

KOMA-Script

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

Note: This document is a translation of the German KOMA-Script manual. Several authors have been involved to this translation. Some of them are native English speakers, others like me are not. Improvements of the translation by native speakers or experts are welcome at all times!

The Guide

KOMA - Script

Markus Kohm

2017-01-02

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, Krickette Murabayashi, Markus Kohm, Jens-Uwe Morawski, Jana Schubert, Jens Hühne, Harald Bongartz, Georg Grandke, Raimund Kohl, Stephan Hennig, Alexander Willand, Melvin Hendrix, and Arndt Schubert.

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 1.3c. 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 an explicit, additional authorization by the authors.

To All Friends of Typography!

Preface to the English KOMA-Script Guide

The translation of the German KOMA-Script guide is still a work in progress and a never ending story. I always try to have an English user guide with all descriptions of the German one. But as long as I have to do the primary translation, these translations not only can but should be improved.

In this release the translation of **chapter 15** has been completed. Nevertheless, a proficient English speaker with basic TeX knowledge could improve the translation. A large part of **chapter 11** has been translated by Alexander Willand. The remaining part may need improvement. The example still has not been translated.

So this English guide is complete but nevertheless not as good as the German one. Currently there are only a few editors for the English guide, who improve my translation for free. Many thanks to them for their very good job! Nevertheless, additional editors or translators would be welcome!

Contents

Preface to the English KOMA-Script Guide	7
1. Introduction	20
1.1. Preface	20
1.2. Structure of the Guide	20
1.3. History of KOMA-Script	21
1.4. Special Thanks	22
1.5. Legal Notes	23
1.6. Installation	23
1.7. Bug Reports and Other Requests	23
1.8. Additional Information	23
 Part I:	
KOMA-Script for Authors	24
 2. Construction of the Page Layout with typearea	25
2.1. Fundamentals of Page Layout	25
2.2. Page Layout Construction by Dividing	27
2.3. Page Layout Construction by Drawing a Circle	29
2.4. Early or late Selection of Options	29
2.5. Compatibility with Earlier Versions of KOMA-Script	30
2.6. Options and Macros to Influence the Page Layout	31
2.7. Paper Format Selection	45
2.8. Tips	47
 3. The Main Classes: scrbook, scrreprt, and scrartcl	51
3.1. Early or late Selection of Options	51
3.2. Compatibility with Earlier Versions of KOMA-Script	53
3.3. Draft Mode	54
3.4. Page Layout	54
3.5. Selection of the Document Font Size	55
3.6. Text Markup	56
3.7. Document Titles	62
3.8. Abstract	68
3.9. Table of Contents	69
3.10. Paragraph Markup	73

3.11.	Detection of Odd and Even Pages	76
3.12.	Head and Foot Using Predefined Page Styles	76
3.13.	Interleaf Pages	81
3.14.	Footnotes	84
3.15.	Demarcation	89
3.16.	Structuring of Documents	90
3.17.	Dicta	109
3.18.	Lists	111
3.19.	Math	119
3.20.	Floating Environments of Tables and Figures	119
3.21.	Margin Notes	136
3.22.	Appendix	137
3.23.	Bibliography	137
3.24.	Index	140
4.	The New Letter Class <code>scrlettr2</code>	142
4.1.	Variables	142
4.2.	Pseudo-Lengths	147
4.3.	Early or late Selection of Options	148
4.4.	Compatibility with Earlier Versions of KOMA-Script	149
4.5.	Draft Mode	150
4.6.	Page Layout	151
4.7.	General Structure of Letter Documents	151
4.8.	Selection of the Document Font Size	161
4.9.	Text Markup	163
4.10.	Note Paper	167
4.11.	Paragraph Markup	197
4.12.	Detection of Odd and Even Pages	198
4.13.	Head and Foot Using Predefined Page Style	199
4.14.	Interleaf Pages	203
4.15.	Footnotes	205
4.16.	Lists	208
4.17.	Math	211
4.18.	Floating Environments of Tables and Figures	211
4.19.	Margin Notes	211
4.20.	Closing	212
4.21.	Letter Class Option Files	214
4.22.	Address Files and Circular Letters	220

5. Adapting Page Headers and Footers with scrlayer-scrpage	225
5.1. Early or late Selection of Options	225
5.2. Head and Foot Height	227
5.3. Text Markup	227
5.4. Usage of Predefined Page Styles	230
5.5. Manipulation of Defined Page Styles	240
6. The Day of the Week Using scrdate	252
7. Getting the Time with Package scrtime	257
8. Access to Address Files with scraddr	259
8.1. Overview	259
8.2. Usage	260
8.3. Package Warning Options	261
9. Creating Address Files from an Address Database	263
10. KOMA-Script Features for other Classes with Package scrextend	264
10.1. Early or late Selection of Options	264
10.2. Compatibility with Earlier Versions of KOMA-Script	266
10.3. Optional, Extended Features	266
10.4. Draft Mode	267
10.5. Selection of the Document Font Size	267
10.6. Text Markup	268
10.7. Document Titles	270
10.8. Detection of Odd and Even Pages	274
10.9. Head and Foot Using Predefined Page Styles	275
10.10. Interleaf Pages	275
10.11. Footnotes	278
10.12. Dicta	282
10.13. Lists	283
10.14. Margin Notes	285
11. Support for the Law Office by scrjura	286
11.1. Early or late Selection of Options	286
11.2. Text Markup	287
11.3. Table of Contents	288
11.4. Environment for Contracts	289
11.4.1. Clauses	289
11.4.2. Paragraphs	292

11.4.3. Sentences	294
11.5. Cross References	295
11.6. Additional Environments	297
11.7. Support for Different Languages	299
11.8. A Detailed Example	299
11.9. State of Development	304

Part II:

KOMA-Script for Advanced Users and Experts307

12. Basic Functions of Package scrbase	308
12.1. Loading the Package	308
12.2. Keys as Attributes of Families and their Members	309
12.3. Conditional Execution	321
12.4. Definition of Language-Dependent Terms	325
12.5. Identification of KOMA-Script	328
12.6. Extension of the L ^A T _E X Kernel	329
12.7. Extension of the Mathematical Features of ε -T _E X	329
13. Control Package Dependencies with scrfile	331
13.1. About Package Dependencies	331
13.2. Actions Prior to and After Loading	332
13.3. Replacing Files at Input	336
13.4. Prevent File Loading	339
14. Economise and Replace Files Using scrwfile	342
14.1. General Modifications of the L ^A T _E X Kernel	342
14.2. The Single File Feature	343
14.3. The Clone File Write Feature	343
14.4. Note on State of Development	344
14.5. Known Package Incompatibilities	345
15. Management of Tables and Lists of Contents Using tocbasic	346
15.1. Basic Commands	346
15.2. Creating a Table or List of Contents	349
15.3. Configuration of Entries to a Table or List of Contents	356
15.4. Internal Commands for Class and Package Authors	367
15.5. A Complete Example	370
15.6. Everything with One Command Only	373

16. Hacks for Third-Party Packages by Package scrhack	378
16.1. State of Development Note	378
16.2. Early or late Selection of Options	378
16.3. Usage of tocbasic	379
16.4. Incorrect Expectations to \@ptsize	380
16.5. Special Case hyperref	381
16.6. Inconsistent Handling of \textwidth and \textheight	381
17. Defining Layers and Page Styles Using sclayer	382
17.1. State of Development Note	382
17.2. Early or late Selection of Options	382
17.3. Some Generic Information	383
17.4. Declaration of Layers	385
17.5. Declaration and Management of Page Styles	396
17.6. Head and Foot Height	404
17.7. Manipulation of Defined Page Styles	405
17.8. End User Interfaces	411
18. Additional Features of sclayer-scrpage	413
18.1. Manipulation of Defined Page Styles	413
18.2. Definition of new Pairs of Page Styles	416
18.3. Definition of Simple Page Styles with Three Parts in Head and Foot	418
18.4. Definition of Complex Page Styles	421
19. Note Columns with sclayer-notecolumn	424
19.1. Note about the State of Development	424
19.2. Early or late Selection of Options	425
19.3. Text Markup	426
19.4. Declaration of new Note Columns	427
19.5. Making a Note	431
19.6. Force Output of Note Columns	435
20. Additional Information about package typearea	437
20.1. Experimental Features	437
20.2. Expert Commands	438
20.3. Local Settings with File typearea.cfg	440
20.4. More or Less Obsolete Options and Commands	440
21. Additional Information about the Main Classes and scrextend	441
21.1. Additional Information to User Commands	441
21.2. Cooperation and Coexistence of KOMA-Script and Other Packages	441

21.3.	Expert Commands	441
21.4.	More or Less Obsolete Options and Commands	463
22.	Additional Information about the Letter Class <code>scrlettr2</code> and the Letter Package <code>scrletter</code>	464
22.1.	Pseudo-Lengths for Experienced Users	464
22.1.1.	Folding Marks	470
22.1.2.	Letterhead	472
22.1.3.	Addressee	473
22.1.4.	Sender's Extensions	475
22.1.5.	Business Line	476
22.1.6.	Subject	477
22.1.7.	Closing	478
22.1.8.	Letter Footer	478
22.2.	Variables for Experienced Users	479
22.3.	Differences in the Page Styles of <code>scrletter</code>	481
22.4.	Differences in the Handling of <code>lco</code> -Files in <code>scrletter</code>	482
22.5.	<code>lco</code> -Files for Experienced Users	482
22.5.1.	Survey of Paper Size	483
22.5.2.	Visualization of Positions	483
22.6.	Language Support	486
22.7.	From Obsolete <code>scrlettr</code> to Current <code>scrlettr2</code>	489
A.	Japanese Letter Support for <code>scrlettr2</code>	491
A.1.	Japanese standard paper and envelope sizes	491
A.1.1.	Japanese paper sizes	491
A.1.2.	Japanese envelope sizes	492
A.2.	Provided <code>lco</code> files	496
A.3.	Examples of Japanese letter usage	498
A.3.1.	Example 1:	498
A.3.2.	Example 2:	499
	Change Log	500
	Bibliography	509
	Index	514
	General Index	514
	Index of Commands, Environments, and Variables	518
	Index of Lengths and Counters	529
	Index of Elements with Capability of Font Adjustment	530

Index of Files, Classes, and Packages

532

Index of Class and Package Options

533

List of Figures

2.1.	Double-sided layout with the box construction of the classical division factor of 9, after subtraction of a binding correction	28
3.1.	Parameters that control the footnote layout	87
3.3.	Example: Usage of <code>\captionaboveof</code> inside another floating environment	126
3.2.	Example: A rectangle	126
3.4.	Example: Figure beside description	128
3.5.	Example: Description centered beside figure	128
3.6.	Example: Figure title top beside	129
3.7.	Example: Default figure caption	132
3.8.	Example: Figure caption with slightly hanging indentation	132
3.9.	Example: Figure caption with hanging indentation and line break	132
3.10.	Example: Figure caption with hanging indentation at the second line	132
4.1.	General structure of a letter document with several individual letters	152
4.2.	General structure of a single letter within a letter document	152
4.3.	Example: letter with addressee and opening	156
4.4.	Example: letter with addressee, opening, text, and closing	157
4.5.	Example: letter with addressee, opening, text, closing, and postscript	158
4.6.	Example: letter with addressee, opening, text, closing, postscript, and distribution list	160
4.7.	Example: letter with addressee, opening, text, closing, postscript, distribution list, and enclosure	161
4.8.	Example: letter with addressee, opening, text, closing, postscript, distribution list, enclosure, and insane large font size	164
4.9.	schematic display of the note paper with the most important commands and variables for the drafted elements	169
4.10.	Example: letter with addressee, opening, text, closing, postscript, distribution list, enclosure, and hole puncher mark	171
4.11.	Example: letter with sender, addressee, opening, text, closing, postscript, distribution list, and enclosure	174
4.12.	Example: letter with sender, separation rule, addressee, opening, text, closing, signature, postscript, distribution list, enclosure, and puncher hole mark	176
4.13.	Example: letter with extended sender, separation rule, addressee, opening, text, closing, signature, postscript, distribution list, enclosure, and puncher hole mark; standard vs. extended letterhead	179

4.14.	Example: letter with extended sender, separation rule, addressee, opening, text, closing, signature, postscript, distribution list, enclosure, and puncher hole mark; left vs. right aligned letterhead	181
4.15.	Example: letter with extended sender, logo, separation rule, addressee, opening, text, closing, signature, postscript, distribution list, enclosure, and puncher hole mark; left vs. right aligned vs. centered sender	183
4.16.	Example: letter with extended sender, logo, addressee, additional sender information, opening, text, closing, signature, postscript, distribution list, enclosure, and puncher hole mark	188
4.17.	Example: letter with extended sender, logo, addressee, additional sender information, place, date, opening, text, closing, signature, postscript, distribution list, enclosure, and puncher hole mark	192
4.18.	Example: letter with extended sender, logo, addressee, additional sender information, place, date, subject, opening, text, closing, signature, postscript, distribution list, enclosure, and puncher hole mark	195
4.19.	Example: letter with extended sender, logo, addressee, additional sender information, place, date, subject, opening, text, closing, modified signature, postscript, distribution list, enclosure, and puncher hole mark	214
4.20.	Example: letter with extended sender, logo, addressee, additional sender information, place, date, subject, opening, text, closing, modified signature, postscript, distribution list, enclosure, and puncher hole mark using a lco-file .	217
5.1.	Commands to define the page head	232
5.2.	Commands to define the page footer	235
11.1.	Example: Three pages from the front of the example club statutes of section 11.8	305
15.1.	Illustrations of some attributes of a TOC-entry with style <code>dottedtocline</code>	358
15.2.	Illustrations of some attributes of a TOC-entry with style <code>largetocline</code>	359
15.3.	Illustrations of some attributes of a TOC-entry with style <code>tocline</code>	359
15.4.	Illustration of some attributes of style <code>undottedtocline</code> by the example of a chapter title	360
18.1.	Elements of a three parts page style	420
19.1.	An example page to the example of chapter 19	436
22.1.	Schematic of the pseudo-lengths for a letter	469

List of Tables

2.1.	Type area dimensions dependent on <i>DIV</i> for A4	33
2.2.	Predefined settings of <i>DIV</i> for A4	34
2.3.	Symbolic values for the <i>DIV</i> option and the <i>DIV</i> argument to <code>\typearea</code>	36
2.4.	Symoblic <i>BCOR</i> arguments for <code>\typearea</code>	38
2.5.	Standard values for simple switches in KOMA-Script	39
2.6.	Output driver for option <code>pagesize=output driver</code>	48
3.1.	Class correspondence	51
3.2.	Elements whose type style can be changed with the KOMA-Script command <code>\setkomafont</code> or <code>\addtokomafont</code>	58
3.3.	Font defaults for the elements of the title	66
3.4.	Main title	66
3.5.	Possible values of option <code>toc</code>	70
3.6.	Font style defaults of the elements of the table of contents	73
3.7.	Possible values of option <code>parskip</code>	74
3.8.	Default values for the elements of a page style	78
3.9.	Macros to set up page style of special pages	80
3.10.	Available numbering styles of page numbers	81
3.11.	Available values for option <code>footnotes</code>	85
3.12.	Available values for option <code>open</code>	90
3.13.	Available values for option <code>headings</code>	92
3.14.	Available values of option <code>numbers</code>	94
3.15.	Default font sizes for different levels of document structuring	98
3.16.	Default settings for the elements of a dictum	109
3.17.	Available values for option <code>captions</code>	121
3.18.	Font defaults for the elements of figure or table captions	124
3.19.	Example: Measure of the rectangle in figure 3.2	126
3.20.	Available values for option <code>listof</code>	135
3.21.	Available values of option <code>bibliography</code>	139
3.22.	Available values of option <code>index</code>	141
4.1.	Alphabetical list of all supported variables in <code>scrlltr2</code>	142
4.2.	Alphabetical list of elements whose font can be changed in <code>scrlltr2</code> using the commands <code>\setkomafont</code> and <code>\addtokomafont</code>	165
4.3.	Combinable values for the configuration of folding marks with option <code>foldmarks</code>	168
4.4.	Available values for option <code>fromalign</code> with <code>scrlltr2</code>	172
4.5.	Possible values of option <code>fromrule</code> with <code>scrlltr2</code>	173

4.6.	Predefined descriptions of the variables of the letterhead	177
4.7.	Predefined description and content of the separators at the letterhead without option <code>symbolicnames</code>	178
4.8.	available values for option <code>addrfield</code> using <code>scrlltr2</code>	184
4.9.	Predefined font style for the elements of the address field.	185
4.10.	available values for option <code>priority</code> in <code>scrlltr2</code>	186
4.11.	Possible values of option <code>locfield</code> with <code>scrlltr2</code>	187
4.12.	Possible value of option <code>refline</code> with <code>scrlltr2</code>	190
4.13.	predefined descriptions of variables of the reference line	190
4.14.	font style default of elements of the reference line	191
4.15.	predefined descriptions of subject-related variables	193
4.16.	available values of option <code>subject</code> with <code>scrlltr2</code>	194
4.17.	available values of option <code>pagenumber</code> with <code>scrlltr2</code>	201
4.18.	predefined <code>lco</code> -files	218
5.1.	Elements of <code>scrlayer-scrpage</code> whose type style can be changed with <code>KOMA-Script</code> command <code>\setkomafont</code> or <code>\addtokomafont</code>	228
5.2.	Possible values for option <code>markcase</code>	245
5.3.	Symbolic values for options <code>headwidth</code> and <code>footwidth</code>	250
10.1.	optional available extended features of <code>scrxextend</code>	267
11.1.	Possible properties for the optional argument of <code>\Clause</code> and <code>\SubClause</code>	290
11.2.	Possible values for option <code>clausemark</code> for activation of running heads	292
11.3.	Possible values for option <code>ref</code> to configure the cross reference format	296
11.4.	Example outputs of the <code>ref</code> -independent cross reference commands	297
11.5.	Meanings and English defaults of language dependent terms	299
12.1.	Overview of usual language dependent terms	327
15.1.	Attributes of the predefined TOC-entry styles of <code>tocbasic</code>	360
15.2.	Options for command <code>\DeclareNewTOC</code>	373
15.3.	Comparison of example environment <code>remarkbox</code> and environment <code>figure</code>	377
17.1.	Options for the definition of page layers with description of the corresponding layer attribute	387
17.2.	The <i>hook</i> options for page styles (in order of execution)	398
18.1.	The layers <code>scrlayer-scrpage</code> defines to a page style	422
19.1.	Options for the declaration of note columns	429

21.1.	Style-independent attributes at the declaration of section-like commands	447
21.2.	Attributes of the style <code>section</code> declaring a section-like command	448
21.3.	Attributes of the style <code>chapter</code> declaring a section-like command	449
21.4.	Attributes of the style <code>part</code> declaring a section-like command	450
21.6.	Default of the headings of <code>scrbook</code> and <code>scrreprt</code>	451
21.5.	Defaults of the chapter headings of <code>scrbook</code> and <code>scrreprt</code> subject to option <code>headings</code>	455
22.1.	Pseudo-lengths provided by class <code>scrlettr2</code>	465
22.2.	Language-dependent forms of the date	488
22.3.	Default settings for language-dependent terms	490
A.1.	ISO and JIS standard paper sizes	492
A.2.	Japanese B-series variants	492
A.3.	Main Japanese contemporary stationary	493
A.4.	Japanese ISO envelope sizes	494
A.5.	Japanese envelope sizes 3	495
A.6.	Supported Japanese envelope types and the window sizes and locations	497
A.7.	<code>lco</code> files provided by <code>scrlettr2</code> for Japanese window envelopes	498

Introduction

This chapter includes important information about the structure of the manual and the history of KOMA-Script, which begins years before the first version. You will also find information for those who have not installed KOMA-Script or encounter errors.

1.1. Preface

KOMA-Script is very complex. This is evidenced by the fact that it consists of not only a single class or a single package, but a bundle of many classes and packages. Although the classes are designed as a counterpart to the standard classes, that does not necessarily mean that they only have the commands, environments, and setting of the standard classes or imitate their appearance. The capabilities of KOMA-Script surpass the capabilities of the standard classes considerably. Some of them are to be regarded as a supplement to the basic skills of the \LaTeX kernel.

The foregoing means that the documentation of KOMA-Script has to be extensive. In addition, KOMA-Script usually is not taught. That means there is no teacher who knows his students and can therefore choose the teaching and learning materials and adapt them accordingly. It would be easy to write the documentation for any specific audience. The difficulty is, however, that the guide must service all potential audiences. We, the authors, have tried to create a guide that is suited for the computer scientist as well as the secretary or the fishmonger. We have tried, although this is actually an impossible task. The result consists of several compromises and we hope that you will keep this in mind when using it. Your suggestions for improvement are, of course, always welcome.

Despite the volume of the manual, it is recommended to consult the documentation. Attention is drawn to the multi-part index at the end of this document. In addition to this guide, documentation includes all the text documents that are part of the bundle. See `manifest.tex` for a list of all of them.

1.2. Structure of the Guide

This manual consists of several parts. There's a part for average users, another part for advanced users and experts, and an appendix with additional examples and information for those who always want to know more.

Part I is recommended for all KOMA-Script users. This means that you may find here even some information for newcomers to \LaTeX . In particular, this part is enhanced by many examples to the average user that are intended to illustrate the explanations. Do not be afraid to try these examples yourself and in modifying them to find out how KOMA-Script works. Nevertheless the KOMA-Script user guide is not intended to be a \LaTeX primer. Those new

to L^AT_EX should look at *The Not So Short Introduction to L^AT_EX 2_ε* [OPHS11] or *L^AT_EX 2_ε for Authors* [Tea05b] 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* [FAQ13]. Although the length of the *T_EX Frequently Asked Questions on the Web* is considerably long, it is nevertheless quite useful not only to those having problems using L^AT_EX or KOMA-Script.

Part II is recommended for advanced KOMA-Script users. These are all of you who already know L^AT_EX, maybe worked with KOMA-Script for a while, and want to learn more about KOMA-Script internals, interaction of KOMA-Script with other packages, and how to use KOMA-Script as an answer to special demands. For this purpose we will have a closer look at some aspects from **part I** again. In addition some instructions that have been implemented for advanced users and experts, especially, will be documented. This is complemented by the documentation of packages that are normally hidden to the user insofar as they do their work under the surface of the classes and user packages. These packages are specifically designed to be used by other authors of classes and packages.

The appendix, which may be found only in the German book version, contains information which is beyond what is covered in **part I** and **part II**. Advanced users may find background information on issues of typography to give them a basis for their own decisions. In addition, the appendix provides examples for aspiring authors of packages. These examples are less intended to be simply transferred. Rather, they convey knowledge of planning and implementation of projects as well as some basic L^AT_EX instructions for authors of packages.

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, scrreprt, and scrartcl) are introduced together in **chapter 3**. Where an example or note only applies to one or two of the three classes, e.g.,

scrartcl

scrartcl, it is called out in the margin, as shown here with scrartcl.

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 L^AT_EX was L^AT_EX 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. At that time neither the standard style nor *Script* provided support for A5 paper. Therefore it did not take long until Markus made the first changes to *Script*. 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` and `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 \TeX e.V, (the German-Language \TeX User Group). Without the DANTE server, *KOMA-Script* could not have been released and distributed. Thanks as well to everybody in the \TeX 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

The three most important T_EX distributions, MacT_EX, MiK_T_EX, and T_EX Live, make KOMA-Script available by their package management software. It is recommended to make installations and updates of KOMA-Script using these tools. Nevertheless the manual installation without using the package managers has been described in the file `INSTALL.txt`, that is part of every legal KOMA-Script bundle. You should also read the documentation that comes with the T_EX distribution you are using.

1.7. Bug Reports 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; if a newer version is available, 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 L^AT_EX 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. 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.

Please report KOMA-Script (only) bugs to komascript@gmx.info. If you want to ask your question in a Usenet group, mailing list, or Internet forum, you should also include such an example as part of your question.

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.

Part I.

KOMA-Script for Authors

In this part you may find information for authors of articles, reports, books, and letters. It is assumed that the average user is less interested in the implementation of KOMA-Script or in the problems of implementing KOMA-Script. Also, the average user isn't interested in obsolete options and instructions. He wants to know how he can achieve things using current options and instructions. Some users may be interested in typographic background information.

In this part, the few passages that contain additional information and justification, and therefore are of less interest for the impatient reader, have been set in sans serif font and can be skipped if necessary. For those who are interested in more information about implementation, side effects with other packages, and obsolete options and instructions, please refer to [part II](#) on [page 308](#). Furthermore, that part of the KOMA-Script guide describes all the features that were created specially for writers of packages and classes.

Construction of the Page Layout with typearea

Many L^AT_EX classes, including the standard classes, present the user with the largely fixed configuration of margins and typearea. With the standard classes, the configuration determined is very much dependent on the chosen font size. There are separate packages, such as *geometry* (see [Ume10]), which give the user complete control, but also full responsibility, of the settings of typearea and margins.

KOMA-Script takes a somewhat different approach with its *typearea* package. Here the user is given several construction setting and automatization possibilities based on established typography standards in order to help guide him or her in making a good choice.

It should be noted that the *typearea* package makes use of the *scrbase* package. The latter is explained in the expert section of this document in [chapter 12](#) from [page 308](#) onwards. The majority of the rules documented there are however not directed at the user, but rather at authors of classes and packages.

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, bottom, left, and right margins, a (running) head area, the text block, and a (running) foot area. There is also a 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 contains much discussion of different algorithms and heuristic approaches for constructing a good page layout [Koh02]. 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 other words, a single page contributes only half of the inner margin.

In the previous paragraph, we mentioned and emphasized *the page*. Erroneously, it is often thought that with the page format the page is the same as the paper format. However, if you look at a bound document, it is obvious that part of the paper vanishes in the binding and is no longer part of the visible page. For the page layout, it is not the format of the paper which is important, 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 is therefore calculated as part of the *gutter*, not the visible inner margin.

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.

$$\text{textblock height} : \text{textblock width} = \text{page height} : \text{page width}$$

$$\text{top margin} : \text{foot margin} = 1 : 2$$

$$\text{left margin} : \text{right margin} = 1 : 1$$

$$\text{half inner margin} : \text{outer margin} = 1 : 2$$

$$\text{page width} = \text{paper width} - \text{binding correction}$$

$$\text{top margin} + \text{bottom margin} = \text{page height} - \text{textblock height}$$

$$\text{left margin} + \text{right margin} = \text{page width} - \text{textblock width}$$

$$\text{half inner margin} + \text{outer margin} = \text{page width} - \text{textblock width}$$

$$\text{half inner margin} + \text{binding correction} = \text{gutter}$$

The values *left margin* and *right margin* only exist in a single-sided document while *half 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 only half of the inner margin, *half 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 font used
- line spacing
- word length
- available room

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

With leading is meant the vertical distance between individual lines of text. In \LaTeX , the leading is set at about 20% of the font size. With commands like `\linespread` or, better, packages like `setspace` (see [TF11]), the leading can be changed. A wider leading helps the eye to follow the

line. A very wide leading, on the other hand, disturbs reading because the eye has to move a wide distance between lines. Also, the reader becomes uncomfortable because of the visible stripe effect. The uniform gray value of the page is thereby spoiled. Still, with a wider leading, the lines can be longer.

The literature gives different values for good line lengths, depending on the author. To some extent, this is related to the native language of the author. Since the eye jumps from word to word, short words make this task easier. Considering all languages and fonts, a line length of 60 to 70 characters, including spaces and punctuation, forms a usable compromise. This requires well-chosen leading, but \LaTeX 's default is usually good enough. Longer line lengths should only be considered for highly-developed readers who spend several hours daily reading. However, even for such readers, line lengths greater than 80 characters are unsuitable. In any case, the leading must be appropriately chosen. An extra 5% to 10% is recommended as a good rule of thumb. With fonts such as Palatino, which require some 5% more leading even at normal line lengths, even more can be required.

Before looking at the actual construction of the page layout, there are just some minor things left to know. \LaTeX does not start the first line in the text block of a page at the upper edge of the text block, but sets the baseline at a defined distance from the top of the text block. 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 combination of the end of the page with titles, figures, tables or similar, even more. 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 set to 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, subtract the part *BCOR*, required for the binding correction, from the inner edge of the paper, and divide the rest of the page vertically into *DIV* rows of equal height.
- Next, divide the page horizontally into the same number (*DIV*) of columns.
- Then, take the uppermost row as the upper margin and the two lowermost rows as the lower margin. If you are printing double-sided, you similarly take the innermost column as the inner margin and the two outermost columns as the outer margin.

