines a new page style only if there is no page style with that name already present.

Using `\defpagestyle` as an example, the syntax of the four commands is explained below.

- name* – the name of the page style for `\pagestyle{name}`
- header definition* – the declaration of the header, consisting of five element; elements in round parenthesis are optional:
 $(ALL, ALT)\{EP\}\{OP\}\{OS\}(BLL, BLT)$
- footer definition* – the declaration of the footer, consisting of five element; elements in round parenthesis are optional:
 $(ALL, ALT)\{EP\}\{OP\}\{OS\}(BLL, BLT)$

As can be seen, header and footer declaration have identical structure. The individual parameters have the following meanings:

ALL – above line length: (header = outside, footer = separation line)

ALT – above line thickness

EP – definition for *even* pages

OP – definition for *odd* pages

OS – definition for *one-sided* layout

BLL – below line length: (header = separation line, footer = outside)

BLT – below line thickness

If the optional line-parameters are omitted, then the line behaviour remains configurable by the commands introduced in [section 4.1.3, page 128](#).

The three elements *EP*, *OP* and *OS* are boxes with the width of page header or footer, as appropriate. The corresponding definitions are set left-justified in the boxes. To set something left- and right-justified into the boxes, the space between two text elements can be stretched using `\hfill`, in order to write the first text element on the left edge *and*:

```
{\headmark\hfill\pagemark}
```

If one would like a third text-element centered in the box, then an extended definition must be used. The commands `\rlap` and `\llap` simply write the given arguments, but for L^AT_EX they take up no horizontal space. Only in this way is the middle text really centered.

```
{\rlap{\headmark}\hfill centered text\hfill\llap{\↵
pagemark}}
```

This and the use of the expert interface in connection with other commands provided by `scpage2` follows now in the final example.

Example: This examples uses the document class `scrbook`, which means that the default page layout is two-sided. The package `scrpage2` is loaded with options `automark` and `headsepline`. The first switches on the automatic update of running headings, the second determines that a separation line between header and text body is drawn in the `scrheadings` page style.

```
\documentclass{scrbook}
\usepackage[automark,headsepline]{scrpage2}
```

The expert interface is used to define two page styles. The page style `withoutLines` does not define any line parameters. The second page style `withLines` defines a line thicknes of 1pt for the line above the header and 0pt for the separation line between header and text.

```
\defpagestyle{withoutLines}{%
  {Example\hfill\headmark}{\headmark\hfill ←
    without lines}
  {\rlap{Example}\hfill\headmark\hfill%
    \llap{without lines}}
}%
  {\pagemark\hfill}{\hfill\pagemark}
  {\hfill\pagemark\hfill}
}

\defpagestyle{withLines}{%
  (\textwidth,1pt)
  {with lines\hfill\headmark}{\headmark\hfill ←
    with lines}
  {\rlap{\KOMAScript}\hfill \headmark\hfill%
    \llap{with lines}}
  (0pt,0pt)
}%
  (\textwidth,.4pt)
  {\pagemark\hfill}{\hfill\pagemark}
  {\hfill\pagemark\hfill}
  (\textwidth,1pt)
```

```
}
```

Right at the beginning of the document the page style `scrheadings` is activated. The command `\chapter` starts a new chapter and automatically sets the page rstyle for this page to `plain`. Even though not a prime example, the command `\thead` shows how running headings can be created even on a `plain` page. However, in principle running headings on chapter start-pages are to be avoided, since otherwise the special character of the `plain` page style is lost. It is more important to indicate that a new chapter starts here than that a section of this page has a special title.

Instead of `\leftmark` one would expect the use of `\rightmark` in the parameter of `\thead`, since the chapter starts on an even page. But, because of internal \LaTeX definitions, this does not work. It only returns an empty string.

```
\begin{document}
\pagestyle{scrheadings}
\chapter{Thermodynamics}
```

```
\thead[\leftmark]{}

```

```
\section{Main Laws}

```

```
Every system has an extensive state quantity  $\leftarrow$ 
called
Energy. In a closed system the energy is  $\leftarrow$ 
constant.
```

1. Thermodynamics

1. Thermodynamics

1.1 Main Laws

Every System has an extensive state quantity

After starting a new page the page style `scrheadings` is active and thus the separation line below the header is visible.

There is a state quantity of a system, called entropy, whose temporal change consists of entropy flow and entropy generation.

1. Thermodynamics

There is a condition unit of a system, called entropy, whose temporal change consists of entropy flow and entropy generation.

After switching to the next page, the automatic update of the running headings is disabled using `\manualmark`, and the page style `withoutLines` becomes active. Since no line parameters are given in the definition of this page style, the default configuration is used, which draws a separation line between header and text body because `scrpage2` was called with `headsepline`.

Energy Conversion

without lines

1.2 Exergy and Anergy

During the transition of a system to an equilibrium state with its environment, the maximum work gainable is called exergy.

```
\manualmark
```

```
\pagestyle{withoutLines}
```

```
\section{Exergy and Anergy}\markright{Energy  
Conversion}
```

During the transition of a system to an equilibrium state

with its environment, the maximum work gainable is called

exergy.

At the next page of the document, the page style `withLines` is activated. The line settings of its definition are taken in account and the lines are drawn accordingly.

```

\pagestyle{mitLinien}
\renewcommand{\headfont}{\itshape\bfseries}
The portion of an energy not convertible in  $\leftarrow$ 
    exergy
is named anergy  $\text{\Var{B}}$ .
\[ B = U + T (S_1 - S_u) - p (V_1 - V_u) \]
\end{document}

```

with lines *1. Thermodynamics*

The portion of an energy not convertible in
exergy is named anergy B .

$$B = U + T(S_1 - S_u) - p(V_1 - V_u)$$

4.2.3 Managing Page Styles

Before long the work with different page styles will establish a common set of employed page styles, depending on taste and tasks. In order to make the management of page styles easier and avoid time-consuming copy operations each time a new project is started, `scrpage2` reads the file `scrpage.cfg` after initialisation. This file can contain a set of user-defined page styles which many projects can share.

Weekday and Time Using `scrdate` and `scrttime`

There are two packages included in KOMA-Script to improve and extend the handling of date and time over and above what is provided by the standard commands `\today` and `\date`. Like all the other packages from the KOMA-Script bundle these two packages may be used not only with KOMA-Script classes but also with the standard and many other classes.

5.1 The Name of the Current Day of the Week Using `scrdate`

The first problem is the question of the current day of the week. The answer may be given using the package `scrdate`.

`\todayname`

You should know that with `\today` one obtains the current date in a language-dependent spelling. `scrdate` offers you the command `\todayname` with which one can obtain the name of the current day of the week in a language-dependent spelling.

Example: In your document you want to show the name of the weekday on which the `dvi` file was generated using `LATEX`. To do this, you write:

```
I have done the {\LaTeX} run of this document↔
on a \todayname.
```

This will result in, e. g.:

```
I have done the LATEX run of this document on a
Friday.
```

Note that the package is not able to decline words. The known terms are the nominative singular that may be used, e. g., in the date of a letter. Given this limitation, the example above can work correctly only for some languages.

Tip: The names of the weekdays are saved in capitalized form, i. e., the first letter is a capital letter, all the others are lowercase letters. But for some languages you may need the names completely in lowercase. You may achieve this using the standard `LATEX` command `\MakeLowercase`. You simply have to write `\MakeLowercase{\todayname}`.

`\nameday{name}`

Analogous to how the output of `\today` can be modified using `\date`, so the output of `\today'sname` can be changed to *name* by using `\nameday`.

Example: You change the current date to a fixed value using `\date`. You are not interested in the actual name of the day, but want only to show that it is a workday. So you set:

```
\nameday{workday}
```

After this the previous example will result in:

I have done the L^AT_EX run of this document on a
workday.

Currently the package `scrdate` knows the languages english (english, american, USenglish, UKenglish and british), german (german, ngerman and austrian), french, italian, spanish, croatian, and finnish. If you want to configure it for other languages, see `scrdate.dtx`.

In the current implementation it does not matter whether you load `scrdate` before or after `german`, `ngerman`, `babel` or similar packages. The current language will be set up at `\begin{document}`.

To explain a little bit more exactly: while you are using a language selection which works in a compatible way to `babel` or `german`, the correct language will be used by `scrdate`. If you are using another language selection you will get (US) english names. In `scrdate.dtx` you will find the description of the `scrdate` commands for defining the names.

5.2 Getting the Time with Package `scrtime`

The second problem is the question of the current time. The solution may be found using package `scrtime`.

`\thistime[delimiter]`
`\thistime*[delimiter]`

`\thistime` results in the current time. The delimiter between the values of hour, minutes and seconds can be given in the optional argument. The default symbol of the delimiter is “:”.

`\thistime*` works in almost the same way as `\thistime`. The only difference is that unlike with `\thistime`, with `\thistime*` the value of the minute field is not preceded by a zero when its value is less than 10. Thus, with `\thistime` the minute field has always two places.

Example: The line

```
Your train departs at \thistime.
```

results, for example, in:

```
Your train departs at 11:56.
```

or:

```
Your train departs at 23:09.
```

In contrast to the previous example a line like:

```
This day is already \thistime*[\ hours and\ ] ←  
minutes old.
```

results in:

```
This day is already 11 hours and 56 minutes old.
```

or:

```
This day is already 12 hours and 25 minutes old.
```

`\settime{time}`

`\settime` sets the output of `\thistime` and `\thistime*` to the value *time*. Now the optional parameter of `\thistime` or `\thistime*` is ignored, since the result of `\thistime` or `\thistime*` was completely determined using `\settime`.

`12h`
`24h`

Using the options `12h` and `24h` one can select whether the result of `\thistime` and `\thistime*` is in 12- or in 24-hour format. The default is `24h`. The option has no effect on the results of `\thistime` and `\thistime*` if `\settime` is used.

The New Letter Class `scrlettr2`

v2.8q

Since the June 2002 release KOMA-Script provides a completely rewritten letter class. Although part of the code is identical to that of the main classes described in [chapter 3](#), letters are quite different from articles, reports, books, and suchlike. That alone justifies a separate chapter about the letter class. But there is another reason for a chapter on `scrlettr2`. This class has been redeveloped from scratch and provides a new user interface different from every other class the author knows of. This new user interface may be uncommon, but the author is convinced both experienced and new KOMA-Script users will benefit from its advantages.

6.1 Looking Back on the Old Letter Class

With the June 2002 release the old letter class `scrlettr` became obsolete. It is recommended not to use that class for new applications. There is no more active development of the old letter class, and support is very restricted. However, if you really need the documentation of the old letter class, you can still find it in the file `scrlettr.dtx`, but only in German. You should run it through \LaTeX several times, like this:

```
latex scrlettr.dtx
latex scrlettr.dtx
latex scrlettr.dtx
```

Then you obtain the file `scrlettr.dvi` containing the old German manual.

To facilitate the transition to the new class, there is a compatibility option. In general, the complete older functionality still remains in the new class. Without that compatibility option, the user interface and the defaults will be different. More details on this option are provided in [section 6.2.8](#), [table 6.10](#) and [section 6.9](#).

6.2 Options

The letter class `scrlettr2` uses the package `keyval` to handle options. This is part of the `graphics` package (see [[Car99b](#)]). Since `graphics` is part of the *required* section of \LaTeX , it should be found in every \LaTeX distribution. Should your \TeX distribution contain \LaTeX , but not the packages `graphics`

and keyval, please complain to your T_EX distributor. If you want to use `scrlltr2`, you will have to install the `graphics` package yourself in that case.

The special feature of the `keyval` package is that options can have values. Thus, you not only need a lot less options, but perhaps also fewer optional arguments. You will see that when discussing the `letter` environment in [section 6.4.3, page 185](#). The class will automatically load the `keyval` package. If you need to supply options to the `keyval` package, you should use the `\PassOptionsToPackage` command before `\documentclass`.

6.2.1 Defining Options Later

This section anticipates a feature of the new `letter` class. The meaning of this feature will not become clear until the structure of a document with more than one letter inside and another feature of `scrlltr2` is understood. However, to keep the number of forward references low, it is reasonable to describe the feature this early.

`\KOMAoptions{option list}`

A special feature of the `scrlltr2` class is the possibility to change many options even after loading the class. The `\KOMAoptions` command serves this purpose, taking options and their values as arguments. You can list multiple options, separated by commas, just like in the optional argument of `\documentclass`. If an option is only available when loading the class, i. e., as an optional argument to `\documentclass`, this will be explicitly mentioned in the option's description.

If you set an option to an illegal value within the *option list*, L^AT_EX will stop and show an error message. By entering “h” you will get an explanation that will also list possible values for that particular option.

6.2.2 Options for Compatibility

People who archive their letters in source code format generally place the highest priority on obtaining exactly the same results in future L^AT_EX runs. In some cases however, improvements and corrections to the class can lead to changes in behaviour, particularly as regards page breaking.

`version=value`

v2.9t

With `scrlltr2` there is the choice, whether a source file should give, as far as is technically possible, the same in any future L^AT_EX runs, or whether the document should be set according to the latest version of the class. To

which version compatibility should be retained is determined by the option `version`. The default is `version 2.9t`. The same effect can be obtained with

```
version=first
```

or

```
version=2.9
```

or

```
version=2.9t
```

If an unknown version is entered for *value*, a warning is output and for safety's sake `version=first` is assumed. With

```
version=last
```

the current newest version can be selected. In this case future compatibility is waived. If the option is used without any value, then `last` is assumed as well.

The question of compatibility concerns first and foremost page breaking. New features which have no effect on page breaking will be available even when compatibility to an earlier version is chosen by this option. The option also has no effect on changes in page breaking which result from the removal of old errors in the new version class. If page breaking compatibility is required absolutely, to the point of incorporating previous class errors, then the document should rather be archived together with the relevant version of KOMA-Script.

It should be noted that the option `version` can no longer be changed after loading of the class.

6.2.3 Page Layout Options

In contrast to the old `scrlttr` class, but in agreement with the other KOMA-Script classes, the `scrlttr2` class refers to the `typearea` package for the construction of the page layout (see [chapter 2](#)). The package will be loaded by the class automatically, and the class then controls the package. The necessary options will be explained in this section.

`paper=format`

This option defines the paper format. Theoretically, all paper formats known by the `typearea` are supported. However, you must leave out the suffix `paper` when entering a value. So, for letter format, you would use

the value `letter`. The formats of the ISO A, B, C, and D series must be entered in lowercase letters, e. g., `a4` for ISO A4. See also [section 2.5](#).

Although every paper size supported by `typearea` can be used, several formats may result in unexpected results on the first page of a letter. That is not a conceptual problem with the class, but a reflection that there currently exist only parameter sets for ISO A4. Unfortunately, there are no general rules to calculate the placement of the address field and so on for arbitrary paper sizes. However, it is possible to define additional parameter sets for different paper sizes. See [section 6.2.8](#) for more information.

```
BCOR=length
DIV=value
headlines=count
```

The options for the divisor, the binding correction, and the number of headlines will be translated directly into the corresponding options of the `typearea` package and passed to it. If the options are set using `\KOMAOptions` and not as class options, the `\typearea` command from the `typearea` package will be used instead. See [section 2.4](#), [page 24](#).

```
enlargefirstpage
```

As described later in this chapter, the first page of a letter always uses a different page layout. The `scrlltr2` class provides a mechanism to calculate height and vertical alignment of header and footer of the first page independently of the following pages. If, as a result, the footer of the first page would reach into the text area, this text area is automatically made smaller using the `\enlargethispage` macro. On the other hand, if the text area should become larger, supposing that the footer on the first page allows that, you can also use this option. At best, a little more text will then fit on the first page. See also the description of the pseudo-length `firstfootvpos` in [section 6.4.2](#), [page 181](#). This option can take the standard values for simple switches, as listed in [table 6.1](#), [page 149](#). Default is `false`.

6.2.4 Other Layout Options

In this subsection, you will find all options, except the specific page layout options, that have an influence on the layout in general. Strictly speaking, all page layout options (see [6.2.3](#)) are also layout options, and vice versa for some of them.

Table 6.1: Standard values for simple switches in `srlttr2`

Value	Description
<code>true</code>	activates the option
<code>on</code>	activates the option
<code>false</code>	deactivates the option
<code>off</code>	deactivates the option

`mpinclude`
`mpexclude`

These two options of the `typearea` package should not be used with the `srlttr2` class, because the first page in particular does not take this option into account. To anticipate any complaints, a warning will be issued when this option is used. If you feel adventurous you could try how these options, especially `mpinclude`, interact with other class options.

`twoside`

From the author's point of view, double-sided letters do not make much sense. Therefore, the option `twoside` only partially switches to double-sided layout. You thus get the possibility to have different margins on left and right pages, but this possibility is not used. So this option really means *activate the possibilities of a double-sided document, but stay with the one-sided layout as far and as long as possible*. This option can take the standard values for simple switches, as listed in [table 6.1, page 149](#). Default is `false`.

By the way, double-sided letters are not supported, because they seem entirely unreasonable.

`cleardoublepage=style`

If you want pages inserted by the `\cleardoublepage` command to just contain a page number in the header and footer, or to be empty, this can be accomplished with this option. There are three different styles supported that are listed at [table 6.2](#). Default is `standard`.

`headsepline`
`footsepline`

These two options insert a separator line below the header or above the footer, respectively, on consecutive pages. In the terminology of this manual, all pages of a letter except the first one are consecutive pages. This

Table 6.2: Possible values of option `cleardoublepage` for selection of page style of empty left pages with `scrlltr2`

<code>empty</code>	switches to page style <code>empty</code> for inserted pages
<code>plain</code>	switches to page style <code>plain</code> for inserted pages
<code>standard</code>	keeps the current page style for inserted pages

option can take the standard values for simple switches, as listed in [table 6.1, page 149](#). Default is `false`. If one of the options is used without a value, like in the declaration above, this evaluates as `true`, so the separator line will be activated. When used as a `\documentclass` option, the option `headinclude` or `footinclude`, respectively, will be passed on to the `typearea` package (see [section 2.4, page 27](#)).

`pagenumber=position`

This option defines if and where a page number will be placed on consecutive pages. All pages without a letterhead are consecutive pages. This option affects the page styles `headings` and `plain`. It also affects the default page styles of the `scrpage2` package, if set before loading the package (see [chapter 4](#)). It can take values only influencing horizontal, only vertical, or both positions. Possible value are shown in [table 6.3](#). Default is `botcenter`.

`parskip=value`

Especially in letters you often encounter paragraphs marked not with indentation of the first line, but with a vertical skip between them. This is a matter of tradition. Apparently, it was easier for a secretary to operate the carriage return lever twice than to set an indentation using a tab stop or the space bar. Correct justification is almost impossible using a typewriter, so letters are traditionally typeset unjustified.

However, typographers like Jan Tschichold take the view that letters, written using means available to modern typesetting, should take advantage of their possibilities just like other documents do. Under these circumstances, letters should also be typeset using paragraph indentation and justification.