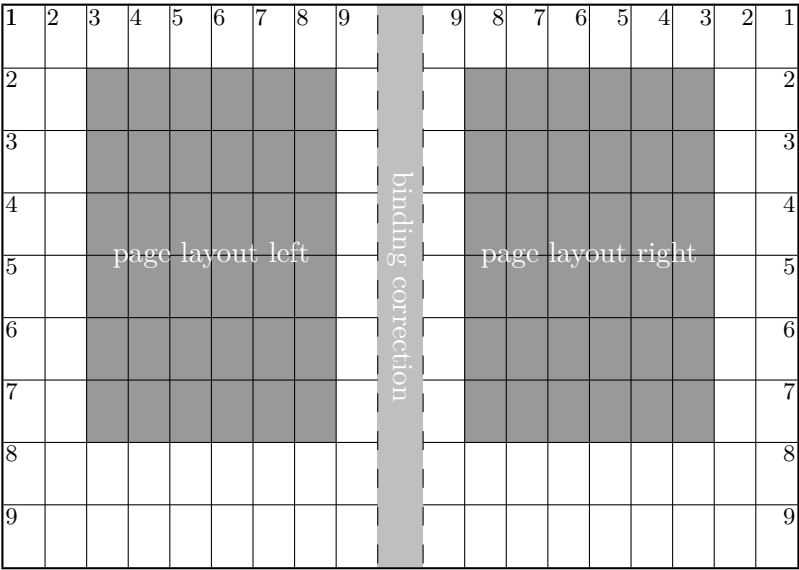


Figure 2.1.: Double-sided layout with the box construction of the classical division factor of 9, after subtraction of a binding correction

- Then add the binding correction $BCOR$ to the inner margin.

What now remains of the page is the text area. The width and the height of the text area and margins result automatically from the number of rows and columns DIV . Since the margins always need three stripes, 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 also known as the *classical division factor of 9* (see [figure 2.1](#)).

In KOMA-Script, this kind of construction is implemented in the `typearea` package, where the bottom margin may drop any fractions of a line in order to conform with the minor condition for the text area height mentioned in the previous paragraph, and thereby to minimize the mentioned problem with `\flushbottom`. For A4 paper, DIV is predefined according to the font size (see [table 2.2, page 34](#)). If there is no binding correction ($BCOR = 0\text{pt}$), the results roughly match the values of [table 2.1, page 33](#).

In addition to the predefined values, one can specify $BCOR$ and DIV as options when loading the package (see [section 2.4, from page 31](#) onwards). There is also a command to explicitly calculate the type area by providing these values as parameters (also see [section 2.4, page 37](#)).

The `typearea` package can automatically determine the optimal value of DIV for the font and leading used. Again, see [section 2.4, page 34](#).

2.3. Page Layout Construction by Drawing a Circle

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

A disadvantage of this late Middle Age method is that the width of the text area is no longer dependent on the font. Thus, one does not 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 of the late Middle Age method as closely as possible. See also [section 2.4, page 34](#).

2.4. Early or late Selection of Options

In this section a peculiarity of KOMA-Script is presented, which, apart from the typearea package, is also relevant to other KOMA-Script packages and classes. Such that the user can find all information corresponding to a single package or a single class in the relevant chapter, this section is found almost identically in several chapters. Users who are not only interested in a particular package or class, but wish to gain an overview of KOMA-Script as a whole, may read the section in one chapter and may thereafter skip it wherever coming across it in the document.

```
\documentclass[option list]{KOMA-Script class}
\usepackage[option list]{package list}
```

In L^AT_EX, provision is made for the user to pass class options as a comma-separated list of keywords as optional arguments to `\documentclass`. Apart from being passed to the class, these options are also passed on to all packages which can understand the options. Provision is also made for the user to pass optional arguments as a comma-separated list of keywords as optional arguments to `\usepackage`. KOMA-Script expands the option mechanism for the KOMA-Script classes and various packages to use further possibilities. Thus, most KOMA-Script options can also take a value. An option may have not only the form *Option*, but may also have the form *option=value*. Apart from this difference `\documentclass` and `\usepackage` function the same in KOMA-Script as described in [Tea05b] or any introduction to L^AT_EX, for example [OPHS11].

When using a KOMA-Script class, no options should be passed on unnecessary, explicit loading of the typearea or scrbase packages. The reason for this is that the class already loads these packages without options and L^AT_EX refuses multiple loadings of a package with different option settings.

You should note, that in opposite to the interface described below the options interface of `\documentclass` and `\usepackage` is not robust. So commands, lengths, counters and such constructs may break inside the optional argument of these commands. Because of this, the usage of a \LaTeX length inside the value of an option would cause an error before KOMA-Script can get the control over the option execution. So, if you want to use a \LaTeX length, counter or command a part of the value of an option, you have to use `\KOMAOptions` or `\KOMAoption`. These commands will be described next.

```
\KOMAOptions{option list}
\KOMAoption{option}{value list}
```

v3.00

KOMA-Script offers most class and package options the opportunity to change the value of options even after loading of the class or package. One may then change the values of a list of options at will with the `\KOMAOptions` command. Each option in the *option list* has the form *option=value*.

Some options also have a default value. If one does not give a value, i. e., gives the option simply in the form *option*, then the default value will be used.

Some options can assume several values simultaneously. For such options there exists the possibility, with the help of `\KOMAoption`, to pass a single *option* a list of values. The individual values are given as a comma-separated *value list*.

To implement this possibility KOMA-Script uses the commands `\FamilyOptions` and `\FamilyOption` with the family “KOMA”. For more information in these commands see [part II, section 12.2, page 313](#).

2.5. Compatibility with Earlier Versions of KOMA-Script

Those who achieve their documents in source code set utmost value to the fact that future \LaTeX runs will yield exactly the same result. Nevertheless, in some cases improvement and bug corrections of packages will result in changes of the behaviour and make-up. But sometimes this is not wanted.

```
version=value
version=first
version=last
```

v3.20b

Since version 3.01b of typearea it’s your choice if your source code should result in the same make-up at future \LaTeX runs or if you like to participate in all improvements of new releases. You may select the compatible version of KOMA-Script with option `version`. Compatibility to the lowest supported KOMA-Script release may be achieved by `version=first` or `version=2.9` or `version=2.9t`. Setting *value* to an unknown release number will result in a warning message and selects `version=first` for safety.

v3.01a

With `version=last` the most recent version will be selected at every L^AT_EX run. Be warned, though, that using `version=last` poses possibilities of compatibility issues for future L^AT_EX runs. Option `version` without any *value* means the same. This is the default behaviour as long as you do not use any deprecated options.

If you use a deprecated option of KOMA-Script 2, KOMA-Script 3 will switch to `version=first` automatically. This will also result in a warning message that explains how to prevent this switching. Alternatively you may select another adjustment using option `version` with the wanted compatibility after the deprecated option.

Compatibility is primarily make-up compatibility. New features not related to the mark-up will be available even if you switch compatibility to a version before first implementation of the feature. Option `version` does not influence make-up changes that are motivated by bug fixes. If you need bug compatibility you should physically save the used KOMA-Script version together with your document.

Please note that you cannot change option `version` anymore after loading the package `typearea`. Therefore, the usage of option `version` within the argument of `\KOMAOPTIONS` or `\KOMAOPTION` is not recommended and will cause an error.

2.6. Options and Macros to Influence the Page Layout

The package `typearea` offers two different user interfaces to influence type area construction. The more important 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. [OPHS11] and [Tea05b], 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](#)).

In this section the `protocol` class will be used, not an existing KOMA-Script class but a hypothetical one. This documentation assumes that ideally there exists a class for every specific task.

BCOR=correction

v3.00

With the aid of the option `BCOR=correction` one may specify the absolute value of the binding correction, i. e., the width of the area which will be lost from the paper width in the binding process. This value is then automatically taken into account in the page layout construction and in the final output is added to the inner (or the left) margin. For the *correction* specification any measurement unit understood by T_EX is valid.

Example: Assume one is creating a financial report, which should be printed out single-sided on A4 paper, and finally kept in a clamp folder. The clamp will hide 7.5 mm. The stack of pages is very thin, thus through paging at most another 0.75 mm will be lost. Therefore, one may write:

```
\documentclass[a4paper]{report}
\usepackage[BCOR=8.25mm]{typearea}
```

or

```
\documentclass[a4paper,BCOR=8.25mm]{report}
\usepackage{typearea}
```

when using BCOR as a global option.

When using a KOMA-Script class, the explicit loading of the typearea package can be omitted:

```
\documentclass[BCOR=8.25mm]{scrreprt}
```

The option `a4paper` could be omitted with `scrreprt`, since this is a predefined setting for all KOMA-Script classes.

If the option is only later set to a new value, one may then use, for example, the following:

```
\documentclass{scrreprt}
\KOMAoptions{BCOR=8.25mm}
```

Thus, at the loading of the `scrreprt` class standard settings will be used. When changing the setting with the use of the command `\KOMAoptions` or `\KOMAoption` a new page layout with new margins will automatically be calculated.

Please note that when using this option with one of the KOMA-Script classes as in the example above, it must be used either as a class option, or passed via `\KOMAoptions` or `\KOMAoption` after loading the class. The `typearea` package should neither be loaded explicitly with `\usepackage` when using a KOMA-Script class, nor should the option be given as an optional argument thereto. If the option is changed via `\KOMAoptions` or `\KOMAoption` after loading the package, the textblock and margins are automatically recalculated anew.

DIV=*factor*

v3.00

With the aid of the option `DIV=factor` the number of stripes into which the page is divided horizontally and vertically during the page layout construction is set. The exact construction method is found in [section 2.2](#). Of importance is that the larger the *factor*, the larger the text block and the smaller the margins. Any integer value greater than 4 is valid for *factor*. Please note that large values can lead to unfulfillment of various minor conditions in the type area, depending on further options chosen. Thus, in an extreme case, the header may fall outside of the page. Users applying the option `DIV=factor` are themselves responsible for fulfillment of the marginal conditions and setting of a typographically aesthetic line length.

In [table 2.1](#) are found the type area sizes for several *DIV* factors for an A4 page without binding correction. Here the minor conditions dependent on font size are not considered.

Table 2.1.: Type area dimensions dependent on *DIV* for A4 regardless of `\topskip`

<i>DIV</i>	Type area		Margins	
	width [mm]	height [mm]	top [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

Example: Assume one wants to write a meeting protocol, using the `protocol` class. The document should be double-sided. In the company 12pt Bookman font is used. This font, which belongs to the standard PostScript fonts, is activated in \LaTeX with the command `\usepackage{bookman}`. The Bookman font is a very wide font, meaning that the individual characters have a large width relative to their height. Therefore, the predefined value for *DIV* in `typearea` is insufficient. Instead of the value of 12 it appears after thorough study of this entire chapter that a value of 15 should be most suitable. The protocol will not be bound but punched and kept in a folder. Thus, no binding correction is necessary. One may then write:

```
\documentclass[a4paper,twoside]{protocol}
\usepackage{bookman}
\usepackage[DIV=15]{typearea}
```

On completion, it is decided that the protocols will from now on be collected and bound quarterly into book format. The binding is to be a simple glue binding, because it is only done to conform with ISO 9000 and nobody is actually going to read them. For the binding including space lost in turning the pages, an average of 12 mm is required. Thus, one may change the options of the `typearea` package accordingly, and use the class for protocols conforming to ISO 9000 regulations:

```
\documentclass[a4paper,twoside]{iso9000p}
\usepackage{bookman}
\usepackage[DIV=15,BCOR=12mm]{typearea}
```

Of course, it is equally possible to use here a KOMA-Script class:

```
\documentclass[twoside,DIV=15,BCOR=12mm]{scrartcl}
```

Table 2.2.: Predefined settings of *DIV* for A4

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

```
\usepackage{bookman}
```

The `a4paper` option can be left out when using the `scrartcl` class, as it is predefined in all KOMA-Script classes.

Please note that when using the *DIV* option with one of the KOMA-Script classes as in the example above, it must be used either as a class option, or passed via `\KOMAOPTIONS` or `\KOMAoption` after loading the class. The `typearea` package should neither be loaded explicitly with `\usepackage` when using a KOMA-Script class, nor should the option be given as an optional argument thereto. If the option is changed via `\KOMAOPTIONS` or `\KOMAoption` after loading the package, the textblock and margins are automatically recalculated anew.

```
DIV=calc
DIV=classic
```

v3.00

As already mentioned in [section 2.2](#), for A4 paper there are fixed predefined settings for the *DIV* value. These can be found in [table 2.2](#). If a different paper format is chosen, then the `typearea` package independently calculates an appropriate *DIV* value. Of course this same calculation can be applied also to A4. To obtain this result, one simply uses the `DIV=calc` option in place of the `DIV=factor` option. This option can just as easily be explicitly given for other paper formats. If one desires an automatic calculation, this also makes good sense, since the possibility exists to configure different predefined settings in a configuration file (see [section 20.3](#)). An explicit passing of the `DIV=calc` option then overwrites such configuration settings.

The classical page layout construction, the Middle Age book design canon, mentioned in [section 2.3](#), is similarly selectable. Instead of the `DIV=factor` or `DIV=calc` option, one may use the `DIV=classic` option. A *DIV* value closest to the Middle Age book design canon is then chosen.

Example: In the example using the Bookman font with the `DIV=factor` option, exactly that problem of choosing a more appropriate *DIV* value for the font arose. As a variation on that example, one could simply leave the choice of such a value to the `typearea` package:

```
\documentclass[a4paper,twoside]{protocol}
\usepackage{bookman}
\usepackage[DIV=calc]{typearea}
```

Please note that when using this option with one of the KOMA-Script classes as in the example above, it must be used either as a class option, or passed via `\KOMAoptions` or `\KOMAoption` after loading the class. The `typearea` package should neither be loaded explicitly with `\usepackage` when using a KOMA-Script class, nor should the option be given as an optional argument thereto. If the option is changed via `\KOMAoptions` or `\KOMAoption` after loading the package, the textblock and margins are automatically recalculated anew.

DIV=current
DIV=last

v3.00 Readers who have followed the examples with acuity actually already know how to calculate a *DIV* value dependent on the chosen font, when a KOMA-Script class is used together with a font package.

The problem is that the KOMA-Script class already loads the `typearea` package itself. Thus, it is not possible to pass options as optional arguments to `\usepackage`. It would also be pointless to pass the `DIV=calc` option as an optional argument to `\documentclass`. This option would be evaluated immediately on loading the `typearea` package and as a result the text block and margin would be chosen according to the L^AT_EX standard font and not for the later loaded font. However, it is quite possible to recalculate the text block and margins anew after loading the font, with the aid of `\KOMAoptions{DIV=calc}` or `\KOMAoption{DIV}{calc}`. Via `calc` an appropriate *DIV* value for a good line length is then chosen.

As it is often more practical to set the *DIV* option not after loading the font, but at a more visible point, such as when loading the class, the `typearea` package offers two further symbolic values for this option.

v3.00 With `DIV=current` a renewed calculation of text block and margin is requested, in which the currently set *DIV* will be used. This is less of interest for renewed type area calculations after loading a different font; it is rather more useful for determining, for example, after changing the leading, while keeping *DIV* the same, that the marginal condition is fulfilled that `\textheight` less `\topskip` is a multiple of `\baselineskip`.

v3.00 With `DIV=last` a renewed calculation of text block and margin is requested, where exactly the same setting is used as in the last calculation.

Example: Let us take up the previous example again, in which a good line length is required for a type area using the Bookman font. At the same time, a KOMA-Script class is to be used. This is easily possible using the symbolic value `last` and the command `\KOMAoptions`:

```
\documentclass[BCOR=12mm,DIV=calc,twoside]{scrartcl}
\usepackage{bookman}
\KOMAoptions{DIV=last}
```

If it should later be decided that a different *DIV* value is required, then only the setting of the optional argument to `\documentclass` need be changed.

Table 2.3.: Possible symbolic values for the DIV option or the *DIV* argument to `\typearea[BCOR]{DIV}`

<code>areaset</code>	Recalculate page layout.
<code>calc</code>	Recalculate type area including choice of appropriate <i>DIV</i> value.
<code>classic</code>	Recalculate type area using Middle Age book design canon (circle-based calculation).
<code>current</code>	Recalculate type area using current <i>DIV</i> value.
<code>default</code>	Recalculate type area using the standard value for the current page format and current font size. If no standard value exists, <code>calc</code> is used.
<code>last</code>	Recalculate type area using the same <i>DIV</i> argument as was used in the last call.

A summary of all possible symbolic values for the DIV option can be found in [table 2.3](#). At this point it is noted that the use of the `fontenc` package can also lead to L^AT_EX loading a different font.

Often the renewed type area calculation is required in combination with a change in the line spacing (*leading*). Since the type area should be calculated such that an integer number of lines fit in the text block, a change in the leading normally requires a recalculation of the page layout.

Example: For a thesis document, a font of size 10 pt and a spacing of 1.5 lines is required. By default, L^AT_EX sets the leading for 10 pt at 2 pt, in other words 1.2 lines. Therefore, an additional stretch factor of 1.25 is needed. Additionally, a binding correction of 12 mm is stipulated. Then the solution could be written as follows:

```
\documentclass[10pt,twoside,BCOR=12mm,DIV=calc]{scrreprt}
\linespread{1.25}
\KOMAOPTIONS{DIV=last}
```

Since `typearea` always executes the command `\normalsize` itself upon calculation of a new type area, it is not necessary to activate the chosen leading with `\selectfont` after `\linespread`, since this will be used already in the recalculation.

When using the `setspace` package (see [\[TF11\]](#)), the same example would appear as

follows:

```
\documentclass[10pt,twoside,BCOR=12mm,DIV=calc]{scrreprt}
\usepackage{setspace}
\onehalfspacing
\KOMAOptions{DIV=last}
```

As can be seen, with the use of the `setspace` package one no longer needs to know the correct stretch value.

At this point it should be noted that the line spacing for the title page should be reset to the normal value.

```
\documentclass[10pt,twoside,BCOR=12mm,DIV=calc]
{scrreprt}
\usepackage{setspace}
\onehalfspacing
\AfterTOCHead{\singlespacing}
\KOMAOptions{DIV=last}
\begin{document}
\title{Title}
\author{Markus Kohm}
\begin{spacing}{1}
\maketitle
\end{spacing}
\tableofcontents
\chapter{0k}
\end{document}
```

See further also the notes in [section 2.8](#). The command `\AfterTOCHead` will be described in [chapter 15](#) of [part II](#) on [page 353](#).

Please note that when using this option with one of the KOMA-Script classes as in the example above, it must be used either as a class option, or passed via `\KOMAOptions` or `\KOMAoption` after loading the class. The `typearea` package should neither be loaded explicitly with `\usepackage` when using a KOMA-Script class, nor should the option be given as an optional argument thereto. If the option is changed via `\KOMAOptions` or `\KOMAoption` after loading the package, the textblock and margins are automatically recalculated anew.

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

If the `DIV` option or the `BCOR` option is set after loading of the `typearea` package, then internally the command `\typearea` is called. When setting the `DIV` option the symbolic value `current` is used internally for `BCOR`, which for reasons of completeness is found also in [table 2.4](#). When setting the `BCOR` option, the symbolic value `last` is used internally for `DIV`. If it is instead desired

Table 2.4.: Possible symbolic *BCOR* arguments for `\typearea[BCOR]{DIV}`

<code>current</code>
Recalculate type area with the currently valid <i>BCOR</i> value.

that the text block and margins should be recalculated using the symbolic value *current* for *DIV*, then `\typearea[current]{current}` can be used directly.

If both *BCOR* and *DIV* need changing, then it is recommended to use `\typearea`, since then the text block and margins are recalculated only once. With `\KOMAOPTIONS{DIV=DIV,BCOR=BCOR}` the text block and margins are recalculated once for the change to *DIV* and again for the change to *BCOR*.

The command `\typearea` is currently defined so as to make it possible to change the type area anywhere within a document. Several assumptions about the structure of the \LaTeX kernel are however made and internal definitions and sizes of the kernel changed. There is a definite possibility, but no guarantee, that this will continue to function in future versions of $\text{\LaTeX} 2_{\epsilon}$. When used within the document, a page break will result.

Since `\typearea[current]{last}` or `\KOMAOPTIONS{DIV=last}` are often needed for recalculation of the type area, there exists specially the abbreviated command `\recalctypearea`.

v3.00

Example: If one finds the notation

```
\KOMAOPTIONS{DIV=last}
```

or

```
\typearea[current]{last}
```

for the recalculation of text block and margins too complicated for reasons of the many special characters, then one may use more simply the following.

```
\recalctypearea
```

```
twoside=simple switch
twoside=semi
```

As already explained in [section 2.1](#), the margin configuration is dependent on whether the document is to be typeset single- or double-sided. For single-sided typesetting, the left and right margins are equally wide, whereas for double-sided printing the inner margin of one page is only half as wide as the corresponding outer margin. In order to implement this distinction, the `typearea` package must be given the `twoside` option, if the document is to be typeset double-sided. Being a *simple switch*, any of the standard values for simple switches in [table 2.5](#) are valid. If the option is passed without a value, the value `true` is assumed, so double-sided typesetting is carried out. Deactivation of the option leads to single-sided typesetting.

Table 2.5.: Standard values for simple switches in KOMA-Script

Value	Description
<code>true</code>	activates the option
<code>on</code>	activates the option
<code>yes</code>	activates the option
<code>false</code>	deactivates the option
<code>off</code>	deactivates the option
<code>no</code>	deactivates the option

v3.00

Apart from the values in [table 2.5](#) the value `semi` can also be given. The value `semi` results in a double-sided typesetting with single-sided margins and single-sided, i. e., not alternating, margin notes. Nevertheless, since KOMA-Script version 3.12 binding corrections (see [BCOR](#), [page 31](#)) will be part of the left margin on odd pages but part of the right margin on even pages. But if you use compatibility with prior versions of KOMA-Script (see [section 2.5](#), [page 30](#)), binding correction will be part of the left margin on both pages while using `twoside=semi`.

v3.12

The option can also be passed as class option in `\documentclass`, as package option to `\usepackage`, or even after loading of the `typearea` package with the use of `\KOMAOPTIONS` or `\KOMAOPTION`. Use of the option after loading the `typearea` package results automatically in recalculation of the type area using `\recalc\typearea` (see [page 37](#)). If double-sided typesetting was active before the option was set, then before the recalculation a page break is made to the next odd page.

twocolumn=simple switch

For the calculation of a good type area with the help of `DIV=calc` it is useful to know in advance if the document is to be typeset one-column or two-column. Since the observations about line length in [section 2.1](#) then apply to each column, the width of a type area in a two-column document can be up to double that in a one-column document.

To implement this difference, the `typearea` package must be told via the `twocolumn` option whether the document is to be two-column. Since this is a *simple switch*, any of the standard values for simple switches from [table 2.5](#) is valid. If the option is passed without a value, the value `true` is assumed, i. e., two-column typesetting. Deactivation of the option results in one-column typesetting.

The option can also be passed as class option in `\documentclass`, as package option to `\usepackage`, or even after loading of the `typearea` package with the use of `\KOMAOPTIONS` or `\KOMAOPTION`. Use of the option after loading the `typearea` package results automatically in recalculation of the type area using `\recalc\typearea` (see [page 37](#)).

```
headinclude=simple switch
footinclude=simple switch
```

So far we have discussed how the type area is calculated and the relationship of the margins to one another and between margins and text block. However, one important question has not been answered: What constitutes the margins?

At first glance the question appears trivial: Margins are those parts on the right, left, top and bottom which remain empty. But this is only half the story. Margins are not always empty. There may be margin notes, for example (see `\marginpar` command in [OPHS11] or section 3.21).

One could also ask whether headers and footers belong to the upper and lower margins or to the text. This can not be answered unambiguously. Of course an empty footer or header belongs to the margins, since they can not be distinguished from the rest of the margin. A header or footer that contains only a page number¹ will optically appear more like a margin. For the optical appearance it is not important whether headers or footers are easily recognized as such during reading. Of importance is only how a well-filled page appears when viewed *out of focus*. One could use the glasses of one's far-sighted grandparents, or, lacking those, adjust one's 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.² In the 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 an empty margin. It is more difficult if the pagination is in the footer, and the length of the titles varies so that the header may appear as a margin on one page and as text on another. However, these pages should not be treated differently under any circumstances, as this would lead to vertically jumping headers. In this case it is probably best to count the header as part of 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; only the appearance when viewed out of focus is important.

The typearea package cannot make the decision whether or not to count headers and footers as part of the text body or the margin. Options `headinclude` and `footinclude` cause the header or footer to be counted as part of the text. These options, being a *simple switch*, understand the standard values for simple switches in table 2.5. One may use the options without specifying a value, in which case the value `true` is used for the *simple switch*, i.e.,

v3.00

¹Pagination refers to the indication of the page number.

²Running headings refer to the repetition of a title in titling font, which is more often typeset in the page header, less often in the page footer.

the header or footer is counted as part of the text.

Readers who are unsure about the the correct setting should re-read the above explanations. Default is usually `headinclude=false` and `footinclude=false`, but this can change depending on KOMA-Script class and KOMA-Script packages used (see [section 3.1](#) and [chapter 5](#)).

Please note that when using these options with one of the KOMA-Script classes as in the example above, they must be used either as a class option, or passed via `\KOMAOPTIONS` or `\KOMAOPTION` after loading the class. Changing of these options after loading the typearea package does not result in an automatic recalculation of the type area. Instead, the changes only take effect at the next recalculation of the type area. For recalculation of the type area, refer to the `DIV` option with the values `last` or `current` (see [page 35](#)) or the `\recalctypearea` command (see [page 37](#)).

`mpinclude=simple switch`

v2.8q

Besides documents where the head and foot are part of the text area, there are also documents where the margin-note area must be counted as part of the text body as well. The option `mpinclude` does exactly this. The option, as a *simple switch*, understands the standard values for simple switches in [table 2.5](#). One may also pass this option without specifying a value, in which case the value `true` for *simple switch* is assumed.

v3.00

The effect of `mpinclude=true` is that one width-unit of the text body is taken for the margin-note area. Using option `mpinclude=false`, the default setting, 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=true` is mainly for experts and so is 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 set up. The length can be easily enlarged with the command `\addtolength`. How much the length has to be enlarged depends on the special situation and it requires some flair. This is another reason the `mpinclude` option is primarily left for experts. Of course one can set up 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 to either not use the option `mpinclude` or to set `mpinclude` to `false`, and 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 cannot be combined with automatic calculation of the *DIV* value. In contrast `DIV=calc` (see [page 34](#)) needs `\mpinclude`.

Please note that when using this option with one of the KOMA-Script classes as in the example above, it must be used either as a class option, or passed via `\KOMAOPTIONS` or `\KOMAOPTION` after loading the class. Changing of this option after loading the `typearea` package does not result in an automatic recalculation of the type area. Instead, the changes only take effect at the next recalculation of the type area. For recalculation of the type area, refer to the `DIV` option with the values `last` or `current` (see [page 35](#)) or the `\recalctypearea` command (see [page 37](#)).

```
headlines=number of lines
headheight=height
```

We have seen how to calculate the type area using the `typearea` package and how to specify whether header and footer are part of the text or the margins. However, in particular for the header, we still have to specify the height. This is achieved with the options `headlines` and `headheight`.

v3.00

The option `headlines` is set to the number of header lines. The `typearea` package 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 margin 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 an “`overfull \vbox`” warning for each page. To prevent this from happening, the `typearea` package is told to calculate an appropriate type area:

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

If you use a KOMA-Script class, it is recommended to pass this option directly as a class option:

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

Commands that can be used to define the contents of a header with two lines are described in [chapter 5](#).

In some cases it is useful to be able to specify the header height not in lines but directly as a length measurement. This is accomplished with the aid of the alternative option `headheight`. For `height` any lengths and sizes that \LaTeX understands are valid. It should be noted though that when using a \LaTeX length such as `\baselineskip` its value at the time of the calculation of the type area and margins, not at the time of setting of the option, is decisive.

Please note that when using these options with one of the KOMA-Script classes as in the example above, they must be used either as a class option, or passed via `\KOMAOPTIONS` or

`\KOMAOption` after loading the class. Changing of these options after loading the `typearea` package does not result in an automatic recalculation of the type area. Instead, the changes only take effect at the next recalculation of the type area. For recalculation of the type area, refer to the `DIV` option with the values `last` or `current` (see page 35) or the `\recalctypearea` command (see page 37).

```
footlines=number of lines
footheight=height
\footheight
```

v3.12

As well as we needed a height value for the head, we need a height value for the page footer. But in difference to the height of the head, \LaTeX itself do not provide a length for the height of the page footer. So `typearea` defines the new length `\footheight`, if it does not exist. Whether or not this length will be used by classes or packages depends on the classes and packages, that will be used. The KOMA-Script package `scrlayer-scrpage` incorporates `\footheight` and actively cooperates with `typearea`. The KOMA-Script classes do not recognize `\footheight`, because without any package assistance they provide only page styles with single-line page footers.

You can use `footlines` to setup the *number of lines* in the page footer, similar to `headlines` for the number of lines in the page header. By default `typearea` uses 1.25 footlines. This is a compromise, large enough for overlining or underlining footers and small enough that the relative weight of the bottom margin is not affected too much when the footer is neither over- nor underlined. Thus in most cases you may leave *number of lines* at its default value and adapt it only in special cases.

Example: Assume a two-lined copyright note should be placed in the page footer. Indeed, \LaTeX itself does not test, whether or not the footer has room enough for that, exceeding of the available height would probably could result in unbalanced margins. Moreover, for example package `scrlayer-scrpage`, that may be used to define such a page footer, would definitely do such a test and would notify a recognised oversize. So it makes sense, to declare the needed `footheight` already for the calculation of the text area and the margins:

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

Again, if you use a KOMA-Script class, it is recommended to pass this option directly a class option:

```
\documentclass[footlines=2.1]{scrartcl}
```

Commands that can be used to define the contents of a footer with two lines are described in chapter 5.

In some cases it is useful to be able to specify the footer height not in lines but directly as a length measurement. This is accomplished with the aid of the alternative option `footheight`. For *height* any lengths and sizes that L^AT_EX understands are valid. It should be noted though that when using a L^AT_EX length such as `\baselineskip` its value at the time of the calculation of the type area and margins, not at the time of setting of the option, is decisive.

Please note that when using these options with one of the KOMA-Script classes as in the example above, they must be used either as a class option, or passed via `\KOMAOPTIONS` or `\KOMAOPTION` after loading the class. Changing of these options after loading the `typearea` package does not result in an automatic recalculation of the type area. Instead, the changes only take effect at the next recalculation of the type area. For recalculation of the type area, refer to the `DIV` option with the values `last` or `current` (see page 35) or the `\recalcTypearea` command (see page 37).

```
\areaset[BCOR]{width}{height}
```

So far we have seen how a good or even very good type area 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 some specified dimensions. At the same time the margins 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 margins will then be calculated automatically, taking account of the options `headinclude`, `headinclude=false`, `footinclude` and `footinclude=false` where needed. On the other hand, the options `headlines` and `headheight` are ignored!

The default of `BCOR` is 0pt. If you want to re-use the current binding correction, e.g. the value set by option `BCOR`, you can use the symbolic value `current` at the optional argument.

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, as long as the height of the

topmost line is less than `\topskip`. Thus, the uppermost line does not require any height. The descenders of characters on the lowermost line, on the other hand, hang below the dimensions 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{poetry}
\usepackage{typearea}
\areaset[1cm]{15cm}{15cm}
```

DIV=areaset

v3.00 In rare cases it is useful to be able to reconstruct the current type area anew. This is possible via the option `DIV=areaset`, where `\KOMAOPTIONS{DIV=areaset}` corresponds to the

```
\areaset[current]{\textwidth}{\textheight}
```

command. The same result is obtained if one uses `DIV=last` and the typearea was last set with `\areaset`.

The typearea package was not made to set up predefined margin values. If you have to do so you may use package `geometry` (see [Ume10]).

2.7. Paper Format Selection

The paper format is a definitive characteristic of any document. As already mentioned in the description of the supported page layout constructions (see section 2.1 to section 2.3 from page 25 onwards), the entire page division and document layout depends on the paper format. Whereas the L^AT_EX standard classes are restricted to a few formats, KOMA-Script supports in conjunction with the typearea package even exotic paper sizes.

paper=format

v3.00 The option `paper` is the central element for format selection in KOMA-Script. *Format* supports first of all the American formats `letter`, `legal`, and `executive`. In addition, it supports the ISO formats of the series A, B, C, and D, for example `A4` or — written in lowercase — `a4`.

v3.02c Landscape formats are supported by specifying the option again, this time with value `landscape` or `seascape`. The difference is that application dvips rotates at `landscape` by -90° , while it rotates by $+90^\circ$ at `seascape`. So you may use `seascape` whenever a PostScript viewer application shows landscape pages upside-down. But you may see the difference only if you do not deactivate option `pagesize`, which will be described next.

v3.01b Additionally, the *format* can also be specified in the form `height:width` or
v3.22 `width:height`. Whether the first or the second value is the *height* or *width* depends

on the paper orientation. With `paper=landscape` or `paper=seascape` the smaller value is the *height* and the larger one is the *width*. With `paper=portrait` the smaller value is the *width* and the larger one is the *height*.

Note that until version 3.01a the first value was always the *height* and the second one the *width*. From version 3.01b until version 3.21, the first value was always the *width* and the second one the *height*. This is important if you use compatibility settings (see option [version](#), [section 2.5](#), [page 30](#)).

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

```
\documentclass{article}
\usepackage[headinclude=false,footinclude=false,
            paper=A8,landscape]{typearea}
\areaset{7cm}{5cm}
\pagestyle{empty}
\begin{document}
\section*{Supported Paper Sizes}
letter, legal, executive, a0, a1 \dots\ %
b0, b1 \dots\ c0, c1 \dots\ d0, d1 \dots
\end{document}
```

If the file cards have the special format (height:width) 5 cm:3 cm, this can be achieved using the following code.

```
\documentclass{article}
\usepackage[headinclude=false,footinclude=false,%
            paper=landscape,paper=5cm:3cm]{typearea}
\areaset{4cm}{2.4cm}
\pagestyle{empty}
\begin{document}
\section*{Supported Paper Sizes}
letter, legal, executive, a0, a1 \dots\ %
b0, b1 \dots\ c0, c1 \dots\ d0, d1 \dots
\end{document}
```

As part of the predefined defaults, KOMA-Script uses A4 paper in portrait orientation. This is in contrast to the standard classes, which by default use the American letter paper format.

Please note that when using these options with one of the KOMA-Script classes, it must be used either as a class option, or passed via `\KOMAOPTIONS` or `\KOMAOPTION` after loading the class. Changing of this option after loading the `typearea` package does not result in an automatic recalculation of the type area. Instead, the changes only take effect at the next recalculation of the type area. For recalculation of the type area, refer to the `DIV` option with the values `last` or `current` (see [page 35](#)) or the `\recalctypearea` command (see [page 37](#)).

```
pagesize=output driver
```

The above-mentioned mechanisms for choice of paper format only affect the output insofar as internal L^AT_EX lengths are set. The typearea package then uses them in the division of the page into type area and margins. The specification of the DVI formats, however, does not include any indications of paper format. If printing is done directly from DVI format to a low-level printer language such as PCL or ESC/P2, this is usually not an issue since with this output also the zero-position is at the top left, identical to DVI. If, however, translation is made into a language such as PostScript or PDF, in which the zero-position is at a different point, and in which also the paper format should be specified in the output data, then this information is missing. To solve this problem, the respective drivers use a predefined paper size, which the user can change either by means of an option or via a corresponding command in the T_EX source file. When using the DVI driver dvips the information can be given in the form of a `\special` command. With pdfT_EX or VT_EX one sets instead two lengths.

With option `pagesize` you may select an output driver for writing the paper size into the destination document. Supported output drivers are listed at [table 2.6](#). The default is `pagesize`. This usage without value is same like `pagesize=auto`.

v3.17

Example: Assume that a document should be available both as a DVI data file and in PDF format for online viewing. Then the preamble might begin as follows:

```
\documentclass{article}
\usepackage[paper=A4,pagesize]{typearea}
```

If the pdfT_EX engine is used *and* PDF output is activated, then the two lengths `\pdfpagewidth` and `\pdfpageheight` are set appropriately. If, however, a DVI data file is created—regardless of whether by L^AT_EX or by pdfL^AT_EX—then a `\special` is written at the start of this data file.

It is recommended always to specify this option. Generally the method without *output driver*, or with `auto` or `automedia`, is useful.

2.8. Tips

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 1980s, it was 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 margins were kept narrow and the line spacing was increased to 1.5 for corrections. Before the advent of modern text processing systems, single-spaced would have been the only alternative—other than with T_EX. In such a single-spaced document even correction signs

Table 2.6.: Output driver for option `pagesize=output driver`

auto	Uses output driver <code>pdftex</code> if pdf _T E _X -specific registers <code>\pdfpagewidth</code> and <code>\pdfpageheight</code> are defined. In addition, output driver <code>dvips</code> will be used.
automedia	Almost the same as <code>auto</code> but if the V _T E _X -specific registers <code>\mediawidth</code> and <code>\mediaheight</code> are defined, they will be set additionally.
false, no, off	Does not set any output driver and does not send page size information to the output driver.
dvipdfmx	Writes paper size into DVI files using <code>\special{pagesize=width,height}</code> . The name of the output driver is <code>dvipdfmx</code> because application <code>dvipdfmx</code> handles such specials not only at document preamble but at the document body too.
dvips	Using this option at the document preamble sets paper size using <code>\special{pagesize=width,height}</code> . While application <code>dvips</code> cannot handle changes of paper size at the inner document pages a hack is needed to achieve such changes. Use changes of paper size after <code>\begin{document}</code> on your own risk, if you are using <code>dvips</code> !
pdftex	Sets paper size using the pdf _T E _X -specific registers <code>\pdfpagewidth</code> and <code>\pdfpageheight</code> . You may do this at any time in your document.

v3.05a

would have been difficult to add. When computers became more 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 margin sizes prescribed.

L_AT_EX already uses sufficient line spacing, and the margins are wide enough for corrections. Thus a page will look generous, even when quite full of text.

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 available. 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 cannot work and results in a fragmented and agitated-looking text. If one uses L^AT_EX for one’s paper, some of these rules have to be either ignored or at least interpreted generously. For example one may interpret “60 characters per line” not as a fixed, but as an average or maximal value.

As executed, record regulations are usually intended to obtain a usable result even if the author does not know what needs to be considered. *Usable* frequently means readable and correctable. In the author’s opinion the type area of a text set with L^AT_EX and the `typearea` package meets these criteria well right from the start. Thus if one is confronted with regulations which deviate obviously substantially from it, then the author recommends submitting an extract from the text to the responsible person and inquiring whether it is permitted to submit the work despite deviations in the format. If necessary the type area can be moderately adapted by modification of option `DIV`. The author advises against the use of `\areaset` for this purpose however. In the worst case one may make use of the `geometry` package (see [Ume10]), which is not part of KOMA-Script, or change the type area parameters of L^AT_EX. One may find the values determined by `typearea` in the `log` file of one’s document. Thus moderate adjustments should be possible. However, one should 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 prove absolutely necessary to set the text with a line spacing of 1.5, then one should not under any circumstances redefine `\baselinestretch`. Although this procedure is recommended all too frequently, it has been obsolete since the introduction of L^AT_EX 2_ε in 1994. In the worst case one may use the instruction `\linespread`. The author recommends the package `setspace` (see [TF11]), which is not part of KOMA-Script. Also one should let `typearea` recalculate a new type area after the conversion of the line spacing. However, one 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 `DIV=calc`, 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 for the respective document. Thus one can select a smaller value easily after the first L^AT_EX run.

The question is not infrequently put to the author, why he spends an entire chapter discussing type area calculations, when it would be very much simpler to merely give the world a package with which anyone can adjust the margins like in a word processor. Often it is added that such a package would in any case be the better solution, since everyone can judge for themselves how good margins are to be chosen, and that the margins calculated by KOMA-Script are anyway not that great. The author takes the liberty of translating a suitable quotation from [WF00]. One may find the original German words in the German `scrguide`.

The practice of doing things oneself is long-since widespread, but the results are often dubious because layman typographers do not see what is incorrect and cannot know what is important. Thus one becomes accustomed to incorrect and poor typography. [...] Now the objection could be made that typography is dependent on taste. If it concerned decoration, perhaps one could let that argument slip by; however, since typography is primarily concerned with information, errors cannot only irritate, 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 L^AT_EX 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 4](#)).

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](#). For example, you may replace `\documentclass{book}` by `\documentclass{scrbook}`. Normally, the document should be processed without errors by L^AT_EX, 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.

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 the size of the text area and the margins could be difficult to answer initially. On the other hand, the main business of an author — planning the document structure, writing the text, preparing figures, tables, lists, index, and other data — should be almost independent of those settings. As an author you should concentrate initially on this work. When that is done, you can concentrate on the fine points of presentation. Besides the choice of options, this means correcting hyphenation, optimizing page breaks, and the placement of tables and figures.

3.1. Early or late Selection of Options

All of what is described in [section 2.4](#) is generally applicable. So if you have already read and understood [section 2.4](#) you can switch to [section 3.2, page 53](#).

In this section a peculiarity of KOMA-Script is presented, which, apart from the classes `scrbook`, `scrreprt`, and `scrartcl` is also relevant to other KOMA-Script classes and packages. Such that the user can find all information corresponding to a single package or a single class in the relevant chapter, this section is found almost identically in several chapters. Users

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

standard class	KOMA-Script class
article	scrartcl
report	scrreprt
book	scrbook
letter	scrletter2

who are not only interested in a particular package or class, but wish to gain an overview of KOMA-Script as a whole, may read the section in one chapter and may thereafter skip it wherever coming across it in the document.

```
\documentclass[option list]{KOMA-Script class}
\usepackage[option list]{package list}
```

In L^AT_EX, provision is made for the user to pass class options as a comma-separated list of keywords as optional arguments to `\documentclass`. Apart from being passed to the class, these options are also passed on to all packages which can understand the options. Provision is also made for the user to pass optional arguments as a comma-separated list of keywords as optional arguments to `\usepackage`. KOMA-Script expands the option mechanism for the KOMA-Script classes and various packages to use further possibilities. Thus, most KOMA-Script options can also take a value. An option may have not only the form *Option*, but may also have the form *option=value*. Apart from this difference `\documentclass` and `\usepackage` function the same in KOMA-Script as described in [Tea05b] or any introduction to L^AT_EX, for example [OPHS11].

When using a KOMA-Script class, no options should be passed on loading of the `typearea` or `scrbase` packages. The reason for this is that the class already loads these packages without options and L^AT_EX refuses multiple loadings of a package with different option settings. Actually, it is no longer necessary when using any KOMA-Script class to explicitly load either one of these packages.

You should note, that in opposite to the interface described below the options interface of `\documentclass` and `\usepackage` is not robust. So commands, lengths, counters and such constructs may break inside the optional argument of these commands. Because of this, the usage of a L^AT_EX length inside the value of an option would cause an error before KOMA-Script can get the control over the option execution. So, if you want to use a L^AT_EX length, counter or command a part of the value of an option, you have to use `\KOMAOPTIONS` or `\KOMAOPTION`. These commands will be described next.

```
\KOMAOPTIONS{option list}
\KOMAOPTION{option}{value list}
```

KOMA-Script offers most class and package options the opportunity to change the value of options even after loading of the class or package. One may then change the values of a list of options at will with the `\KOMAOPTIONS` command. Each option in the *option list* has the form *option=value*.

Some options also have a default value. If one does not give a value, i. e., gives the option simply in the form *option*, then the default value will be used.

Some options can assume several values simultaneously. For such options there exists the possibility, with the help of `\KOMAOPTION`, to pass a single *option* a list of values. The individual values are given as a comma-separated *value list*.

To implement this possibility KOMA-Script uses the commands `\FamilyOptions` and `\FamilyOption` with the family “KOMA”. For more information in these commands see [part II, section 12.2, page 313](#).

3.2. Compatibility with Earlier Versions of KOMA-Script

It applies, mutatis mutandis, what is written in [section 2.5](#). So if you have already read and understood [section 2.5](#) you can switch to [page 54, page 54](#).

Those who achieve their documents in source code set utmost value to the fact that future L^AT_EX runs will yield exactly the same result. However, sometimes improvement and bug corrections of classes will result in changes of the behaviour and make-up. But sometimes this is not wanted.

```
version=value
version=first
version=last
```

v3.2.0a

Since version 2.96a of KOMA-Script it's your choice if your source code should result in the same make-up at future L^AT_EX runs or if you like to participate in all improvements of new releases of the class. You may select the compatible version of KOMA-Script with option `version`. Compatibility to the lowest supported KOMA-Script release may be achieved by `version=first` or `version=2.9` or `version=2.9t`. Setting `value` to an unknown release number will result in a warning message and selects `version=first` for safety.

With `version=last` the most recent version will be selected at every L^AT_EX run. Be warned, though, that using `version=last` poses possibilities of compatibility issues for future L^AT_EX runs. Option `version` without any `value` means the same. This is the default behaviour as long as you do not use any deprecated options.

v3.01a

If you use a deprecated option of KOMA-Script 2, KOMA-Script 3 will switch to `version=first` automatically. This will also result in a warning message that explains how to prevent this switching. Alternatively you may select another adjustment using option `version` with the wanted compatibility after the deprecated option.

Compatibility is primarily make-up compatibility. New features not related to the mark-up will be available even if you switch compatibility to a version before first implementation of the feature. Option `version` does not influence make-up changes that are motivated by bug fixes. If you need bug compatibility you should physically save the used KOMA-Script version together with your document.

Please note that you cannot change option `version` anymore after loading the class. Therefore, the usage of option `version` within the argument of `\KOMAOPTIONS` or `\KOMAOPTION` is not recommended and will cause an error.

3.3. Draft Mode

Many classes and packages provide a draft mode aside from the final typesetting mode. The difference of draft and final mode may be as manifold as the classes and package that support these modes. For instance, the `graphics` and the `graphicx` packages do not actually output the graphics in their own draft mode. Instead they output a framed box of the appropriate size containing only the graphic's file name (see [Car05]).

`draft=simple switch`

v3.00

This option is normally used to distinguish between the draft and final versions of a document. *simple switch* value may be any standard value from [table 2.5, page 39](#). In particular, switching on the option 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 default `draft=false` option no such boxes are shown. Such overly long lines often vanish using package `microtype` [Sch13].

3.4. Page Layout

Each page of a document is separated into several different layout elements, e.g., margins, head, foot, text area, margin note column, and the distances between all these elements. KOMA-Script additionally distinguishes the page as a whole also known as the paper and the viewable area of the page. Without doubt, the separation of the page into the several parts is one of the basic features of a class. Nevertheless at KOMA-Script the classes delegate that business to the package `typearea`. This package may be used with other classes too. In difference to those other classes the KOMA-Script classes load `typearea` on their own. Because of this, there's no need to load the package explicitly with `\usepackage` while using a KOMA-Script class. Nor would this make sense or be useful. See also [section 3.1](#).

Some settings of KOMA-Script classes do influence the page layout. Those effects will be documented at the corresponding settings.

For more information about page size, separation of pages into margins and type area, and about selection of one or two column typesetting see the documentation of package `typearea`. You may find it at [chapter 2](#) from [page 25](#) onwards.

`\flushbottom`
`\raggedbottom`

In double-sided documents, it's preferred to have the same visual baseline in not only the first lines of the text areas in a double-side spread, but also in the last lines. If pages consist of text without paragraphs or headlines only, this is the result in general. But a paragraph distance of half of a line would be enough to prevent achieving this, if the difference in the number of paragraphs on each page of the double-page spread is odd-numbered. In this case at least some of the paragraph

distances need to be shrunk or stretched to fit the rule again. T_EX knows shrinkable and stretchable distances for this purpose. L^AT_EX provides an automatism for this kind of vertical adjustment.

Using double-sided typesetting with option `twoside` (see [section 2.4, page 38](#)) or two-column typesetting with option `twocolumn` (see [page 39](#)) switches on vertical adjustment also. But with compatibility selection to a KOMA-Script version prior to 3.17 (see [section 3.2, page 53](#), option `version`) this is not the case, if you use `\KOMAOption` or `\KOMAoptions` to change the setting of these options.

Alternatively, vertical adjustment may be switched on at any time valid from the current page using `\flushbottom`. `\raggedbottom` would have the opposite effect, switching off vertical adjustment from the current page on. This is also the default at one-sided typesetting.

By the way, KOMA-Script uses a slightly modified kind of abdication of vertical adjustment. This has been done to move footnotes to the bottom of the text area instead of having them close to the last used text line.

3.5. Selection of the Document Font Size

The main document font size is one of the basic decisions for the document layout. The maximum width of the text area, and therefore splitting the page into text area and margins, depends on the font size as stated in [chapter 2](#). The main document font will be used for most of the text. All font variations either in mode, weight, declination, or size should relate to the main document font.

`fontsize=size`

In contrast to the standard classes and most other classes that provide only a very limited number of font sizes, the KOMA-Script classes offer the feature of selection of any desired *size* for the main document font. In this context, any well known T_EX unit of measure may be used and using a number without unit of measure means `pt`.

If you use this option inside the document, the main document font size and all dependent sizes will change from this point. This may be useful, e. g., if the appendix should be set using smaller fonts on the whole. It should be noted that changing the main font size does not result in an automatic recalculation of type area and margins (see `\recalctypearea`, [section 2.4, page 37](#)). On the other hand, each recalculation of type area and margins will be done on the basis of the current main font size. The effects of changing the main font size to other additionally loaded packages or the used document class depend on those packages and the class. This may even result in error messages or typesetting errors, which cannot be considered a fault of KOMA-Script, and even the KOMA-Script classes do not change all lengths if the main font size changes after loading the class.

This option is not intended to be a substitution for `\fontsize` (see [\[Tea05a\]](#)). Also, you should not use it instead of one of the main font depending font size commands `\tiny` up to `\Huge`! The default at `scrbook`, `scrreprt`, and `scartcl` is `fontsize=11pt`. In contrast, the default

of the standard classes would be 10pt. You may attend to this if you switch from a standard class to a KOMA-Script class.

3.6. Text Markup

L^AT_EX offers different possibilities for logical and direct markup of text. Selection of the font family commands, as well as choosing the font size and width is supported. More information about the standard font facilities may be found at [OPHS11], [Tea05b], and [Tea05a].

```
\textsuperscript{Text}
\textsubscript{Text}
```

The L^AT_EX-Kern already defines the command `\textsuperscript` to superscript text. Unfortunately, until release 2015/01/01 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 C₆H₁₂O₆ and O₂ to produce H₂O and CO₂. 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{#1\textsubscript{#2}}}
```

Now the output in the whole document changes to:

The cell produces its energy partly from reaction of C₆H₁₂O₆ and O₂ to produce H₂O and CO₂. 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 ”.

```
\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`, and `\scshape`, as well as the font size commands `\Huge`, `\huge`, `\LARGE`, `\Large`, `\large`, `\normalsize`, `\small`, `\footnotesize`, `\scriptsize`, and `\tiny`. The description of these commands can be found in [OPHS11], [Tea05b], or [Tea05a]. Color switching commands like `\normalcolor` (see [Car05] and [Ker07]) are also acceptable. The behavior when using other commands, especially those that make redefinitions 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 inside the document body, but only in the document preamble. Usage examples can be found in the paragraphs on the corresponding element. Names and meanings of the individual items are listed in table 3.2. The default values are shown in the corresponding paragraphs.

With command `\usekomafont` the current font style may be changed into the font style of the selected *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 3.2.: Elements whose type style can be changed with the KOMA-Script command `\setkomafont` or `\addtokomafont`

v3.12	<p>author</p> <p>author of the document on the main title, i.e., the argument of <code>\author</code> when <code>\maketitle</code> will be used (see section 3.7, page 65)</p>
	<p>caption</p> <p>text of a table or figure caption (see section 3.20, page 123)</p>
	<p>captionlabel</p> <p>label of a table or figure caption; used according to the element <code>caption</code> (see section 3.20, page 123)</p>
	<p>chapter</p> <p>title of the sectional unit <code>\chapter</code> (see section 3.16, page 95)</p>
	<p>chapterentry</p> <p>table of contents entry of the sectional unit <code>\chapter</code> (see section 3.9, page 72)</p>
	<p>chapterentrypagenumber</p> <p>page number of the table of contents entry of the sectional unit <code>\chapter</code>, variation on the element <code>chapterentry</code> (see section 3.9, page 72)</p>
	<p>chapterprefix</p> <p>chapter number line at setting <code>chapterprefix=true</code> or <code>appendixprefix=true</code> (see section 3.16, page 91)</p>
v3.12	<p>date</p> <p>date of the document on the main title, i.e., the argument of <code>\date</code> when <code>\maketitle</code> will be used (see section 3.7, page 65)</p>
v3.12	<p>dedication</p> <p>dedication page after the main title, i.e., the argument of <code>\dedication</code> when <code>\maketitle</code> will be used (see section 3.7, page 67)</p>
	<p>descriptionlabel</p> <p>labels, i.e., the optional argument of <code>\item</code> in the <code>description</code> environment (see section 3.18, page 113)</p>
	<p>dictum</p> <p>dictum, wise saying, or smart slogan (see section 3.17, page 109)</p>

Table 3.2.: Elements whose type style can be changed (*continuation*)

dictumauthor	author of a dictum, wise saying, or smart slogan; used according to the element dictum (see section 3.17 , page 109)
dictumtext	another name for dictum
disposition	all sectional unit titles, i. e., the arguments of \part down to \subparagraph and \minisec , including the title of the abstract; used before the element of the corresponding unit (see section 3.16 ab page 90)
footnote	footnote text and marker (see section 3.14 , page 85)
footnotelabel	mark of a footnote; used according to the element footnote (see section 3.14 , page 85)
footnotereference	footnote reference in the text (see section 3.14 , page 85)
footnoterule	horizontal rule above the footnotes at the end of the text area (see section 3.14 , page 88)
labelinglabel	labels, i. e., the optional argument of \item in the labeling environment (see section 3.18 , page 114)
labelingseparator	separator, i. e., the optional argument of the labeling environment; used according to the element labelinglabel (see section 3.18 , page 114)
minisec	title of \minisec (see section 3.16 ab page 100)
pagefoot	only used if package scrlayer-scrpage has been loaded (see chapter 5 , page 234)

Table 3.2.: Elements whose type style can be changed (*continuation*)

`pagehead`

another name for `pageheadfoot`

`pageheadfoot`

the head of a page, but also the foot of a page (see [section 3.12](#) ab [page 76](#))

`pagenumber`

page number in the header or footer (see [section 3.12](#))

`pagination`

another name for `pagenumber`

`paragraph`

title of the sectional unit `\paragraph` (see [section 3.16](#), [page 95](#))

`part`

title of the `\part` sectional unit, without the line containing the part number (see [section 3.16](#), [page 95](#))

`partentry`

table of contents entry of the sectional unit `\part` (see [section 3.9](#), [page 72](#))

`partentrypagenumber`

Page number of the table of contents entry of the sectional unit `\part` variation on the element `partentry` (see [section 3.9](#), [page 72](#))

`partnumber`

line containing the part number in a title of the sectional unit `\part` (see [section 3.16](#), [page 95](#))

`publishers`

publishers of the document on the main title, i.e., the argument of `\publishers` when `\maketitle` will be used (see [section 3.7](#), [page 65](#))

`section`

title of the sectional unit `\section` (see [section 3.16](#), [page 95](#))

`sectionentry`

table of contents entry of sectional unit `\section` (only available in `scrartcl`, see [section 3.9](#), [page 72](#))

Table 3.2.: Elements whose type style can be changed (*continuation*)

sectionentrypagenumber

page number of the table of contents entry of the sectional unit `\section`, variation on element `sectionentry` (only available in `scrartcl`, see [section 3.9](#), [page 72](#))

sectioning

another name for `disposition`

subject

categorization of the document, i. e., the argument of `\subject` on the main title page (see [section 3.7](#), [page 65](#))

subparagraph

title of the sectional unit `\subparagraph` (see [section 3.16](#), [page 95](#))

subsection

title of the sectional unit `\subsection` (see [section 3.16](#), [page 95](#))

subsubsection

title of the sectional unit `\subsubsection` (see [section 3.16](#), [page 95](#))

subtitle

subtitle of the document, i. e., the argument of `\subtitle` on the main title page (see [section 3.7](#), [page 65](#))

title

main title of the document, i. e., the argument of `\title` (for details about the title size see the additional note in the text of [section 3.7](#) from [page 65](#))

titlehead

head above the main title of the document, i. e., the argument of `\titlehead` when `\maketitle` will be used (see [section 3.7](#), [page 65](#))

```

\usefontofkomafont{element}
\useencodingofkomafont{element}
\usesizeofkomafont{element}
\usefamilyofkomafont{element}
\useseriesofkomafont{element}
\useshapeofkomafont{element}

```

v3.12

Sometimes and despite the recommendation users use the font setting feature of elements not only for font settings but for other settings too. In this case it may be useful to switch only to the font setting of an element but not to those other settings. You may use `\usefontofkomafont` in such cases. This will activate the font size and baseline skip, the font encoding, the font family, the font series, and the font shape of an element, but no further settings as long as those further settings are local.

You may also switch to one of those attributes only using one of the other commands. Note, that `\usesizeofkomafont` will activate both, the font size and the baseline skip.

You should not misunderstand these commands as a legitimization of using all kind of commands at the font setting of an element. Hence this would result in errors sooner or later (see [section 21.3, page 444](#)).

3.7. Document Titles

In general we distinguish two kinds of document titles. First known are title pages. In this case the document title will be placed together with additional document information, like the author, on a page of its own. Besides the main title page, several further title pages may exist, like the half-title or bastard title, publisher data, dedication, or similar. The second known kind of document title is the in-page title. In this case, the document title is placed at the top of a page and specially emphasized, and may be accompanied by some additional information too, but it will be followed by more material in the same page, for instance by an abstract, or the table of contents, or even a section.