Table 6.3: Possible values of option `pagenumber` for the position of the page number in page styles `headings` and `plain` with `srlttr2`

<code>bot, foot</code>	page number in footer, horizontal position not changed
<code>botcenter, botcentered, botmittle, footcenter, footcentered, footmiddle</code>	page number in footer, centered
<code>botleft, footleft</code>	page number in footer, left justified
<code>botright, footright</code>	page number in footer, right justified
<code>center, centered, middle</code>	page number centered horizontally, vertical position not changed
<code>false, no, off</code>	no page number
<code>head, top</code>	page number in header, horizontal position not changed
<code>headcenter, headcentered, headmiddle, topcenter, topcentered, topmiddle</code>	page number in header, centered
<code>headleft, topleft</code>	page number in header, left justified
<code>headright, topright</code>	page number in header, right justified
<code>left</code>	page number left, vertical position not changed
<code>right</code>	page number right, vertical position not changed

As a reaction to many serious requests, `scrlltr2` offers the possibility to mark paragraphs not only by indentation of the first line, but alternatively by a vertical skip. You can choose between a full or half a line of vertical space. When using paragraph spacing, it is often useful to keep the last line of a paragraph shorter so that paragraph recognition will be eased. All these features are controlled by different values for the `parskip` option, shown in [table 6.4](#). Default is `false`.

6.2.5 Font Options

Fonts options are any options which influence the size of the base font or of fonts for particular parts of the letter. In theory, options affecting the font type would also count as font options. At present there is only one option for font size in `scrlltr2`.

`fontsize=size`

Whereas in the main classes you choose the font size for the document using the `10pt`, `12pt`, etc., options, in the `scrlltr2` class the desired `size` is set using the `fontsize` option. The functionality is however the same. This option can only be used with `\documentclass`, not with `\KOMAsoptions`. Default is `12pt`.

6.2.6 Options for Letterhead and Address

The `scrlltr2` class offers numerous extensions for the design of the letterhead. There are also options for address formatting, extending the possibilities of the standard letter class, although these features could already be found in the now obsolete `scrlettr` class.

`fromalign`

This option defines the placement of the from address in the letterhead of the first page. At the same time, this option serves as a switch to activate or deactivate the extended letterhead options. If these extensions are deactivated, some other options will have no effect. This will be noted in the explanations of the respective options. Possible values for `fromalign` are shown in [table 6.5](#). Default is `left`.

`fromrule`

This option is part of the letterhead extensions (see option `fromalign` above). It allows you to place a horizontal line within the return address.

Table 6.4: Possible values of option `parskip` to select the paragraph mark with `scrlltr2`

<code>false</code> , <code>off</code>	paragraph indentation instead of vertical space; the last line of a paragraph may be arbitrarily filled
<code>full</code> , <code>on</code> , <code>true</code>	one line vertical space between paragraphs; there must be at least 1 em free space in the last line of a paragraph
<code>full*</code>	one line vertical space between paragraphs; there must be at least a quarter of a line free space at the end of a paragraph
<code>full+</code>	one line vertical space between paragraphs; there must be at least a third of a line free space at the end of a paragraph
<code>full-</code>	one line vertical space between paragraphs; the last line of a paragraph may be arbitrarily filled
<code>half</code>	half a line vertical space between paragraphs; there must be at least 1 em free space in the last line of a paragraph
<code>half*</code>	half a line vertical space between paragraphs; there must be at least a quarter of a line free space at the end of a paragraph
<code>half+</code>	half a line vertical space between paragraphs; there must be at least a third of a line free space at the end of a paragraph
<code>half-</code>	one line vertical space between paragraphs

Table 6.5: Possible values of option `fromalign` for setting the position of the from address in the letterhead with `scrlttr2`

<code>center, centered, middle</code>	return address centered; an optional logo will be above the extended return address; letterhead extensions will be activated
<code>false, no, off</code>	standard design will be used for the return address; the letterhead extensions are deactivated
<code>left</code>	left-justified return address; an optional logo will be right justified; letterhead extensions will be activated
<code>right</code>	right-justified return address; an optional logo will be left justified; letterhead extensions will be activated

The possible values are shown in [table 6.6](#). Default is `false`. You can not activate more than one line at a time. Regarding the length of the line, see [section 6.4.7, page 180](#).

fromphone

This option is part of the letterhead extensions (see option `fromalign` above). It defines whether the phone number will be part of the return address. This option can take the standard values for simple switches, as listed in [table 6.1, page 149](#). Default is `false`.

Table 6.6: Possible values of option `fromrule` for the position of the rule in the from address with `scrlttr2`

<code>afteraddress, below, on, true, yes</code>	rule below the return address
<code>aftername</code>	rule directly below the sender's name
<code>false, no, off</code>	no rule

fromfax

This option is part of the letterhead extensions (see option **fromalign** above). It defines whether the facsimile number will be part of the return address. This option can take the standard values for simple switches, as listed in [table 6.1, page 149](#). Default is **false**.

fromemail

This option is part of the letterhead extensions (see option **fromalign** above). It defines whether the email address will be part of the return address. This option can take the standard values for simple switches, as listed in [table 6.1, page 149](#). Default is **false**.

fromurl

This option is part of the letterhead extensions (see option **fromalign** above). It defines whether the URL will be part of the return address. This option can take the standard values for simple switches, as listed in [table 6.1, page 149](#). Default is **false**.

fromlogo

This option is part of the letterhead extensions (see option **fromalign** above). It defines whether the logo will be part of the return address. This option can take the standard values for simple switches, as listed in [table 6.1, page 149](#). Regarding the placement of the logo, see also the explanation of the option **fromalign** above. Default is **false**.

addrfield

This option defines whether an address field will be set. Default is to use the address field. This option can take the standard values for simple switches, as listed in [table 6.1, page 149](#). Default is **true**.

backaddress

This option defines whether a return address for window envelopes will be set. Default is to use the return address. If the address field is suppressed (see option **addrfield**), there will be no return address either. This option can take the standard values for simple switches, as listed in [table 6.1, page 149](#). Default is **true**.

subject

This option serves two purposes: first, you can choose if your subject should have a title, given by the *subject* variable (see [table 6.17](#),

Table 6.7: Possible values of option `subject` for the position of the subject with `scrlttr2`

<code>afteropening</code>	set subject after opening
<code>beforeopening</code>	set subject before opening
<code>titled</code>	add title to subject
<code>untitled</code>	do not add title to subject

[page 191](#)); second, you can choose if the subject should be set before or after the opening. Possible values for this option are shown in [table 6.7](#). Defaults are `beforeopening` and `untitled`.

locfield

`scrlttr2` places a field with additional sender attributes next to the address field. This can be used, for example, for bank account or similar additional information. Depending on the `fromalign` option, it will also be used for the sender logo. The width of this field may be defined within an `lco` file (see [section 6.2.8](#)). If the width is set to 0 in that file, then the `locfield` option can toggle between two presets for the field width. See the explanation on the `locwidth` pseudo length in [section 6.4.4, page 187](#). Possible values for this option are shown in [table 6.8](#). Default is `narrow`.

foldmarks

This option activates fold marks for two- or three-panel folding of the letter. The exact placement of the fold marks for three-panel letter folding

Table 6.8: Possible values of option `locfield` for setting the width of the field with additional sender attributes with `scrlttr2`

<code>narrow</code>	narrow sender supplement field
<code>wide</code>	wide sender supplement field

Table 6.9: Possible value of option **refline** for setting the width of the reference fields line with **scrlttr2**

narrow	reference fields line restricted to type area
wide	reference fields line corresponds to address and sender attributes

depends on user settings or the **lco** file (see [section 6.2.8](#)). The folding need not result in equal-sized parts. This option can take the standard values for simple switches, as listed in [table 6.1, page 149](#). Default is **true**, which implies setting the fold marks.

numericaldate

This option toggles between the standard, language-dependent date presentation, and a short, numerical one. KOMA-Script does not provide the standard presentation. It should be defined by packages such as **german**, **babel**, or **isodate**. The short, numerical presentation, on the other hand, is produced by **scrlttr2** itself. This option can take the standard values for simple switches, as listed in [table 6.1, page 149](#). Default is **false**, which results in standard date presentation. In the now obsolete **scrlettr** class, the opposite effect was achieved using the **orgdate** option.

refline

With the **scrlttr2** class, the header, footer, address, and sender attributes may extend beyond the normal type area to the left and to the right. This option defines whether that should also apply to the reference fields line. Normally, the reference fields line contains at least the date, but it can hold additional data. Possible values for this option are shown in [table 6.9](#). Default is **narrow**.

6.2.7 Formatting Options

Formatting options are those which influence form or formatting of the output and do not belong to another section. You might also call them the *miscellaneous options*.

draft

This option toggles between the final and the draft version of a document. In particular, enabling the **draft** option activates little black boxes

that will be drawn at the end of overfull lines. These boxes allow the unpracticed eye to more easily identify paragraphs that need manual intervention. When the `draft` option is disabled, there will be no such boxes. This option can take the standard values for simple switches, as listed in [table 6.1, page 149](#). Default is `false`, as usual. However, I strongly recommend enabling the `draft` option when designing a letter, as for every other document.

6.2.8 The Letter Class Option Files

Normally, you would not redefine parameters like the distance between the address field and the top edge of the paper every time you write a letter. Instead, you would reuse a whole set of parameters for certain occasions. It will be much the same for the letterhead and footer used on the first page. Therefore, it is reasonable to save these settings in a separate file. For this purpose, the `scrlettr2` class offers the `lco` files. The `lco` suffix is an abbreviation for *letter class option*.

In an `lco` file you can use all commands available to the document at the time the `lco` file is loaded. Additionally, it can contain internal commands available to package writers. For `scrlettr2`, these are in particular the commands `\@newplength`, `\@setplength`, and `\@addtoplength` (see [section 6.3.4](#)).

There are already some `lco` files included in the KOMA-Script distribution. The `DIN.lco`, `DINmtext.lco`, `SNleft.lco`, and `SN.lco` files serve to adjust KOMA-Script to different layout standards. They are well suited as templates for your own parameter sets. The `KOMAold.lco` file, on the other hand, serves to improve compatibility with the old letter class `scrlettr`. Since it contains internal commands not open to package writers, you should not use this as a template for your own `lco` files. You can find a list of predefined `lco` files in [table 6.10, page 162](#).

If you have defined a parameter set for a letter standard not yet supported by KOMA-Script, you are explicitly invited to send this parameter set to the KOMA-Script support address. Please do not forget to include the permission for distribution under the KOMA-Script license (see the `lpp1.txt` file). If you know the necessary metrics for an unsupported letter standard, but are not able to write a corresponding `lco` file yourself, you can also contact the KOMA-Script author, Markus Kohm, directly.

`\LoadLetterOption{name}`

Usually, the `lco` files will be loaded by the `\documentclass` command.

You enter the name of the `lco` file without suffix as an option. The `lco` file will be loaded right after the class file.

However, it is also possible to load an `lco` file later, or even from within another `lco` file. This can be done with the `\LoadLetterOption` command, which takes the *name* of the `lco` file without suffix as a parameter.

Example: You write a document containing several letters. Most of them should comply with the German DIN standard. So you start with:

```
\documentclass{sclttr2}
```

However, one letter should use the `DINmtext` variant, with the address field placed more toward the top, which results in more text fitting on the first page. The folding will be modified so that the address field still matches the address window in a DIN C6/5 envelope. You can achieve this as follows:

```
\begin{letter}{Markus Kohm\\
  Freiherr-von-Drais-Stra\ss e 66\\68535 ←
  Edingen-Neckarhausen}
\LoadLetterOption{DINmtext}
\opening{Hello,}
```

Since construction of the page does not start before the `\opening` command, it is sufficient to load the `lco` file before this. In particular, the loading need not be done before `\begin{letter}`. Therefore the changes made by loading the `lco` file are local to the corresponding letter.

If an `lco` file is loaded via `\documentclass`, it may nevertheless take the name of an option, provided it is an option that does not take an argument. Nevertheless, it would be possible to give the name `fromalign=left.lco` to an `lco` file. This will then be loaded every time the `\documentclass` option `fromalign` is used with the value `left`. Admittedly, this is only of academic interest. Of course you can use this feature only if your operating system and file system support this kind of file name. Otherwise you have to choose another file name and add the corresponding option, if needed.

Example: You do not want to enter your sender address every time, so you create an `lco` file with the necessary data, like this:

```

\ProvidesFile{mkohm.lco}[2002/02/25 letter ←
  class option]
\setkomavar{fromname}{Markus Kohm}
\setkomavar{fromaddress}{Freiherr-von-Drais←
  Stra\ss e 66\\
                                68535 Edingen←
                                Neckarhausen}

```

The command `\setkomavar` used above, and the principle of variables will be explained in detail in [section 6.3.3, page 169](#). In the example given here, knowledge of the exact function of the command is not critical. It is only important to note what can be done with `lco` files, less so exactly how this might be accomplished. Please note that the German sharp s, “ß”, was entered using the \TeX macro `\ss`, because directly after `\documentclass` no packages for input encoding, for example `\usepackage[latin1]{inputenc}` for Unix or `\usepackage[ansinew]{inputenc}` for Windows, and no language packages, like `\usepackage{ngerman}` for the new German orthography, are loaded.

However, if you would always use the same input encoding, you could also include it into your `lco` file. This would look as follows:

```

\ProvidesFile{mkohm.lco}[2002/02/25 letter ←
  class option]
\RequirePackage[latin1]{inputenc}
\setkomavar{fromname}{Markus Kohm}
\setkomavar{fromaddress}{Freiherr-von-Drais←
  Stra\ss e 66\\
                                68535 Edingen←
                                Neckarhausen}

```

There is one distinct disadvantage with this usage: you can no longer load this `lco` file later in your document. If you want to have letters with different senders in one document, you should therefore refrain from loading packages in your `lco` file.

Let us further assume that I always typeset letters using the preset parameters `KOMAold`. Then I could add the following line to my `mkohm.lco` file:


```
\LoadLetterOption{KOMAdId}
```

Anyway, now you can preset my sender address using

```
\documentclass[mkohm]{scr1ttr2}
```

In [table 6.10, page 162](#) you can find a list of all predefined lco files. If you use a printer that has large unprintable areas on the left or right side, you might have problems with the SN option. Since the Swiss standard SN 101 130 defines the address field to be placed 8 mm from the right paper edge, the headline and the sender attributes too will be set with the same small distance from the paper edge. This also applies to the reference fields line when using the `refline=wide` option (see [section 6.2.6, page 157](#)). If you have this kind of problem, create your own lco file that loads SN first and then changes `toaddrhpos` (see [section 6.4.3, page 183](#)) to a smaller value. Additionally, also reduce `toaddrwidth` accordingly.

```
\LetterOptionNeedsPapersize{option name}{paper size}
```

As mentioned in [section 6.2.3](#), at present there exist only parameter sets and lco files for A4-sized paper. In order that you will at least be warned when using another *paper size*, you will find a `\LetterOptionNeedsPapersize` command in every lco file distributed with KOMA-Script. The first argument is the name of the lco file without the “.lco” suffix. The second argument is the paper size for which the lco file is designed.

If several lco files are loaded in succession, a `\LetterOptionNeedsPapersize` command can be contained in each of them, but the `\opening` command will only check the last given *paper size*. As shown in the following example, an experienced user can thus easily write lco files with parameter sets for other paper sizes. If you do not plan to set up such lco files yourself, you may just forget about this option and skip the example.

Example: Suppose you use A5-sized paper in normal, i. e., upright or portrait, orientation for your letters. We further assume that you want to put them into standard C6 window envelopes. In that case, the position of the address field would be the same as for a DIN standard letter on A4-sized paper. The main difference is that A5 paper needs only one fold. So you want to disable the upper and lower fold marks. The easiest way to achieve this is to place the marks outside of the paper area.

Table 6.10: The predefined lco files

lco file	Description and features
DIN	parameter set for letters on A4-size paper, complying with German standard DIN 676; suitable for window envelopes in the sizes C4, C5, C6, and C6/5 (C6 long).
DINmtext	parameter set for letters on A4-size paper, complying with DIN 676, but using an alternate layout with more text on the first page; only suitable for window envelopes in the sizes C6 and C6/5 (C6 long).
KOMAold	parameter set for letters on A4-size paper using a layout close to the now obsolete <code>scrlettr</code> letter class; suitable for window envelopes in the sizes C4, C5, C6, and C6/5 (C6 long); some additional commands to improve compatibility with obsolete <code>scrlettr</code> commands are defined; <code>scrlettr2</code> may behave slightly different when used with this lco file than with the other lco files.
SN	parameter set for Swiss letters with address field on the right side, according to SN 010 130; suitable for Swiss window envelopes in the sizes C4, C5, C6, and C6/5 (C6 long).
SNleft	parameter set for Swiss letters with address field on the left side; suitable for Swiss window envelopes with window on the left side in the sizes C4, C5, C6, and C6/5 (C6 long).

```

\ProvidesFile{paper=a5.lco}[2002/05/02 letter ↵
    class option]
\LetterOptionNeedsPapersize{paper=a5}{a5}
\@setlength{tfoldmarkvpos}{\paperheight}
\@setlength{bfoldmarkvpos}{\paperheight}

```

Besides this, the placement of the foot, that is, the pseudo-length `firstfootvpos`, must be adjusted. It is left to the reader to find an appropriate value. When using such an lco file, you must only take care that other lco file options, like `SN`, are declared before

the paper size, i.e., before loading “paper=a5.lco”. Does this seem too complicated? Only before you have used it the first time. Anyway, how often do you write letters not using your standard formats for A4-size or letter-size paper?

By the way, the DIN lco file will always be loaded as the first lco file. This ensures that all pseudo-lengths will have more or less reasonable default values.

Please note that it is not possible to use `\PassOptionsToPackage` to pass options to packages from within an lco file that have already been loaded by the class. Normally, this only applies to the `typearea`, `scrfile`, and `keyval` packages.

6.3 General Document Properties

Some document properties are not assigned to any particular part of the document such as to the letterhead or the letter body. Several of these properties have already been mentioned or explained in [section 6.2](#).

6.3.1 Font Selection

Commands for defining, extending and querying the font of a specific element can be found [3.2.1](#). These commands work exactly the same in `scrlettr2`. The elements which can be influenced in this way are listed in [table 6.11](#).

6.3.2 Page Style

One of the general properties of a document is the page style. Please refer also to [section 3.2.2](#) and [chapter 4](#).

```
\pagestyle{empty}
\pagestyle{plain}
\pagestyle{headings}
\pagestyle{myheadings}
\thispagestyle{local page style}
```

In letters written with `scrlettr2` there are four different page styles.

empty is the page style, in which the header and footer of subsequent pages (all pages apart from the first) are completely empty. This page style is also used for the first page, because header and footer of this page are set by other means using the macro `\opening` (see [section 6.4.1](#), [section 6.4.2](#), as well as [section 6.5.1](#), [page 193](#)).