```

titlepage=simple switch
titlepage=firstiscover

```

v3.00

Using `\maketitle` (see [page 63](#)), this option switches between document title pages and in-page title. For *simple switch*, any value from [table 2.5, page 39](#) may be used.

The option `titlepage=true` or `titlepage` makes L^AT_EX use separate pages for the titles. Command `\maketitle` sets these pages inside a `titlepage` environment and the pages normally have neither header nor footer. In comparison with standard L^AT_EX, KOMA-Script expands the handling of the titles significantly.

The option `titlepage=false` 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.

v3.12

The third choice, `titlepage=firstiscover` does not only select title pages. It additionally prints the first title page of `\maketitle`, this is either the extra title or the main title, as a cover page. Every other setting of option `titlepage` would cancel this setting. The margins of the cover page are given by `\coverpagetopmargin`, `\coverpagebottommargin`, `\coverpageleftmargin` und `\coverpagerightmargin`. The defaults of these depend on the lengths `\topmargin` and `\evensidemargin` and can be changed using `\renewcommand`.

The default of classes `scrbook` and `scrreprt` is usage of title pages. Class `scartcl`, on the other hand, uses in-page titles as default.

```
\begin{titlepage}...\end{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—and in single column mode. 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 pages provided by `\maketitle`, that will be described next; 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}
```

It’s simple, isn’t it?

```
\maketitle[page number]
```

While the the standard classes produce at least one 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 count 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 3.12, page 79](#).

The following commands do not lead immediately to the ship-out of the titles. The typesetting and ship-out of the title pages are always done by `\maketitle`. By the way, you should note that `\maketitle` should not be used inside a `titlepage` environment. Like shown in the examples, one should use either `\maketitle` or `titlepage` only, but not both.

The commands explained below only define the contents of the title pages. Because of this, they have to be used before `\maketitle`. It is, however, not necessary and, when using, e.g., the `babel` package, not recommended to use these in the preamble before `\begin{document}` (see [\[BB13\]](#)). 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 horizontally 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{title head}
\subject{subject}
\title{title}
\subtitle{subtitle}
\author{author}
\date{date}
\publishers{publisher}
\and
\thanks{footnote}
```

The contents of the full title page are defined by seven elements. The output of the full title page occurs as part of the title pages of `\maketitle`, whereas the now listed elements only define the corresponding elements.

The *title head* is defined with the command `\titlehead`. It is typeset with the font of the homonymous element in regular justification and full width at the top of the page. It can be freely designed by the user.

The *subject* is output with the font of the homonymous element immediately above the *title*.

v2.8p

The *title* is output with a very large font size. Beside all other element the font size is, however, not affected by the font switching element `\title` (see [table 3.4, page 66](#)).

v2.97c

The *subtitle* is output with the font of the homonymous element just below the title.

Below the *subtitle* appears the *author*. Several authors can be specified in the argument of `\author`. They should be separated by `\and`. The font of element `author` is used for the output of the authors.

Below the author or authors appears the date in the font of the homonymous element. The default value is the present date, as produced by `\today`. The `\date` command accepts arbitrary information — 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 instead of centering it. Then it is to be considered equivalent to the title head. However, note that this field is put above any existing footnotes. The font of element `publishers` is used for the output.

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. Note, that `\thanks` has to be used inside the argument of another command, e.g., at the argument *author* of the command `\author`.

Table 3.3.: Font defaults for the elements of the title

Element name	Default
author	\Large
date	\Large
dedication	\Large
publishers	\Large
subject	\normalfont\normalcolor\bfseries\Large
subtitle	\usekomafont{title}\large
title	\usekomafont{disposition}
titlehead	

v3.12

While printing the title elements the equal named font switching elements will be used for all them. The defaults, that may be found in [table 3.3](#), may be changed using the commands `\setkomafont` and `\addtokomafont` (see [section 3.6](#), [page 57](#)).

With the exception of `titlehead` and possible footnotes, all the items are centered horizontally. The information is summarised in [table 3.4](#). Please note, that for the main title `\huge` will be used after the font switching element `title`. So you cannot change the size of the main title using `\setkomafont` or `\addtokomafont`.

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}
\usepackage[english]{babel}
\begin{document}
\titlehead{{\Large Unseen University
\hfill SS~2002\\}
Higher Analytical Institute\\}
```

Table 3.4.: Font 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	Orientation
Title head	\titlehead	\usekomafont{titlehead}	justified
Subject	\subject	\usekomafont{subject}	centered
Title	\title	\usekomafont{title}\huge	centered
Subtitle	\subtitle	\usekomafont{subtitle}	centered
Authors	\author	\usekomafont{author}	centered
Date	\date	\usekomafont{date}	centered
Publishers	\publishers	\usekomafont{publishers}	centered

```

    Mythological Rd\\
    34567 Etherworld}
\subject{Dissertation}
\title{Digital space simulation with the DSP\,56004}
\subtitle{Short but sweet?}
\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.

Nevertheless, since KOMA-Script 3.12 the first title page of `\maketitle` can be printed as a cover page with different margins. For more information about this see the description of option `titlepage=firstiscover` on page 62.

```

\uppertitleback{titlebackhead}
\lowertitleback{titlebackfoot}

```

With the standard classes, the back of the title page of a double-side print 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 given by the font of the homonymous element. The font can be changed using command `\setkomafont` or `\addtokomafont` (see section 3.6, page 56). 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}
\usepackage[english]{babel}
\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.

3.8. Abstract

Particularly with articles, more rarely with reports, there is a summary directly under the title and before the table of contents. When using an in-page title, this summary is normally a kind of left- and right-indented block. In contrast to this, a kind of chapter or section is printed using title pages.

abstract=*simple switch*

In the standard classes the **abstract** environment sets the text “Abstract” centered before the summary text. This was normal practice in the past. 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 **abstract**. For *simple switch*, any value from [table 2.5, page 39](#) may be used.

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.

scartcl,
scrreprt

```
\begin{abstract}...\end{abstract}
```

Some L^AT_EX classes offer a special environment for this summary, the `abstract` environment. This is output directly, as 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 option `abstract` (see above).

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. See commands `\chapter*` and `\addchap` or `\addchap*` at [section 3.16](#) from [page 99](#) onwards.

When using an in-page title (see option `titlepage`, [section 3.7](#), [page 62](#)), the abstract is set using the environment `quotation` (see [section 3.18](#), [page 116](#)) internally. Thereby paragraphs will be set with indentation of the first line. If that first paragraph of the abstract should not be indented, this indent may be disabled using `\noindent` just after `\begin{abstract}`.

3.9. Table of Contents

The table of contents is normally set after the document title and an optional existing abstract. Often one may find additional lists of floating environments, e. g., the list of tables and the list of figures, after the table of contents (see [section 3.20](#)).

```
toc=selection
```

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 `selection listof` 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 [\[Lin01\]](#)) or the `floatrow` (see [\[Lap08\]](#)). All these lists are unnumbered, since they contain entries that reference other sections of the document. If one wants to ignore this general agreement, one may use `selection listofnumbered`.

The option `index=totoc` 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. Despite objection of the author, KOMA-Script does also support to ignore this general agreement using `toc=indexnumbered`.

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 `toc=bibliographynumbered` has this effect, including the generation of the corresponding entry

v3.00

v3.18

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 `toc=bibliography`.

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 default setup is selected with the option `toc=graduated`.

v3.00

When there are many sections, the corresponding numbering tends to become very wide, so that the reserved field overflows. The German FAQ [Wik] 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 `toc=flat` 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.

The [table 3.5](#) shows an overview of possible values for *selection* of `toc`.

Table 3.5.: Possible values of option `toc` to set form and contents of the table of contents

bibliography, bib	
The bibliography will be represented by an entry at the table of contents, but will not be numbered.	
bibliographynumbered, bibnumbered, numberedbibliography, numberedbib	
The bibliography will be represented by an entry at the table of contents and will be numbered.	
chapterentrywithdots, chapterentrydotfill	
v3.15	The chapter entries of classes <code>scrbook</code> and <code>screpr</code> also use dots to separate the headings text from the page number.
chapterentrywithoutdots, chapterentryfill	
v3.15	The chapter entries of classes <code>scrbook</code> and <code>screpr</code> use white space to separate the headings text from the page number.
flat, left	
The table of contents will be set in table form. The numbers of the headings will be at the first column, the heading text at the second column, and the page number at the third column. The amount of space needed for the numbers of the headings will be determined by the detected needed amount of space at the previous <code>L^AT_EX</code> run.	

Table 3.5.: Possible values of option `toc` (*continuation*)

	graduated, indent, indented
	The table of contents will be set in hierarchical form. The amount of space for the heading numbers is limited.
	index, idx
	The index will be represented by an entry at the table of contents, but will not be numbered.
	indexnumbered
v3.18	The index will be represented by an entry at the table of contents and will be numbered.
	listof
	The lists of floating environments, e. g., the list of figures and the list of tables, will be represented by entries at the table of contents, but will not be numbered.
	listofnumbered, numberedlistof
	The lists of floating environments, e. g., the list of figures and the list of tables, will be represented by entries at the table of contents and will be numbered.
	nobibliography, nobib
	The bibliography will not be represented by an entry at the table of contents.
	noindex, noidx
	The index will not be represented by an entry at the table of contents.
	nolistof
	The lists of floating environments, e. g., the list of figures and the list of tables, will not be represented by entries at the table of contents.
	sectionentrywithdots, sectionentrydotfill
v3.15	The section entries of class <code>scrartcl</code> also use dots to separate the headings text from the page number.
	sectionentrywithoutdots, sectionentryfill
v3.15	The section entries of class <code>scrartcl</code> use white space to separate the headings text from the page number.

```
chapterentrydots=simple switch
sectionentrydots=simple switch
```

v3.15
scrbook,
scrreprt
scartcl

These options configure a dotted separation line between the text and the page number of the chapter entries of classes `scrbook` and `scrreprt`, or of the section entries of class `scartcl` in the table of contents. For *simple switch*, any value from [table 2.5, page 39](#) may be used. The default is `false`. It selects an empty gap instead of dots.

If a dotted line is selected, you can change their font using element `chapterentrydots` or `sectionentrydots`. The font also depends on the element of the page number of the entry (see also [section 3.6, page 57](#) and [table 3.2, page 58](#)). The defaults of the elements are shown in [table 3.6](#). Please note that the dots of all entries are equally aligned only if all dots use the same font.

`\tableofcontents`

The production of the table of contents is done by the `\tableofcontents` command. To get a correct table of contents, at least two L^AT_EX runs are necessary after every change. The contents and the form of the table of contents may be influenced with the above described option `toc`. After changing the settings of this option, at least two L^AT_EX runs are needed again.

The entry for the highest sectional unit below `\part`, i.e., `\chapter` with `scrbook` and `scrreprt` or `\section` with `scartcl` is not indented. 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. The font style is, however, affected by the settings of the element `partentry`, and for classes `scrbook` and `scrreprt` by `chapterentry`, and for class `scartcl` by `sectionentry`. The font style of the page numbers may be set dissenting from these elements using `partentrypagenumber` and `chapterentrypagenumber` respectively `sectionentrypagenumber` (see [section 3.6, page 57](#), and [table 3.2, page 58](#)). If the optional dots of the entries of `\chapter` or `\section` are used, you can change their font using element `chapterentrydots` or `sectionentrydots`. The font also depends on the element of the page number of the entry (see also [section 3.6, page 57](#) and [table 3.2, page 58](#)). The defaults of the elements are shown in [table 3.6](#). Please note that the dots of all entries are aligned same only if all dots use the same font.

v2.97c

`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 `scartcl`. 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 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.

Table 3.6.: Font style defaults of the elements of the table of contents

Element	Default font style
<code>partentry</code>	<code>\usekomafont{disposition}\large</code>
<code>partentrypagenumber</code>	
<code>chapterentry</code>	<code>\usekomafont{disposition}</code>
<code>chapterentrydots</code>	<code>\normalfont</code>
<code>chapterentrypagenumber</code>	
<code>sectionentry</code>	<code>\usekomafont{disposition}</code>
<code>sectionentrydots</code>	<code>\normalfont</code>
<code>sectionentrypagenumber</code>	

scartcl Please note that for `\part` the values of `tocdepth` and `secnumdepth` (see section 3.16, page 106) are not the same in `scartcl`. Therefore, you should not use `\partnumdepth` to set the value of `tocdepth`.

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

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

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

```
\documentclass{scartcl}
\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`.

3.10. Paragraph Markup

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 to 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 extremely 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.

parskip=*manner*

Once in a while there are requests for a document layout with vertical inter-paragraph spacing instead of indentation. The KOMA-Script classes provide with option `parskip` several capabilities to use inter-paragraph spacing instead of paragraph indent. The *manner* consists of two elements. The first element is either `full` or `half`, meaning the space amount of one line or only half of a line. The second element is “*”, “+”, or “-”, and may be omitted. Without the second element the last line of a paragraph will end with white space of at least 1em. With the plus character as second element the white space amount will be a third, and with the asterisk a fourth, of the width of a normal line. The minus variant does not take care about the white space at the end of the last line of a paragraph.

The setting may be changed at any place inside the document. In this case the command `\selectfont` will be called implicitly. The change will be valid and may be seen from the next paragraph.

Besides the resulting eight possible combinations for *manner*, the values for simple switches shown at [table 2.5, page 39](#) may be used. Switching on the option would be the same as using `full` without annex and therefore will result in inter-paragraph spacing of one line with at least 1em white space at the end of the last line of each paragraph. Switching off the options would reactivate the default of 1em indent at the first line of the paragraph instead of paragraph spacing. All the possible values of option `parskip` are shown in [table 3.7](#).

Table 3.7.: Possible values of option `parskip` to select the paragraph mark

<code>false</code> , <code>off</code> , <code>no</code>
paragraph indentation instead of vertical space; the last line of a paragraph may be arbitrarily filled

Table 3.7.: Possible values of option `parskip` (*continuation*)

<code>full, true, on, yes</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; the last line of a paragraph may be arbitrarily filled
<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; 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 1 em free space in the last line of a paragraph
<code>half-</code>	one line vertical space between paragraphs
<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>	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>never</code>	there will be no inter-paragraph spacing even if additional vertical spacing is needed for the vertical adjustment with <code>\flushbottom</code>

All eight `full` and `half` option values 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 `parskip=false`. In this case, there is no

spacing between paragraphs, only an indentation of the first line by 1 em.

3.11. Detection of Odd and Even Pages

In double-sided documents we distinguish left and right pages. Left pages always have an even page number, right pages always have an odd page number. Thus, they are most often referred to as even and odd pages in this guide. This also means that the detection of a left or right page is same as detection of even and odd page numbers.

There's no distinction in left and right pages in single-sided documents. Nevertheless there are pages with even or odd page numbers.

```
\ifthispageodd{true part}{false part}
```

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 part* argument is executed only if the command falls on an odd page. Otherwise the *false part* argument is executed.

Example: Assume that you simply want to show whether a text will be placed onto an even or odd page. You may achieve that using

```
This page has an \ifthispageodd{odd}{even}
page number.
```

which will result in the output

```
This page has an even 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.

At [section 21.1, page 441](#) experts may find more information about the problems detecting left and right pages or even and odd page number.

3.12. Head and Foot Using Predefined Page Styles

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

```
headsepline=simple switch
footsepline=simple switch
```

v3.00

In order to have or not to have a rule separating the header from the text body, use the option `headsepline` with any value shown in [table 2.5, page 39](#). Activation of the option will result in such a separation line. Similarly, activation of option `footsepline` switches on a rule above the foot line. Deactivation of any of the options will deactivate the corresponding rule.

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 does not 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. KOMA-Script behaves similar to `footinclude` using `footsepline`. Package `scrlayer-scrpage` (see [chapter 5](#)) adds additional features to this.

```
\pagestyle{page style}
\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.

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 `screpr` 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. `scartcl` 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.

`scrbook`,
`screpr`

`scartcl`

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 into consideration.

Beyond this KOMA-Script classes support rules below the head and above the foot using options `headsepline` and `footsepline` which are described above.

Table 3.8.: Default values for the elements of a page style

Element	Default value
pagefoot	
pageheadfoot	\normalfont\normalcolor\slshape
pagenumber	\normalfont\normalcolor

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

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.

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. If one uses `\pagestyle` just before a command, that results in an implicit page break and if the new page style should be used at the resulting new page first, a `\cleardoublepage` just before `\pagestyle` will be useful. But 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{\chapterpagestyle}` 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 `sclayer-scrpage` package, but instead via special instructions. The page styles **headings** and **myheadings** should not be used together with this package.

v2.8p

In order to change the font style used in the header, footer, or for the page number, please use the interface described in [section 3.6, page 57](#). The same element is used for header and footer, which you can designate with `pageheadfoot`. The element for the page number within the header or footer is called `pagenumber`. The element `pagefoot`, that is additionally supported by the KOMA-Script classes, will be used only if a page style has been defined that has text at the foot line, using package `sclayer-scrpage` (see [chapter 5, page 234](#)).

The default settings can be found in [table 3.8](#).

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{pageheadfoot}{%
  \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{pagehead}{\small}
```

As you can see, the last example uses the element `pagehead`. You can achieve the same result using `pageheadfoot` instead (see [table 3.2](#) on [page 58](#)).

It is not possible to use these methods to force capitals to be used automatically for the running headings. For that, please use the `scrlayer-scrpage` package (see [chapter 5](#), [page 244](#)).

If you define your own page styles, the commands `\usekomafont{pageheadfoot}`, `\usekomafont{pagenumber}`, and `\usekomafont{pagefoot}` can be useful. If you do not use the KOMA-Script package `scrlayer-scrpage` (see [chapter 5](#)) for that, but, for example, the package `fancyhdr` (see [\[vO04\]](#)), you can use these commands in your definitions. Thereby you can remain 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 package `scrlayer-scrpage` takes care to keep the maximum possible compatibility with other packages itself.

```
\markboth{left mark}{right mark}
\markright{right mark}
```

With page style `myheadings`, there's no automatic setting of the running head. Instead of this one would set it with the help of commands `\markboth` and `\markright`. Thereby *left mark* normally will be used at the head of even pages and *right mark* at the heads of odd pages. With one-sided printing, only the *right mark* exists. Using package `scrlayer-scrpage`, the additional command `\markleft` exists.

The commands may be used with other page styles too. Combination with automatic running head, e. g., with page style `headings`, limits the effect of the commands until the next automatic setting of the corresponding marks.

```
\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 `scartcl`. The default value for all four cases is `plain`. The meaning of these macros can be taken from [table 3.9](#). The page styles can be redefined with the `\renewcommand` macro.

Table 3.9.: Macros to set up page style of special pages

<code>\titlepagestyle</code>	Page style for a title page when using <i>in-page</i> titles.
<code>\partpagestyle</code>	Page style for the pages with <code>\part</code> titles.
<code>\chapterpagestyle</code>	Page style for the first page of a chapter.
<code>\indexpagestyle</code>	Page style for the first page of the index.

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 77](#), 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 `scrlayer-scrpage` (see [chapter 5](#)). 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 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 do not need to know the current page style before the change to switch back at the end.

```
\clearpage
\begin{group}
```


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

Table 3.10.: Available numbering styles of page numbers

```
\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}
```

should read more about the background of `\rightmark` at [section 21.1, page 441](#).

\pagenumbering{numbering style}

This command works the same way in KOMA-Script as in the standard classes. More precisely it is a feature neither of the standard classes nor of the KOMA-Script classes but of the L^AT_EX 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. It is recommended to use `\cleardoubleoddpge` to close the last page and start a new odd page before. The possible settings can be found in [table 3.10](#).

Using the command `\pagenumbering` also resets the page counter. Thus the page number of the next page which T_EX outputs will have the number 1 in the style *numbering style*.

3.13. Interleaf Pages

Interleaf pages are pages that are intended to stay blank. Originally these pages were really completely white. L^AT_EX, on the other hand, by default sets those pages with the current valid page style. So those pages may have a head and a pagination. KOMA-Script provides several extensions to this.

Interleaf pages may be found in books mostly. Because chapters in books commonly start on odd pages, sometimes a left page without contents has to be added before. This is also the

reason that interleaf pages only exist in double-sided printing. The unused back sides of the one-sided printings are not interleaf pages, really, although they may seem to be such pages.

```
cleardoublepage=page style
cleardoublepage=current
```

v3.00

With this option, you may define the page style of the interleaf pages created by the `\cleardoublepage`, `\cleardoubleoddpage`, or `\cleardoubleevenpage` to break until the wanted page. Every already defined *page style* (see [section 3.12](#) from [page 76](#) and [chapter 5](#) from [page 225](#)) may be used. Besides this, `cleardoublepage=current` is valid. This case is the default until KOMA-Script 2.98c and results in interleaf page without changing the page style. Since KOMA-Script 3.00 the default follows the recommendation of most typographers and has been changed to blank interleaf pages with page style `empty` unless you switch compatibility to an earlier version (see option [version](#), [section 3.2](#), [page 53](#)).

v3.00

Example: Assume you want interleaf pages almost empty but with pagination. This means you want to use page style `plain`. You may use following to achieve this:

```
\KOMAoptions{cleardoublepage=plain}
```

More information about page style `plain` may be found at [section 3.12](#), [page 78](#).

```

\clearpage
\cleardoublepage
\cleardoublepageusingstyle{page style}
\cleardoubleemptypage
\cleardoubleplainpage
\cleardoublestandardpage
\cleardoubleoddpager
\cleardoubleoddpagerusingstyle{page style}
\cleardoubleoddpageremptypage
\cleardoubleoddpagerplainpage
\cleardoubleoddpagerstandardpage
\cleardoubleevenpage
\cleardoubleevenpageusingstyle{page style}
\cleardoubleevenemptypage
\cleardoubleevenplainpage
\cleardoubleevenstandardpage

```

The L^AT_EX kernel 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 2.4, page 38](#)) starts a new right-hand page. An empty left page in the current page style is output if necessary.

v3.00

With `\cleardoubleoddpagerstandardpage`, KOMA-Script works as described above. The `\cleardoubleoddpagerplainpage` command changes the page style of the empty left page to `plain` in order to suppress the running head. Analogously, the page style `empty` is applied to the empty page with `\cleardoubleoddpageremptypage`, suppressing the page number as well as the running head. The page is thus entirely empty. If another *page style* is wanted for the interleaf page it may be set with the argument of `\cleardoubleoddpagerusingstyle`. Every already defined *page style* (see [chapter 5](#)) may be used.

Sometimes chapters should not start on the right-hand page but the left-hand page. This is in contradiction to the classic typography; nevertheless, it may be suitable, e.g., if the double-page spread of the chapter start is of special contents. KOMA-Script therefore provides the commands `\cleardoubleevenstandardpage`, `\cleardoubleevenplainpage`, `\cleardoubleevenemptypage`, and `\cleardoubleevenpageusingstyle`, which are equivalent to the odd-page commands.

However, the approach used by the KOMA-Script commands `\cleardoublestandardpage`, `\cleardoubleemptypage`, `\cleardoubleplainpage`, and `\cleardoublepageusingstyle` is dependent on the option `cleardoublepage` described above and is similar to one of the corresponding commands above. The same is valid for the standard command `\cleardoublepage`, that may be either `\cleardoubleoddpager` or `\cleardoubleevenpage`.

Example: Assume you want to set next in your document a double-page spread with a picture at the left-hand page and a chapter start at the right-hand page. The picture should have the same size as the text area without any head line or pagination. If the last chapter ends with a left-hand page, an interleaf page has to be added, which should be completely empty.

First you will use

```
\KOMAOptions{cleardoublepage=empty}
```

to make interleaf pages empty. You may use this setting at the document preamble already. As an alternative you may set it as the optional argument of `\documentclass`.

At the relevant place in your document, you'll write:

```
\cleardoubleevenemptypage
\thispagestyle{empty}
\includegraphics[width=\textwidth,%
                  height=\textheight,%
                  keepaspectratio]%
                  {picture}
\chapter{Chapter Headline}
```

The first of these lines switches to the next left page. If needed it also adds a completely blank right-hand page. The second line makes sure that the following left-hand page will be set using page style `empty` too. From third down to sixth line, an external picture of wanted size will be loaded without deformation. Package `graphicx` will be needed for this command. The last line starts a new chapter on the next page which will be a right-hand one.

The commands `\cleardoubleoddpag` respective `\cleardoubleevenpag` leads to the next odd respectively even page. The page style of an interleaf page will be set depending on option `cleardoublepag`.

3.14. Footnotes

KOMA-Script, unlike the standard classes, provides features for configuration of the footnote block format.

Table 3.11.: Available values for option `footnotes` setting up footnotes

<code>multiple</code>	At sequences of immediately following footnote marks, consecutive marks will be separated by <code>\multifootsep</code> .
<code>nomultiple</code>	Immediately following footnotes will be handled like single footnotes and not separated from each other.

`footnotes=setting`

v3.00

Footnotes will be marked with a tiny superscript number in text by default. If more than one footnote falls at the same place, one may think that it is only one footnote with a very large number instead of multiple footnotes (i. e., footnote 12 instead of footnotes 1 and 2). Using `footnotes=multiple` will separate multiple footnotes immediately next to each other by a separator string. The predefined separator at `\multifootsep` is a single comma without space. The whole mechanism is compatible with package `footmisc`, Version 5.3d (see [Fai11]). It is related not only to footnotes placed using `\footnote`, but `\footnotemark` too.

Command `\KOMAOPTIONS` or `\KOMAOPTION` may be used to switch back to the default `footnotes=nomultiple` at any time. If any problems using another package that influences footnotes occur, it is recommended not to use the option anywhere and not to change the *setting* anywhere inside the document.

A summary of the available *setting* values of `footnotes` may be found at [table 3.11](#).

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

Similar to the standard classes, footnotes in KOMA-Script are produced with the `\footnote` command, or alternatively the paired usage 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 L^AT_EX no choice but to break the footnote onto the next page. KOMA-Script, unlike the standard classes, can recognize and separate consecutive footnotes automatically. See the previously documented option `footnotes` for this.

v3.00

If you want to set the separator manually, you may use `\multiplefootnoteseparator`. Note that this command should not be redefined, because it has been defined not only to be the separator string but also the type style, i. e., font size and superscript. The separator string without type style may be found at `\multifootsep`. The predefined default is

```
\newcommand*{\multfootsep}{,}
```

and may be changed by redefining the command.

Example: Assume you want to place two footnotes following a single word. First you may try

```
Word\footnote{1st footnote}\footnote{2nd footnote}
```

for this. Assume that the footnotes will be numbered with 1 and 2. Now the reader may think it's a single footnote 12, because the 2 immediately follows the 1. You may change this using

```
\KOMAoptions{footnotes=multiple}
```

which would switch on the automatic recognition of footnote sequences. As an alternative you may use

```
Word\footnote{1st footnote}%  
\multiplefootnoteseparator  
\footnote{2nd footnote}
```

This should give you the wanted result even if the automatic solution would fail or could not be used.

Further, assume you want the footnotes separated not only by a single comma, but by a comma and a white space. In this case you may redefine

```
\renewcommand*{\multfootsep}{, \nobreakspace}
```

at the document preamble. `\nobreakspace` instead of a usual space character has been used in this case to avoid paragraph or at least page breaks within footnote sequences.

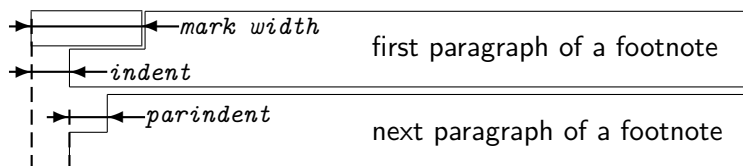
```
\footref{reference}
```

v3.00

Sometimes there are single footnotes to multiple text passages. The least sensible way to typeset this would be to repeatedly use `\footnotemark` with the same manually set number. The disadvantages of this method would be that you have to know the number and manually fix all the `\footnotemark` commands, and if the number changes because of adding or removing a footnote before, each `\footnotemark` would have to be changed. Because of this, KOMA-Script provides the use of the `\label` mechanism in such cases. After simply setting a `\label` inside the footnote, `\footref` may be used to mark all the other text passages with the same footnote mark.

Example: Maybe you have to mark each trade name with a footnote which states that it is a registered trade name. You may write, e. g.,

Figure 3.1.: Parameters that control the footnote layout



```
Company SplishSplash\footnote{This is a registered trade name.
  All rights are reserved.\label{refnote}}
produces not only SplishPlump\footref{refnote}
but also SplishSplash\footref{refnote}.
```

This will produce the same footnote mark three times, but only one footnote text. The first footnote mark is produced by `\footnote` itself, and the following two footnote marks are produced by the additional `\footref` commands. The footnote text will be produced by `\footnote`.

Because of setting the additional footnote marks using the `\label` mechanism, changes of the footnote numbers will need at least two \LaTeX runs to ensure correct numbers for all `\footref` marks.

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

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 at page 87 illustrates the layout parameters. The default configuration of the KOMA-Script classes is:

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

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

Theont 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 `\setkomafont` and `\addtokomafont` (see [section 3.6, page 57](#)). Please refer also to [table 3.2, page 58](#). 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 3.2, page 58](#)). Thus the footnote marks in the text and in the footnote itself are identical. The font can be changed with the commands `\setkomafont` and `\addtokomafont` (see [section 3.6, page 57](#)).

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 `\setkomafont` and `\addtokomafont` (see [section 3.6, page 57](#)):

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

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

```
\setfootnoterule[thickness]{length}
```

v3.06

Generally a horizontal rule will be placed between the text area and the footnote area. But normally this rule is not as long as the width of the typing area. With Command `\setfootnoterule` you may change the thickness and the width of that rule. Thereby the parameters *thickness* and *length* will be evaluated not at definition time but when setting the rule itself. If optional argument *thickness* has been omitted the thickness of the rule will not be changed. Empty arguments *thickness* or *length* are also allowed and do not change the corresponding parameter. Using implausible values may result in warning messages not only setting the arguments but also when KOMA-Script uses the parameters.

v3.07

With element `footnoterule` the color of the rule may be changed using the commands `\setkomafont` and `\addtokomafont` for element `footnoterule` (see [section 3.6, page 57](#)). Default is no change of font or color. For color changes a color package like `xcolor` would be needed.

scrbook

3.15. Demarcation

Sometimes books are roughly separated into front matter, main matter, and back matter. KOMA-Script provides this for `scrbook` also.