Table 6.11: Alphabetical list of elements whose font can be changed in `scrlltr2` using the commands `\setkomafont` and `\addtokomafont`

<code>backaddress</code>	return address for a window envelope
<code>descriptionlabel</code>	label, i.e., the optional argument of <code>\item</code> , in a <code>description</code> environment
<code>fromaddress</code>	sender's address in the letterhead
<code>fromname</code>	sender's address in the letterhead if different from <code>fromaddress</code>
<code>pagefoot</code>	in most cases the footer, sometimes the header of a page
<code>pagehead</code>	in most cases the header, sometimes the footer of page
<code>pagenumber</code>	page number in the footer or header, inserted with <code>\pagemark</code>
<code>subject</code>	subject in the opening of the letter
<code>title</code>	title in the opening of the letter

plain is the page style with only page numbers in the header or footer on subsequent pages. The placement of these page numbers is determined by the option `pagenumber` (see [section 6.2.4, page 150](#)).

headings is the page style for running (automatic) headings on subsequent pages. The inserted marks are the sender's name from the variable `fromname` and the subject from the variable `subject` (see [section 6.4.1, page 179](#) and [section 6.4.6, page 190](#)). At which position these marks and the page numbers are placed, depends on the option `pagenumber` (see [section 6.2.4, page 150](#)). The author can also change these marks manually after the `\opening` command. To this end, the commands `\markboth` and `\markright` are available

as usual, and with the use of package `scrpage2` also `\markleft` (see [section 4.1.2, page 124](#)) is available.

myheadings is the page style for manual page headings on subsequent pages. This is very similar to **headings**, but here the marks must be set by the author using the commands `\markboth` and `\markright`. With the use of package `scrpage2` also `\markleft` can be utilized.

In the terminology of this manual, subsequent pages are all pages of a letter except for the first one.

Page styles are also influenced by the option `headsepline` and `footsepline` (see [section 6.2.4, page 149](#)). The page style beginning with the current page is switched using `\pagestyle`. In contrast, `\thispagestyle` changes only the page style of the current page. The letter class itself uses `\thispagestyle{empty}` within `\opening` for the first page of the letter.

For changing the font style of headers or footers you should use the user interface described in [section 3.2.1](#). For header and footer the same element is used, which you can name either `pagehead` or `pagefoot`. The element for the page number within the header or footer is named `pagenumber`. Default settings are listed in [table 3.4, page 58](#). Please have also a look at the example in [section 3.2.2, page 58](#).

```
\clearpage
\cleardoublepage
\cleardoublestandardpage
\cleardoubleplainpage
\cleardoubleemptypage
```

Please refer to [section 3.2.2, page 61](#). The function of `\cleardoublepage` in `scrlltr2` depends on the option `cleardoublepage` which is described in more detail in [section 6.2.4, page 149](#).

6.3.3 Variables

Apart from options, commands, environments, counters and lengths, additional elements have already been introduced in KOMA-Script. A typical property of an element is the font style and the option to change it (see [section 3.2.1](#)). At this point we now introduce variables. Variables have a name by which they are called, and they have a content. The content of a variable can be set independently from time and location of the actual usage in the same way as the contents of a command can be separated from

its usage. The main difference between a command and a variable is that a command usually triggers an action, whereas a variable only consists of plain text which is then output by a command. Furthermore, a variable can additionally have a description which can be set and output.

This section specifically only gives an introduction to the concept of variables. The following examples have no special meaning. More detailed examples can be found in the explanation of predefined variables of the letter class in the following sections. An overview of all variables is given in [table 6.12](#).

Table 6.12: Alphabetical list of all supported variables in `scrlettr2`

backaddress	return address for window envelopes (section 6.4.3, page 184)
backaddresseseparator	separator within the return address (section 6.4.3, page 184)
ccseparator	separator between title of additional addressees, and additional addressees (section 6.6.2, page 196)
customer	customer number (section 6.4.5, page 189)
date	date (section 6.4.5, page 189)
emailseparator	separator between e-mail name and e-mail address (section 6.4.1, page 180)
enclseparator	separator between title of enclosure, and enclosures (section 6.6.2, page 196)
faxseparator	separator between title of fax, and fax number (section 6.4.1, page 180)
fromaddress	sender's address without sender name (section 6.4.1, page 179)

Table 6.12: Alphabetical list of all supported variables in `scr1ttr2` (*continued*)

<code>frombank</code>	sender's bank account (section 6.4.7, page 192)
<code>fromemail</code>	sender's e-mail (section 6.4.1, page 179)
<code>fromfax</code>	sender's fax number (section 6.4.1, page 179)
<code>fromlogo</code>	commands for inserting the sender's logo (section 6.4.1, page 179)
<code>fromname</code>	complete name of sender (section 6.4.1, page 179)
<code>fromphone</code>	sender's telephone number (section 6.4.1, page 179)
<code>fromurl</code>	a url of the sender (section 6.4.1, page 179)
<code>invoice</code>	invoice number (section 6.4.5, page 189)
<code>location</code>	more details of the sender (section 6.4.4, page 187)
<code>myref</code>	sender's reference (section 6.4.5, page 189)
<code>place</code>	place (section 6.4.5, page 189)
<code>placeseparator</code>	separator between place and date (section 6.4.5, page 189)
<code>phoneseparator</code>	separator between title of telephone, and telephone number (section 6.4.1, page 180)
<code>signature</code>	signature beneath the ending of the letter (section 6.6.1, page 194)

Table 6.12: Alphabetical list of all supported variables in `scrlltr2` (*continued*)

<code>specialmail</code>	mode of dispatch (section 6.4.3, page 185)
<code>subject</code>	subject (section 6.4.6, page 190)
<code>subjectseparator</code>	separator between title of subject, and subject (section 6.4.6, page 190)
<code>title</code>	letter title (section 6.4.6, page 190)
<code>toname</code>	complete name of addressee (section 6.4.3, page 185)
<code>toaddress</code>	address of addressee without addressee name (section 6.4.3, page 185)
<code>yourmail</code>	date of addressee's mail (section 6.4.5, page 189)
<code>yourref</code>	addressee's reference (section 6.4.5, page 189)

```
\newkomavar[description]{name}
\newkomavar*[description]{name}
\removerefferds
\defaultrefferds
\addtoeffields{name}
```

With `\newkomavar` a new variable is defined. This variable is addressed via *name*. As an option you can define a *description* for the variable *name*. Using the command `\addtoeffields` you can add the variable *name* to the reference fields line (see [section 6.4.5](#)). The *description* and the content of the variable are added at the end of the reference fields line. The starred version `\newkomavar*` is similar to the unstarred version, with a subsequent call of the command `\addtoeffields`. Thus, the starred version automatically adds the variable to the reference fields line.

Example: Suppose you need an additional field for direct dialling. You can define this field either with


```
\newkomavar[Direct dialling]{myphone}
\addtoreffields{myphone}
```

or more concisely with

```
\newkomavar*[direct dialling]{myphone}
```

When you define a variable for the reference fields line you should always give it a description.

With the command `\removeeffields` all variables in the reference line can be removed. This also includes the predefined variables of the class. The reference fields line is then empty except for the date which is always appended to the end. This can be useful, for example, if you wish to change the order of the variables in the reference fields line.

The command `\defaultreffields` acts to reset the reference fields line to its predefined format. In doing so, all custom-defined variables are removed from the reference fields line.

```
\setkomavar{name}[description]{content}
\setkomavar*{name}{description}
```

With the command `\setkomavar` you determine the *content* of the variable *name*. Using an optional argument you can at the same time change the *description* of the variable. In contrast, `\setkomavar*` can only set the *description* of the variable *name*.

Example: Suppose you have defined a direct dialling as mentioned above and you now want to set the content. You write:

```
\setkomavar{myphone}{-\,11}
```

In addition, you want to replace the term “direct dialling” with “Connection”. Thus you add the description:

```
\setkomavar*{myphone}{Connection}
```

or you can combine both in one command:

```
\setkomavar{myphone}[Connection]{-\,11}
```

By the way: You may delete the content of a variable using an empty *content* argument. You can also delete the description using an empty *description* argument.

Example: Suppose you have defined a direct dialling as mentioned above and you now no longer want a description to be set. You write:

```
\setkomavar*{myphone}{}
```

You can combine this with the definition of the content:

```
\setkomavar{myphone}[]{-\,11}
```

So you may setup the content and delete the description using only one command.

```
\usekomavar[command]{name}
\usekomavar*[command]{name}
```

v2.9i

In some cases it is necessary for the user to access the content or the description of a variable, and not to leave this only up to the class. This is specially important when you have defined a variable which is not added to the reference fields line. Using the command `\usekomavar` you have access to the content of the variable *name*, whereas the starred version `\usekomavar*` allows you to access the description.

The commands `\usekomavar` and `\usekomavar*` are, similarly to all commands where a starred version exists or which can take an optional argument, not fully expandable. Nevertheless, if used within `\markboth`, `\markright` or similar commands, you need not insert a `\protect` before using them. Of course this is also true for `\markleft` if using package `scrpage2`. However, these kinds of commands can not be used within commands like `\MakeUppercase` which directly influence their argument. `\MakeUppercase{\usekomavar{name}}` would result in `\usekomavar{NAME}`. To avoid this problem you may use commands like `\MakeUppercase` as an optional argument to `\usekomavar` or `\usekomavar*`. Then you will get the uppercase content of a variable using `\usekomavar[\MakeUppercase]{name}`.

```
\ifkomavareempty{name}{true}{false}
\ifkomavareempty*{name}{true}{false}
```

v2.9i

With these commands you may check whether or not the expanded content or description of a variable is empty. The *true* argument will be executed if the content or description is empty. Otherwise the *false* argument will be executed. The starred variant handles the description of a variable, the unstarred variant handles the contents.

It is important to know that the content or description of the variable will be expanded as far as this is possible with `\edef`. If this results in spaces or

unexpandable macros like `\relax`, the result will be not empty even where the use of the variable would not result in any visible output.

Both variants of the command also must not be used as the argument of `\MakeUppercase` or other commands which have similar effects to their arguments (see the description of `\usekomavar` above for more information about using commands like `\usekomavar` or `\ifkomavareempty` at the argument of `\MakeUppercase`). However, they are robust enough to be used as the argument of, e.g., `\markboth` or `\footnote`.

6.3.4 The Pseudo-Lengths

\TeX works with a fixed number of registers. There are registers for tokens, for boxes, for counters, for skips and for dimensions. Overall there are 256 registers for each of these categories. For \LaTeX lengths, which are addressed with `\newlength`, skip registers are used. Once all these registers are in use, you can not define any more additional lengths. The letter class `scrlltr2` would normally use up more than 20 of such registers for the first page alone. \LaTeX itself already uses 40 of these registers. The `typearea` package needs some of them too; thus, approximately a quarter of the precious registers would already be in use. That is the reason why lengths specific to letters in `scrlltr2` are defined with macros instead of lengths. The drawback of this approach is that computations with macros is somewhat more complicated than with real lengths.

It can be pointed out that the now recommended \LaTeX installation with $\varepsilon\text{-TeX}$ no longer suffers from the above-mentioned limitation. However, that improvement came too late for `scrlltr2`.

A list of all pseudo-lengths in `scrlltr2` is shown in [table 6.13](#) starting at [page 171](#). The meaning of the various pseudo-lengths is shown graphically in [figure 6.1](#). The dimensions used in the figure correspond to the default settings of `scrlltr2`. More detailed description of the individual pseudo-lengths is found in the individual sections of this chapter.

Table 6.13: Pseudo-lengths provided by class `scrlltr2`

<code>foldmarkhpos</code>	horizontal distance of all foldmarks from left paper edge (section 6.4.7, page 192)
---------------------------	-------------------------------------------------------------------------------------------------------

Table 6.13: Pseudo-lengths provided by class `scrlltr2` (*continued*)

<code>tfoldmarkvpos</code>	vertical distance of upper foldmark from top paper edge (section 6.4.7, page 191)
<code>bfoldmarkvpos</code>	vertical distance of lower foldmark from top paper edge (section 6.4.7, page 191)
<code>firstheadvpos</code>	vertical distance of letter-head from top paper edge (section 6.4.1, page 178)
<code>firstheadwidth</code>	width of letter-head; letter-head is centered horizontally on letter paper (section 6.4.1, page 178)
<code>fromrulewidth</code>	length of an optional horizontal rule in letter-head (section 6.4.1, page 180)
<code>toaddrvpos</code>	vertical distance of address field from top paper edge (section 6.4.3, page 183)
<code>toaddrhpos</code>	horizontal distance of address field from left paper edge – for positive values – or negative horizontal distance of address field from right paper edge – for negative values – (section 6.4.3, page 183)
<code>toaddrindent</code>	left and right indentation of address within address field (section 6.4.3, page 184)
<code>toaddrwidth</code>	width of address field (section 6.4.3, page 184)
<code>toaddrheight</code>	height of address field (section 6.4.3, page 183)
<code>backaddrheight</code>	height of return address field on top of address field (section 6.4.3, page 184)

Table 6.13: Pseudo-lengths provided by class `scrLtr2` (*continued*)

<code>specialmailindent</code>	left indentation of mode of dispatch within address field (section 6.4.3 , page 185)
<code>specialmailrightindent</code>	right indentation of mode of dispatch within address field (section 6.4.3 , page 185)
<code>locwidth</code>	width of supplemental data field; for zero value width is calculated automatically with respect to option <code>locfield</code> that is described in section 6.2.6 (section 6.4.4 , page 187)
<code>refvpos</code>	vertical distance of reference fields line from top paper edge (section 6.4.5 , page 188)
<code>refwidth</code>	width of reference fields line (section 6.4.5 , page 188)
<code>refhpos</code>	horizontal distance of reference fields line from left paper edge; for zero value reference fields line is centered horizontally on letter paper (section 6.4.5 , page 188)
<code>refaftervskip</code>	vertical skip below reference fields line (section 6.4.5 , page 189)
<code>sigbeforevskip</code>	vertical skip between closing and signature (section 6.6.1 , page 195)
<code>sigindent</code>	indentation of signature with respect to text body (section 6.6.1 , page 195)
<code>firstfootvpos</code>	vertical distance of letter foot from top paper edge (section 6.4.2 , page 181)

Table 6.13: Pseudo-lengths provided by class `scrlltr2` (*continued*)

firstfootwidth

width of letter foot; letter foot is centered horizontally on letter paper (section 6.4.2, page 181)

\@newlength{*name*}

This command defines an new pseudo-length. This new pseudo-length is uniquely identified by its *name*. If with this command a redefinition of an already existing pseudo-length is attempted, the commands exits with an error message.

Since the user in general does not define own pseudo-lengths, this command is not intended as a user command. Thus, it can not be used within a document, but it can, for example, be used within an lco file.

\@setlength[*factor*]{*pseudo-length*}{*value*}
\@addtoplength[*factor*]{*pseudo-length*}{*value*}

Using the command `\@setlength` you can assign the multiple of a *value* to a *pseudo-length*. The *factor* is given as an optional argument (see also `\setlengthtoplength`). The command `\@addtoplength` adds the *value* to a *pseudo-length*. To assign, or to add the multiple of, one *pseudo-length* to another pseudo-length, the command `\useplength` is used within *value*. To subtract the value of one pseudo-length from another *pseudo-length* a minus sign, or -1, is used as the *factor*.

Since the user in general does not define own pseudo-lengths, this command is not intended as a user command. Thus, it can not be used within a document, but can, for example, be used within an lco file.

\useplength{*name*}

Using this command you can access the value of the pseudo-length with the given *name*. This is one of the few user commands in connection with pseudo-lengths. Of course this command can also be used with an lco file.

\setlengthtoplength[*factor*]{*length*}{*pseudo-length*}
\addtolengthplength[*factor*]{*length*}{*pseudo-length*}

While you can simply prepend a factor to a length, this is not possible with pseudo-lengths. Suppose you have a length `\test` with the value 2pt; then `3\test` gives you the value 6pt. Using pseudo-lengths instead,

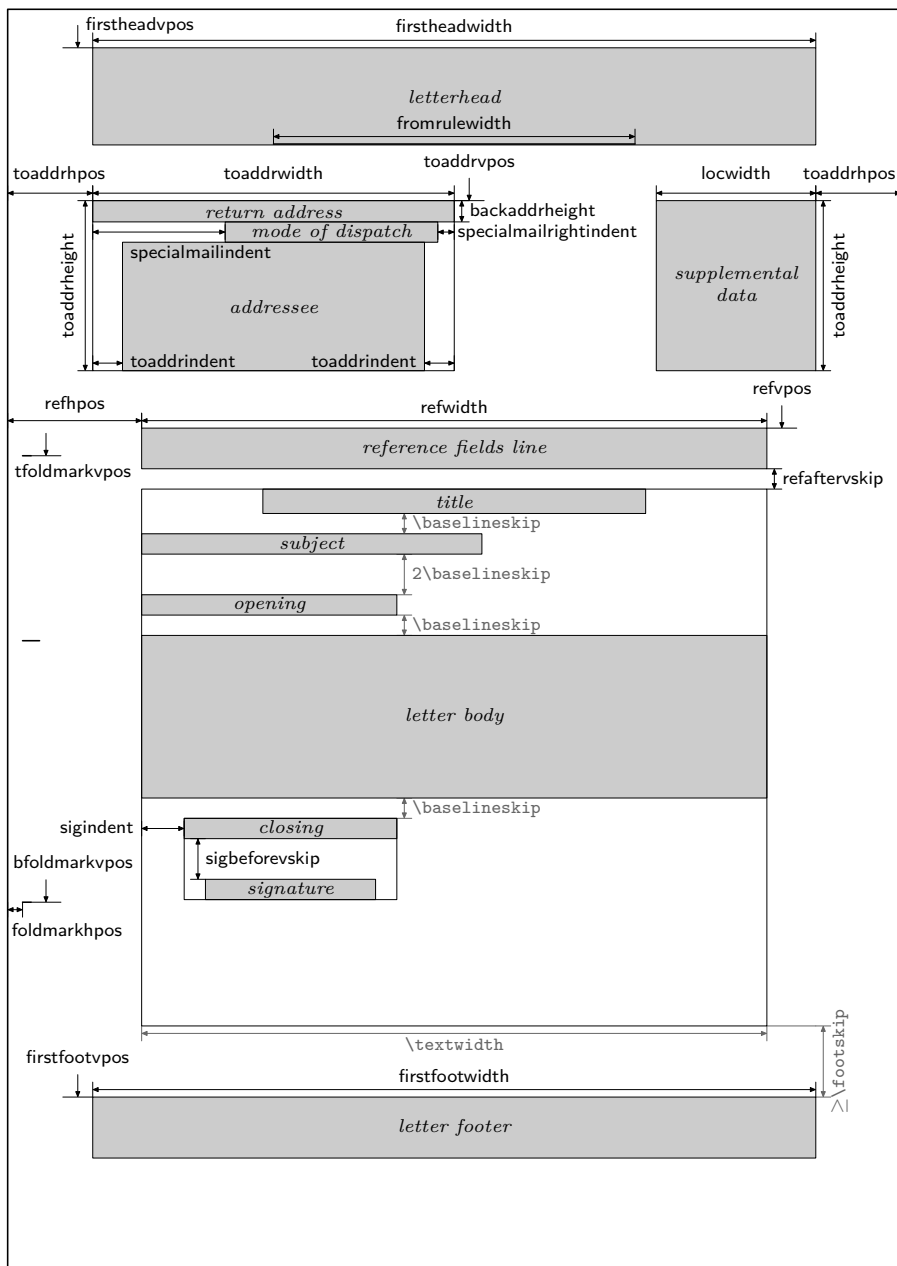


Figure 6.1: Schematic of the pseudo-lengths for a letter

`3\useplength{test}` would give you 32 pt. This is especially annoying if you want a real *length* to take the value of a *pseudo-length*.