```
\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 which 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.

v2.97e

In case the user sees things differently and wishes to use numbered sections in the chapters of the front matter, as of version 2.97e the section numbering no longer contains the chapter number. This change only takes effect when the compatibility option is set to at least version 2.97e (see option [version, section 3.2, page 53](#)). It is explicitly noted that this creates a confusion with chapter numbers! The use of `\addsec` and `\section*` (see [section 3.16, page 99](#) and [page 100](#)) are thus, in the author's opinion, far more preferable.

v2.97e

As of version 2.97e the numbering of float environments, such as tables and figures, and equation numbers in the front matter also contain no chapter number part. To take effect this too requires the corresponding compatibility setting (see option [version, section 3.2, page 53](#)).

`\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 3.12, page 81](#).

Table 3.12.: Available values for option `open` to select page breaks with interleaf pages using `scrbook` or `scrreprt`

any	Parts, chapter, index, and back matter use <code>\clearpage</code> instead of <code>\cleardoublepage</code> ; <code>\cleardoublepageusingstyle</code> , <code>\cleardoublestandardpage</code> , <code>\cleardoubleplainpage</code> , <code>\cleardoubleemptypage</code> , and <code>\cleardoublepage</code> behaves same like using <code>open=right</code> .
left	Part, chapter, index and back matter use <code>\cleardoublepage</code> ; <code>\cleardoublepageusingstyle</code> , <code>\cleardoublestandardpage</code> , <code>\cleardoubleplainpage</code> , <code>\cleardoubleemptypage</code> , and <code>\cleardoublepage</code> result in a page break and add an interleaf page if needed to reach the next left-hand page.
right	Part, chapter, index and back matter use <code>\cleardoublepage</code> ; <code>\cleardoublepageusingstyle</code> , <code>\cleardoublestandardpage</code> , <code>\cleardoubleplainpage</code> , <code>\cleardoubleemptypage</code> , and <code>\cleardoublepage</code> result in a page break and add an interleaf page if needed to reach the next right-hand page.

3.16. Structuring of Documents

Structuring of documents means to divide them into parts, chapters, sections, and several other structural elements.

open=method

scrbook,
scrreprt

v3.00

KOMA-Script classes `scrbook` and `scrreprt` give you the choice of where to start a new chapter with double-sided printing. By default `scrreprt` starts a new chapter at the next page. This is like *method any*. However, `scrbook` starts new chapters at the next right-hand page. This is like *method right* and is usually used in books. But sometimes chapters should start at the left-hand page of a double-page spread. This would be accomplished with *method left*. An overview of the supported methods may be found at [table 3.12](#). The table also describes the effect on `\cleardoublepage`, `\cleardoublepageusingstyle`, `\cleardoublestandardpage`, `\cleardoubleplainpage`, and `\cleardoubleemptypage` (see [section 3.13](#), page 83).

Since \LaTeX does not differentiate between left-hand and right-hand pages in single-sided printing, the option does not have any influence in that case.

In class `scartcl` the section is the first structural element below the part. Because of this, `scartcl` does not support this option.

```
chapterprefix=simple switch
appendixprefix=simple switch
\IfChapterUsesPrefixLine{then code}{else code}
```

scrbook, With the standard classes `book` and `report`, a chapter title consists of a line 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 option `chapterprefix`. Any value from table [table 2.5, page 39](#) may be used as `simple switch`. The default, however, is `chapterprefix=false`, which is opposite of the behaviour of the standard classes, which would correspond to `chapterprefix=true`. These options also affect the automatic running titles in the headers (see [section 3.12, page 77](#)).

Sometimes one wishes to have the chapter titles in simplified form according to `chapterprefix=false`. 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 (see [table 2.5, page 39](#)). Since this results in an inconsistent document layout, I advise against using this option. Final consequence of using the option is, that `\appendix` changes setting of option `chapterprefix`.

v3.18 You can execute code depending on the current setting for the chapter preceding line using `\IfChapterUsesPrefixLine`. If `chapterprefix` is currently active, the *then code* will be executed, otherwise the *else code*.

v2.96a The font style of the chapter number line using `chapterprefix=true` or `appendixprefix=true` may be changed with element `chapterprefix` using commands `\setkomafont` and `\addtokomafont` (see [section 3.6, page 57](#)). Default is the usage of element `chapter` (see [page 95](#), as well as [table 3.15, page 98](#)).

```
headings=selection
```

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 especially problematic for small paper sizes. Consequently, KOMA-Script provides, besides the large title font size defined by the `headings=big` option, the two options `headings=normal` and `headings=small`, that allow for smaller title font sizes. The font sizes for headings resulting from these options for `scrbook` and `scrreprt` are shown in [table 3.15, page 98](#). Respectively, the fonts of the elements `chapter`, `section`, `subsection`, `subsubsection`, `paragraph`, and `subparagraph` will be reset to these default.

scrbook, For `scartcl`, smaller font sizes are generally used. The spacing before and after chapter titles is also influenced by these options.

Chapter titles are also influenced by the options `headings=twolinechapter` and `headings=onelinechapter`, that are same as `chapterprefix=true` and `chapterprefix=false` (see above). The appendix titles are influenced by `headings=twolineappendix` and

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

headings=onelineappendix, that are the same as the options `appendixprefix=true` and `appendixprefix=false` (see also above).

The method of beginning new chapters may be switched by `headings=openany`, `headings=openright`, and `headings=openleft` alternatively to option `open` with the values `any`, `right`, and `left` (see above).

v3.10

Another special feature of KOMA-Script is the handling of the optional argument of the structural commands `\part`, `\chapter`, etc., down to `\subparagraph`. Function and meaning may be influenced by the options `headings=optiontohead`, `headings=optiontotoc`, and `headings=optiontoheadandtoc`.

A summary of all the available selections of option `headings` may be found in [table 3.13](#). Examples are at the following description of the structural commands.

Table 3.13.: Available values for option `headings` to select different kinds of structural headings

`big`

Use very large headings with large distances above and below.

`normal`

Use mid-size headings with medium distances above and below.

`onelineappendix`, `noappendixprefix`, `appendixwithoutprefix`,
`appendixwithoutprefixline`

Chapter headings at the appendix will be set like other headings too.

`onelinechapter`, `nochapterprefix`, `chapterwithoutprefix`,
`chapterwithoutprefixline`

Chapter headings will be set like other headings too.

`openany`

The commands `\cleardoublepageusingstyle`, `\cleardoublestandardpage`, `\cleardoubleplainpage`, `\cleardoubleemptypage`, and `\cleardoublepage` behave same like using `headings=openright`. Parts, chapter, index, and back matter use `\clearpage` instead of `\cleardoublepage`.

`openleft`

The commands `\cleardoublepageusingstyle`, `\cleardoublestandardpage`, `\cleardoubleplainpage`, `\cleardoubleemptypage`, and `\cleardoublepage` generate a page break and if needed insert an interleaf page to reach the next left-hand page at double-page printing. Part, chapter, index and back matter use `\cleardoublepage`.

Table 3.13.: Available values for option `headings` (*continuation*)

openright

The commands `\cleardoublepageusingstyle`, `\cleardoublestandardpage`, `\cleardoubleplainpage`, `\cleardoubleemptypage`, and `\cleardoublepage` generate a page break and if needed insert an interleaf page to reach the next right-hand page at double-page printing. Part, chapter, index and back matter use `\cleardoublepage`.

optiontohead

v3.10

The advanced functionality of the optional argument of the structural commands `\part` down to `\subparagraph` will be activated. By default the optional argument will be used for the running head only.

optiontoheadandtoc, optiontotocandhead

v3.10

The advanced functionality of the optional argument of the structural commands `\part` down to `\subparagraph` will be activated. By default the optional argument will be used for the running head and the table of contents.

optiontotoc

v3.10

The advanced functionality of the optional argument of the structural commands `\part` down to `\subparagraph` will be activated. By default the optional argument will be used for the table of contents only.

small

Use small headings with small distances above and below.

twolineappendix, appendixprefix, appendixwithprefix, appendixwithprefixline

Chapters at the appendix will be set with a number line with the contents of `\chapterformat`.

twolinechapter, chapterprefix, chapterwithprefix, chapterwithprefixline

Chapters will be set with a number line with the contents of `\chapterformat`.

numbers=selection

In German, according to DUDEN, the numbering of sectional units should have no period at the end if only arabic numbers are used (see [DUD96, R3]). On the other hand, if roman numerals or letters appear in the numbering, then a period should appear at the end of the numbering (see [DUD96, R4]). KOMA-Script has an internal mechanism 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 period. The information is saved in the

Table 3.14.: Available values of option `numbers` for selection of the period at the end of numbers of structural headings

<code>autoendperiod</code> , <code>autoenddot</code> , <code>auto</code>
KOMA-Script decides, whether or not to set the period at the end of the numbers. The numbers consists in Arabic digits only, the period will be omitted. If there are alphabetic characters or roman numbers the period will always be set. References to numbers will be set without ending period always.
<code>endperiod</code> , <code>withendperiod</code> , <code>periodatend</code> , <code>enddot</code> , <code>withenddot</code> , <code>dotatend</code>
All numbers of structural commands and all dependent numbers will be set with ending period. Only references will be set without the ending period.
<code>noendperiod</code> , <code>noperiodatend</code> , <code>noenddot</code> , <code>nodotatend</code>
All the numbers are without ending period.

`aux` file and takes effect on the next \LaTeX run.

In some cases the mechanism for placing or leaving off the ending period may fail, or other languages may have different rules. Therefore it is possible to activate the use of the ending period manually with the option `numbers=endperiod` or to deactivate it with `numbers=noendperiod`. Default is `numbers=autoendperiod` with auto detection whether to set the period or not.

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.

The available values are summarized in [table 3.14](#). Unlike most other selections, this option may be changed at the document preamble, before `\begin{document}`, only.

`chapteratlists`

`chapteratlists=value`

As mentioned in [section 3.20, page 133](#), 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 3.2, page 53](#)).

Furthermore, now the option `chapteratlists` can be used to change the spacing, by passing the desired distance as *value*. The default setting with `listof=chaptergapsmall` is 10pt. If `chapteratlists=entry` or `chapteratlists` without value is specified, then instead of a vertical distance, the chapter entry itself will be entered into the lists. This will be done even if there's no floating environment inside of the chapter.

Please note that changes to the option will only become effective in the lists following two more \LaTeX runs.

```

\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.

v3.10

In addition to this, with option `headings=optiontohead`, KOMA-Script does not use the optional argument *short version* at the table of contents, but for the running head only. Nevertheless, such a running head needs an appropriate page style. See [section 3.12](#) and [chapter 5](#) about this. With option `headings=optiontotoc`, KOMA-Script does not use the optional argument *short version* for the running head, but at the table of contents. Nevertheless, the entry will be shown only if counter `tocdepth` (see [section 3.9](#), [page 72](#)) is great enough. With option `headings=optiontoheadandtoc`, KOMA-Script uses the optional argument *short version* in both the table of contents and running head. All these three selections will also activate the extended interpretation of the optional argument *short version*, which is not active by default.

v3.10

The extended interpretation of the optional argument determines whether there's an equality sign in *short version*. If so, the optional argument will be interpreted as *option list* instead of simple *short version*. Thereby the two options `head=running head` and `tocentry=table of contents entry` are supported. Commas or equality signs inside of the values of those options will be accepted only if they are enclosed by braces.

Please note that this mechanism is only functional as long as KOMA-Script controls the described commands. From using a package that controls the sectioning commands or the internal L^AT_EX kernel commands for sectioning commands, KOMA-Script can no longer provide this extended mechanism. This is also valid for the always active extension of KOMA-Script to not create entries to the table of contents if the text of the entry is empty. If you really want an entry with empty heading text, you may use an invisible entry like `\mbox{}` instead.

Example: Assume you're writing a document with some very extensive chapter headings. These headings should be shown in the table of contents too. But for the running head you want only single-line short headings. You will do this using the optional argument of `\chapter`.

```

\chapter[short version of chapter heading]
{The Structural Sectioning Command
 for Chapters Supports not only the
 Heading Text itself but also a

```

```
Short Version with Selectable
Usage}
```

Sometimes later you become aware that the automatic line breaking of this heading is somehow inappropriate. Therefore you want to make the breaking yourself. Nevertheless, the automatic line breaking should be still used at the table of contents. With

```
\chapter[head={short version of chapter heading},
          tocentry={The Structural Sectioning
                    Command for Chapters Supports not
                    only the Heading Text itself but
                    also a Short Version with
                    Selectable Usage}]
{The Structural\\
  Sectioning Command for Chapters\\
  Supports not only\\
  the Heading Text itself\\
  but also\\
  a Short Version\\
  with Selectable Usage}
```

you use independent entries for table of contents, running head, and the chapter heading itself. The arguments of the options `head` and `tocentry` have been enclosed into braces, so the contents of the options cannot influence the interpretation of the optional argument.

The recommendation of the braces in the example above will make more sense with one more example. Assume you're using option `headings=optiontotoc` and now have a heading:

```
\section[head=\emph{value}]
{Option head=\emph{value}}
```

This would result in the entry “Option head=*value*” at the table of contents but “*value*” at the running head. But surely you wanted the entry “head=*value*” at the table of contents and the complete heading text at the running head. You may do this using braces:

```
\section[head={}\emph{value}]
{Option head=\emph{value}}
```

A similar case would be a comma. With the same `headings` option like before:

```
\section[head=0, 1, 2, 3, \dots]
{Natural Numbers Including the Zero}
```

would result in an error, because the comma would be interpreted as the separator between the single options of the option list “0, 1, 2, 3, \dots”. But writing


```
\section[head={0, 1, 2, 3, \dots}]
      {Natural Numbers Including the Zero}
```

will change “0, 1, 2, 3, \dots” into the argument of option `head`.

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 107) is set on a separate page.

`\chapter` only exists in book or report classes, that is, in classes `book`, `scrbook`, `report` and `scrreport`, 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, but can be reverted by the option `chapterprefix` (see page 91).

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 3.12, page 79).

The font of all headings can be changed with the commands `\setkomafont` and `\addtokomafont` (see section 3.6, page 57). In doing this, generally the element `disposition` is used, followed by a specific element for every section level (see table 3.2, page 58). There is a separate element `partnumber` for the number part of parts. The font for the element `disposition` is predefined as `\normalcolor\sffamily\bfseries`. The default font size for the specific elements depends on the options `headings=big`, `headings=normal` and `headings=small` (see page 91). The defaults are listed in table 3.15.

Example: Suppose you are using the class option `headings=big` 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. This is done either by:

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

or simply using:

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

scrbook,
scrreprt

scrbook,
scrreprt

scrbook,
scrreprt

v2.8p

Table 3.15.: Default font sizes for different levels of document structuring in scrbook and screprt

Class Option	Element	Default
headings=big	part	\Huge
	partnumber	\huge
	chapter	\huge
	section	\Large
	subsection	\large
	subsubsection	\normalsize
	paragraph	\normalsize
	subparagraph	\normalsize
headings=normal	part	\huge
	partnumber	\huge
	chapter	\LARGE
	section	\Large
	subsection	\large
	subsubsection	\normalsize
	paragraph	\normalsize
	subparagraph	\normalsize
headings=small	part	\LARGE
	partnumber	\LARGE
	chapter	\Large
	section	\large
	subsection	\normalsize
	subsubsection	\normalsize
	paragraph	\normalsize
	subparagraph	\normalsize

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 49](#) 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 is described below.

`\chapter*` only exists in book and report classes, that is, `book`, `scrbook`, `report` and `scrreport`, but not the article classes `article` and `scartcl`.

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 3.12](#), [page 79](#)).

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.

In the standard classes there are no further structuring commands. In particular, there are no commands which can produce unnumbered chapters or sections which show up in the table of contents and in the running heading.

```
\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 `scartcl`.

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 3.12](#), [page 79](#)).

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.

v2.96a

The font type of the structuring command `\minisec` can be changed using the element `disposition` (see [table 3.2](#), [page 58](#)) and `minisec`. Default setting of element `minisec` is empty, so the default of the element `disposition` is active.

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

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.

```
\raggedsection
\raggedchapter
\raggedpart
```

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.

v3.15

Because some users want for `\chapter` another alignment than for all other sections, you can change only `\chapter` redefining `\raggedchapter`. The default of this command is usage of `\raggedsection`. So every change of `\raggedsection` will also change the alignment of chapter headings.

Unlike all others, the headings of parts (`\part`) will be horizontally centered instead of set ragged right. This is because command `\raggedpart` is defined as

```
\let\raggedpart\centering
```

You may also redefine this using `\renewcommand` too.

Example: You do not want different alignment at headings of `\part`. So you put

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

into the preamble of your document. In this case, and unlike in the example above, `\let` has not been used, because `\let` would give `\raggedpart` the current meaning of `\raggedsection`. Further changes of `\raggedsection` would then stay disregarded at the usage of `\raggedpart`. Doing the redefinition using `\renewcommand` gives `\raggedpart` the meaning of `\raggedsection` not at definition time, but each time `\raggedpart` will be used.

```

\partformat
\chapterformat
\sectionformat
\subsectionformat
\subsubsectionformat
\paragraphformat
\subparagraphformat
\othersectionlevelsformat{sectioningname}{}{counter output}
\IfUsePrefixLine{then code}{else code}
\autodot

```

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

As described for option **numbers** at the beginning of this section (see [page 93](#)), periods 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 followed. In all levels except for `\part`, a dot is followed by a further `\enskip`. This corresponds to a horizontal skip of 0.5 em.

Since KOMA-Script 3.17 the command `\othersectionlevelsformat` is used only in rare circumstances, i. e., if the corresponding format command to a section command does not exist or is `\relax`. This should not happen for all section commands defined by KOMA-Script itself. Therefore the command is not officially documented any longer. Nevertheless, if a compatibility level prior to 3.17 (see option **version**, [section 3.2, page 53](#)) has been selected, commands `\sectionformat` down to `\subparagraphformat` are ignored by KOMA-Script. In this case `\othersectionlevelsformat` indeed will be used.

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}{%
  \mbox{\chapappifchapterprefix{\nobreakspace}\thechapter
    \autodot\IfUsePrefixLine{}{\enskip}}}
\newcommand*{\sectionformat}{\thesection\autodot\enskip}
\newcommand*{\othersectionlevelsformat}[3]{%
  #3\autodot\enskip}

```

The definitions of the lower levels correspond to `\sectionformat`.

Because of `\IfUsePrefixLine` command `\chapterformat` should not be used outside of `\chapter`. `\IfUsePrefixLine` is only valid inside section commands of KOMA-Script. In this case, it executes the *then code* if a prefix line for the number is used, but *else code*

scrbook,
scrreprt

v3.17

v3.17

otherwise.

Do not forget to change `\newcommand` into `\renewcommand` if you like to re-define one of the commands.

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 period has to be applied. However, you thereby give up the possibility to use one of the options `numbers=endperiod` and `numbers=noendperiod` and optionally depart from the rules. More details concerning class options can be found at [page 93](#).

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*{\sectionformat}{%
  \makebox[0pt][r]{\thesection\autodot\enskip}}
\renewcommand*{\subsectionformat}{%
  \makebox[0pt][r]{\thesubsection\autodot\enskip}}
\renewcommand*{\subsubsectionformat}{%
  \makebox[0pt][r]{\thesubsubsection\autodot\enskip}}
\renewcommand*{\paragraphformat}{%
  \makebox[0pt][r]{\theparagraph\autodot\enskip}}
\renewcommand*{\paragrapheformat}{%
  \makebox[0pt][r]{\thesubparagraph\autodot\enskip}}
```

See [Tea05b] for more information about the optional arguments of `\makebox`.

If you would like to change more than only the printing of the counter of a heading, please have a look at `\chapterlineswithprefixformat`, `\chapterlinesformat`, `\sectionlinesformat`, and `\sectioncatchphraseformat` in [section 21.3](#) from [page 457](#).

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

scrbook,
scrreprt

These two commands are not only used internally by KOMA-Script but 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=true` (see [page 91](#)) `\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 `maincls=chapterprefixfalse` is set, then nothing is output.

The command `\chapapp` always outputs the word “Chapter” or “Appendix”. In this case the selection of option `chapterprefix` has no effect.

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

```
\chaptermark{running head}
\addchapmark{running head}
\sectionmark{running head}
\addsecmark{running head}
\subsectionmark{running head}
\chaptermarkformat
\sectionmarkformat
\subsectionmarkformat
```

As mentioned in [section 3.12](#) the page style headings works with automatic running heads. For this, the commands `\chaptermark` and `\sectionmark`, or `\sectionmark` and `\subsectionmark`, respectively, are defined. Every structuring command (`\chapter`, `\section`, etc.) 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`.

Please note, the running heads of `\addchap` and `\addsec` are also based on `\chaptermark` and `\addsecmark` but locally set counter `secnumdepth` to a value that makes chapters respectively sections not numbered. You should pay attention for this especially if you redefine `\chaptermark` or `\sectionmark` (see `\ifnumbered` on [page 107](#)). The star variants `\addchap*` and `\addsec*` use additional commands `\addchapmark` and `\addsecmark` that are defined also basing on `\chaptermark` and `\sectionmark` with local changes of `secnumdepth`.

Of course there is no command `\chaptermark` or `\chaptermarkformat` in `scrartcl`. Accordingly, `\subsectionmark` and `\subsectionmarkformat` exist only in `scrartcl`. This changes when you use the `scrlayer-scrpage` package (see [chapter 5](#)).

Similar to `\partformat` down to `\subparagraphformat` for formatting the numbers in the headings, the commands `\chaptermarkformat` (not in `scrartcl`), `\sectionmarkformat`, and `\subsectionmarkformat` (only in `scrartcl`) define the formatting of the sectioning numbers in the automatic running heads. 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
\partnumdepth
\chapternumdepth
\sectionnumdepth
\subsectionnumdepth
\subsubsectionnumdepth
\paragraphnumdepth
\subparagraphnumdepth
```

Section levels in the classes `scrbook` and `scrreport` are, by default, numbered from `\part` down to `\subsection`. In class `scartcl` the default numbering is from `\part` down to `\subsubsection`. The actual depth to which headings will be numbered is controlled by the L^AT_EX counter `secnumdepth`. Since version 3.12 KOMA-Script provides the commands `\partnumdepth` to `\subparagraphnumdepth` which return the number that corresponds to the level they bear in their name. This saves users the trouble of having to remember abstract numbers and allows them to define the depth to which headings should be numbered with relative ease.

v3.12

KOMA-Script provides the commands `\partnumdepth` to `\subparagraphnumdepth` so users do not have to remember abstract numbers, to be able to define the section level down to which headings should be numbered. These commands stand for the corresponding numbers of the section levels.

Example: For a book project you want the section levels from part down to the section to be numbered. To achieve this, you have to set counter `secnumdepth` to the value represented by `\sectionnumdepth` in the preamble of your document:

```
\setcounter{secnumdepth}{\sectionnumdepth}
```

Redefining these commands is not allowed. Doing so could lead to unexpected results not only with KOMA-Script but also with third party packages. Thus, it is recommended to never redefine any of them.

Do not confuse the counters `secnumdepth` and `tocdepth`, please refer to the explanation concerning the counter `tocdepth` in [section 3.9, page 72](#). Actually, depending on the class you

are using, the meaning of the values of the counters `secnumdepth` and `tocdepth` may deviate from one another for the same section level.

```
\ifnumbered{section level}{then code}{else code}
\ifunnumbered{section level}{then code}{else code}
```

v3.12

After describing above how to define down to which section level headings are numbered, the commands `\ifnumbered` and `\ifunnumbered` can be used to execute code depending on whether a *section level* is numbered or not. If the current setting of `secnumdepth` stipulates that a *section level* will be numbered, the *then code* after `\ifnumbered` gets executed. If it is not numbered, the *else code* gets executed. The `\ifunnumbered` command behaves in exactly the opposite manner, executing the *then code* if the current level is not numbered and the *else code* if it is. The *section level* parameter is simply the L^AT_EX name of a section like `part`, `chapter`, `section`, `subsection`, `subsubsection`, `paragraph` oder `subparagraph`. `part`, `chapter`, `section`, `subsection`, `subsubsection`, `paragraph`, or `subparagraph`.

KOMA-Script itself uses these tests, for example, in the definition of `\chaptermark` within page style `headings`. This indirectly guarantees that headings inserted by `\addchap` do not set a number inside the running head (see also `\addchapmark`, page 105).

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

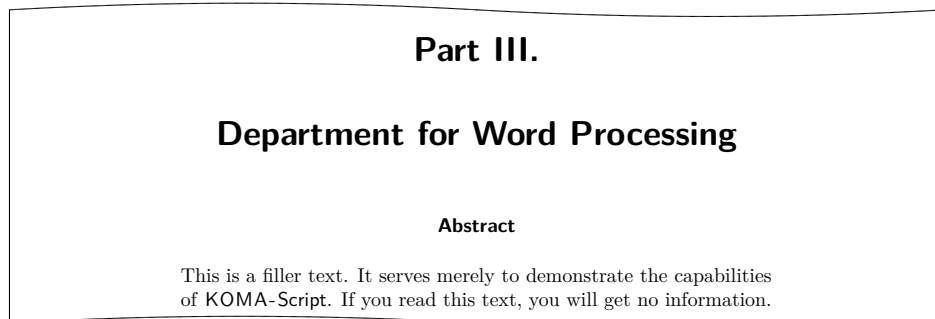
scrbook,
scrreprt

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 [page 91](#)) and the options for the **abstract** environment (see [section 3.8](#), [page 68](#)), 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 3.8](#), [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.

A more often usage of `\setchapterpreamble` would be something like a smart slogan or dictum to a heading. The command `\dictum`, that may be used for this, will be described at the next section. You will also find an example there.

Please note that a *preamble* placed above the chapter headings will be set into the already existing vertical space above the heading. The heading will not be moved down. It is you who is responsible for ensuring that the preamble is small enough and the space is sufficient. See also `\chapterheadstartvskip` in [section 21.3](#), [page 455](#) for this.

Table 3.16.: Default settings
for the elements of a dictum

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

3.17. Dicta

Sometimes you may find a dictum, a kind of smart slogan or excerpt, often ragged left above or below the heading of a chapter or section. The text and the source of the slogan often use special styles.

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

The command `\dictum` inserts such a dictum. This macro can be used as obligatory argument of either the command `\setchapterpreamble` or `\setpartpreamble`. However, this is not obligatory.

The dictum together with an optional *author* is inserted in a `\parbox` (see [Tea05b]) 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 `\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 `dictum`, which can be changed with the commands `\setkomafont` and `\addtokomafont` (see section 3.6, page 57). Default settings are listed in table 3.16.

If there is an *author* name, it is separated from the *dictum* by a rule to the full width of the `\parbox`. This rule is defined as vertical object to command `\dictumrule`:

v3.10

```
\newcommand*{\dictumrule}{\vskip-1ex\hrulefill\par}
```

The alignment is defined with `\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 parentheses. For the element `dictumauthor`, a different font than for the element `dictum` can be defined. Default settings are listed in [table 3.16](#). Changes can be made using the commands `\setkomafont` and `\addtokomafont` (see [section 3.6](#), page 57).

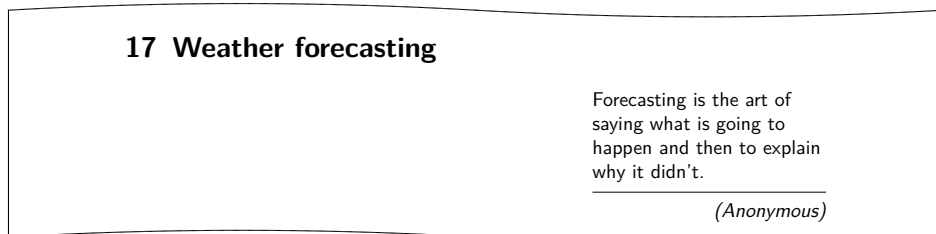
If `\dictum` is used within the macro `\setchapterpreamble` or `\setpartpreamble` you have to take care of the following: the horizontal positioning is always done with `\raggeddictum`. Therefore, the optional argument for horizontal positioning which is implemented for these two commands has no effect. `\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` [[Sch09](#)].

3.18. 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.

```
\begin{itemize}
  \item ...
  :
\end{itemize}
\labelitemi
\labelitemii
\labelitemiii
\labelitemiv
```

The simplest form of a list is an `itemize` list. 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}
\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

```

\begin{enumerate}
\item ...
:
\end{enumerate}
\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 L^AT_EX 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`.

```
\begin{description}
  \item[keyword] ...
  :
\end{description}
```

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 `\setkomafont` and `\addtokomafont` (see [section 3.6, page 57](#)) for the element `descriptionlabel` (see [table 3.2, page 58](#)). Default setting is `\sffamily\bfseries`.

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 3.12](#). The heavily abbreviated source could be:

```

\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.

```

```

\begin{labeling}[delimiter]{widest pattern}
\item[keyword]...
:
\end{labeling}

```

An additional form of a description list is only available in the KOMA-Script classes: the `labeling` environment. Unlike the `description` environment, 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. The font which is responsible for emphasizing the item and the separator can be changed with the commands `\setkomafont` and `\addtokomafont` (see section 3.6, page 57) for the element `labelinglabel` and `labelingseparator` (see table 3.2, page 58).

v3.02

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

```

\setkomafont{labelinglabel}{\ttfamily}
\setkomafont{labelingseparator}{\normalfont}
\begin{labeling}[~--]{myheadings}
\item[empty]
Page style without header and footer
\item[plain]
Page style for chapter beginnings without headings
\item[headings]
Page style for running headings
\item[myheadings]
Page style for manual headings
\end{labeling}

```

As the result we get:

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

As can be seen in this example, a font changing command may be set in the usual way. But if you do not want the font of the separator to be changed in the same way as the font of the label, you have to set the font of the separator as well.

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.

`\begin{verse}... \end{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 could, 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.

```
\begin{quote}... \end{quote}
\begin{quotation}... \end{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?’’

```
\begin{addmargin}[left indentation]{indentation}...\end{addmargin}
\begin{addmargin*}[inner indentation]{indentation}...\end{addmargin*}
```

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:

```
\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 1em. In turn the outer margin is decreased by 1em. The result is a shift by 1em to the outside. Instead of 1em you can of course use a length, for example, 2\parindent.

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` (section 3.11, page 76) and `\ifthispagewasodd` (section 21.1, page 441).

There may be several questions about coexistence of lists and paragraphs. Because of this additional information may be found at the description of option `parskip` in section 21.1, page 441. Also at the expert part, in section 21.1, page 441, you may find additional information about page breaks inside of `addmargin*`.

3.19. Math

There are no math environments implemented in the KOMA-Script classes. Instead of this, the math features of the L^AT_EX kernel have been supported. With this also, the options `leqno` and `fleqn` are available.

You will not find a description of the math environments of the L^AT_EX kernel here. If you want to use `displaymath`, `equation`, and `eqnarray` you should read a short introduction into L^AT_EX like [OPHS11]. But if you want more than very simple mathematics, usage of package `amsmath` would be recommended (see [Ame02]).

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. This option has to be used as an optional argument of `\documentclass`. Usage as an argument of `\KOMAOPTIONS` or `\KOMAOPTION` is not supported. This would not make sense, because the recommended math package `amsmath` supports the option at loading time only too and would not react on run-time changes of the option.

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. This option may be used as an optional argument of `\documentclass` but not as an argument of `\KOMAOPTIONS` or `\KOMAOPTION`. The latter would not make sense, because the recommended math package `amsmath` supports the option at loading time only too and would not react on run-time changes of the option.

3.20. Floating Environments of Tables and Figures

With the floating environments, L^AT_EX 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 [Wik].

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.

`captions=selection`

The standard classes format titles of floating environments, which are placed with `\caption` (see below), like signatures. They differentiate between one-line and multi-line table or figure captions. One-line captions are centered while multi-line captions are left-justified.

For tables, however, headings are often used. That's because there may be tables that span several pages. Surely the reader wants to know the purpose of the table at the first page already. Furthermore tables will be read row by row from top down to bottom. So there are at least two good reasons to generally use table headings. KOMA-Script therefor supports option `captions=tableheading`, which changes the caption format into headings at tables only.

Please note that multi-page tabulars may not use any floating environment. To have an automatic page break at any kind of tabular you also need additional packages like `longtable` (see [Car04]) or `tabu` (see [Che11]).

You may switch back to the default table signatures using `captions=tablesignature`. 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 vice 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. This may change using package `float` and command `\restylefloats` (see [Lin01]).

v3.09

Of course similar features exist for figures using options `captions=figureheading` and `captions=figuresignature`. Nevertheless, figures like photos will be viewed as a whole, and a diagram or graph will mostly be examined from left bottom to the right. Therefore, in general, signatures should be used and it would not be useful to change the caption format from signatures to headings.

v3.09

Nevertheless sometimes all floating environments shall use headings. For this KOMA-Script supports options `captions=heading` and `captions=signature` to switch the format of every floating environment. These options may be used also inside a floating environment but before using `\caption`.

`float` Note that when using the `float` package, the options `captions=tablesignature` and `captions=tableheading` 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` (see page 129).

Furthermore, KOMA-Script supports to switch off the distinguish of captions with only one line or more than one line using option `captions=nooneline`. This may be useful, if one-line captions should not be centered. The default of centering one-line captions corresponds to `captions=oneline`.

Another special feature of KOMA-Script is to alternatively put the caption neither above nor below the floating object but beside it. For this you need Environment `captionbeside`, that will be described from page 126. Several settings for this environment may be done also using `caption`. You may find all the available *settings* at table 3.17.

Table 3.17.: Available values for option `captions` to select formation of captions as headings or signatures at floating environments

`bottombeside, besidebottom`

Captions and contents of environment `captionbeside` (see section 3.20, page 126) will be vertically align depending on the bottommost base lines.

`centeredbeside, besidecentered, middlebeside, besidemiddle`

Captions and contents of environment `captionbeside` (see section 3.20, page 126) will be vertically centered

`figureheading, figureabove, abovefigure, topatfigure`

Captions of figures will use heading formation — maybe in discrepancy to `captions=signature`.

`figuresignature, belowfigure, bottomatfigure`

Captions of figures will use signature formation — maybe in discrepancy to `captions=headings`.

`heading, above, top`

Captions of floating environments will use heading formation. Nevertheless this does not influence whether they are really placed at the top or at the bottom of the object. This options also implies `captions=tableheading` and `captions=figureheading`.

...

Table 3.17.: Available values for option `captions` (*continuation*)

`innerbeside, besideinner`

Captions of environment `captionbeside` (siehe [section 3.20, page 126](#)) will be placed innermost beside the contents of the environment at double-side printing. At single-side printing `captions=leftbeside` will be used.

`leftbeside, besideleft`

Captions of environment `captionbeside` (siehe [section 3.20, page 126](#)) will be placed left beside the contents of the environment.

`nooneline`

Captions with only one line will not be handled different from captions with more than one line.

`oneline`

Captions with only one line will be centered horizontally.

`outerbeside, besideouter`

Captions of environment `captionbeside` (siehe [section 3.20, page 126](#)) will be placed outermost beside the contents of the environment at double-side printing. At single-side printing `captions=rightbeside` will be used.

`rightbeside, besideright`

Captions of environment `captionbeside` (siehe [section 3.20, page 126](#)) will be placed right beside the contents of the environment.

`signature, below, bot, bottom`

Captions of floating environments will use signature formation. Nevertheless this does not influence whether they are really placed at the top or at the bottom of the object. This options also implies `captions=tablesignature` and `captions=figuresignature`.

`tableheading, tableabove, abovetable, abovetabular, topattable`

Captions of tables will use heading formation — maybe in discrepancy to `captions=signature`.

`tablesignature, belowtable, belowtabular, bottomattable`

Captions of tables will use signature formation — maybe in discrepancy to `captions=heading`.

Table 3.17.: Available values for option `captions` (*continuation*)

`topbeside, besidetop`

Captions and contents of environment `captionbeside` (see [section 3.20, page 126](#)) will be vertically align depending on the topmost base lines.

```
\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.

Not only for tables but also for figures or all kind of floating environments the behaviour of `\caption` may be modified using option `captions` described at the beginning of this section. For compatibility reasons the default behaviour of `\caption` used with all kinds of floating environments is similar to `\captionbelow`. Nevertheless it is recommended to use table headings and therefor switch behaviour of `\caption` inside table environments into `\captionabove` using option [3.20](#). Alternatively you may use `\captionabove` instead of `\caption` inside of every table environment.

Example: Instead of using captions below a table you want to place your captions above it, because you have tables which span more then 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:

Table 3.18.: Font defaults for the elements of figure or table captions

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

```
\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 3.20 instead (see page 120). Then you can use `\caption` as you would in the standard classes. You will get the same result as with `\captionabove`.

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 `\setkomafont` and `\addtokomafont` (see section 3.6, page 57). The respective elements for this are `caption` and `captionlabel` (see table 3.2, page 58). 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 3.18.

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}
```

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.

```
\captionof{float type}[entry]{title}
\captionbelowof{float type}[entry]{title}
\captionaboveof{float type}[entry]{title}
```

v3.05 KOMA-Script supports a command `\captionof` similar to packages `caption` and `capt-of`. You may use this command to place a floating environment caption with corresponding entry into the list of that kind of floating environment but even inside a another floating environment or outside any floating environment. In difference to `\caption` the kind of floating environment has to be set as first parameter.

v3.09 Furthermore, KOMA-Script provides the additional commands `\captionaboveof` and `\captionbelowof`. These are like `\captionabove` and `\captionbelow` but with the additional features and parameter of `\captionof`.

v3.09a Of course KOMA-Script takes care of the heading and signature setting of option `captions`. But this feature may be lost, loading package `capt-of` or `caption`. Please note the manual of package `caption` for this!

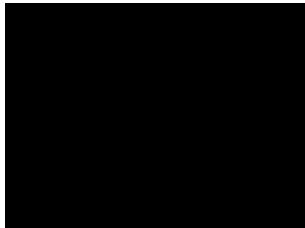
Example: Assumed you want to create a floating object with a table and a figure side by side. As you know, there are now mixed floating environment. Therefor you use a `figure` environment primarily:

```
\begin{figure}
  \begin{minipage}{.5\linewidth}
    \centering
    \rule{4cm}{3cm}
    \caption{A rectangle}\label{fig:rechteck}
  \end{minipage}%
  \begin{minipage}{.5\linewidth}
    \centering
    \captionaboveof{table}
    [Measure of the rectangle in
     figure~\ref{fig:rechteck}]%
    {Measure of the rectangle}
    \label{tab:rechteck}
    \begin{tabular}{ll}
      Width: & 4\,cm\\
      Height: & 3\,cm
    \end{tabular}
  \end{minipage}
\end{figure}
```

Two `minipage` environments have been used to have figure and table side by side. The percent char after the end of the first `minipage` is important. Without an additional inter-word distance would be made between the `minipage` environments.

The figure signature has been done using `\caption`. The table heading has been

Figure 3.3.: Example:
of `\captionaboveof` inside
another floating environment

	Table 3.19.: Rectangle size
Figure 3.2.: A rectangle	Width: 4 cm
	Height: 3 cm

done using `\captionaboveof` with first argument `table`. Because of this KOMA-Script knows, that despite the `figure` environment a table caption should be made.

The optional argument of `\captionaboveof` does make the entry into the list of tables. Without the optional argument, the last mandatory argument would have been used for the list of tables too. Although this caption text is sufficient for the environment itself, it would be very useful at the list of tables. Therefore a somehow more detailed description has been used for the list of tables using the optional argument. The [figure 3.3](#) shows the result of the example code.

A non-floating table with a caption may be produced in the same kind like the table inside a figure environment in the example above. In such a case also a `minipage` environment should be used, to avoid page breaks between table caption and tabular. An additional `flushleft` environment around the `minipage` environment may be used, to have a pleasing distance above and below and to avoid the paragraph indentation of the `minipage` environment.

```
\begin{captionbeside}[entry]{title}[placement][width][offset]... \end{captionbeside}
\begin{captionbeside}[entry]{title}[placement][width][offset]*... \end
{captionbeside}
```

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:

- 1 – left

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

v3.00

Default setting is to the right of the content of the environment. This default may be changed using option **captions** (see [page 120](#)) with values like **innerbeside**, **leftbeside**, **outerbeside**, and **rightbeside**. 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 0 pt gives you a left-aligned output.

Adding a star to the optional parameter *offset* makes 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 0 pt means alignment with the inner margin. As mentioned before, in some cases it takes two L^AT_EX runs for this to work correctly.

v3.00

The default vertical alignment is bottom. This means that the bottommost base lines of the contents of the environment and of the caption are aligned. This setting may be changed using option **captions** (see [page 120](#)) with value **topbeside**, **centeredbeside**, or **bottombeside**. With setting **topbeside** the topmost base lines of the environment contents and caption will be aligned. With **centeredbeside** they will be centered vertically. In this context it should be known, that the base line of a picture is mostly at the bottom of the picture. This may be changed, e. g., using `\raisebox`.

Example: An example for the usage of the `captionbeside` environment can be found in [figure 3.4](#). This figure was typeset with:

```
\begin{figure}
\begin{captionbeside}[Example: Figure beside description]%
  {A figure description which is neither above nor
  below, but beside the figure}[i][\linewidth][%
  i][\linewidth][%
  \dimexpr\marginparwidth+\marginparsep\relax]*
\fbbox{%
  \parbox[b][5\baselineskip][c]{.25\textwidth}
  {%
    \hspace*{\fill}\KOMAScript
    \hspace*{\fill}\par
  }
}
```

Figure 3.4.: A figure description which is neither above nor below, but beside the figure

```

    }%
  }
  \end{captionbeside}
  \label{fig:\LabelBase.captionbeside}
\end{figure}

```

The total width is thus the currently available width `\linewidth`. However, this width is shifted `\marginparwidth + \marginparsep` 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.

With `\dimexp` a ε -TEX command has been used. This should not be a problem at all, because KOMA-Script itself needs ε -TEX and every almost up-to-date L^AT_EX distribution uses ε -TEX already.

Figure 3.5 shows a centered caption with:

```
\KOMAoption{captions}{centeredbeside}
```

Even if you are not a typographer you may see, that this is not a recommended suggestion.

In opposite to the centered example, the top aligned from figure 3.6 may be used. To show how to change the baseline using `\raisebox`, the complete example code follows:

```

\documentclass[captions=topbeside]{scrbook}
\usepackage[english]{babel}
\usepackage{graphics}
\begin{document}
\chapter{An Example}
\begin{figure}

```

Figure 3.5.: A figure description which is neither above nor below, but centered beside the figure

Figure 3.6.: A figure description which is neither above nor below, but top beside the figure

KOMA-Script

```
\begin{captionbeside}%
  [Example: Figure title top beside]%
  {A figure description which is neither above nor
   below, but top beside the figure}%
  [i][\linewidth][%
    \dimexpr\marginparwidth+\marginparsep\relax
  ]*
  \raisebox{%
    \dimexpr\baselineskip-\totalheight\relax
  }{%
    \includegraphics{examplepicture}%
  }%
\end{captionbeside}
\label{fig:\LabelBase.captionbesidetop}
\end{figure}
\end{document}
```

You may use such a movement not only at graphics replacements like show, but also using `\includegraphics` (see [Car05]).

```
\begin{captionofbeside}{float type}[entry]{title}[placement][width][offset]...\end
  {captionofbeside}
\begin{captionofbeside}{float type}[entry]{title}[placement][width][offset]*...\end
  {captionofbeside}
```

v3.10

This environment corresponds to `captionbeside` in the same manner like `\captionof` corresponds to `\caption`. The float type is not defined by a floating environment but by the first mandatory argument.

```
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 `\...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 option `captions`. For details please refer to the explanations of these option at [page 121](#).

Example: For the examples please refer to figures [3.7](#) to [3.10](#). 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.

KOMA-Script

Figure 3.7.: Equivalent to the standard setting, similar to the usage of `\setcaphanging`

KOMA-Script

Figure 3.8.: With slightly hanging indentation starting at the second line using `\setcapindent{1em}`

KOMA-Script

Figure 3.9.:
With hanging indentation starting at the second line and line break before the description using `\setcapindent*{1em}`

KOMA-Script

Figure 3.10.:
With indentation in the second line only and line break before the description using `\setcapindent{-1em}`

```
\setcapwidth[justification]{width}
\setcapdynwidth[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 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`.

v3.20

Please note, `\setcapwidth` sets the width immediately to the value of parameter *width* like `\setlength` would do. If you instead want the value of *width* when the caption is set, you can use `\setcapdynwidth`. There can be differences in the result, if you, e. g., use lengths like `\linewidth` or other commands as argument *width*.

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 3.20 (see page 121) for the captions which are typeset with this margin setting.

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.

Experts and advanced users may find a tricky usage of `\setcapwidth` in [Koh14a].

origlongtable

If the table captions produced by the `longtable` package (see [Car04]) should not be redefined by the KOMA-Script classes, activate the `origlongtable` option. This option has to be used at the optional argument of `\documentclass`. It may not be used as a setting of `\KOMAoptions` or `\KOMAoptions`.

listof=setting

v3.00

Normally lists of floating environments — like list of tables or list of figures will neither get an entry at the table of contents nor have a number at the heading. More information about that may be found in section 3.9. Alternative to the view from the table of contents to the lists of floating environments, you may reconsider a view from the lists of floating environments into the table of contents. Because of this, there are not only the options 3.9, 3.9, and 3.9 described in section 3.9, page 69, but also `listof=notoc`, `listof=totoc`, and `listof=numbered` with the same meaning.

v3.06

By default the headings of the lists of floating environments use the topmost level below `\part`. This is the chapter level at `scrbook` and `scrreprt` and the section level at `scartcl`. With `listof=leveldown` a one step lower level will be used instead.

Example: At a book you want to move the list of figures and the list of tables as sub-lists into a common list named “Figures and Tables”. With

```
\KOMAoption{listof}{leveldown}
```

you first declare to use the section instead of the chapter level for both lists and then you use:

```
\addchap*{Figures and Tables}
\listoffigures
\listoftables
```

for the new list, that contains the list of figures and the list of tables. More information about the command `\addchap*` may be found in [section 3.16](#) at [page 100](#).

v2.8q

Normally the lists of floating environments use a constant with to place the caption number of the entries. Additionally all entries will be indented a little bit. This corresponds to setting `listof=graduated`.

If the numbers of the figures or tables, become very wide — i. e., if you have a lot of tables or figures — their may be not enough width predefined. There’s a setting `listof=flat` for the lists of floating environment similar to [3.9](#) for the table of contents. Thereby the needed width for printing the number will be determined at each L^AT_EX run. See option [3.9](#), [section 3.9](#), [page 70](#) for information about how it works. Please note again, that you need more than one L^AT_EX runs until the lists of floating environments will become their final result.

v3.06

Setting `listof=entryprefix` will automatically activate `listof=flat` too. Normally it would not make sense to add the prefix “figure” to each entry of the list of figures and the prefix “table” to each entry of the list of tables, because nothing else than figures would be and should be expected at the list of figures and nothing else than tables would be and should be expected at the list of tables. So this prefixes would not give any additional information and for this would not be useful. Nevertheless, such prefixes may be added using option `listof=entryprefix`. With this all entries of the same list will get the same prefix. The prefix will depend on the file extension of the helper file, that will be used for the corresponding list. For the list of figures the file extension would be “lof” and therefor `\listoflofentryname` would be used. For the list of tables, the file extension would be “lot” and `\listoflotentryname` would be used.

scrbook,
scrreprt

v3.00

Within classes `scrbook` and `scrreprt` KOMA-Script adds a vertical gap to the lists of floating environments whenever a new chapter starts. This behaviour, that is same at the standard classes, structures the lists by chapters. At KOMA-Script it corresponds to setting `listof=chaptersgapsmall`. In this case a gap of width 10pt will be used. With option `listof=chaptersgapline` a gap of the height of one standard text line will be used. The gap may be switched of with `listof=nochaptersgap`. Option `listof=chapterentry` is somehow special. Instead of a gap it adds the table of contents entry for the chapter additionally to the lists of floating environments. Please note, that this would also happen, if the chapter does not have any floating environment. Additional influence of chapters to the lists of floating environments is available with option `chapteratlists`. See [section 3.16](#), [page 94](#) for more information about that.

An overview about all settings to option `listof` may be found at [table 3.20](#).

Table 3.20.: Available values for option `listof` to modify contents and formation of the lists of floating environments

chapterentry, withchapterentry

Marks chapter starts at the lists of floating environments by a copy of their entries to the table of contents.

chaptergapline, onelinechaptergap

Marks chapter starts at the lists of floating environments by a vertical gap of the height of one standard text line.

chaptergapsmall, smallchaptergap

Marks chapter starts at the lists of floating environments by a small vertical gap.

entryprefix

Adds a prefix depending on the file extension of the list to each entry of the lists of floating environments. The prefix additionally depends on the language, e. g., in English “Figure” would be used for the entries to the list of figures and “Table” for the entries to the list of tables. Both prefixes will be followed by a white space.

flat, left

The lists of floating environments will be printed like a kind of table. The caption numbers will be the first column, the caption texts the second column, and the page numbers the last column. The width of the first column depends on the previous L^AT_EX run.

graduated, indent, indented

The lists of floating environments will be printed in hierarchical form. The width for the caption numbers will be limited.

leveldown

The lists of floating environments will use a heading of one step lower sectioning level than default.

nochaptergap, ignorechapter

Chapter starts are not marked at the lists of floating environments.

notoc, plainheading

The lists of floating environments, e. g., list of figures and list of tables do not generate an entry at the table of contents.

Table 3.20.: Available values for option `listof` (*continuation*)

`numbered, toctocnumbered, tocnumbered, numberedtotoc`

The lists of floating environments, e.g., list of figures and list of tables, would get a numbered heading and therefor generate an entry at the table of contents.

`totoc, toc, notnumbered`

The lists of floating environments, e.g., list of figures and list of tables, would generate an entry at the table of contents, but their headings are not numbered.

```
\listoftables
```

```
\listoffigures
```

These commands generate a list of tables or figures. Changes in the document that modify these lists will require two L^AT_EX runs in order to take effect. The layout of the lists can be influenced by the option `listof` with values `graduated` or `flat` (see [page 133](#)). Moreover, the values `listof` and `listofnumbered` of option `toc` (see [section 3.9](#)) as well as the values `totoc` and `numbered` of the previous described option `listof` have influence to the lists of floating environments.

Mostly the lists of floating environment may be found after the table of contents. But some publishers like to have these lists at the appendix. Nevertheless the author of this guide prefers to find them immediately after the table of contents.

3.21. Margin Notes

Aside from the text area, that normally fills the typing area, usually a marginalia column may be found. Margin notes will be printed at this area. At lot of them may be found in this manual.

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

```
\marginline{margin note}
```

Usually margin notes in L^AT_EX 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 this document, sometimes, the class name `scartcl` can be found in the margin. This can be produced 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.

Experts and advanced users may find information about problems using `\marginpar` at [section 21.1, page 441](#). These are valid for `\marginline` also.

3.22. Appendix

The appendix of a document contains mainly the enclosures to the document. These are typically bibliography, index, glossary. But only for this parts nobody would and should start an appendix, because the formation of these already distinguishes them from the main document. But if there are additional elements at the appendix, i. e., cited third party documents, endnotes, figures or tabulars, the standard elements like the bibliography should also be part of the appendix.

`\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 option `numbers` in [section 3.16, page 93](#).

Die output of the chapter headings in the appendix are influenced by the options `chapterprefix` and `appendixprefix`. See [section 3.16, page 91](#) for more information.

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

3.23. Bibliography

The bibliography opens up external resources. Mainly bibliographies will be made by program BibTeX or biber using an external file in database like structure. Thereby BibTeX style influences not only the formation of the bibliography entries but also their sorting. Using an additional bibliography style like natbib, babelbib, or biblatex limits the influence of KOMA-Script to the bibliography hardly. In such cases it is important so see the manual of the bibliography package! General information about bibliography may be found in [OPHS11].

```
bibliography=selection
```

v3.00

For a start, *selection* may be any already defined bibliography formation style. There are two predefined formation styles at KOMA-Script. You should not misconceive them with the styles used by L^AT_EX which you may select using `\bibstyle`. While L^AT_EX influences not only the sorting but also the contents of the bibliography, KOMA-Script influences only some basic features of the bibliography or a tiny amount of formation features of the entries to the bibliography.

Option `bibliography=oldstyle` selects a compact formation of the bibliography entries. In this case command `\newblock` inside of the entries will only result in a small horizontal distance. The name is a result of the fact, that this is the mostly used classic kind of bibliography. In opposite to this `bibliography=openstyle` selects a more modern and open kind of bibliography. The name is a result of the fact, that command `\newblock` inserts a paragraph break. The entries will be more structured by this. They are less compact and seem more relaxed or open. Information about definition of new formation styles may be found in description of command `\newbibstyle` in [section 21.3](#) at [page 462](#).

Beside the formation style one more feature may be selected using *selection*. The bibliography is a kind of contents list. But instead of listing contents of the document itself, it references to external contents. Because of this, someone may say, that the bibliography is a chapter or section on its own and should have a chapter or section number. You may select this with option `bibliography=totocnumbered` which will therefor also generate an entry to the table of contents. In my opinion the bibliography is nothing you've written on your own and so does not merits a numbered entry to the table of contents. A entry without number may be set with option `bibliography=totoc`. Nevertheless, the default would be neither a number nor an entry to the table of contents and corresponds to `bibliography=nottotoc`. For more information see option `toc` in [section 3.9](#), especially values `bibliographynumbered`, `bibliography`, and `nobibliography` to this option at [page 69](#).

v3.12

Sometimes it is not usefull to have one bibliography for the whole document but a bibliography at every chapter of a document made using `scrbook` or `scrreprt`. In that case you'd need the bibliography itself not to be a chapter but one level below, a section. You may achieve this using Option `bibliography=leveldown`. You may use this also if you'd combine several lists and the bibliography together below one heading. So this option is also available with `scartcl`.

A summary of all available values for option `bibliography` may be found in [table 3.21](#). Nevertheless you should note, that additional values may be generated using `\newbibstyle`.

```
\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 need not be directly in front of it. For example, it could be placed at the beginning of the document. Similar to the options `bibliography=totoc` and `bibliography=`

Table 3.21.: Predefined values of option `bibliography` to select the formation of the bibliography

leveldown

v3.12

The bibliography will use a heading of one step lower section level than default.

nottotoc

The bibliography will neither have an entry at the table of contents nor a number,

oldstyle

The bibliography will use the classic, compact formation, where `\newblock` generates an expandable horizontal distance only.

openstyle

The bibliography will use the structured, open formation, where `\newblock` generates a paragraph break.

totoc

The bibliography will have an entry at the table of contents but no number.

totocnumbered

The bibliography will have an entry at the table of contents and a number at the heading.

`totocnumbered`, 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 [Dal10]).

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.

\BreakBibliography*{interruption code}*

v3.00

This command exists only if the environment `thebibliography` has not been redefined by another package. It provides a break at the bibliography. The *interruption code* will be expanded inside a group. Such a break may be, e. g., a heading using `\minisec`. Unfortunately

it is not possible to add this command to the BibTeX database using, e.g., a special kind of BibTeX entry. Because of this, users may use it currently only if they make the bibliography on their own. Because of this usage is very limited.

```
\AfterBibliographyPreamble{code}
\AtEndBibliography{code}
```

v3.00

In some cases it may be useful to add some *code* after the bibliography preamble or just before the end of the bibliography. This may be achieved using one of these instructions.

Example: You want to set the bibliography not justified but ragged right. This may be achieved using:

```
\AfterBibliographyPreamble{\raggedright}
```

You may place this instruction anywhere before the bibliography. Nevertheless it is recommended to do so at the preamble of the document or inside your own package.

The functionality of this instruction depends on cooperation with packages modifying the bibliography, if such a package should be used (see [section 21.2](#), [page 441](#)).