Using the command `\setlengthtoplength` you can assign the multiple of a *pseudo-length* to a real *length*. Here, instead of prepending the *factor* to the *pseudo-length*, it is given as an optional argument. You should also use this command when you want to assign the negative value of a *pseudo-length* to a *length*. In this case you can either use a minus sign or `-1` as the *factor*. The command `\addtolengthplength` works very similarly; it adds the multiple of a *pseudo-length* to the *length*.

6.3.5 The General Structure of a Letter Document

The general structure of a letter document differs somewhat from the structure of a normal document. Whereas a book document in general contains only one book, a letter document can contain several letters. As illustrated in [figure 6.2](#), a letter document consists of a preamble, the individual letters, and the closing.

The preamble comprises all settings that in general concern all letters. Most of them can also be overwritten in the settings of the individual letters. The only setting which can not be changed within a single letter is the font size (see option `fontsize` in [section 6.2.5, page 152](#)). It is recommended that only general settings such as the loading of packages and the setting of options be placed before `\begin{document}`. All settings that comprise the setting of variables or other text features should be done after `\begin{document}`. This is particularly recommended when the `babel` package (see [[Bra01](#)]) is used, or language-dependent variables of `scr1tr2` are to be changed.

The closing usually consists only of `\end{document}`. Of course you can also insert additional comments at this point.

As shown in [figure 6.3](#), every single letter itself consists of an introduction, the letter body, and the closing. In the introduction, all settings pertaining only to the current letter are defined. It is important that this introduction always ends with `\opening`. Similarly, the closing always starts with `\closing`. The two arguments *opening* and *closing* can be left empty, but both commands must be used and must have an argument.

It should be noted that several settings can be changed between the individual letters. Such changes then have an effect on all subsequent letters. For reasons of maintainability of your letter documents, it is however not recommended to use further general settings with limited scope between the letters.

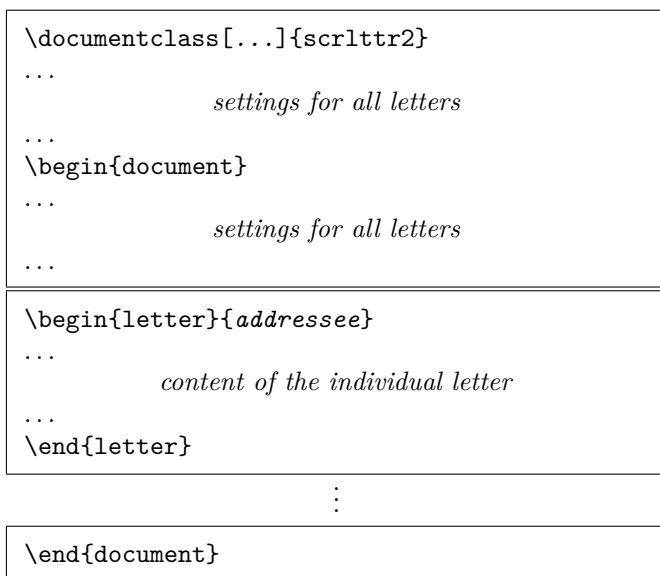


Figure 6.2: General structure of a letter document with several individual letters (the structure of a single letter is shown in [figure 6.3](#))

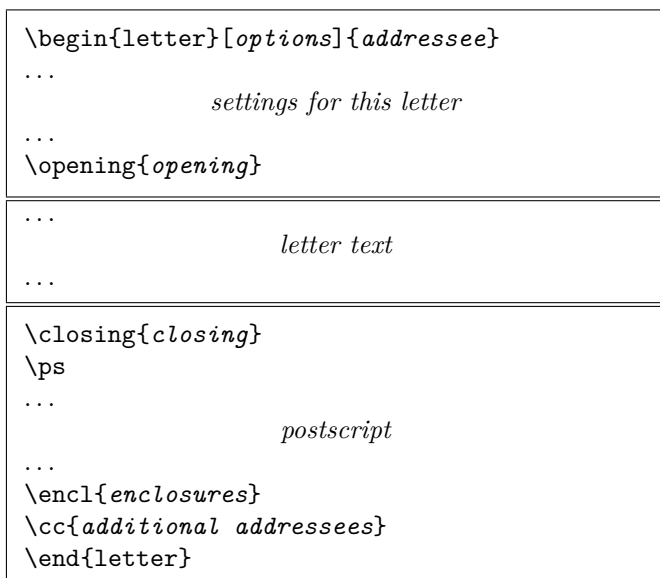


Figure 6.3: General structure of a single letter within a letter document (see also [figure 6.2](#))

As already mentioned, all general settings used in the preamble of a letter document, with the exception of font size, can also be in the preamble of the individual letters. Therefore, you will not find more detailed explanations for the possible settings in this section. Please refer to [section 6.4](#).

6.4 The Letter Declaration

The letter declaration gives all settings for the letter itself, as well as for the first page of the body. The first page consists of more than just the preliminaries of the letter; in fact, it consists of several different parts.

6.4.1 The Letterhead

The term letterhead here refers to all of the data pertaining to the sender and which is set above the addressee's address. It is usually expected that this information is set via the page style settings. In fact, this was the case in the earlier incarnation of the letter class, `scrlettr`. But with `scrlettr2`, the letterhead is made independent of the page style setting, and is set by the command `\opening`. The position of the letterhead is absolute and independent of the type area. In fact, the first page of a letter, the page that holds the letterhead, is set using the page style `empty`.

`firstheadvpos`

The pseudo-length `firstheadvpos` gives the distance between the top edge of the paper and start of the letterhead. This value is set differently in the various predefined `lco` files. A typical value is 8 mm.

`firstheadwidth`

The pseudo-length `firstheadwidth` gives the width of the letterhead. This value is set differently in the various predefined `lco` files. While this value usually depends on the paper width and the distance between the left edge of the paper and the addressee address field, it was the type area width in `KOMAold`.

Table 6.14: The sender's predefined labels for the letterhead

<code>fromemail</code>	<code>\usekomavar*{emailseparator}\usekomavar{emailseparator}</code>
<code>fromfax</code>	<code>\usekomavar*{faxseparator} \usekomavar{faxseparator}</code>
<code>fromname</code>	<code>\headfromname</code>
<code>fromphone</code>	<code>\usekomavar*{phoneseparator}\usekomavar{phoneseparator}</code>
<code>fromurl</code>	<code>\usekomavar*{urlseparator}\usekomavar{urlseparator}</code>

<code>fromname</code>
<code>fromaddress</code>
<code>fromphone</code>
<code>fromfax</code>
<code>fromemail</code>
<code>fromurl</code>
<code>fromlogo</code>

These variables give all information concerning the sender necessary to create the letterhead. Which variables will actually be used to finally build the letterhead can be chosen by use of the letterhead extensions (see option `fromalign` in [section 6.2.6, page 152](#)) and the options given there. The variables `fromname`, `fromaddress` and `fromlogo` will be set in the letterhead without their labels; the variables `fromphone`, `fromfax`, `fromemail` and `fromurl` will be set with their labels. The labels are shown in [table 6.14, page 179](#).

An important hint concerns the sender's address: within the sender's address, parts such as street, P.O. Box, state, country, etc., are separated with a double backslash. Depending on how the sender's address is used, this double backslash will be interpreted differently and therefore is not strictly always a line break. Paragraphs, vertical spacing and the like are usually not allowed within the sender's address declaration. One has to have very good knowledge of `scrlltr2` to use things like those mentioned above, intelligently. Another point to note is the one should most certainly set the variables for return address (see [section 6.4.3, page 184](#)) and

signature (see [section 6.6.1](#), [page 194](#)) oneself.

It is possible, by the way, to load an external picture to use as a logo. For this purpose one can put as content of `fromlogo` an `\includegraphics` command. Naturally, the corresponding package, that is, either `graphics` or `graphicx` (see [[Car99b](#)]), has to be loaded in the preamble of the letter document (see [section 6.3.5](#)).

`fromrulewidth`

Depending on the class option `fromrule` (see [section 6.2.6](#), [page 152](#)), a horizontal rule can be drawn the predefined letterheads under or within the sender address. If the pseudo-length `fromrulewidth` has a value of 0pt, which is the default in the predefined `lco` files, the rule length is calculated automatically taking into account, e.g., letterhead width or an optional logo. Users can adjust rule length manually in their own `lco` files by setting this pseudo-length to positive values using `\setplength` (see [section 6.3.4](#), [page 174](#)).

`phoneseparator`
`faxseparator`
`emailseparator`
`urlseparator`

With these variables, hyphens are defined. If applicable, they are used in the sender's data in the letterhead (see [table 6.14](#)). As a feature, they are labeled and the labels also used in the sender's details of the letterhead. To look up the predefined labels and their contents, see [table 6.15](#).

`\firsthead{construction}`

For most cases, `scrlltr2` with its options and variables offers enough possibilities to create a letterhead. In very rare situations one may wish to have more freedom in terms of layout. In those situations one will have to do without predefined letterheads, which could have been chosen via options. Instead, one needs to create one's own letterhead from scratch. To do so, one has to define the preferred *construction* with the command `\firsthead`. Within `\firsthead`, and with the help of the `\parbox` command (see [[Tea01](#)]), one can set several boxes side by side, or one underneath the other. An advanced user will thus be able to create a letterhead on his own. Of course the *construct* may use variables with the help of `\usekomavar`.

Table 6.15: predefined labels and contents of hyphens for sender's data in the letterhead

name	label	content
emailseparator	\emailname	:~
faxseparator	\faxname	:~
phoneseparator	\phonename	:~
urlseparator	\wwwname	:~

6.4.2 The Footer

As the first page holds a letterhead of its own, it also holds a footer of its own. And, as with the letterhead, it will not be set by the page style settings, but directly with the use of `\opening`.

firstfootvpos

This pseudo-length gives the distance between the footer and the upper edge of the paper. This value is set differently in the various predefined lco files. It also takes care of preventing text from jutting into the footer area. If needed, it can help to shorten the text height on the first page using `\enlargethispage`. Likewise, and if it is needed, the text height can conversely be extended with the help of the option `enlargefirstpage`. This way, the distance between text area and the first page's footer can be reduced to the value `\footskip`. See also [section 6.2.3, page 148](#).

2.9t

With the compatibility option set up to version 2.9t (see [version](#) in [section 6.2.2, page 146](#)) the footer is set independently of the type area in all predefined lco files (see [section 6.2.8](#)) except for KOMAold. The option `enlargefirstpage` also loses its effect. From version 2.9u onwards the footer is set in a position at the bottom edge of the paper. In this situation, the height of the type area also becomes dependent on `enlargefirstpage`.

firstfootwidth

This pseudo-length gives the width of the letter's first page footer. The value is set equal to that of the pseudo-length `firstheadwidth` in the predefined lco files.

\firstfoot{construction}

The first page's footer is preset to empty. However, with the `\firstfoot` command, it is possible to create a *construction* the same way as when defining the letterhead with `\firsthead`.

Example: In the first page's footer, you may want to set the content of the variable `frombank` (the bank account). The double backslash should be exchanged with a comma at the same time:

```
\firstfoot{%
  \parbox[b]{\linewidth}{%
    \centering\def\\{, }\usekomavar{frombank}%
  }%
}
```

For the hyphen you might define a variable of your own if you like. This is left as an exercise for the reader.

Nowadays it has become very common to create a proper footer in order to obtain some balance with respect to the letterhead. This can be done as follows:

```
\firstfoot{%
  \parbox[t]{\textwidth}{\footnotesize
    \begin{tabular}[t]{l@{}}%
      \multicolumn{1}{@{}l@{}}{Partners:}\\
      Jim Smith\\
      Russ Mayer
    \end{tabular}%
    \hfill
    \begin{tabular}[t]{l@{}}%
      \multicolumn{1}{@{}l@{}}{Manager:}\\
      Jane Fonda\\[1ex]
      \multicolumn{1}{@{}l@{}}{Court Of ←
        Jurisdiction:}\\
      Great Plains
    \end{tabular}%
    \ifkomavareempty{frombank}{-}{%
      \hfill
      \begin{tabular}[t]{l@{}}%
        \multicolumn{1}{@{}l@{}}{\usekomavar*{←
          frombank:}}\\
        \usekomavar{frombank}
      \end{tabular}%
    }%
  }%
}
```

This example, by the way, came from Torsten Krüger. With

```
\setkomavar{frombank}{Account No. ←  
12\,345\,678\  
at Citibank\  
bank code no: 876\,543\,21}
```

the bank account can be set accordingly. If the footer will have such a large height then it might happen that you have to shift its position. You can do this with the pseudo-length `firstfootvpos`, which is described above in this section.

In the previous example a multi-line footer was set. With a compatibility setting to version 2.9u (see `version` in [section 6.2.2, page 146](#)) the space will in general not suffice. In that case, you may need to reduce `firstfootvpos` (see [page 181](#)) appropriately.

6.4.3 The Address

The term address here refers to the addressee's name and address which are output in an address field. Additional information can be output within this address field, such as dispatch type or a return address; the latter is especially useful when using window envelopes. The address directly follows the letterhead.

`toaddrvpos`
`toaddrhpos`

These pseudo-lengths define vertical and horizontal position of the address field relative to the top-left corner of the paper. Values are set differently in the various predefined `lco` files, according to standard envelope window measures. A special feature of `toaddrhpos` is that with negative values the offset is that of the right edge of the address field relative to the right edge of the paper. This can be found, for instance, in the case of `SN`. The smallest value of `toaddrvpos` is found with `DINmtext`. Care must be taken to avoid overlap of letterhead and address field. Whether the address field is output or not can be controlled by class option `addrfield` (see [section 6.2.6, page 155](#)).

`toaddrheight`

The pseudo-length `toaddrheight` defines the height of the address field, including the dispatch type. If no dispatch type is specified, then the

address is vertically centered in the field. If a dispatch type is specified, then the address is set below the dispatch type, and vertically centered in the remaining field height.

toaddrwidth

The pseudo-length `toaddrwidth` defines the width of the address field. Values are set differently in the various predefined `lco` files, according to standard envelope window measures. Typical values are between 70 mm and 100 mm.

Example: Assume that your printer has a very wide left or right margin of 15 mm. In this case, when using the option `SN`, the letterhead, sender's extensions and the address can not be completely printed. Thus, you create a new `lco` file with the following content:

```
\ProvidesFile{SNmmarg.lco}
[2002/06/04 v0.1 my own lco]
\LoadLetterOption{SN}
\@addtoplength{toaddrwidth}{%
  -\useplength{toaddrhpos}}
\@setplength{toaddrhpos}{-15mm}
\@addtoplength{toaddrwidth}{%
  \useplength{toaddrhpos}}
\endinput
```

Then, until you can obtain a printer with smaller page margins, you simply use the option `SNmmarg` instead of `SN`.

toaddrindent

Additional indentation of the address within address field can be controlled by the pseudo-length `toaddrindent`. Its value applies to both left and right margin. Default value is 0 pt.

`backaddress`
`backaddressseparator`
`backaddrheight`

When using window envelopes, the sender's return address is often included within the window, placed at the top above the addressee and dispatch type information, separated by a horizontal rule and set in a smaller font size. The contents of the return address, stored in the variable `backaddress`, are

usually built automatically from the variables `fromname` and `fromaddress`. Within the return address, double backslashes are replaced by the content of the variable `backaddresseseparator`, whose default value is a comma followed by a non-breaking space.

The height reserved for the return address within the address field is defined by the pseudo-length `backaddrheight`. In the predefined `lco` files, this is typically set to 5 mm. Whether the return address is output or not is controlled by document class options `addrfield` (see [section 6.2.6, page 155](#)) and `backaddress` (see [section 6.2.6, page 155](#)).

```
specialmail
specialmailindent
specialmailrightindent
```

An optional dispatch type can be output within the address field between the return address and the addressee address, by setting the variable `specialmail`. Left and right alignment are determined by pseudo-lengths `specialmailindent` and `specialmailrightindent`, respectively. In the predefined `lco` files provided by KOMA-Script, `specialmailindent` is set to rubber length `\fill`, while `specialmailrightindent` is set to 1 em. Thus the dispatch type is set 1 em from the address field's right margin.

```
toname
toaddress
```

These two variables contain the addressee's name and address as output in the address field. Usually you will not access these variables directly, but their values are taken from the argument to the `letter` environment. Please see the important hint on address formatting given in [section 6.4.1, page 179](#).

```
letter[options]{addressee}
```

The `letter` environment is only one of the key environments of the letter class. A special `scr1tr2` feature are optional arguments to the `letter` environment. These *options* are executed internally via the `\KOMAOPTIONS` command.

The *addressee* is a mandatory argument passed to the `letter` environment. Parts of the addressee contents are separated by double backslashes. The first part of *addressee* is stored in variable `toname`, while the rest is stored in variable `toaddress` for further use. These parts are output on individual lines in the address field. Nevertheless, the double backslash should not be interpreted as a certain line break. Vertical material such

as paragraphs or vertical space is not permitted within *addressee*, and could lead to unexpected results and error messages, as is the case also for the standard letter class.

The `letter` environment does not actually start the letter output. This is done by the `\opening` command.

`\AtBeginLetter{commands}`

L^AT_EX enables the user to declare *commands* whose execution is delayed until a determined point. Such points are called *hooks*. Known macros for using hooks are `\AtBeginDocument` and `\AtEndOfClass`. The letter class `scrlettr2` provides an additional hook that can be used via the macro `\AtBeginLetter`. Originally, hooks were provided for package and class authors, so they are documented in [Tea99] only, and not in [Tea01]. However, with letters there are useful applications of `\AtBeginLetter` as the following example may illustrate:

Example: It is given that one has to set multiple letters with questionnaires within one document. Questions are numbered automatically within single letters using a counter. Since, in contrast to page numbering, that counter is not known by `scrlettr2`, it would not be reset at the start of each new letter. Given that each questionnaire contains ten questions, question 1 would get number 11 in the second letter. A solution is to reset this counter at the beginning of each new letter:

```
\newcounter{Question}
\newcommand{\Question}[1]{%
  \refstepcounter{Question}\par
  \@hangfrom{\makebox[2em][r]{\theQuestion←
    :~}}{#1}}
\AtBeginLetter{\setcounter{Question}{0}}
```

This way question 1 remains question 1, even in the 1001st letter. Of course definitions like those mentioned above need to be stated either between macros `\makeatletter` and `\makeatother` (see [RNH02]) in letter declarations (see [section 6.3.5](#) and [figure 6.2, page 177](#)), in a unique package, or in an `lco` file (see [section 6.2.8](#)).

6.4.4 The Sender's Extensions

Often, especially with business letters, the space for the letterhead or page footer seems to be too tight to include all you want. To give more details about the sender, often the space right beside the addressee's field is used. In this manual this field is called the *sender's extension*

locwidth

This pseudo-length `locwidth` declares the width of the sender's extension field. Its value is typically set to 0 pt in the predefined `lco` files. This value takes on a special meaning: it does not mean that the sender's extension has no width; instead the actual width is set with `\opening` when the paper width, address window width, and the distance between the left edge of the paper and the address window are known. The option `locfield` (see [section 6.2.6, page 156](#)) is also taken into account.

location

The contents of the sender's extension field is determined by the variable `location`. To set this variable's content, it is permitted to use formatting commands like `\raggedright`. One has to consider that depending on the use of the options `fromalign` and `fromlogo`, a part of the space for the sender's extension may already be reserved for a logo (see [section 6.2.6, page 152](#) and [page 155](#)).

Example: Assume that you would like to put the names of your partners, manager, or court of jurisdiction in the sender's extension field. You can do this as follows:

```
\KOMAOptions{locfield=wide}
\setkomavar{location}{\raggedright
  \textbf{Partners:}\
  \quad Hugo Mayer\
  \quad Bernd Miller\|[1ex]
  \textbf{Manager:}\
  \quad Liselotte Mayer\|[1ex]
  \textbf{Court of jurisdiction:}\
  \quad Washington, DC
}
```

The option `locfield=wide` is set to make the details fit horizontally. Sender details like those mentioned in the above

example can be written, together with the common sender address details, into your own `lco` file.

6.4.5 The Reference Fields Line

Especially with business letters, a line can be found that gives initials, dial code, customer number, invoice number, or a reference to a previous letter. In this manual this line is called the *reference fields line*. The reference fields line can consist of more than just one line and is set only if one of those variables mentioned above is given. Only those fields will be set that are given. To set a seemingly empty field, one needs to give as value at least a white space or `\null`. If you want to have your letter without a reference fields line, then instead of it the label and contents of the variable `date` will be set.

`refvpos`

This pseudo-length gives the distance between the upper edge of the paper and the reference fields line. Its value is set differently in the various predefined `lco` files. Typical values are between 80.5 mm and 98.5 mm.

`refwidth`
`refhpos`

This pseudo-length gives the width that is available for the reference fields line. The value is set typically to 0 pt in the predefined `lco` files. This value has a special meaning: in no way does it determine that there is no available width for the business line; instead, this value means that the width will be calculated with the `\opening`. Thus the calculated width depends on the determination of the options `refline` (see [section 6.2.6, page 157](#)). At the same time, `refhpos` will be set according to this option. With `refline=wide`, the reference fields line is centered, while with `refline=narrow` it is aligned on the left.

If `refwidth` non-null, i. e., the width of the reference fields line is therefore not determined by the option `refline`, then `refhpos` gives the distance of the reference fields line from the left edge of the paper. If this distance is null, then the reference fields line is set so that the ratio between its distances from the left and right edges of the paper equal the ratio of distance of the type area from the left and right edges of the paper. Thus, for a type area horizontally centered on the paper, the reference fields line too will be centered.

Table 6.16: predefined labels of typical variables of the reference fields line. The content of the macros depend on language.

name	label	in english
<code>yourref</code>	<code>\yourrefname</code>	Your reference
<code>yourmail</code>	<code>\yourmailname</code>	Your letter from
<code>myref</code>	<code>\myrefname</code>	Our reference
<code>customer</code>	<code>\customername</code>	Customer No.:
<code>invoice</code>	<code>\invoicename</code>	Invoice No.:
<code>date</code>	<code>\datename</code>	date

As a rule, these special cases are likely to be of little interest to the normal user. The simplest rule is as follows: either `refhpos` is left at null and so the width and alignment of the reference fields line are left to the option `refline`, or `refwidth` as well as `refhpos` are set by the user.

refaftervskip

This pseudo-length gives the vertical space that has to be inserted beneath the reference fields line. The value is set in the predefined `lco` files. It directly affects the text height of the first page. A typical value lies between one and two lines.

```
yourref
yourmail
myref
customer
invoice
date
```

These variables are typical reference fields. Their meanings are given in [table 6.12](#) on [page 166](#). Each variable has also a predefined label, shown in [table 6.16](#). The field width that belongs to each variable, adjusts itself automatically to its label and content.

```
place
placeseparator
```

As said before in the introduction of this subsection, the reference fields line can be omitted. This happens if all variables of the business line are empty with the exception of the variable for the date. In this case, the content of `place` and `placeseparator` will be set, followed by the content of `date`.

The predefined content of the `placeseparator` is a comma followed by a non-breaking space. If the variable `place` has no value then the hyphen remains unset also. The predefined content of `date` is `\today` and depends on the setting of the option `numericaldate` (see [section 6.2.6, page 157](#)).

6.4.6 The Title and the Subject Line

Business letters very often carry a subject line. The subject line indicates briefly the respect of the letter. Usually the subject should be short and precise and not run across several lines. Apart from the subject, such a letter may also carry a title. Titles find usage most often with irregular letters such as a warning, an invoice or a reminder.

`title`

With `scrlltr2` a letter can carry an additional title. The title is centered and set with font size `\LARGE` directly after and beneath the reference fields line. The predefined font setup for this element (`\normalcolor\sffamily\bfseries`) can be changed with help of the interface described in [section 3.2.1](#). Font size declarations are allowed.

Example: Assume that you are to write a reminder. Thus you put as title:

```
\setkomavar{title}{Reminder}
```

This way the addressee will recognize a reminder as such.

`subject`
`subjectseparator`

In case a subject should be set, the contents of the variable `subject` need to be defined. Depending on what the option `subject` is set to, a label can be placed in front of the subject contents; also, the vertical position of the subject contents can be changed (see [section 6.2.6, page 155](#)). The predefined labels are shown in [table 6.17](#). The predefined value of `subjectseparator` is a colon followed by a non-breaking space.

The subject line is set in a separate font. To change this use the user interface described in [section 3.2.1](#). For the element `subject` the predetermined font in `scrlltr2` is `\normalfont\normalcolor\bfseries`.

Example: Assume you are a board member and want to write a letter to another member of that board about a few internals of the organization. You want to clarify with your subject line what

Table 6.17: Predefined labels of subject-related variables.

name	label
<code>subject</code>	<code>\usekomavar*{subjectseparator}% \usekomavar{subjectseparator}</code>
<code>subjectseparator</code>	<code>\subjectname</code>

this letter is all about, but without labeling it thus. You can do this as follows:

```
\setkomavar{subject}[Subject ]{%
    organization's internals}
```

or easier:

```
\setkomavar{subject}[]{%
    about organization's internals}
```

Furthermore, if you want to set the subject line not only in bold but also in sans serif:

```
\addtokomafont{subject}{\sffamily}
```

As you can see, it is really easy to solve such problems.

6.4.7 Further Settings

In this paragraph variables and settings are listed which could not be assigned to any other part of the letter declaration but somehow belong to this section.

`tfoldmarkvpos`
`bfoldmarkvpos`

The letterclass `scr1tr2` knows about 3 fold marks. The one in the middle serves to halve the paper and is therefore printed always in the middle of the paper height. The position of the upper fold mark, seen from the upper edge of the paper, is determined by the pseudo-length `tfoldmarkvpos`; the lower one is determined by the pseudo-length `bfoldmarkvpos`. These two fold marks do not serve to exactly fold into three equally-sized panels. Instead the idea is to have the paper folded so that the address field is seen in the window of a window envelope. The settings therefore are different

in the various predefined `lco` files. A special case is `DINmtext`. In this case the envelope format C6/5 (also called “C6 long”) is necessary. Letters produced in this format are not compatible with envelopes in either C5 or C4 formats.

`foldmarkhpos`

This pseudo-length gives the distance between all three fold marks and the left edge of the paper. Usually this is 3.5 mm. In case you work with a printer with a broader unprintable left margin, this value can be changed in your own `lco` files. The length of the upper and lower fold marks is identically 2 mm, while the center fold mark is 4 mm long. The thickness of all three is 2 pt. At the moment there are no plans to change these value. If the fold marks will be set at all depends on the option `foldmarks` (see [section 6.2.6, page 156](#)).

`frombank`

This variable at the moment takes on a special meaning: it is not used internally at this point, and the user can make use of it to set, for example, his bank account within the sender’s extension field or the footer.

`\nexthead{construction}` `\nextfoot{construction}`

The possibilities that are offered with variables and options in `scrlettr2` should be good enough in most cases to create letterheads and footers for those pages that follow the first letter page. Even more so since you can additionally change with `\markboth` and `\markright` the sender’s statements that `scrlettr2` uses to create the letterhead. The term “subsequent pages” in this manual refers to all pages following the first letter page. The commands `\markboth` and `\markright` can in particular be used together with `pagestyle myheadings`. If the package `scrpage2` is used then this, of course, is valid also for `pagestyle scrheadings`. There the command `\markleft` is furthermore available.

At times one wants to have more freedom with creating the letterhead or footer of subsequent pages. Then one has to give up the possibilities of predefined letterheads or footers that could have been chosen via the option `pagenumber` (see [section 6.2.4, page 150](#)). Instead one is free to create the letterhead and footer of subsequent pages just the way one wants to have them set. For that, one creates the desired letterhead or footer *construction* using the command `\nexthead` or `\nextfoot`, respectively. Within `\nexthead` and `\nextfoot` you can, for example, have several boxes

side by side or one beneath the other by use of the `\parbox` command (see [Tea01]). A more advanced user should have no problems creating letterheads or footers of his own. Within *construction* you can of course also make use of the variables by using `\usekomavar`.

6.5 The Text

In contrast to an article, a report or a book, a letter normally has no chapter or section structure. Even float environments with tables and figure are unusual. Therefore, a letter has no table of contents, lists of figures and tables, index, bibliography, glossary or similar things. The letter text mainly consists of an opening and the main text. Thereupon follow the signature, a postscript and various listings.

6.5.1 The Opening

In the early days of computer-generated letters, programs did not have many capabilities, therefore the letters seldom had an opening. Today the capabilities have been enhanced. Thus personal openings are very common, even in mass-production advertising letters.

`\opening{opening}`

This is one of the most important commands in `scrlettr2`. For the user it may seem that only the opening is typeset, but the command also typesets the folding marks, letterhead, address field, reference fields line, subject, the page footer and others. In short, without `\opening` there is no letter.

6.5.2 Footnotes

In letters footnotes should be used more sparingly than in normal documents. However, `scrlettr2` is equipped with all mechanisms mentioned in [section 3.6.3](#) for the main document classes. Therefore they will not be discussed here again.

6.5.3 Lists

Lists have the same validity in letters as in normal documents. Thus `scrlettr2` provides the same possibilities as mentioned in [section 3.6.4](#) for the main document classes. Therefore they will not be discussed here again.

6.5.4 Margin Notes

Margin notes are quite uncommon in letters. Therefore the option `mpinclude` is not actively supported by `scrlettr2`. However, `scrlettr2` is equipped with all mechanisms mentioned in [section 3.6.5](#) for the main document classes. Therefore they will not be discussed here again.

6.5.5 Text Emphasis

The distinction of text has the same importance in letters as in other documents. Thus the same rules apply, meaning: emphasize text sparingly. Even letters should be readable and a letter where each word is typeset in another font is quite unreadable.

The class `scrlettr2` is equipped with all mechanisms mentioned in [section 3.6.7](#) for the main document classes. Therefore it will not be discussed here again.

6.6 The Closing Part

A letter always ends with a closing phrase. Even computer-generated letters without signature have such a phrase. Sometimes this is a sentence such as, “This letter has been generated automatically and is valid without a signature.”. Sometimes a sentence like this will even be used as a signature. Thereupon can follow a postscript and various listings.

6.6.1 Closing

The closing consists of three parts: besides the closing phrase there are a hand-written inscription and the signature, which acts as an explanation for the inscription.

signature

The variable `signature` holds an explanation for the inscription. The content is predefined as `\usekomavar{fromname}`. The explanation may consist of multiple lines. The lines should then be separated by a double backslash. Paragraphs in the explanation are however not permitted.

`\closing{closing phrase}`

The command `\closing` not only typesets the closing phrase, but also the content of the variable `signature`. The closing phrase may consists of multiple lines, but paragraphs are not permitted.

```
sigindent
sigbeforevskip
\raggedsignature
```

Closing phrase and signature will be typeset in a box. The width of the box is determined by the length of the longest line of the closing phrase or signature.

The box will be typeset with indentation of the length set in pseudo-length `sigindent`. In the predefined `lco` files this length is set to 0 mm.

The command `\raggedsignature` defines the alignment inside the box. In the predefined `lco` files the command is either defined as `\centering` (all besides KOMAold) or `\raggedright` (KOMAold). In order to obtain flush-right or flush-left alignment inside the box, the command can be redefined in the same way as `\raggedsection` (see [section 3.6.2, page 79](#)).

Between closing phrase and signature a vertical space is inserted, the height of which is defined in the pseudo-length `sigbeforevskip`. In the predefined `lco` files this is set to 2 lines. In this space you can then write your inscription.

Example: You are writing as the directorate of a society a letter to all members. Moreover, you want on the one hand to elucidate that you are writing in the name of the board of directors, and on the other hand you want indicate your position on the board of directors.

```
\setkomavar{signature}{John McEnvy\\
  {\small (Vice-President ‘‘The Other Society’’)\leftarrow
  }}
\closing{Regards\\
  (for the board of directors)}
```

You can of course set the variable `signature` in your private `lco` files. Otherwise it is advisable to define the variable in the letter preamble (see [section 6.4](#)).

6.6.2 Postscript, Carbon Copy and Enclosures

After the closing can follow some other statements. Besides the postscript, there are the distribution list of carbon copies, and the reference to enclosures.

\ps

In the time when letters were written by hand it was quite common to use a postscript because this was the only way to add information which one had forgotten to mention in the main part of the letter. Of course, in letters written with \LaTeX you can insert additional lines easily. Nevertheless, it is still popular to use the postscript. It gives one a good possibility to underline again the most important or sometimes the less important things of the particular letter.

This instruction merely switches to the postscript. Hence, a new paragraph begins, and a vertical distance—usually below the signature—is inserted. The command `\ps` is followed by normal text. If you want the postscript to be introduced with the acronym “PS:”, which by the way is written without a full stop, you have to type this yourself. The acronym is typeset neither automatically nor optionally by the class `scrletter2`.

`\cc{distribution list}`
`ccseparator`

With the command `\cc` it is possible to typeset a *distribution list*. The command takes the *distribution list* as its argument. If the content of the variable `ccseparator` is not empty, then the name and the content of this variable is inserted before *distribution list*. In this case the *distribution list* will be indented appropriately. It is a good idea to set the *distribution list* `\raggedright` and to separate the individual entries with a double backslash.

Example: You want to indicate that your letter is sent to all members of a society and to the board of directors:

```
\cc{%
  the board of directors\\
  all society members}
```

This instruction should be written below the `\closing` instruction from the previous example, or below a possible postscript.

A vertical space is inserted automatically before the distribution list.

`\encl{enclosures}`
`enclseparator`

Enclosures have the same structure as the distribution list. The only difference is that here the enclosures starts with the name and content of the variable `enclseparator`.

6.7 Language Support

The document class `scrlltr2` supports many languages. These include German (`german` for old German orthography, `ngerman` for the new orthography, and `austrian` for Austrian), English (`english` without specification as to whether American or British should be used, `american` and `USenglish` for American, and `british` and `UKenglish` for British), French, Italian, Spanish, Dutch, Croatian and Finnish.

6.7.1 Language Selection

If the package `babel` (see [Bra01]) is used, one can switch between languages with the command `\selectlanguage{language}`. Other packages like `german` (see [Rai98a]) and `ngerman` (see [Rai98b]) also define this command. As a rule though, the language selection takes place already as a direct consequence of loading such a package. Further information can be obtained in the documentation of the relevant packages.

There is one thing more to mention about language packages. The package `french` (see [Gau03]) redefines not only the terms of section 6.7.2, but also other, for instance it even redefines the command `\opening`, since it assumes that the definition of the standard letter is used. With `scrlltr2` this is not the case, therefore the package `french` destroys the definition in `scrlltr2` and does not work correctly with KOMA-Script. The author views this is a fault in the `french` package.

If one utilizes the `babel` package in order to switch to language `french` while the package `french` is simultaneously installed, then the same problems will likely occur, since `babel` employs definitions from the `french` package. If the package `french` is not installed then there are no problems. Aimilarly, there is no problem if for `babel` instead of `french` other languages like `acadian`, `canadien`, `francais` or `frenchb` are chosen.

From `babel` version 3.7j this problem only occurs when it is indicated explicitly by means of an option that `babel` should use the `french` package.

If it cannot be ascertained that a new version of `babel` is being used, it is recommended to use

```
\usepackage[...frenchb,...]{babel}
```

in order to select `french`.

Other languages can possibly cause similar problems. Currently there are no known problems with the `babel` package for the german language and the various english language selections.

```

\captionsenglish
\captionUSenglish
\captionamerican
\captionbritish
\captionUKenglish
\captionsgerman
\captionsgerman
\captionsaustrian
\captionsfrench
\captionssitalian
\captionsspanish
\captionsdutch
\captionscroatian
\captionsfinnish

```

If one switches the language then using these commands the language-dependent terms from [section 6.7.2](#) are redefined. If the used language selection scheme does not support this then the commands above can be used directly.

```

\dateenglish
\dateUSenglish
\dateamerican
\datebritish
\dateUKenglish
\dategerman
\datengerman
\dateaustrian
\datefrench
\dateitalian
\datespanish
\datedutch
\datecroatian
\datefinnish

```

The numerical representation of the date (see option `numericaldate` in [section 6.2.6](#)) will be written depending on the selected language. Some examples can be found in [table 6.18](#).

6.7.2 Language-Dependent Terms

As is usual in L^AT_EX, the language-dependent terms are defined by commands which are then redefined when one switches the language.