3.24. Index

For general information about making an index see [OPHS11], [Lam87], and [Keh97]. Using a package, that redefines commands or environments for the index, may limit the influence of KOMA-Script to the index hardly. This would be valid, e.g., for usage of package `index` but not for usage of package `splitidx` (see [Koh14b]).

```
index=selection
```

v3.00

The index is chapter (`scrbook` or `scrreprt`) or section (`scartcl`) without heading number or entry at the table of contents by default or option `index=default`. The index does not need an entry at the table of contents, because it should always be the last element of a document. Nevertheless, such an entry may be achieved using option `index=totoc`. You can even number the index using option `index=numbered`. See also option `toc` with value `index` or `indexnumbered` in [section 3.9](#) from [page 69](#) onward.

v3.18

A summary of all available values for option `index` may be found in [table 3.22](#).

Table 3.22.: Available values of option `index`

`default`, `nottotoc`, `plainheading`

The index will not have an entry at the table of contents.

`numbered`, `totocnumbered`

The index will have an entry at the table of contents and also will have a heading number.

`totoc`, `toc`, `notnumbered`

The index will have an entry at the table of contents, but will not have a heading number.

v3.18

```
\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 3.12, page 79](#)).

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

The New Letter Class `scrlettr2`

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.

4.1. 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 4.9, page 164](#)). 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 4.1](#).

Table 4.1.: Alphabetical list of all supported variables in `scrlettr2`

<code>addresseeimage</code>	instuctions, that will be used to print the Port-Payé head of option <code>addrfield=backgroundimage</code> or the Port-Payé addressee of option <code>addrfield=image</code> (section 4.10, page 182)
<code>backaddress</code>	return address for window envelopes (section 4.10, page 182)
<code>backaddressseparator</code>	separator within the return address (section 4.10, page 182)

Table 4.1.: Alphabetical list of all supported variables in scrLtr2 (*continuation*)

ccseparator	
separator between title of additional addressees, and additional addressees (section 4.7, page 159)	
customer	
customer number (section 4.10, page 189)	
date	
date (section 4.10, page 189)	
emailseparator	
separator between e-mail name and e-mail address (section 4.10, page 176)	
enclseparator	
separator between title of enclosure, and enclosures (section 4.7, page 159)	
faxseparator	
separator between title of fax, and fax number (section 4.10, page 176)	
v3.08	firstfoot
	page foot of the note paper (section 4.10, page 196)
v3.08	firsthead
	page head of the note paper (section 4.10, page 182)
fromaddress	
sender’s address without sender name (section 4.10, page 172)	
frombank	
sender’s bank account (section 4.10, page 196)	
fromemail	
sender’s e-mail (section 4.10, page 176)	
fromfax	
sender’s fax number (section 4.10, page 176)	
fromlogo	
commands for inserting the sender’s logo (section 4.10, page 180)	
v3.12	frommobilephone
	sender’s mobile telephone number (section 4.10, page 176)

Table 4.1.: Alphabetical list of all supported variables in scrLtr2 (*continuation*)

fromname	complete name of sender (section 4.10, page 172)
fromphone	sender’s telephone number (section 4.10, page 176)
fromurl	a URL of the sender, e. g., the URL of his homepage (section 4.10, page 176)
fromzipcode	zip code or postcode of the sender used at the Port-Payé head of option addrfield=PP (section 4.10, page 182)
invoice	invoice number (section 4.10, page 189)
location	more details of the sender (section 4.10, page 187)
myref	sender’s reference (section 4.10, page 189)
nextfoot	page foot using page style headings or myheadings (section 4.13, page 203)
nexthead	page head using page style headings or myheadings (section 4.13, page 203)
phoneseparator	separator between title of telephone and telephone number (section 4.10, page 176)
place	sender’s place used near date (section 4.10, page 182)
placeseparator	separator between place and date (section 4.10, page 190)
PPdatamatrix	instruction, that print the data array of option addrfield=PP (section 4.10, page 182)

Table 4.1.: Alphabetical list of all supported variables in scrLtr2 (*continuation*)

PPcode	instructions for the sender’s identification code of option <code>addrfield=PP</code> (section 4.10, page 182)
signature	signature beneath the ending of the letter (section 4.20, page 212)
specialmail	mode of dispatch (section 4.10, page 182)
subject	letter’s subject (section 4.10, page 193)
subjectseparator	separator between title of subject and subject (section 4.10, page 193)
title	letter title (section 4.10, page 192)
toaddress	address of addressee without addressee name (section 4.10, page 182)
toname	complete name of addressee (section 4.10, page 182)
yourmail	date of addressee’s referenced mail (section 4.10, page 189)
yourref	addressee’s reference (section 4.10, page 189)
zipcodeseparator	separator between the zip code’s or postcode’s title and the code itself (section 4.10, page 182)

```
\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 or title. In [section 22.2, page 480](#) you may find more information about defining variable on your own.

```
\ifkomavar{name}{true-code}{false-code}
```

v3.03

This command may be used to test, whether or not a variable has already been defined. The *true-code* will be executed only, if the variable already exists. The contents of the variable will not be examined, so it may be empty. The *false-code* will be executed if the variable does not yet exist. Such tests make sense if a variable will be defined at one lco-file (see [section 4.21](#) from [page 214](#) onward), but used in another lco-file if it exists only.

```
\ifkomavareempty{name}{true-code}{false-code}
\ifkomavareempty*{name}{true-code}{false-code}
```

v2.9i

With these commands you may check whether or not the expanded content or description of a variable is empty. The *true-code* will be executed if the content or description is empty. Otherwise the *false-code* will be executed. The starred variant handles the description of a variable, the unstarred variant handles the contents.

4.2. Pseudo-Lengths

L^AT_EX processes length with commands `\newlength`, `\setlength`, `\addtolength`, and `\the length`. Many packages also use macros, that are commands, to store lengths. KOMA-Script extends the method of storing length at macros by some commands similar to the commands above, that are used to handle real lengths. KOMA-Script calls this kind of lengths, that are stored at macros instead of real L^AT_EX lengths, pseudo-lengths.

A list of all pseudo-lengths in scrlltr2 is shown in [table 22.1](#) starting at [page 465](#). The meaning of the various pseudo-lengths is shown graphically in [figure 22.1](#). 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.

Normally users would not need to define a pseudo-length on their own. Because of this, definition of pseudo-lengths will be described in the expert part at [section 22.1](#), [page 468](#). Setting pseudo-lengths to new values is also a work for advanced users. So this will be described in the expert part too at [page 470](#).

Please note: Even though these pseudo-lengths are internally implemented as macros, the commands for pseudo-length management expect only the names of the pseudo-lengths not the macros representing the pseudo-lengths. The names of pseudo-lengths are without backslash at the very beginning similar to the names of L^AT_EX counters and in opposite to macros or L^AT_EX lengths.

```
\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 (see [section 4.21](#) ab [page 214](#)).

```
\setlengthtoplength[factor]{length}{pseudo-length}
\addtolengthlength[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 2 pt; then `3\test` gives you the value 6 pt. Using pseudo-lengths instead, `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 `\addtolengthlength` works very similarly; it adds the multiple of a *pseudo-length* to the *length*.

4.3. Early or late Selection of Options

All of what is described in [section 2.4](#) is generally applicable. So if you have already read and understood [section 2.4](#) you can switch to [section 4.4, page 149](#).

In this section a peculiarity of KOMA-Script is presented, which, apart from the class `scrlltr2` is also relevant to other KOMA-Script classes and packages. Such that the user can find all information corresponding to a single package or a single class in the relevant chapter, this section is found almost identically in several chapters. Users who are not only interested in a particular package or class, but wish to gain an overview of KOMA-Script as a whole, may read the section in one chapter and may thereafter skip it wherever coming across it in the document.

```
\documentclass[option list]{KOMA-Script class}
\usepackage[option list]{package list}
```

In L^AT_EX, provision is made for the user to pass class options as a comma-separated list of keywords as optional arguments to `\documentclass`. Apart from being passed to the class, these options are also passed on to all packages which can understand the options. Provision is also made for the user to pass optional arguments as a comma-separated list of keywords as optional arguments to `\usepackage`. KOMA-Script expands the option mechanism for the KOMA-Script classes and various packages to use further possibilities. Thus, most KOMA-Script options can also take a value. An option may have not only the form *Option*, but may also have the form *option=value*. Apart from this difference `\documentclass` and `\usepackage` function the same in KOMA-Script as described in [\[Tea05b\]](#) or any introduction to L^AT_EX, for example [\[OPHS11\]](#).

When using a KOMA-Script class, no options should be passed on loading of the `typearea` or `scrbase` packages. The reason for this is that the class already loads these packages without options and L^AT_EX refuses multiple loadings of a package with different option settings. Actually, it is no longer necessary when using any KOMA-Script class to explicitly load either one of these packages.

You should note, that in opposite to the interface described below the options interface of `\documentclass` and `\usepackage` is not robust. So commands, lengths, counters and such constructs may break inside the optional argument of these commands. Because of this, the usage of a L^AT_EX length inside the value of an option would cause an error before KOMA-Script

can get the control over the option execution. So, if you want to use a L^AT_EX length, counter or command a part of the value of an option, you have to use `\KOMAOPTIONS` or `\KOMAOPTION`. These commands will be described next.

```
\KOMAOPTIONS{option list}
\KOMAOPTION{option}{value list}
```

v3.00

KOMA-Script offers most class and package options the opportunity to change the value of options even after loading of the class or package. One may then change the values of a list of options at will with the `\KOMAOPTIONS` command. Each option in the *option list* has the form *option=value*.

Some options also have a default value. If one does not give a value, i. e., gives the option simply in the form *option*, then the default value will be used.

Some options can assume several values simultaneously. For such options there exists the possibility, with the help of `\KOMAOPTION`, to pass a single *option* a list of values. The individual values are given as a comma-separated *value list*.

To implement this possibility KOMA-Script uses the commands `\FamilyOPTIONS` and `\FamilyOPTION` with the family “KOMA”. For more information in these commands see [part II, section 12.2, page 313](#).

4.4. Compatibility with Earlier Versions of KOMA-Script

It applies, mutatis mutandis, what is written in [section 2.5](#). However, this feature exists at scrlltr2 since version 2.9t. o if you have already read and understood [section 2.5](#) you can switch to [page 150, page 150](#).

In some cases improvement and bug corrections of classes will result in changes of the behaviour and make-up. But sometimes this is not wanted.

```
version=value
version=first
version=last
```

v3.00a

Since version 2.9t of scrlltr2 it's your choice if your source code should result in the same make-up at future L^AT_EX runs or if you like to participate in all improvements of new releases of the class. You may select the compatible version of KOMA-Script with option `version`. Compatibility to the lowest supported KOMA-Script release may be achieved by `version=first` or `version=2.9` or `version=2.9t`. Setting *value* to an unknown release number will result in a warning message and selects `version=first` for safety.

v3.01a

With `version=last` the most recent version will be selected at every L^AT_EX run. Be warned, though, that using `version=last` poses possibilities of compatibility issues for future L^AT_EX runs. Option `version` without any *value* means the same. This is the default behaviour as long as you do not use any deprecated options.

If you use a deprecated option of KOMA-Script 2, KOMA-Script 3 will switch to `version=first` automatically. This will also result in a warning message that explains how to prevent this switching. Alternatively you may select another adjustment using option `version` with the wanted compatibility after the deprecated option.

Compatibility is primarily make-up compatibility. New features not related to the mark-up will be available even if you switch compatibility to a version before first implementation of the feature. Option `version` does not influence make-up changes that are motivated by bug fixes. If you need bug compatibility you should physically save the used KOMA-Script version together with your document.

Example: The letter examples of this chapter should use and show all the features of the recent KOMA-Script release. To achieve this, we set the corresponding compatibility:

```
\documentclass[version=last]{screlttr2}
```

In this case the symbolic value `last` has been used to select the latest version.

Please note that you cannot change option `version` anymore after loading the class. Therefore, the usage of option `version` within the argument of `\KOMAOPTIONS` or `\KOMAOPTION` is not recommended and will cause an error.

4.5. Draft Mode

What is written in [section 3.3](#) applies, mutatis mutandis. So if you have already read and understood [section 3.3](#) you can jump to [section 4.6](#) on [page 151](#).

Many classes and packages provide a draft mode aside from the final typesetting mode. The difference of draft and final mode may be as manifold as the classes and package that support these modes. For instance, the `graphics` and the `graphicx` packages do not actually output the graphics in their own draft mode. Instead they output a framed box of the appropriate size containing only the graphic's file name (see [\[Car05\]](#)).

`draft=simple switch`

v3.00

This option is normally used to distinguish between the draft and final versions of a document. *simple switch* value may be any standard value from [table 2.5](#), [page 39](#). In particular, switching on the option 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 default `draft=false` option no such boxes are shown. Such overly long lines often vanish using package `microtype` [\[Sch13\]](#).

4.6. Page Layout

Each page of a document is separated into several different layout elements, e. g., margins, head, foot, text area, margin note column, and the distances between all these elements. KOMA-Script additionally distinguishes the page as a whole also known as the paper and the viewable area of the page. Without doubt, the separation of the page into the several parts is one of the basic features of a class. Nevertheless at KOMA-Script the classes delegate that business to the package `typearea`. This package may be used with other classes too. In difference to those other classes the KOMA-Script classes load `typearea` on their own. Because of this, there's no need to load the package explicitly with `\usepackage` while using a KOMA-Script class. Nor would this make sense or be useful. See also [section 4.3](#).

Some settings of KOMA-Script classes do influence the page layout. Those effects will be documented at the corresponding settings.

For more information about page size, separation of pages into margins and type area, and about selection of one or two column typesetting see the documentation of package `typearea`. You may find it at [chapter 2](#) from [page 25](#) onwards.

Normally it makes no sense to distinguish letters with single-side layout and letters with double-side layout. Mostly letters are not bound like books. Therefor each page will be viewed on its own. This is also true if both sides of the paper sheet will be used for printing. Vertical adjustment is unusual at letters too. Nevertheless, if you need or want it, you may read the description of `\raggedbottom` and `\flushbottom` in [section 3.4](#) at [page 54](#).

4.7. General Structure of Letter Documents

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 4.1](#), 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 compatibility to prior versions of `scrlettr2` (see option [version](#) in [section 4.4](#), [page 149](#)).

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 [\[BB13\]](#)) is used, or language-dependent variables of `scrlettr2` 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 4.2](#), every single letter itself consists of an introduction, the letter body,

Figure 4.1.: General structure of a letter document with several individual letters (the structure of a single letter is shown in figure 4.2)

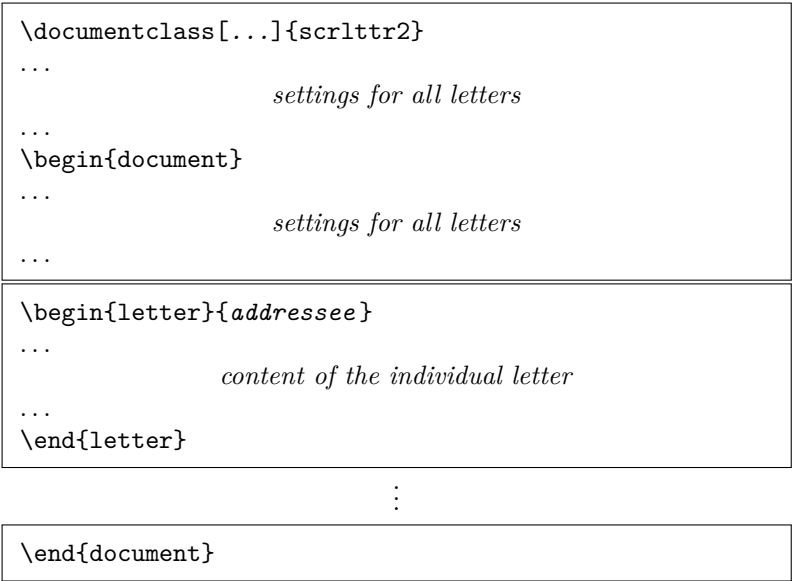
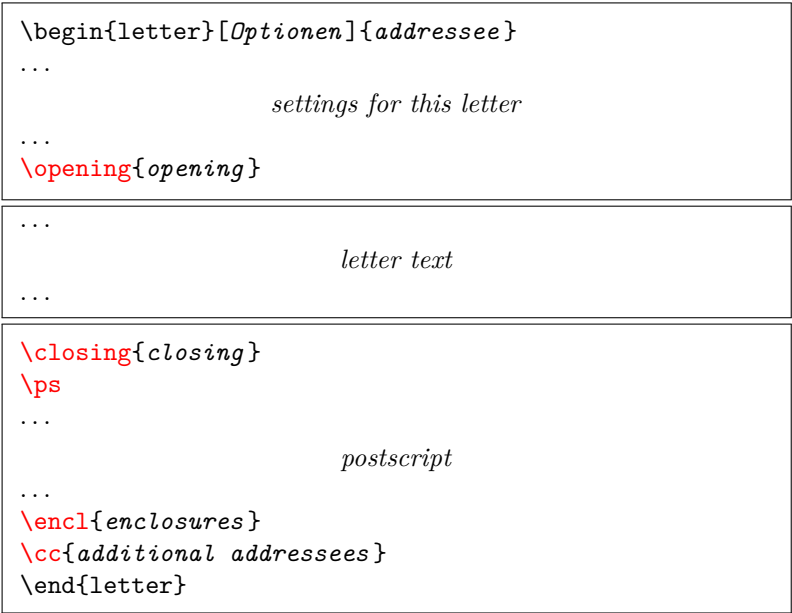


Figure 4.2.: General structure of a single letter within a letter document (see also figure 4.1)



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.

```
\begin{letter}[options]{addressee}...\end{letter}
```

The `letter` environment is one of the key environments of the letter class. A special `scrlettr2` 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. 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.

Example: Assumed, someone wants to send a letter to Joana Public. A minimalistic letter document for this may be:

```
\documentclass[version=last]{scrlettr2}
\usepackage[english]{babel}
\begin{document}
\begin{letter}{Joana Public\\
    Hillside 1\\
    12345 Public-City}
\end{letter}
\end{document}
```

However, this would not result in any printable output. At least there would not be an addressee at the note paper sheet. The reason for this will be explained at the description of command `\opening` at [page 155](#).

```
\AtBeginLetter{instruction code}
\AtEndLetter{instruction code}
```

L^AT_EX enables the user to declare *instruction code* whose execution is delayed until a determined point. Such points are called *hooks*. Known macros for using hooks are `\AtBeginDocument` and `\AtEndOfClass` at the L^AT_EX kernel. The class `scrlltr2` provides two more hooks. The *instruction code* for these may be declared using `\AtBeginLetter` and `\AtEndLetter`. Originally, hooks were provided for package and class authors, so they are documented in [Tea06] only, and not in [Tea05b]. However, with letters there are useful applications of `\AtBeginLetter` as the following example may illustrate.

v2.95

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 `scrlltr2`, 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
  \noindent\begin{tabularx}{\textwidth}{1@{}X}
    \theQuestion:~ & #1\\
  \end{tabularx}%
}%
\AtBeginLetter{\setcounter{Question}{0}}
```

This way first question remains question 1, even in the 1001st letter. Of course the definition at this example needs package `tabularx` (see [Car99b]).

```
letter
\thisletter
\letterlastpage
```

v3.19

If you have more than one letter in one document, it is useful to have a letter number. Since version 3.19 KOMA-Script provides counter `letter` and increases it at every `\begin{letter}`.

Example: Have one more look into the `\AtBeginLetter` example. Instead of resetting the counter explicitly at `\begin{letter}`, we can do it implicitly by defining counter `Question` depending on counter `letter`:

```
\newcounter{Question}[letter]
\newcommand{\Question}[1]{%
  \refstepcounter{Question}\par
```

```

\noindent\begin{tabularx}{\textwidth}{l@{X}}
  \theQuestion:~ & #1\\
\end{tabularx}%
}%

```

Now, the new counter will be reset at every start of a new letter and the first question of every letter will be number one.

If you want the output of current value of `letter`, you may usually use `\theletter`. Indeed the letter can also be used for cross-references. So you can use `\label{name}` to generate a label immediately after `\begin{letter}` and reference it somewhere in the document using `\ref{name}`. Inside the same letter you can simply use `\thisletter` without generating a label to get the same result.

KOMA-Script itself uses `\thisletter` to put a label onto the last page of every letter. You can use `\letterlastpage` to reference the last page number of the current letter. Please note, the value of `\letterlastpage` is valid after some L^AT_EX runs, because it uses `\label` and `\pageref`. So you need at least two or three L^AT_EX runs, if you use `\letterlastpage`. Please have a look at *Rerun* terminal or log-file messages about labels that have been changed.

`\opening{opening}`

This is one of the most important commands in `sclrttr2`. For the user it may seem that only the *opening*, e.g., “Dear Mrs ...”, 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. And if you want to print a letter without opening you have to use an `\opening` command with an empty argument.

Example: Let’s extend the example from [page 153](#) by an opening:

```

\documentclass[version=last]{sclrttr2}
\usepackage[english]{babel}
\begin{document}
\begin{letter}{%
  Joana Public\\
  Hillside 1\\
  12345 Public-City%
}
\opening{Dear chairman,}
\end{letter}
\end{document}

```

This will result in a note paper sheet shown in [figure 4.3](#).

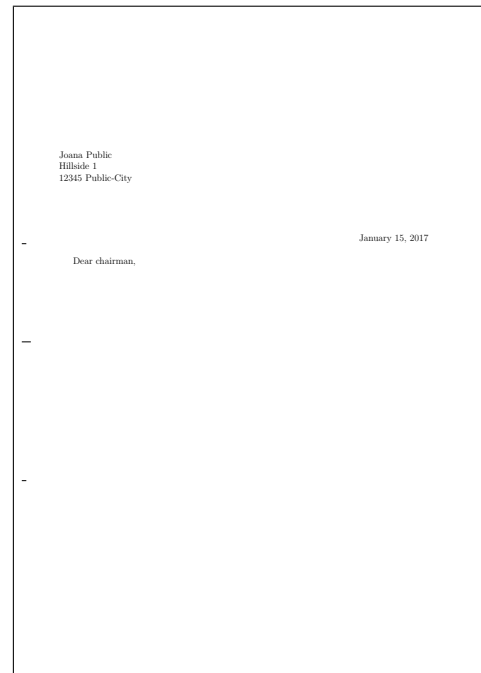


Figure 4.3.: result of a minimalistic letter with addressee and opening only (date and folding marks are defaults of DIN-letters)

`\closing{closing phrase}`

The main purpose of the command `\closing` is to typeset the *closing phrase*. This may even consists of multiple lines. The lines should be separated by double backslash. Paragraph breaks inside the *closing phrase* are not allowed.

Beyond that the command also typesets the content of the variable `signature`. More information about the signature and the configuration of the signature may be found at [section 4.20](#) ab [page 212](#).

Example: Let's extend the our example by some lines of text and a closing phrase:

```
\documentclass[version=last]{scr1ttr2}
\usepackage[english]{babel}
\begin{document}
\begin{letter}{%
  Joana Public\\
  Hillside 1\\
  12345 Public-City%
}
\opening{Dear chairman,}
the last general meeting was about one year ago.
I want to remind you, that the constitution of our
club advises you to make a general meeting every
```

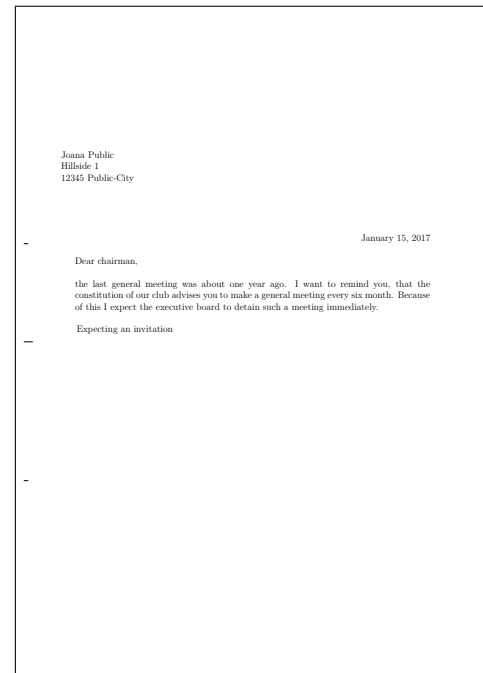


Figure 4.4.: result of a small letter with addressee, opening, text, and closing (date and folding marks are defaults of DIN-letters)

```
six month. Because of this I expect the executive
board to detain such a meeting immediately.
\closing{Expecting an invitation}
\end{letter}
\end{document}
```

This will result in a the letter shown in [figure 4.4](#).

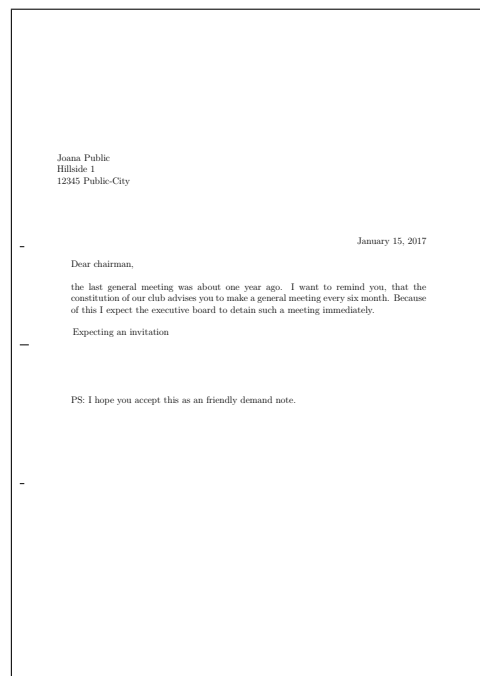
\ps

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 `scr1ttr2`.

Example: The example letter extended by a postscript

```
\documentclass[version=last]{scr1ttr2}
\usepackage[english]{babel}
\begin{document}
\begin{letter}{%
  Joana Public\\
```

Figure 4.5.: result of a small letter with addressee, opening, text, closing, and postscript (date and folding marks are defaults of DIN-letters)



```
Hillside 1\\
12345 Public-City%
}
\opening{Dear chairman,}
the last general meeting was about one year ago.
I want to remind you, that the constitution of our
club advises you to make a general meeting every
six month. Because of this I expect the executive
board to detain such a meeting immediately.
\closing{Expecting an invitation}
\ps PS: I hope you accept this as an friendly
demand note.
\end{letter}
\end{document}
```

results in [figure 4.5](#).

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.

```
\cc{distribution list}
\setkomavar{ccseparator}[description]{contents}
```

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: This time, the example letter should be send not only to the chairman, but also to all club members:

```
\documentclass[version=last]{scrlltr2}
\usepackage[english]{babel}
\begin{document}
\begin{letter}{%
  Joana Public\\
  Hillside 1\\
  12345 Public-City%
}
\opening{Dear chairman,}
the last general meeting was about one year ago.
I want to remind you, that the constitution of our
club advises you to make a general meeting every
six month. Because of this I expect the executive
board to detain such a meeting immediately.
\closing{Expecting an invitation}
\ps PS: I hope you accept this as an friendly
demand note.
\cc{executive board\\all members}
\end{letter}
\end{document}
```

The result is shown in [figure 4.6](#).

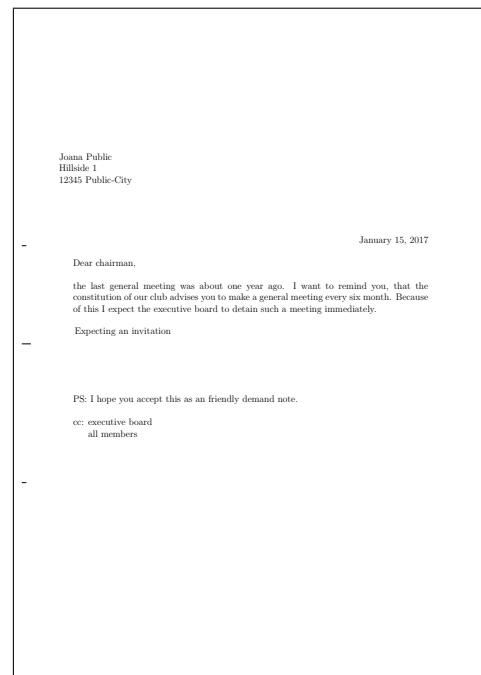
In front of the distribution list a vertical gap is inserted automatically.