Table 6.18: Language-dependent forms of the date

Command	Date example
<code>\dateenglish</code>	1/12/1993
<code>\dateUSenglish</code>	12/1/1993
<code>\dateamerican</code>	12/1/1993
<code>\datebritish</code>	1/12/1993
<code>\dateUKenglish</code>	1/12/1993
<code>\dategerman</code>	1. 12. 1993
<code>\datengerman</code>	1. 12. 1993
<code>\dateaustrian</code>	1. 12. 1993
<code>\datefrench</code>	1. 12. 1993
<code>\dateitalian</code>	1. 12. 1993
<code>\datespanish</code>	1. 12. 1993
<code>\datedutch</code>	1. 12. 1993
<code>\datecroatian</code>	1. 12. 1993.
<code>\datefinnish</code>	1. 12. 1993.

```

\yourrefname
\yourmailname
\myrefname
\customername
\invoicename
\subjectname
\ccname
\enclname
\headtoname
\headfromname
\datename
\pagename
\phonename
\faxname
\emailname
\wwwname
\bankname

```

The commands above contain the language-dependent terms. These definitions can be modified in order to support a new language or for private customization. How this can be done is described in [section 6.7.3](#). The definitions become active only at `\begin{document}`. Therefore they are not available in the \LaTeX preamble and cannot be redefined there. In

Table 6.19: Default settings for languages `english` and `ngerman`

Command	<code>english</code>	<code>ngerman</code>
<code>\bankname</code>	Bank account	Bankverbindung
<code>\ccname</code> ¹	cc	Kopien an
<code>\customername</code>	Customer no.	Kundennummer
<code>\datename</code>	Date	Datum
<code>\emailname</code>	Email	E-Mail
<code>\enclname</code> ¹	encl	Anlagen
<code>\faxname</code>	Fax	Fax
<code>\headfromname</code>	From	Von
<code>\headtoname</code> ¹	To	An
<code>\invoicename</code>	Invoice no.	Rechnungsnummer
<code>\myrefname</code>	Our ref.	Unser Zeichen
<code>\pagename</code> ¹	Page	Seite
<code>\phonename</code>	Phone	Telefon
<code>\subjectname</code>	Subject	Betrifft
<code>\wwwname</code>	Url	URL
<code>\yourmailname</code>	Your letter of	Ihr Schreiben vom
<code>\yourrefname</code>	Your ref.	Ihr Zeichen

¹Normally these terms are defined by language packages like `babel`. In this case they are not redefined by `scrlettr2` and may differ from the table above.

table 6.19 the default settings for `english` and `ngerman` can be found.

6.7.3 Defining and Changing Language-dependent Terms

Normally one has to change or define the language-dependent terms of **section 6.7.1** in such a way that in addition to the available terms the new or redefined terms are defined. This is made more difficult by the fact that some packages like `german` or `ngerman` redefine those settings when the packages are loaded. This definitions unfortunately occurs in such a manner as to destroy all previous private settings. That is also the reason why `scrlettr2` delays its own changes, with `\AtBeginDocument` until `\begin{document}`, that is, after package loading is completed. The user can also use `\AtBeginDocument`, or redefine the language-dependent terms after `\begin{document}`, that is, not put them in the preamble at all. The class `scrlettr2` even provides some additional commands for defining language-dependent terms.


```

\providecaptionname{language}{term}{definition}
\newcaptionname{language}{term}{definition}
\renewcaptionname{language}{term}{definition}

```

Using one of the commands above, the user can assign a *definition* for a particular *language* to a *term*. The *term* is always a macro. The commands differ dependent on whether a given *language* or a *term* within a given *language* are already defined or not at the time the command is called.

If *language* is not defined, then `\providecaptionname` does nothing other than writes a message in the log file. This happens only once for each language. If *language* is defined but *term* is not yet defined for it, then it will be defined using *definition*. The *term* will not be redefined if the *language* already has such a definition; instead, an appropriate message is written to the log file.

The command `\newcaptionname` has a slightly different behaviour. If the *language* is not yet defined, then a new language command (see [section 6.7.1](#)) will be created and a message written to the log file. If *term* is not yet defined in *language*, then it will be defined using *definition*. If *term* already exists in *language*, then this results in an error message.

The command `\renewcaptionname` again behaves differently. It requires an existing definition of *term* in *language*. If neither *language* nor *term* exist or *term* is unknown in a defined language then a error message will be given. Otherwise, the *term* for *language* will be redefined according to *definition*.

The class `scrlltr2` itself employs `\providecaptionname` in order to define the commands in [section 6.7.2](#).

Example: If you prefer “Your message of” instead of “Your letter of”, you have to redefine the definition of `\yourmailname`.

```

\renewcaptionname{english}{\yourmailname}{%
  Your message of}

```

Since only existing terms in available languages can be redefined, you have to put the command after `\begin{document}` or delay the command by using `\AtBeginDocument`. Furthermore, you will get an error message if there is no package used that switches language selection to *english*.

6.8 Address Files and Circular Letters

When people write circular letters one of the more odious tasks is the typing of many different addresses. The class `scr1ttr2`, as did its predecessor `scrlettr` as well, provides basic support for this task. Currently there are plans for much enhanced support.

```
\adrentry{Lastname}{Firstname}{Address}{Phone}{F1}{F2}{Comment}{Key}
```

The class `scr1ttr2` supports the use of address files which contain address entries, very useful for circular letters. The file extension of the address file has to be `.adr`. Each entry is an `\adrentry` command with eight parameters, for example:

```
\adrentry{McEnvy}
      {Flann}
      {Main Street 1\\ Glasgow}
      {123 4567}
      {male}
      {}
      {niggard}
      {FLANN}
```

The 5th and 6th elements, `F1` and `F2`, can be used freely: for example, for the gender, the academic grade, the birthday, or the date on which the person joined a society. The last parameter *Key* should only consist of uppercase letters in order to not interfere with existing $\text{T}_{\text{E}}\text{X}$ or $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ commands.

Example: Mr. McEnvy is one of your most important business partners, but every day you receive correspondence from him. Before long you do not want to bother typing his boring address again and again. Here `scr1ttr2` can help. Assume that all your business partners have an entry in your `partners.adr` address file. If you now have to reply to Mr. McEnvy again, then you can save typing as follows:

```
\input{partners.adr}
\begin{letter}{\FLANN}
  Your correspondence of today \dots
\end{letter}
```

Your \TeX system must be configured to have access to your address file. Without access, the \input command results in an error. You can either put your address file in the same directory where you are running \LaTeX , or configure your system to find the file in a special directory.

 $\text{\address}{\textit{Lastname}}{\textit{Firstname}}{\textit{Address}}{\textit{Phone}}{\textit{F1}}{\textit{F2}}{\textit{F3}}{\textit{F4}}{\textit{Key}}$

Over the years people have objected that the \address has only two free parameters. To cater to this demand, there now exists a new command called \addreentry —note the additional “d”—which supports four freely-definable parameters. Since \TeX supports maximally nine parameters per command, the comment parameter has fallen away. Other than this difference, the use is the same as that of \address .

Both \address and \addreentry commands can be freely mixed in the **adr** files. However, it should be noted that there are some packages which are not suited to the use of \addreentry . For example, the **adrconv** by Axel Kielhorn can be used to create address lists from **adr** files, but it has currently no support for command \addreentry . In this case, the only choice is to extend the package yourself.

Besides the simple access to addresses, the address files can be easily used in order to write circular letters. Thus, there is no requirement to access a complicated database system via \TeX .

Example: Suppose you are member of a society and want write an invitation for the next general meeting to all members.

```

\documentclass{scr1ttr2}
\begin{document}
\renewcommand*{\address}[8]{
  \begin{letter}{#2 #1\#3}
    \opening{Dear members,} Our next general ↵
      meeting will be on
      Monday, August 12, 2002. The following ↵
      topics are \dots
    \closing{Regards,}
  \end{letter}
}
\input{members.adr}
\end{document}

```

If the address file contains `\addrentry` commands too, than an additional definition for `\addrentry` is required before loading the address file:

```
\renewcommand*{\addrentry}[9]{%
  \adrentry{#1}{#2}{#3}{#4}{#5}{#6}{#7}{#9}%
}
```

In this simple example the extra freely-definable parameter is not used, and therefore `\addrentry` is defined with the help of `\adrentry`.

With some additional programming one can let the content of the letters depend on the address data. For this the free parameters of the `\adrentry` and `\addrentry` commands can be used.

Example: Suppose the 5th parameter of the `\adrentry` command contains the gender of a member (m/f), and the 6th parameter contains what amount of subscription has not yet been paid by the member. If you would like to write a more personal reminder to each such member, then the next example can help you:

```
\renewcommand*{\adrentry}[8]{
  \ifdim #6pt>0pt\relax
  % #6 is an amount greater than 0.
  % Thus, this selects all members with due ←
    subscription.
  \begin{letter}{#2 #1\|#3}
    \if #5m \opening{Dear Mr.\|#2,} \fi
    \if #5f \opening{Dear Mrs.\|#2,} \fi
```

```

    Unfortunately we have to remind you that ←
      you have
    still not paid the member subscription for←
      this
    year.
```

```

    Please remit EUR #6 to the account of the ←
      society.
```

```

  \closing{Regards,}
\end{letter}
```

```

\fi
}

```

As you can see, the letter text can be made more personal by depending on attributes of the letter's addressee. The number of attributes is only restricted by number of two free parameters of the `\adrentry` command, or four free parameters of the `\addrchar` command.

```

\adrchar{initial letter}
\addrchar{initial letter}

```

As already mentioned above, it is possible to create address and telephone lists using `adr` files. For that, the additional package `adrconv` by Axel Kielhorn (see [Kie99]) is needed. This package contains interactive L^AT_EX documents which help to create those lists.

The address files have to be sorted already in order to obtain sorted lists. It is recommended to separate the sorted entries at each different initial letter of *Lastname*. As a separator, the commands `\adrchar` and `\addrchar` can be used.. These commands will be ignored if the address files are utilized in `scrletter2`.

Example: Suppose you have the following short address file:

```

\adrchar{A}
\adrentry{Angel}{Gabriel}
    {Cloud 3\\12345 Heaven's Realm}
    {000\\,01\\,02\\,03}{-}{-}{archangel}{←
    GABRIEL}
\adrentry{Angel}{Michael}
    {Cloud 3a\\12345 Heaven's Realm}
    {000\\,01\\,02\\,04}{-}{-}{archangel}{←
    MICHAEL}
\adrchar{K}
\adrentry{Kohm}{Markus}
    {Freiherr-von-Drais-Stra\\ss e 66\\68535←
    Edingen-Neckarhausen}
    {+49~62\\,03~1\\,??\\,??}{-}{-}{no angel at ←
    all}
    {KOMA}

```

This address file can be treated with `adrdirex.tex` of the `adrconv` package [Kie99]. The result should look like this:

			A
<hr/>			
ANGEL, Gabriel			
Cloud 3			
12345 Heaven's Realm	GABRIEL		
(archangel)	000 01 02 03		
ANGEL, Michael			
Cloud 3a			
12345 Heaven's Realm	MICHAEL		
(archangel)	000 01 02 04		

The letter in the page header is created by the `\adrchar` command. The definition can be found in `adrdir.tex`.

More about the `adrconv` package can be found in its documentation. There you should also find information about whether the current version of `adrconv` supports the `\addrentry` and `\addrchar` commands. Former versions only know the commands `\adrentry` and `\adrchar`.

6.9 From `scrlettr` to `scrlttr2`

The first step in the conversion of an old letter written with the `scrlettr` class is to load the appropriate `lco` file using option `KOMAold` at `\documentclass`. Thereupon most commands of the old class should work. However, you will encounter some differences in the output, since the page layout of the old class is not realized exactly. The reason is that the calculation of the type area in `scrlettr` has some minor bugs. For example, the position of the folding marks used to depend on the height of the page header, which again depended on the font size. That was unambiguously a design error.

There is no compatibility regarding the defined lengths in `scrlettr`. Thus, if a user has changed the page layout of `scrlettr`, then the relevant statements should be deleted or commented out. In some cases, the modification of a length can cause an error, since this length is no longer defined in `scrlttr2`. The user should delete or comment out such modifications as well.

After the switch from `scrlettr` to `scrlttr2`, the old letter example can be successfully compiled already alone through the setting of the option `KOMAold`:

```

\documentclass[10pt,KOMAold]{scrlettr2}
\name{{\KOMAScript} team}
\address{Class Alley 1\\12345 {\LaTeX} City}
\signature{Your {\KOMAScript} team}
\begin{document}
  \begin{letter}{ {\KOMAScript} users\\
                  Everywhere\\world-wide}
    \opening{Dear {\KOMAScript} users,}
    the {\KOMAScript} team is proud to announce \dots
    \closing{Happy {\TeX}ing}
  \end{letter}
\end{document}

```

The next step is that while wanting the old commands to be still be available, the layout of the old letter should no longer be used. If, for example, one wants to sue the layout of the letter class option DIN, then this option can be given in `\documentclass`, but it has to be specified *after* the option `KOMAold`:

```

\documentclass[10pt,KOMAold,DIN]{scrlettr2}
\name{{\KOMAScript} team}
\address{Class Alley 1\\12345 {\LaTeX} City}
\signature{Your {\KOMAScript} team}
\begin{document}
  \begin{letter}{ {\KOMAScript} users\\
                  Everywhere\\world-wide}
    \opening{Dear {\KOMAScript} users,}
    the {\KOMAScript} team is proud to announce \dots
    \closing{Happy {\TeX}ing}
  \end{letter}
\end{document}

```

By using more options this way, you can further influence the layout. However, the author recommends a more inherent change right away.

The last step is to replace all old commands with their new equivalents, and to omit the option `KOMAold`. For this task, it may help to read the contents of `KOMAold.lco`. In that file the old commands are defined using the new commands and variables.

```

\documentclass{scrlettr2}
\setkomavar{fromname}{ {\KOMAScript} team}
\setkomavar{fromaddress}{Class Alley 1\\

```

```

12345 {\LaTeX} City}
\setkomavar{signature}{Your {\KOMAScript} team}
\let\raggedsignature=\raggedright
\begin{document}
  \begin{letter}{ {\KOMAScript} users\
    Everywhere\
    world-wide}
    \opening{Dear {\KOMAScript} users,}
    the {\KOMAScript} team is proud to announce \dots
    \closing{Happy {\TeX}ing}
  \end{letter}
\end{document}

```

This example shows also the possibility to change the alignment of the signature by redefining the command `\raggedsignature`. This is always recommended when the width of the explanation of the signature as defined by the command `\setkomavar{signature}{...}` is greater than the width of the argument of `\closing`.

Access to Address Files with scraddr

7.1 Overview

The package `scraddr` is a small extension to the KOMA-Script letter class. Its aim is to make access to the data of address files more flexible and easier. Basically, the package implements a new loading mechanism for address files which contain address entries in the form of `\adrentry` and newer `\addrentry` commands, as described in the previous chapter.

```
\InputAddressFile{file name}
```

The command `\InputAddressFile` is the main command of the `scraddr`, and reads the content of the address file given as its parameter. If the file does not exist the command returns an error message.

For every entry in the address file the command generates a set of macros for accessing the data. For large address files this will take a lot of T_EX memory.

```
\adrentry{Lastname}{Firstname}{Address}{Phone}{F1}{F2}{Comment}{Key}
\addrentry{Lastname}{Firstname}{Address}{Phone}{F1}{F2}{F3}{F4}{Key}
\adrchar{initial}
\addrchar{initial}
```

The structure of the address entries in the address file was discussed in detail in [section 6.8](#) from [page 202](#) onwards. The division of the address file with the help of `\adrchar` or `\addrchar`, also discussed therein, has no meaning for `scraddr` and is simply ignored.

The commands for accessing the data are given by the name of the data field they are intended for.

```
\Name{Key}
\FirstName{Key}
\LastName{Key}
\Address{Key}
\Telephone{Key}
\FreeI{Key}
\FreeII{Key}
\Comment{Key}
\FreeIII{Key}
\FreeIV{Key}
```

These commands give access to data of your address file. The last parameter, i. e., parameter 8 for the `\adrentry` entry and parameter 9 for the `\addrentry` entry, is the identifier of an entry, thus the *Key* has to be unique and non-blank. The *Key* should only be composed of letters.

If the file contains more than one entry with the same *Key* value, the last occurrence will be used.

7.2 Usage

First of all, we need an address file with valid address entries. In this example the file has the name `lotr.adr` and contains the following entries.

```
\addrentry{Baggins}{Frodo}%
    {The Hill\\ Bag End/Hobbiton in the Shire}{}%
    {Bilbo Baggins}{pipe-weed}%
    {the Ring-bearer}{Bilbo's heir}{FRODO}
\adrentry{Gamgee}{Samwise}%
    {Bagshot Row 3\\Hobbiton in the Shire}{}%
    {Rosie Cotton}{taters}%
    {the Ring-bearer's faithful servant}{SAM}
\adrentry{Bombadil}{Tom}%
    {The Old Forest}{}%
    {Goldberry}{trill queer songs}%
    {The Master of Wood, Water and Hill}{TOM}
```

The 4th parameter, the telephone number, has been left blank. If you know the story behind these addresses you will agree that a telephone number makes no sense here, and besides, it should simply be possible to leave them out. The command `\InputAddressFile` is used to load the address file shown above:

```
\InputAddressFile{lotr}
```

With the help of the commands introduced in this chapter we can now write a letter to old TOM BOMBADIL. In this letter we ask him if he can remember two fellow-travelers from Elder Days.

```
\begin{letter}{\Name{TOM}\\\Address{TOM}}
  \opening{Dear \FirstName{TOM} \LastName{TOM},}
```

```
  or \FreeIII{TOM}, how your delightful \FreeI{TOM} ←
    calls you. Can
```

```

you remember Mr.\,\LastName{FRODO}, strictly speaking
\Name{FRODO}, since there was Mr.\,\FreeI{FRODO} too. ←
    He was
\Comment{FRODO} in the Third Age and \FreeIV{FRODO} \←
    Name{SAM},
\Comment{SAM}, has attended him.

```

```

Their passions were very worldly. \FirstName{FRODO} ←
    enjoyed
smoking \FreeII{FRODO}, his attendant appreciated a ←
    good meal with
\FreeII{SAM}.

```

```

Do you remember? Certainly Mithrandir has told you ←
    much
    about their deeds and adventures .
\closing{'‘0 spring-time and summer-time
        and spring again after!\\
        0 wind on the waterfall,
        and the leaves' laughter!'''}
\end{letter}

```

In the address of letters often both firstname and lastname are required. Thus, the command `\Name{Key}` is an abridgement for `\FirstName{Key}` `\LastName{Key}`.

The 5th and 6th parameters of the `\adrenentry` or `\adrenentry` commands are for free use. They are accessible with the commands `\FreeI` and `\FreeII`. In this example, the 5th parameter contains the name of a person who is the most important in the life of the entry's person, the 6th contains the person's passion. The 7th parameter is a comment or in general also a free parameter. The commands `\Comment` or `\FreeIII` give access to this data. Use of `\FreeIV` is only valid for `\addrenentry` entries; for `\adrenentry` entries it results in an error. More on this is covered in the next section.

7.3 Package Warning Options

As mentioned above, the command `\FreeIV` leads to an error if it is used for `\adrenentry` entries. How `scraddr` reacts in such a situation is decide by package options.

```

adrFreeIVempty
adrFreeIVshow
adrFreeIVwarn
adrFreeIVstop

```

These four options allow the user to choose between *ignore* and *rupture* during the \LaTeX run if \FreeIV has been used with an \adrenentry entry.

adrFreeIVempty – the command \FreeIV will be ignored

adrFreeIVshow – “(entry \FreeIV undefined at *Key*)” will be written as warning in the text

adrFreeIVwarn – writes a warning in the logfile

adrFreeIVstop – the \LaTeX run will be interrupted with an error message

To choose the desired reaction, one of these options can be given in the optional argument of the \usepackage command. The default setting is **adrFreeIVshow**.

Creating Address Files from a Address Database

In former versions of KOMA-Script the package `addrconv` was a permanent part of the KOMA-Script system. The chief involvement with KOMA-Script was that with the help of `addrconv` it was possible from an address database in `BIBTEX` format to create address files compatible with the KOMA-Script letter class or with the package `scraddr`.

```
@address{HMUS,
  name = {Carl McExample},
  title = {Dr.},
  city = {Anywhere},
  zip = 01234,
  country = {Great Britain},
  street = {A long Road},
  phone = {01234 / 5 67 89},
  note = {always forget his birthday},
  key = {HMUS},
}
```

From entries such as that given above, address files can be generated. For this `addrconv` employs `BIBTEX` and various `BIBTEX` styles. Additionally, there are some `LATEX` files which can help to create various telephone and address lists for printing.

However, the package `addrconv` was actually an independent package, since besides what is required for KOMA-Script it includes several more interesting features. Therefore, the package `addrconv` has for some time already been removed from the KOMA-Script system. The package `addrconv`, with a single *d*, entirely replaces `addrconv`. If it is not included in your `TEX` distribution then it can be downloaded from [Kie99] and you can install it separately.

Control Package Dependencies with `scrfile`

The introduction of \LaTeX 2_ε in 1994 brought many changes in the handling of \LaTeX extensions. Today the package author has many macros available to determine if another package or class is employed and whether specific options are used. The author can load other packages or can specify options in the case that the package is loaded later. This has led to the expectation that the order in which package are loaded would not be important. Sadly this hope has not been fulfilled.

9.1 About Package Dependencies

More and more frequently, different packages either newly define or redefine the same macro again. In such a case the order in which a package is loaded becomes very important. For the user it sometimes becomes very difficult to understand the behaviour, and in some cases the user wants only to react to the loading of a package. This too is not really a simple matter.

Let us take the simple example of loading the package `longtable` with a KOMA-Script document class. The `longtable` package defines table captions very well suited to the standard classes, but the captions are totally unsuitable for documents using KOMA-Script and also do not react to the options of the provided configuration commands. In order to solve this problem, the `longtable` package commands which are responsible for the table captions need to be redefined. However, by the time the `longtable` package is loaded, the KOMA-Script class has already been processed.

Until the present, the only way for KOMA-Script to solve this problem was to delay the redefinition until the beginning of the document with help of the macro `\AtBeginDocument`. If the user wants to change the definitions too, it is recommended to do this in the preamble of the document. However, this is impossible since later at `\begin{document}` KOMA-Script will again overwrite the user definition with its own. Therefore, the user too has to delay his definition with `\AtBeginDocument`.

Actually, KOMA-Script should not need to delay the redefinition until `\begin{document}`. It would be enough to delay exactly until the package `longtable` has been loaded. Unfortunately, the \LaTeX kernel does not define appropriate commands. The package `scrfile` provides redress here.

Likewise, it might be conceivable that before a package is loaded one would like to save the definition of a macro in a help-macro, in order to restore its

meaning after the package has been loaded. The package `scrfile` allows this, too.

The employment of `scrfile` is not limited to package dependencies only. Even dependencies on any other file can be considered. For example, the user can be warned if the not uncritical file `french.ldf` has been loaded.

Although the package is particularly of interest for package authors, there are of course applications for normal \LaTeX users, too. Therefore, this chapter gives and explains examples for both groups of users.

9.2 Actions Prior to and After Loading

`scrfile` can execute actions both before and after the loading of files. In the commands used to do this, distinctions are made between general files, classes, and packages.

```
\BeforeFile{file}{instructions}
\AfterFile{file}{instructions}
```

The macro `\BeforeFile` ensures that *instructions* are only executed before the next time *file* is loaded. `\AfterFile` works in a similar fashion, and the *instructions* will be executed only after the *file* has been loaded. If *file* is never loaded then the *instructions* will never be executed.

In order to implement those features `scrfile` redefines the well known \LaTeX command `\InputIfFileExists`. If this macro does not have the expected definition then `scrfile` issues a warning. This is for the case that in future \LaTeX versions the macro can have a different definition, or that another package has already redefined it.

The command `\InputIfFileExists` is used by \LaTeX every time a file is to be loaded. This is independent of whether the actual load command is `\include`, `\LoadClass`, `\documentclass`, `\usepackage`, `\RequirePackage`, or similar. Exceptionally, the command

```
\input foo
```

loads the file `foo` without utilizing `\InputIfFileExists`. Therefore, one should always use

```
\input{foo}
```

instead. Notice the parentheses surrounding the file name!

```
\BeforeClass{class}{instructions}
\BeforePackage{package}{instructions}
```

These two commands work in the same way as `\BeforeFile`. The only difference is that the document class *class* and the L^AT_EX package *package* are specified with their names and not with their file names. That means that the file extensions `.cls` and `.sty` can be omitted.

```
\AfterClass{class}{instructions}
\AfterClass*{class}{instructions}
\AfterPackage{package}{instructions}
\AfterPackage*{package}{instructions}
```

The commands `\AfterClass` and `\AfterPackage` work in the same way as `\AfterFile`. The only difference is that the document class *class* and the L^AT_EX package *package* are specified with their names and not with their file names. That means that the file extensions `.cls` and `.sty` can be omitted. The starred versions execute the *instructions* not only at next time that the class or package is loaded, but also immediately if the class or package has been loaded already.

Example: In the following, an example for class and package authors shall be given. It shows how KOMA-Script itself employs the new commands. The class `scrbook` contains:

```
\AfterPackage{hyperref}{%
  \@ifpackagelater{hyperref}{2001/02/19}{%
    \ClassWarningNoLine{scrbook}{%
      You are using an old version of hyperref ↵
      package!%
    \MessageBreak%
    This version has a buggy hack at many ↵
    drivers%
    \MessageBreak%
    causing \string\addchap\space to behave ↵
    strange.%
    \MessageBreak%
    Please update hyperref to at least version
    6.71b}}}
```

Old versions of the `hyperref` package redefine a macro of the `scrbook` class in such a way that does not work with newer KOMA-Script versions. New versions of `hyperref` desist from

making these changes if a new KOMA-Script version is detected. For the case that `hyperref` is loaded at a later stage, therefore, the code in `scrbook` verifies that a acceptable `hyperref` version is used. If not, the command issues a warning.

At other places in three KOMA-Script classes the following can be found:

```
\AfterPackage{caption2}{%
  \renewcommand*{\setcapindent}{%
```

After the package `caption2` has been loaded, and only if it has been loaded, KOMA-Script redefines its own command `\setcapindent`. The exact code of the redefinition is not important. It should only be noted that `caption2` takes control of the `\caption` macro and that therefore the normal definition of the `\setcapindent` macro would become ineffective. The redefinition improves the collaboration with `caption2`.

There are however also useful examples for normal \LaTeX user. Suppose a document that should be available as a PS file, using \LaTeX and `dvips`, as well as a PDF file, using `pdf\text{\LaTeX}`. In addition, the document should contain hyperlinks. In the list of tables there are entries longer than one line. This is not a problem for the `pdf\text{\LaTeX}` method, since here hyperlinks can be broken across multiple lines. However, if a `hyperref` driver for `dvips` or `hyper\text{\LaTeX}` is used then this is not possible. In this case one desires that for the `hyperref` setup `linktocpage` is used. The decision which `hyperref` driver to use happens automatically via `hyperref.cfg`. The file has, for example, the following content:

```
\ProvidesFile{hyperref.cfg}
\@ifundefined{pdfoutput}{\ExecuteOptions{dvips←
  }}
                                {\ExecuteOptions{pdftex}}
\endinput
```

All the rest can now be left to `\AfterFile`.

```
\documentclass{article}
\usepackage{scrfile}
\AfterFile{hdvips.def}{\hypersetup{linktocpage←
```

```

    }}
\AfterFile{hypertex.def}{\hypersetup{↵
    linktocpage}}
\usepackage{hyperref}
\begin{document}
\listoffigures
\clearpage
\begin{figure}
  \caption{This is an example for a fairly long↵
    figure caption, but
    which does not employ the optional caption ↵
    argument that would
    allow one to write a short caption in the ↵
    list of figures.}
\end{figure}
\end{document}

```

If now the `hyperref` drivers `hypertex` or `dvips` are used, then the useful `hyperref` option `linktocpage` will be set. In the `pdfLATEX` case, the option will not be set, since in that case another `hyperref` driver, `hpdftex.def`, will be used. That means neither `hdvips.def` nor `hypertex.def` will be loaded.

Furthermore, the loading of package `scrfile` and the `\AfterFile` statement can be written in a private `hyperref.cfg`. If you do so, then instead of `\usepackage` the macro `\RequirePackage` ought be used (see [Tea99]). The new lines have to be inserted directly after the `\ProvidesFile` line, thus immediately prior to the execution of the options `dvips` or `pdftex`.

Change Log

At this list of changes you will find all significant changes of the user interface of the KOMA-Script bundle at the last few versions. The list was sorted about the names of the classes and packages and their version. The numbers behind the versions are the pages, where the changes are described. At the margins of these pages you will find corresponding version marks.

scrbook , scrreprt , scrartcl

2.96a	79
v2.96a	71

scrbook, scrreprt

v2.8o	82
v2.8p	85
v2.8q	85
v2.96a	49

scrbook, scrreprt, scrartcl

v2.8p	53, 58, 66, 74, 77, 78, 93, 105
v2.8q	28, 48, 49, 88, 89, 103, 108
v2.96a	39

scrllttr2

2.9t	181
v2.8q	145
v2.9i	170
v2.9t	146

scrpage2

2.2	124, 128
-----------	----------

Bibliography

In the following you can find many references. All of them are referenced from the main text. In many cases the reference points to documents or directories which can be accessed via Internet. In these cases the reference includes a URL instead of a publisher. If the reference points to a \LaTeX package then the URL is written in the form “[CTAN://destination](#)”. The prefix “[CTAN://](#)” means the \TeX archive on a CTAN server or mirror. For example, you can substitute the prefix with <ftp://ftp.ctan.org/tex-archive/>. For \LaTeX packages it is also important to mention that we have tried to give a version number appropriate to the text that cites the reference. But for some packages it is very difficult to find a consistent version number and release date. Additionally the given version is not always the current version. If you want install new packages take care that the package is the most up-to-date version and check first whether the package is already available on your system or not.

- [Bra01] Johannes Braams:
Babel, a multilingual package for use with \LaTeX 's standard document classes, February 2001.
[CTAN://macros/latex/required/babel/](#).
- [Car98] David Carlisle:
The longtable package, May 1998.
[CTAN://macros/latex/required/tools/](#).
- [Car99a] David Carlisle:
The ifthen package, September 1999.
[CTAN://macros/latex/base/](#).
- [Car99b] David P. Carlisle:
Packages in the ‘graphics’ bundle, February 1999.
[CTAN://macros/latex/required/graphics/](#).
- [Dal99] Patrick W. Daly:
Natural sciences citations and references, May 1999.
[CTAN://macros/latex/contrib/natbib/](#).
- [DUD96] DUDEN:
Die deutsche Rechtschreibung. DUDENVERLAG, Mannheim, 21. edition, 1996.
- [Fai99] Robin Fairbairns:

- topcapt.sty*, March 1999.
[CTAN://macros/latex/contrib/misc/topcapt.sty](http://www.ctan.org/ctan/macros/latex/contrib/misc/topcapt.sty).
- [FAQ] *Tex frequently asked questions on the web*.
<http://www.tex.ac.uk/faq/>.
- [Gau03] Bernard Gaulle:
Les distributions de fichiers de francisation pour latex,
 December 2003.
[CTAN://language/french/](http://www.ctan.org/ctan/language/french/).
- [KDP] *KOMA-Script homepage*.
<http://www.komascript.de>.
- [Kie99] Axel Kielhorn:
adrconv, November 1999.
[CTAN://macros/latex/contrib/adrconv/](http://www.ctan.org/ctan/macros/latex/contrib/adrconv/).
- [Kil99] James Kilfiger:
extsizes, a non standard L^AT_EX-package, November 1999.
[CTAN://macros/latex/contrib/extsizes/](http://www.ctan.org/ctan/macros/latex/contrib/extsizes/).
- [Lin01] Anselm Lingnau:
An improved environment for floats, July 2001.
[CTAN://macros/latex/contrib/float/](http://www.ctan.org/ctan/macros/latex/contrib/float/).
- [Mit00] Frank Mittelbach:
An environment for multicolumn output, July 2000.
[CTAN://macros/latex/required/tools/](http://www.ctan.org/ctan/macros/latex/required/tools/).
- [OPHS99] Tobias Oetker, Hubert Partl, Irene Hyna, and Elisabeth Schlegl:
The Not So Short Introduction to L^AT_EX 2_ε, April 1999.
[CTAN://info/lshort/](http://www.ctan.org/ctan/info/lshort/).
- [Rah01] Sebastian Rahtz:
Hypertext marks in L^AT_EX: the hyperref package, February 2001.
[CTAN://macros/latex/contrib/hyperref/](http://www.ctan.org/ctan/macros/latex/contrib/hyperref/).
- [Rai98a] Bernd Raichle:
german package, July 1998.
[CTAN://language/german/](http://www.ctan.org/ctan/language/german/).
- [Rai98b] Bernd Raichle:
ngerman package, July 1998.
[CTAN://language/german/](http://www.ctan.org/ctan/language/german/).

- [RNH02] Bernd Raichle, Rolf Niepraschk, and Thomas Hafner:
De-TEX-/dante-faq, May 2002.
<http://www.dante.de/faq/de-tex-faq/>.
- [Sch03] Martin Schröder:
The Ragged2e package, January 2003.
CTAN://macros/latex/contrib/ms/.
- [Som04] Harald Axel Sommerfeldt:
caption package, July 2004.
CTAN://macros/latex/contrib/caption/.
- [SU03] Tom Sgouros and Stefan Ulrich:
The mparhack package, May 2003.
CTAN://macros/latex/contrib/mparhack/.
- [Tea99] L^AT_EX3 Project Team:
L^AT_EX 2_ε for class and package writers, March 1999.
CTAN://macros/latex/doc/clsguide.pdf.
- [Tea00] L^AT_EX3 Project Team:
L^AT_EX 2_ε font selection, September 2000.
CTAN://macros/latex/doc/fntguide.pdf.
- [Tea01] L^AT_EX3 Project Team:
L^AT_EX 2_ε for authors, July 2001.
CTAN://macros/latex/doc/usrguide.pdf.
- [Tob00] Geoffrey Tobin:
setspace L^AT_EX package, December 2000.
CTAN://macros/latex/contrib/setspace/.
- [Tsc87] Jan Tschichold:
Ausgewählte Aufsätze über Fragen der Gestalt des Buches und der Typographie. Birkhäuser Verlag, Basel, 2 edition, 1987.
- [Ume00] Hideo Umeki:
The geometry package, June 2000.
CTAN://macros/latex/contrib/geometry/.
- [vO00] Piet van Oostrum:
Page layout in L^AT_EX, October 2000.
CTAN://macros/latex/contrib/fancyhdr/.
- [WF00] Hans Peter Willberg and Friedrich Forssman:
Erste Hilfe in Typografie. Verlag Hermann Schmidt, Mainz, 2000.

Index

There are two kinds of page numbers at this index. The bold printed numbers show the pages of declaration or explanation of the topic. The normal printed numbers show the pages of using a topic.

General Index

A

address **152–157, 183–186**
 database 213
 file **202–206, 209, 213**
 list **205**
 aphorism 85
 appendix 45, 82, 114
 author **66**

B

back matter 72
 bibliography 73, 114, 116
 binding correction 41, 148
 boxed (float style) 106

C

caption
 of figure 101
 of table 101
 chapter **42, 82**
 title **44**
 circular letters **202–206**
 citations 96
 class
 → Index of Files etc. 231
 CM fonts 76
 color
 in footer 128
 in header 128
 columns 41
 command
 → Index of Commands etc. .. 225
 Compatibility **39–40**
 compatibility **146–147**

contents

 table of **46, 69, 77**
 counter
 → Index of Commands etc. .. 225
 → Index of Lengths etc. 230
secnumdepth **83**
tocdepth **70–72**

D

date 67, 142, 198
 dedication 68
 document structure 46
 double-sided 56
 draft version 157
 DVI 33

E

EC fonts 76
 element
 → Index of Elements 230
 empty (page style) ... 42, **56–57, 150, 163–165**
 environment
 → Index of Commands etc. .. 225
 equation 51

F

figure 101
 file
 → Index of Files etc. 231
 final version 157
 floating environments **72, 101**
 float style
 boxed 106

- komaabove **106**
 - komabelow **106**
 - plain **106**
 - ruled **106**
 - font **53–56, 74–76, 93**
 - size . **46, 53–56, 74–76, 152, 163**
 - style **105–106, 163, 165**
 - footer
 - color **128**
 - footnotes **67, 87**
 - front matter **72**
- H**
- half-title **65**
 - header **77**
 - color **128**
 - heading **78, 82, 84, 134**
 - headings (page style) **56–57, 163–165**
- I**
- indentation **43, 100**
 - index **73, 114**
- K**
- komaabove (float style) **106**
 - komabelow (float style) **106**
- L**
- language **197–201**
 - dependent terms **198–201**
 - definition **200**
 - selection **197**
 - lco **158–163**
 - length
 - Index of Commands etc. .. **225**
 - Index of Lengths etc. **230**
 - letter
 - class option **158–163**
 - footer **181–183**
 - head **152–157, 178–180**
 - letters **145–208**
 - line **131**
 - list
 - of figures **72**
 - of tables **72**
 - lists **89–100**
 - LM fonts **76**
 - logical markup **113**
- M**
- macro
 - Index of Commands etc. .. **225**
 - main matter **72**
 - margin **98**
 - notes **100**
 - markright **165**
 - markup **113**
 - myheadings (page style) **56–57, 163–165, 192**
- N**
- numbering **77, 83, 92**
- O**
- option
 - Index of Options **232**
 - options **146**
- P**
- package
 - Index of Files etc. **231**
 - page
 - counter **63**
 - layout **40, 147–148**
 - number **63**
 - style . **56, 120, 134, 136, 150, 192**
 - page style
 - empty . **42, 56–57, 150, 163–165**
 - headings **56–57, 163–165**
 - myheadings ... **56–57, 163–165, 192**
 - plain . **42, 56–57, 150, 163–165**
 - scrheadings **120–122, 133, 192**
 - scrplain **120–122**
 - useheadings **124**
 - paper format **31, 40, 147–148**
 - paragraph **43**
 - PDF **33**
 - plain (float style) **106**
 - plain (page style) ... **42, 56–57, 150, 163–165**

poems 95
 PostScript 33
 pseudo-length
 → Index of Lengths etc. 230
 pseudo-lengths 171–176
 publisher 67

R

reference fields line 188–190
 rule 131
 ruled (float style) 106
 running heading 28, 70, 82

S

scrheadings (page style) .. 120–122,
 133, 192
 scrplain (page style) 120–122
 secnumdepth
 → Index of Lengths etc. 230
 structuring 73
 subject 66
 subscript 113
 summary 50, 69
 superscript 113

T

table 101
 caption 102
 of contents 46, 69, 77
 telephone list 205
 terms
 language-dependent ... 198–201
 text
 subscript 113
 superscript 113
 time 142, 143
 title 46
 head 66
 tocdepth
 → Index of Lengths etc. 230
 twoside 101
 type style 57–59

U

uppercase letters 134
 useheadings (page style) 124

V

variables 165–170

Index of Commands, Environments, and Variables

\@addtoplength 174
 \@newplength 174
 \@setplength 174

A

abstract (environment) 69, 84
 \addchap 77–78
 \addchap* 77–78
 addmargin (environment) ... 98–100
 \addpart 77–78
 \addpart* 77–78
 \addrchar 205–206, 209
 \adrentry 203, 209
 \Address 209–210
 \addsec 77–78
 \addsec* 77–78
 \addtokomafont 53–56, 74, 128

\addtolengthlength 174–176
 \addtoeffields 168–169
 \adrchar 205–206, 209
 \adrentry 202–203, 209
 \AfterClass 216
 \AfterClass* 216
 \AfterFile 215
 \AfterPackage 34, 216
 \AfterPackage* 216
 \and 66–68
 \appendix 114–115
 \appendixmore 115–116
 \areaset 30–31
 \AtBeginLetter 186
 \author 66–68
 \autodot 80–82
 \automark 124–125, 133

B

backaddress (variable) 166,
184–185
backaddresseseparator (variable)
..... 166, 184–185
\backmatter 73
\bankname 199–200
\BeforeClass 216
\BeforeFile 215
\BeforePackage 216
\bigskip 86, 95, 116
boxed
→ General Index 223

C

\capfont 118
\caplabelfont 118
\caption 51, 101–106
\captionabove 51, 101–106
\captionbelow 51, 101–106
captionbeside (environment)
..... 103–105
\captionformat 106–107
\captionsamerican 198
\captionsaustrian 198
\captionsbritish 198
\captionscroatian 198
\captionsdutch 198
\captionsenglish 198
\captionsfinnish 198
\captionsfrench 198
\captionsgerman 198
\captionsitalian 198
\captionsngerman 198
\captionsspanish 198
\captionsUKenglish 198
\captionsUSenglish 198
\cc 196
\ccname 199–200
ccseparator (variable) 166, 196
\cefoot 120–123
\cehead 120–123
\cfoot 120–123
\chapapp 82
\chapappifchapterprefix 82

\chapter 73–76, 83
\chapter* 77
\chapterformat 80–82
\chaptermark 82–83
\chaptermarkformat 82–83
\chapterpagestyle 59–61
\chead 120–123
\cleardoubleemptypage 61–62, 165
\cleardoublepage .. 42, 61–62, 165
\cleardoubleplainpage 61–62, 165
\cleardoublestandardpage . 61–62,
165
\clearpage 61–62, 165
\clearscrheadfoot 122–123
\clearscrheadings 122–123
\clearscrplain 122–123
\closing 194–195
\cofoot 120–123
\cohead 120–123
\Comment 209–210
\contentsname 70
customer (variable) 166, 189
\customername 199–200

D

\date 66–68, 143
date (variable) 166, 189
\dateamerican 198
\dateaustrian 198
\datebritish 198
\datecroatian 198
\datedutch 198
\dateenglish 198
\datefinnish 198
\datefrench 198
\dategerman 198
\dateitalian 198
\datename 199–200
\datengerman 198
\datespanish 198
\dateUKenglish 198
\dateUSenglish 198
\dedication 68–69
\defaulttreffields 168–169
\deffootnote 88–89

- `\deffootnotemark` **88–89**
 - `\defpagestyle` **136–141**
 - `\deftripstyle` **134–136**
 - `\descfont` **118**
 - `description` (environment) **93**
 - `\dictum` **85–87**
 - `\dictumauthorformat` **85–87**
 - `\dictumwidth` **85–87**
- E**
- `\emailname` **199–200**
 - `emailseparator` (variable) . **166, 180**
 - `empty`
 - General Index **223**
 - `\encl` **196**
 - `\enclname` **199–200**
 - `enclseparator` (variable) .. **166, 196**
 - `\enlargethispage` **148**
 - `enumerate` (environment) **92–93**
 - `\extratitle` **65–66**
- F**
- `\faxname` **199–200**
 - `faxseparator` (variable) ... **166, 180**
 - `figure` (environment) **108**
 - `\figureformat` **107**
 - `\firstfoot` **181–183**
 - `\firsthead` **180**
 - `\FirstName` **209–210**
 - `\flushbottom` **41**
 - `\footnote` **87–88**
 - `\footnotemark` **87–88**
 - `\footnotetext` **87–88**
 - `\FreeI` **209–210**
 - `\FreeII` **209–210**
 - `\FreeIII` **209–210**
 - `\FreeIV` **209–210**
 - `fromaddress` (variable) **166, 179–180**
 - `frombank` (variable) **167, 192**
 - `fromemail` (variable) . **167, 179–180**
 - `fromfax` (variable) **167, 179–180**
 - `fromlogo` (variable) ... **167, 179–180**
 - `fromname` (variable) **164, 167, 179–180**
 - `fromphone` (variable) . **167, 179–180**
 - `fromurl` (variable) **167, 179–180**
 - `\frontmatter` **73**
- H**
- `\headfont` **125**
 - `\headfromname` **199–200**
 - `headings`
 - General Index **223**
 - `\headmark` **123**
 - `\headtoname` **199–200**
- I**
- `\ifkomavareempty` **170**
 - `\ifoot` **120–123**
 - `\ifpdfoutput` **33–34**
 - `\ifthispageodd` **62–63**
 - `\ifthispagewasodd` **62–63**
 - `\ihead` **120–123**
 - `\indexpagestyle` **59–61**
 - `\InputAddressFile` **209–210**
 - `invoice` (variable) **167, 189**
 - `\invoicename` **199–200**
 - `\isopaper` **31–32**
 - `\item` **90–91, 92–94**
 - `itemize` (environment) **90–91**
- K**
- `komaabove`
 - General Index **223**
 - `komabelow`
 - General Index **223**
 - `\KOMAOptions` **146**
- L**
- `\labelenumi` **92–93**
 - `\labelenumii` **92–93**
 - `\labelenumiii` **92–93**
 - `\labelenumiv` **92–93**
 - `labeling` (environment) **93–94**
 - `\labelitemi` **90–91**
 - `\labelitemii` **90–91**
 - `\labelitemiii` **90–91**
 - `\labelitemiv` **90–91**
 - `\LastName` **209–210**
 - `\lefoot` **120–123**

`\leftmark` 123
`\lehead` 120–123
`letter` (environment) . 185, 185–186
`\LetterOptionNeedsPapersize`
 161–163
`\linespread` 26
`\listfigurename` 72
`\listoffigures` 72
`\listoftables` 72
`\listtablename` 72
`\LoadLetterOption` 158–161
`location` (variable) ... 167, 187–188
`\lofoot` 120–123
`\lohead` 120–123
`\lowertitleback` 68

M

`\mainmatter` 73
`\maketitle` 64–69
`\MakeUppercase` 170, 171
`\manualmark` 124
`\marginline` 100–101
`\marginpar` 100–101
`\markboth` 57, 124, 125, 165, 170, 192
`\markleft` 125, 170, 192
`\markright` ... 57, 124, 125, 170, 192
`\medskip` 95
`\minisec` 78–79
`myheadings`
 → General Index 223
`myref` (variable) 167, 189
`\myrefname` 199–200

N

`\Name` 209–210
`\nameday` 143
`\newcaptionname` 201
`\newkomavar` 168–169
`\newkomavar*` 168–169
`\newpagestyle` 136–141
`\nextfoot` 192–193
`\nexthead` 192–193
`\noindent` 96
`\nopagebreak` 96

O

`\ohead` 120–123
`\opening` 163, 165, 193
`\othersectionlevelsformat` . 80–82

P

`\pagemark` 123–124
`\pagename` 199–200
`\pagenumbering` 63
`\pagestyle` ... 56–57, 124, 163–165
`\paperheight` 32
`\paperwidth` 32
`\paragraph` 73–76
`\paragraph*` 77
`\parbox` 85
`\part` 73–76, 83
`\part*` 77
`\partformat` 80–82
`\partpagestyle` 59–61
`\pdfoutput` 34
`\pdfpageheight` 33
`\pdfpagewidth` 33
`\phonename` 199–200
`phoneseparator` (variable) . 167, 180
`place` (variable) 167, 189–190
`placeseparator` (variable) 167,
 189–190
`plain`
 → General Index 223
`\pnumfont` 125
`\protect` 170
`\providecaptionname` 201
`\providepagestyle` 136–141
`\ps` 196
`\publishers` 66–68

Q

`quotation` (environment) 96–98
`quote` (environment) 96–98

R

`\raggedbottom` 41
`\raggeddictum` 85–87
`\raggeddictumauthor` 85–87
`\raggeddictumtext` 85–87
`\raggedleft` 86

- `\raggedright` 86
- `\raggedsection` 79–80
- `\raggedsignature` 195
- `\refoot` 120–123
- `\rehead` 120–123
- `\removeeffields` 168–169
- `\renewcaptionname` 201
- `\renewpagestyle` 136–141
- `\rfoot` 120–123
- `\rightmark` 123
- `\rofoot` 120–123
- `\rohead` 120–123
- ruled
 - General Index 223
- S**
- scrheadings
 - General Index 223
- scrplain
 - General Index 223
- secnumdepth
 - Index of Lengths etc. 230
- `\sectfont` 117
- `\section` 73–76, 83
- `\section*` 77
- `\sectionmark` 82–83
- `\sectionmarkformat` 82–83
- `\setbibpreamble` 116–117
- `\setcaphanging` 107–108
- `\setcapindent` 107–108
- `\setcapindent*` 107–108
- `\setcapmargin` 108–113
- `\setcapmargin*` 108–113
- `\setcapwidth` 108–113
- `\setchapterpreamble` 84–85
- `\SetDIVList` 34
- `\setfootbotline` 128–130
- `\setfootseptline` 128–130
- `\setfootwidth` 126–128
- `\setheadsepline` 128–130
- `\setheadtopline` 128–130
- `\setheadwidth` 126–128
- `\setindexpreamble` 117
- `\setkomafont` 53–56, 74, 128
- `\setkomavar` 169–170
- `\setkomavar*` 169–170
- `\setlengthtoplength` 174–176
- `\setpartpreamble` 84–85
- `\settime` 144
- signature (variable) 167, 194
- specialmail (variable) 168, 185
- `\subject` 66–68
- subject (variable) 164, 168, 190–191
- `\subjectname` 199–200
- subjectseparator (variable) ... 168, 190–191
- `\subparagraph` 73–76
- `\subparagraph*` 77
- `\subsection` 73–76, 83
- `\subsection*` 77
- `\subsectionmark` 82–83
- `\subsectionmarkformat` 82–83
- `\subsubsection` 73–76, 83
- `\subsubsection*` 77
- T**
- `\tableformat` 107
- `\tableofcontents` 70
- `\Telephone` 209–210
- `\textsubscript` 113–114
- `\textsuperscript` 88–89, 113
- `\thanks` 66–68
- `\theenumi` 92–93
- `\theenumii` 92–93
- `\theenumiii` 92–93
- `\theenumiv` 92–93
- `\thefootnotemark` 88–89
- `\thispagestyle` ... 56–57, 163–165
- `\thistime` 143–144
- `\thistime*` 143–144
- `\title` 66–68
- title (variable) 168, 190
- `\titlehead` 66–68
- titlepage (environment) 64
- `\titlepagestyle` 59–61
- toaddress (variable) 168, 185
- tocdepth
 - Index of Lengths etc. 230
- `\today` 67, 142

<code>\todaysname</code>	142	<code>\useplength</code>	174
<code>toname</code> (variable)	168, 185		
<code>\typearea</code>	24–27		
		V	
		<code>verse</code> (environment)	94–96
		W	
		<code>\wwwname</code>	199–200
		Y	
		<code>yourmail</code> (variable)	168, 189
		<code>\yourmailname</code>	199–200
		<code>yourref</code> (variable)	168, 189
		<code>\yourrefname</code>	199–200

Index of Lengths and Counters

	B		
<code>backaddrheight</code>	172, 184–185	<code>refvpos</code>	173, 188
<code>bfoldmarkvpos</code>	172, 191–192	<code>refwidth</code>	173, 188–189
	F		
<code>firstfootvpos</code>	173, 181, 183		
<code>firstfootwidth</code>	174, 181		
<code>firsttheadvpos</code>	172, 178		
<code>firsttheadwidth</code>	172, 178		
<code>foldmarkhpos</code>	171, 192		
<code>fromrulewidth</code>	172, 180		
	L		
<code>locwidth</code>	173, 187		
	R		
<code>refaftervskip</code>	173, 189		
<code>refhpos</code>	173, 189		
	S		
	<code>secnumdepth</code> (counter)		83
	<code>sigbeforevskip</code>		173, 195
	<code>sigindent</code>		173, 195
	<code>specialmailindent</code>		173, 185
	<code>specialmailrightindent</code>		173, 185
	T		
	<code>tfoldmarkvpos</code>		172, 191–192
	<code>toaddrheight</code>		172, 183–184
	<code>toaddrhpos</code>		161, 172, 183
	<code>toaddrindent</code>		172, 184
	<code>toaddrvpos</code>		172, 183
	<code>toaddrwidth</code>		172, 184
	<code>tocdepth</code> (counter)		70–72

Index of Elements with Capability of Font Adjustment

	B		
<code>backaddress</code>	164	<code>chapter</code>	54, 75
	C		
<code>caption</code>	54, 105		
<code>captionlabel</code>	54, 105		
	D		
	<code>descriptionlabel</code>		54, 93, 164
	<code>dictum</code>		54
	<code>dictumauthor</code>		54

dictumtext	54		P	
disposition	55, 66, 70, 74, 76, 79, 80		pagefoot	55, 58, 164, 165
		F	pagehead	55, 58, 164, 165
footbotline	128		pagenumber	55, 58, 164, 165
footbottomline	128		pagination	55
footnote	55, 88		paragraph	55, 75
footnotelabel	55, 88		part	55, 75
footnotereference	55, 89		partnumber	55, 75
footsepline	128			S
fromaddress	164		section	55, 75
fromname	164		sectioning	56
		H	subject	164, 190
headsepline	128		subparagraph	56, 75
headtopline	128		subsection	56, 75
		M	subsubsection	56, 75
minisec	55			T
			title	56, 66, 164

Index of Files, Classes, and Packages

	A		graphics (package)	52
addrconv (package)	213		graphicx (package)	52
article (class)	37			I
	B		ifthen (package)	116
babel (package) 65, 143, 157, 176, 197			isodate (package)	157
book (class)	37, 134			K
	C		keyval (package)	145
caption (package)	103			L
caption2 (package)	52, 103		letter (class)	37
color (package)	129		longtable (package)	52, 103, 109
	E			M
extsizes (package)	46		mparhack (package)	101
	F		multicol (package)	41
fancyhdr (package)	58			N
float (package) ... 47, 49, 51, 103, 106			natbib (package)	116
french (package)	197		ngerman (package) .	143, 160, 197, 200
	G			R
german (package) ..	143, 157, 197, 200		report (class)	37

S

scraddr (package)	209–211
scrartcl (class)	57, 70, 83
scrbook (class)	56, 70, 83
scrdate (package)	142–143
scrlettr (class)	145
scrfile (package)	214–218
scrlltr2 (class)	145–208
scrpage (package)	59
scrpage.cfg	141
scrpage2 (package) .	59, 71, 119–141, 192
scrreprt (class)	56, 70, 83
scrttime (package)	143–144

T

topcapt (package)	103
typearea (package)	40, 126
typearea.cfg	34

Index of Class and Package Options

Xpt	46
10pt	46
11pt	46
12h	144
12pt	46
24h	144
chapterprefix	44–45
cleardoubleempty	42
cleardoublepage	149
cleardoubleplain	42
cleardoublestandard	42
clines	131–132

A

a0paper	31–32, 40
abstractoff	50
abstracton	50
addrfield	155
adrFreeIVempty	212
adrFreeIVshow	212
adrFreeIVstop	212
adrFreeIVwarn	212
appendixprefix	45
automark	132–133
autooneside	133

B

b0paper	31–32, 40
backaddress	155
BCOR	20–21, 41, 148
bibtotoc	47–48
bibtotocnumbered	47–48
bigheadings	46

C

c0paper	31–32, 40
chapteratlists	49–50

D

d0paper	31–32, 40
DIN	162
DINmtext	162
DIV	21–23, 41, 148
DIVcalc	23–27, 29, 41
DIVclassic	23–27, 41
dotlessnumbers	51
dottednumbers	51
draft	52, 157–158
dvips	32–33

E

enlargefirstpage	148
executivepaper	31–32, 40

F

final	52
fleqn	51
foldmarks	156–157
fontsize	152
footbotline	130–131
footexclude	27–28, 130
footinclude	27–28, 130

- footnosepline 44
 - footsepline 44, 130–131, 149–150, 165
 - fromalign 152
 - fromemail 155
 - fromfax 155
 - fromlogo 155
 - fromphone 154
 - fromrule 152–154
 - fromurl 155
- H**
- halfparskip 43–44
 - halfparskip* 43–44
 - halfparskip+ 43–44
 - halfparskip- 43–44
 - header lines 148
 - headexclude 27–28, 130
 - headinclude 27–28, 130
 - headlines 29–30, 41
 - headnosepline 44
 - headsepline 44, 130–131, 149–150, 165
 - headtopline 130–131
- I**
- idxtotoc 47–48
 - ilines 131–132
- K**
- KOMAold 162
 - komastyle 133
- L**
- landscape 31–32, 40
 - legalpaper 31–32, 40
 - leqno 51
 - letterpaper 31–32, 40
 - listsindent 49
 - listsleft 49
 - liststotoc 47–48
 - liststotocnumbered 47–48
 - locfield 156
- M**
- manualmark 132–133
- N**
- markuppercase 133
 - markusedcase 133
 - mpexclude 28–29, 149
 - mpinclude 28–29, 149
- O**
- noappendixprefix 45
 - nochapterprefix 44–45
 - noonelinecaption 45
 - normalheadings 46
 - notitlepage 42
 - nouppercase 133–134
 - numericaldate 157
- P**
- olines 131–132
 - onecolumn 41
 - onelinecaption 45
 - oneside 41, 149
 - openany 42
 - openbib 52
 - openright 42
 - origlongtable 52
- R**
- refline 157
- S**
- smallheadings 46

SN	161, 162	titlepage	42
SNleft	162	tocindent	48–49
standardstyle	133	tocleft	48–49
subject	155–156	twocolumn	41, 84
T		twoside	41, 149
tablecaptionabove	51–52, 102	V	
tablecaptionbelow	51–52, 102	version	39–40, 146–147