```
\encl{enclosures}
\setkomavar{enclseparator}[description]{contents}
```

The *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`.

Example: Now, the example letter will be extended by some paragraphs from the constitution. These will be added as an enclosure. The description title will be changed

Figure 4.6.: result of a small letter with addressee, opening, text, closing, postscript, and distribution list (date and folding marks are defaults of DIN-letters)



also, because there is only one enclosure and the default may be prepared for several enclosures:

```
\documentclass[version=last]{srlttr2}
\usepackage[english]{babel}
\begin{document}
\begin{letter}{%
  Joana Public\\
  Hillside 1\\
  12345 Public-City%
}
\opening{Dear chairman,}
the last general meeting was about one year ago.
I want to remind you, that the constitution of our
club advises you to make a general meeting every
six month. Because of this I expect the executive
board to detain such a meeting immediately.
\closing{Expecting an invitation}
\ps PS: I hope you accept this as an friendly
demand note.
\setkomavar*{enclseparator}{Enclosure}
\encl{Compendium of the constitution with the
general meeting paragraphs.}
```

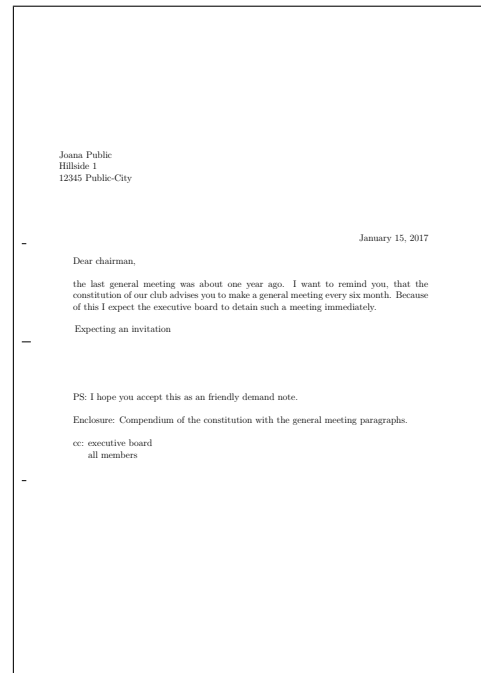



Figure 4.7.: result of a small letter with addressee, opening, text, closing, postscript, distribution list, and enclosure (date and folding marks are defaults of DIN-letters)

```
\cc{executive board\\all members}
\end{letter}
\end{document}
```

This will result in [figure 4.7](#).

4.8. Selection of the Document Font Size

What is described in [section 3.5](#) applies, mutatis mutandis. So if you have already read and understood [section 3.5](#) you can jump to the example at the end of this section on [page 162](#).

The main document font size is one of the basic decisions for the document layout. The maximum width of the text area, and therefore splitting the page into text area and margins, depends on the font size as stated in [chapter 2](#). The main document font will be used for most of the text. All font variations either in mode, weight, declination, or size should relate to the main document font.

`fontsize=size`

In contrast to the standard classes and most other classes that provide only a very limited number of font sizes, the KOMA-Script classes offer the feature of selection of any desired *size* for the main document font. In this context, any well known T_EX unit of measure may be used and using a number without unit of measure means **pt**.

If you use this option inside the document, the main document font size and all dependent sizes will change from this point. This may be useful, e.g., if one more letter should be set using smaller fonts on the whole. It should be noted that changing the main font size does not result in an automatic recalculation of type area and margins (see `\recalctypearea`, [section 2.4, page 37](#)). On the other hand, each recalculation of type area and margins will be done on the basis of the current main font size. The effects of changing the main font size to other additionally loaded packages or the used document class depend on those packages and the class. This may even result in error messages or typesetting errors, which cannot be considered a fault of KOMA-Script, and even the KOMA-Script classes do not change all lengths if the main font size changes after loading the class.

This option is not intended to be a substitution for `\fontsize` (see [Tea05a]). Also, you should not use it instead of one of the main font depending font size commands `\tiny` up to `\Huge`! Default at scrllttr2 is `fontsize=12pt`.

Example: Assumed, the example is a letter to “*The friends of insane font sizes*” and therefore it should be printed with 14 pt instead of 12 pt. Only a simple change of the first line is needed:

```
\documentclass[version=last,fontsize=14pt]{scrllttr2}
\usepackage[english]{babel}
\begin{document}
\begin{letter}{%
  Joana Public\\
  Hillside 1\\
  12345 Public-City%
}
\opening{Dear chairman,}
the last general meeting was about one year ago.
I want to remind you, that the constitution of our
club advises you to make a general meeting every
six month. Because of this I expect the executive
board to detain such a meeting immediately.
\closing{Expecting an invitation}
\ps PS: I hope you accept this as an friendly
demand note.
\setkomavar*{enclseparator}{Enclosure}
\encl{Compendium of the constitution with the
general meeting paragraphs.}
```

```
\cc{executive board\\all members}
\end{letter}
\end{document}
```

Alternatively the option may be set at the optional argument of the `letter` environment:

```
\documentclass[version=last]{srlttr2}
\usepackage[english]{babel}
\begin{document}
\begin{letter}[fontsize=14pt]{%
  Joana Public\\
  Hillside 1\\
  12345 Public-City%
}
\opening{Dear chairman,}
the last general meeting was about one year ago.
I want to remind you, that the constitution of our
club advises you to make a general meeting every
six month. Because of this I expect the executive
board to detain such a meeting immediately.
\closing{Expecting an invitation}
\ps PS: I hope you accept this as an friendly
demand note.
\setkomavar*{enclseparator}{Enclosure}
\encl{Compendium of the constitution with the
general meeting paragraphs.}
\cc{executive board\\all members}
\end{letter}
\end{document}
```

In the case of this late change of the font size no recalculation of the type area will happen. Because of this, the two results of [figure 4.8](#) differ.

4.9. Text Markup

What is described in [section 3.6](#) applies, *mutatis mutandis*. So if you have already read and understood [section 3.7](#) you can switch to [page 167](#). But you should have a look at [table 4.2](#), [page 165](#).

L^AT_EX offers different possibilities for logical and direct markup of text. Selection of the font family commands, as well as choosing the font size and width is supported. More information about the standard font facilities may be found at [\[OPHS11\]](#), [\[Tea05b\]](#), and [\[Tea05a\]](#).

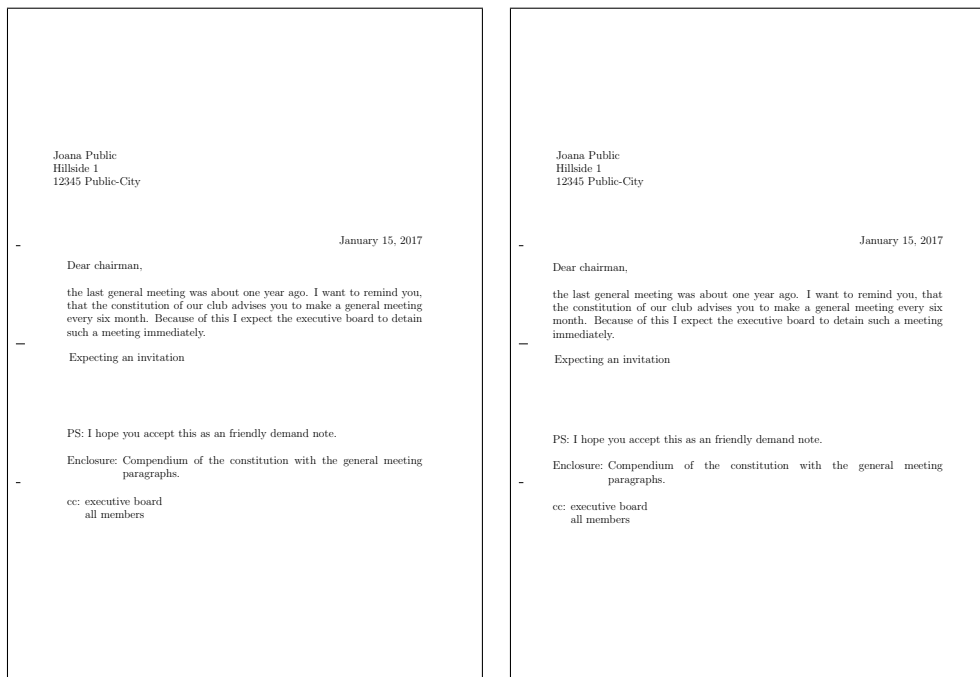


Figure 4.8.: result of a small letter with addressee, opening, text, closing, postscript, distribution list, enclosure, and insane large font size (date and folding marks are defaults of DIN-letters): at left one the font size has been defined by the optional argument of `letter`, at the right one the optional argument of `\documentclass` has been used

```
\textsuperscript{Text}
```

```
\textsubscript{Text}
```

The L^AT_EX-Kern already defines the command `\textsuperscript` to superscript text. Unfortunately, until release 2015/01/01 L^AT_EX itself does not offer a command to produce text in subscript instead of superscript. KOMA-Script defines `\textsubscript` for this purpose. You may find an example of usage at [section 3.6, page 56](#).

```
\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`, and `\scshape`, as well as the font size commands `\Huge`, `\huge`, `\LARGE`, `\Large`, `\large`, `\normalsize`, `\small`, `\footnotesize`, `\scriptsize`, and `\tiny`. The description of these commands can be found in [OPHS11], [Tea05b], or [Tea05a]. Color switching commands like `\normalcolor` (see [Car05] and [Ker07]) are also acceptable. The behavior when using other commands, especially those that make redefinitions 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 inside the document body, but only in the document preamble. Usage examples can be found in the paragraphs on the corresponding element. Names and meanings of the individual items are listed in table 4.2. The default values are shown in the corresponding paragraphs.

With command `\usekomafont` the current font style may be changed into the font style of the selected *element*. A general example for the usage of `\setkomafont` and `\usekomafont` may be found in section 3.6 at page 57.

Table 4.2.: Alphabetical list of elements whose font can be changed in scrlltr2 using the commands `\setkomafont` and `\addtokomafont`

addressee	name und address in address field (section 4.10, page 182)
backaddress	return address for a window envelope (section 4.10, page 182)
descriptionlabel	label, i.e., the optional argument of <code>\item</code> , in a description environment (section 4.16, page 209)
foldmark	foldmark on the letter page; intended for color settings (section 4.10, page 168)
footnote	footnote text and marker (see section 4.15, page 206)
footnotelabel	mark of a footnote; used according to the element footnote (see section 4.15, page 206)
footnotereference	footnote reference in the text (see section 4.15, page 206)

Table 4.2.: Elements whose font can be changed (*continuation*)**footnoterule**

v3.07

horizontal rule above the footnotes at the end of the text area (see [section 3.14](#), [page 88](#))

labelinglabel

labels, i. e., the optional argument of `\item` in the **labeling** environment (see [section 4.16](#), [page 209](#))

labelingseparator

separator, i. e., the optional argument of the **labeling** environment; used according to the element **labelinglabel** (see [section 4.16](#), [page 209](#))

pagefoot

used after element **pageheadfoot** for the page foot, that has been defined with variable **nextfoot**, or for the page foot of package `scrlayer-scrpage` ([chapter 5](#), [page 234](#))

pagehead

another name for **pageheadfoot**

pageheadfoot

the head of a page, but also the foot of a page at all page style, that has been defined using KOMA-Script (see [section 4.13](#), [page 202](#))

pagenumber

page number in the header or footer (see [section 4.13](#), [page 202](#))

pagination

another name for **pagenumber**

placeanddate

v3.12

place and date, if a date line will be used instead of a normal reference line ([section 4.10](#), [page 190](#))

refname

description or title of the fields in the reference line ([section 4.10](#), [page 189](#))

refvalue

content of the fields in the reference line ([section 4.10](#), [page 189](#))

specialmail

mode of dispatch in the address field ([section 4.10](#), [page 182](#))

Table 4.2.: Elements whose font can be changed (*continuation*)**lettersubject**

v3.17

subject in the opening of the letter (section 4.10, page 193)

lettertitle

v3.17

title in the opening of the letter (section 4.10, page 192)

toaddressvariation of the element **addressee** for setting the addressee address (less the name) in the address field (section 4.10, page 182)**toname**variation of the element **addressee** for the name (only) of the addressee in the address field (section 4.10, page 182)

```

\usefontofkomafont{element}
\useencodingofkomafont{element}
\usesizeofkomafont{element}
\usefamilyofkomafont{element}
\useseriesofkomafont{element}
\useshapeofkomafont{element}

```

v3.12

Sometimes and despite the recommendation users use the font setting feature of elements not only for font settings but for other settings too. In this case it may be useful to switch only to the font setting of an element but not to those other settings. You may use `\usefontofkomafont` in such cases. This will activate the font size and baseline skip, the font encoding, the font family, the font series, and the font shape of an element, but no further settings as long as those further settings are local.

You may also switch to one of those attributes only using one of the other commands. Note, that `\usesizeofkomafont` will activate both, the font size and the baseline skip.

You should not misunderstand these commands as a legitimization of using all kind of commands at the font setting of an element. Hence this would result in errors sooner or later (see section 21.3, page 444).

4.10. Note Paper

The note paper is the first page and therefore the signboard of each letter. In business scope often preprinted forms are used, that already contains elements like the letter head with the sender's information and logo. KOMA-Script provides to position these elements independent. With this it is not only possible to replicate the note paper directly, but also to complete

Table 4.3.: Combinable values for the configuration of folding marks with option `foldmarks`

B	activate upper horizontal foldmark on left paper edge
b	deactivate upper horizontal foldmark on left paper edge
H	activate all horizontal folding marks on left paper edge
h	deactivate all horizontal folding marks on left paper edge
L	activate left vertical foldmark on upper paper edge
l	deactivate left vertical foldmark on upper paper edge
M	activate middle horizontal foldmark on left paper edge
m	deactivate middle horizontal foldmark on left paper edge
P	activate punch or center mark on left paper edge
p	deactivate punch or center mark on left paper edge
T	activate lower horizontal foldmark on left paper edge
t	deactivate lower horizontal foldmark on left paper edge
V	activate all vertical folding marks on upper paper edge
v	deactivate all vertical folding marks on upper paper edge

the destined fields instantaneously. The independent position is provided by pseudo-lengths (see [section 4.2](#) from [page 147](#) onward). A schematic display of the note page and the used variable is shown by [figure 4.9](#). Thereby the names of the variables are printed boldly for better distinction from the commands and their arguments.

Following pages are different from the note paper. Following pages in the meaning of this manual are all letter pages but the first one.

foldmarks=selection

Foldmarks or folding marks are tiny horizontal rules at the left margin or tiny vertical rules at the top margin. KOMA-Script currently provides three configurable horizontal folding marks and one configurable vertical folding mark. Additionally it provides a horizontal hole puncher mark, also known as page middle mark. This additional mark cannot be moved vertically.

This option activates or deactivates folding marks for vertical two-, three- or four-panel folding, and a single horizontal folding, of the letter, whereby the folding need not result in equal-sized parts. The position of the four horizontal and the single vertical marks are configurable via pseudo-lengths (see [section 22.1.1](#) from [page 472](#) onwards).

The user has a choice: Either one may use the standard values for simple switches, as described in [table 2.5](#), [page 39](#), to activate or deactivate at once all configured folding marks on the left and upper edges of the paper; or one may specify by one or more letters, as listed in [table 4.3](#), the use of the individual folding marks independently. Also in the latter case the folding marks will only be shown if they have not been switched off generally with one of `false`, `off`, or `no`. The exact positioning of the folding marks is specified in the user settings, that is, the `lco` files (see [section 4.21](#) from [page 214](#) onward) chosen for a letter. Default values are `true` and `TBMPL`.

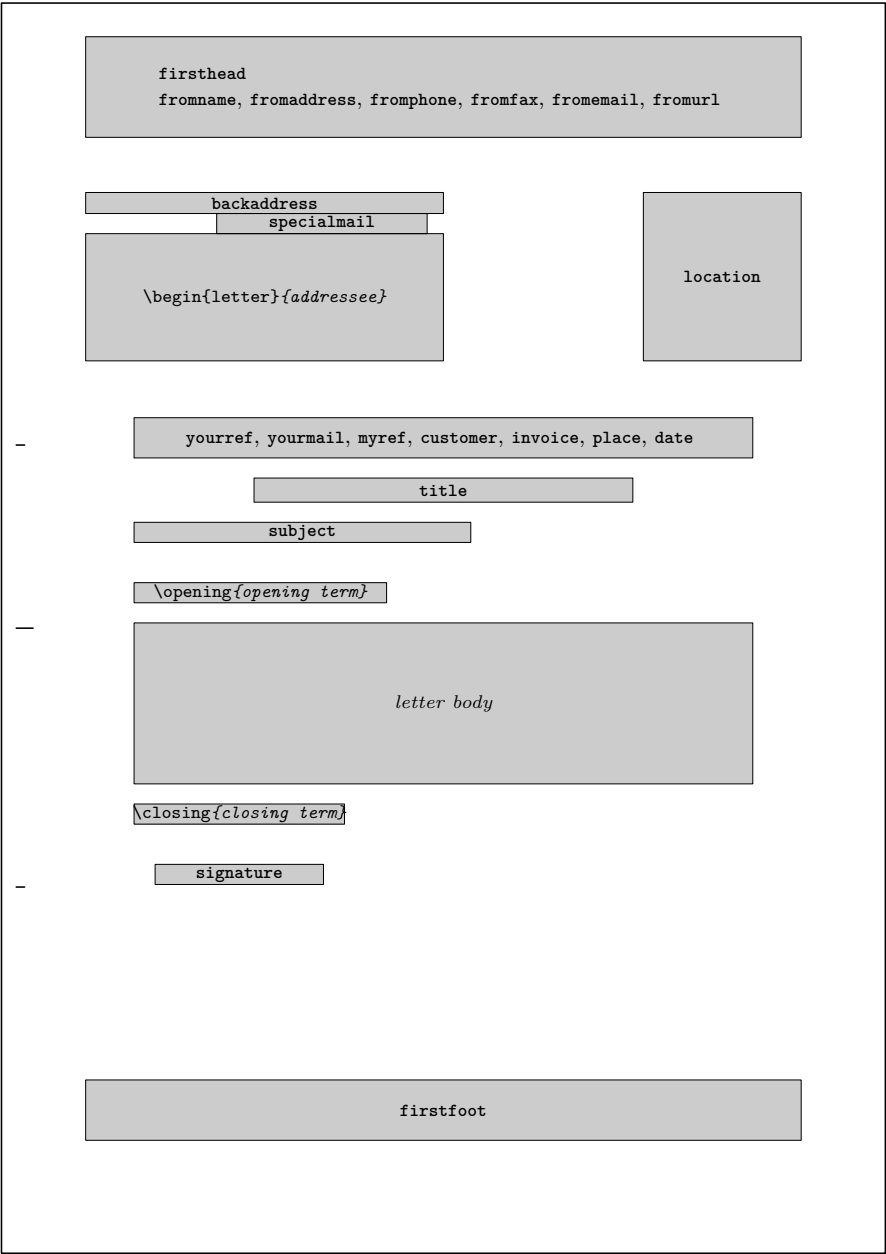


Figure 4.9.: schematic display of the note paper with the most important commands and variables for the drafted elements

Example: Assume that you would like to deactivate all folding marks except the punching mark. This you can accomplish with, for example:

```
\KOMAOptions{foldmarks=blmt}
```

as long as the defaults have not been changed previously. If some changes might have been made before, a safer method should be used. This changes our example a little bit:

```
\documentclass[foldmarks=true,foldmarks=blmtP,
  version=last]{scrlettr2}
\usepackage[english]{babel}
\begin{document}
\begin{letter}{%
  Joana Public\\
  Hillside 1\\
  12345 Public-City%
}
\opening{Dear chairman,}
the last general meeting was about one year ago.
I want to remind you, that the constitution of our
club advises you to make a general meeting every
six month. Because of this I expect the executive
board to detain such a meeting immediately.
\closing{Expecting an invitation}
\ps PS: I hope you accept this as an friendly
demand note.
\setkomavar*{enclseparator}{Enclosure}
\encl{Compendium of the constitution with the
  general meeting paragraphs.}
\cc{executive board\\all members}
\end{letter}
\end{document}
```

The result is shown in [figure 4.10](#).

v2.97c

The color of the folding mark may be changed using using the commands `\setkomafont` and `\addtokomafont` (see [section 4.9, page 164](#)) with element `foldmark`. Default is not change.

enlargefirstpage=simple switch

The first page of a letter always uses a different page layout. The `scrlettr2` 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 use this option. At best, a little more text will then fit

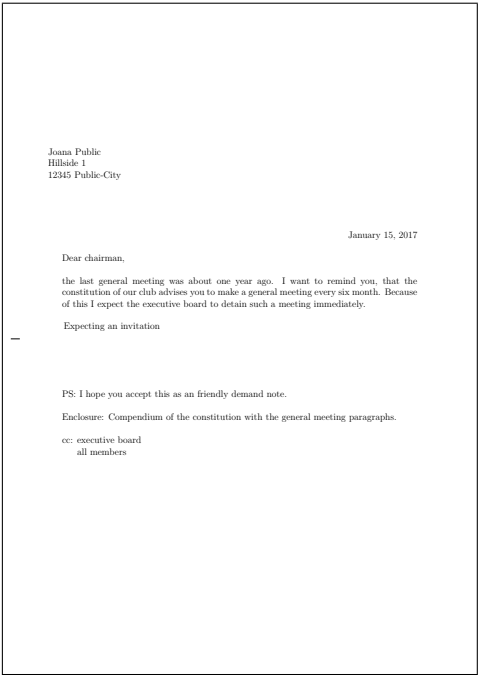


Figure 4.10.: result of a small letter with addressee, opening, text, closing, postscript, distribution list, enclosure, and hole puncher mark (the date is a default of DIN-letters)

on the first page. See also the description of the pseudo-length `firstfootvpos` on [page 479](#). This option can take the standard values for simple switches, as listed in [table 2.5](#), [page 39](#). Default is `false`.

`firsthead=simple switch`

v2.97e

The letterhead is usually the topmost element of the note paper. This option determines whether the letterhead will be typeset at all. The option accepts the standard values for simple keys, given in [table 2.5](#) at [page 39](#). Default is for the letterhead to be set.

`fromalign=method`

v2.97e

Option `fromalign` defines the placement of the return address in the letterhead of the first page. Apart from the various options for positioning the return address in the letterhead, there is also the option of adding the return address to the sender’s extension. Further, this option serves as a switch to activate or deactivate the letterhead extensions. 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 4.4](#). Default is `left`.

Table 4.4.: Available values for option `fromalign` to define the position of the from address in the letterhead with `scrLtr2`

<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>locationleft, leftlocation</code>	return address is set left-justified in the sender’s extension; a logo, if applicable, will be placed above it; the letterhead is automatically deactivated but can be reactivated using option <code>firsthead</code> .
<code>locationright, rightlocation, location</code>	return address is set right-justified in the sender’s extension; a logo, if applicable, will be placed above it; the letterhead is automatically deactivated but can be reactivated using option <code>firsthead</code> .
<code>right</code>	right-justified return address; an optional logo will be left justified; letterhead extensions will be activated

```
fromrule=position
\setkomavar{fromname}[description]{contents}
\setkomavar{fromaddress}[description]{contents}
```

The sender’s name will be determined by variable `fromname`. Thereby the *description* (see also [table 4.6, page 177](#)) will not be used by the predefined letterheads.

At the extended letterhead an optional horizontal rule below the name may be selected using `fromrule=aftername`. Alternatively this rule may be placed below the while sender using `fromrule=afteraddress`. A summary of all available rule position settings shows [table 4.5](#). The length of this rule is determined by pseudo-length `fromrulewidth`.

Default for the rule at the extended letterhead is `false`. But at the standard letterhead the rule will always be placed below the sender’s name.

The sender’s address follows below the name. The *content* of variable `fromaddress` determines this address. The *description* (see also [table 4.6](#)) will not be used at the predefined

Table 4.5.: Possible values of option `fromrule` for the position of the rule in the from address with `scrllttr2`

<code>afteraddress</code> , <code>below</code> , <code>on</code> , <code>true</code> , <code>yes</code> rule below the return address
<code>aftername</code> rule directly below the sender’s name
<code>false</code> , <code>no</code> , <code>off</code> no rule

letterheads

The font of the whole address is determined by the element `fromaddress`. Modifications to this may be defined with element `fromname` for the sender’s name and with element `fromrule` for the rule, that may be activated using option `fromrule`. Nevertheless changing the font style of the rule would make sense. But you may use the elements also to change the color, e.g. color the rule gray instead of black. See [Ker07] for information about colors.

Example: Let’s now define the name of the sender at our letter example:

```
\documentclass[foldmarks=true,foldmarks=blmtP,
  fromalign=false,
  version=last]{scrllttr2}
\usepackage[english]{babel}
\begin{document}
\setkomavar{fromname}{John Public}
\setkomavar{fromaddress}{Valley 2\\
                        54321 Public-Village}
\setkomavar{fromphone}{0\,12\,34-56\,78}
\setkomavar{fromemail}{Peter@Public.invalid}
\setkomavar{fromlogo}{\includegraphics{musterlogo}}
\begin{letter}{%
  Joana Public\\
  Hillside 1\\
  12345 Public-City%
}
\opening{Dear chairman,}
the last general meeting was about one year ago.
I want to remind you, that the constitution of our
club advises you to make a general meeting every
six month. Because of this I expect the executive
board to detain such a meeting immediately.
\closing{Expecting an invitation}
\ps PS: I hope you accept this as an friendly
```

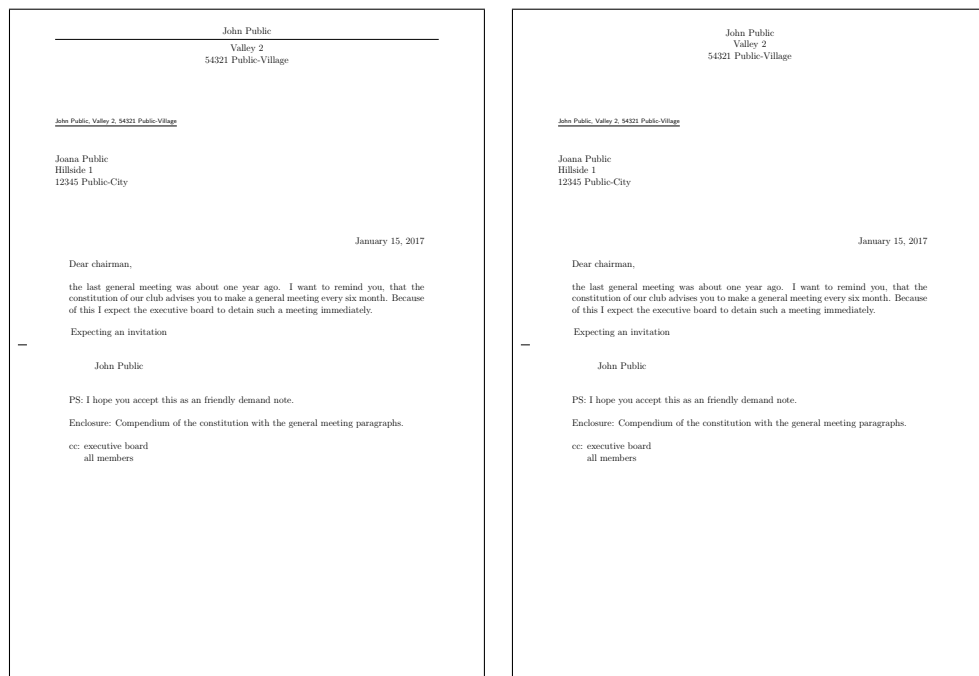


Figure 4.11.: result of a small letter with sender, addressee, opening, text, closing, postscript, distribution list, and enclosure (date and folding marks are defaults of DIN-letters): at left one the standard letterhead using `fromalign=false`, at right one the extended letterhead using `fromalign=center`

```

demand note.
\setkomavar*{enclseparator}{Enclosure}
\encl{Compendium of the constitution with the
general meeting paragraphs.}
\cc{executive board\all members}
\end{letter}
\end{document}

```

First of all not the extended but the standard letterhead has been used. The result is shown at the left side of figure 4.11. The right side shows almost the same letter but with `fromalign=center` and therefore with the extended letterhead. You may see, that this variation is without any rule.

For the first time figure 4.11 also shows a signature below the closing phrase. This has been generated automatically from the sender's name. More information about configuration of the signature may be found in section 4.20 from page 212 onward.

Now, the letter with extended letterhead should use option `fromrule` to print a rule below the sender's name:

```

\documentclass[foldmarks=true,foldmarks=blmtP,
  fromalign=center,fromrule=aftername,
  version=last]{scrlettr2}
\usepackage[english]{babel}
\begin{document}
\setkomavar{fromname}{John Public}
\setkomavar{fromaddress}{Valley 2\\
                        54321 Public-Village}

\begin{letter}{%
  Joana Public\\
  Hillside 1\\
  12345 Public-City%
}
\opening{Dear chairman,}
the last general meeting was about one year ago.
I want to remind you, that the constitution of our
club advises you to make a general meeting every
six month. Because of this I expect the executive
board to detain such a meeting immediately.
\closing{Expecting an invitation}
\ps PS: I hope you accept this as an friendly
demand note.
\setkomavar*{enclseparator}{Enclosure}
\encl{Compendium of the constitution with the
  general meeting paragraphs.}
\cc{executive board\\all members}
\end{letter}
\end{document}

```

The result may be found at the right side of [figure 4.12](#). In difference to this, the left letter has been set once again with the standard letter head, that does not react on the additional option.

An important note 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 `scrlettr2` 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 variable [backaddress](#), [page 182](#)) and signature (see variable [signature](#), [page 212](#)) oneself.

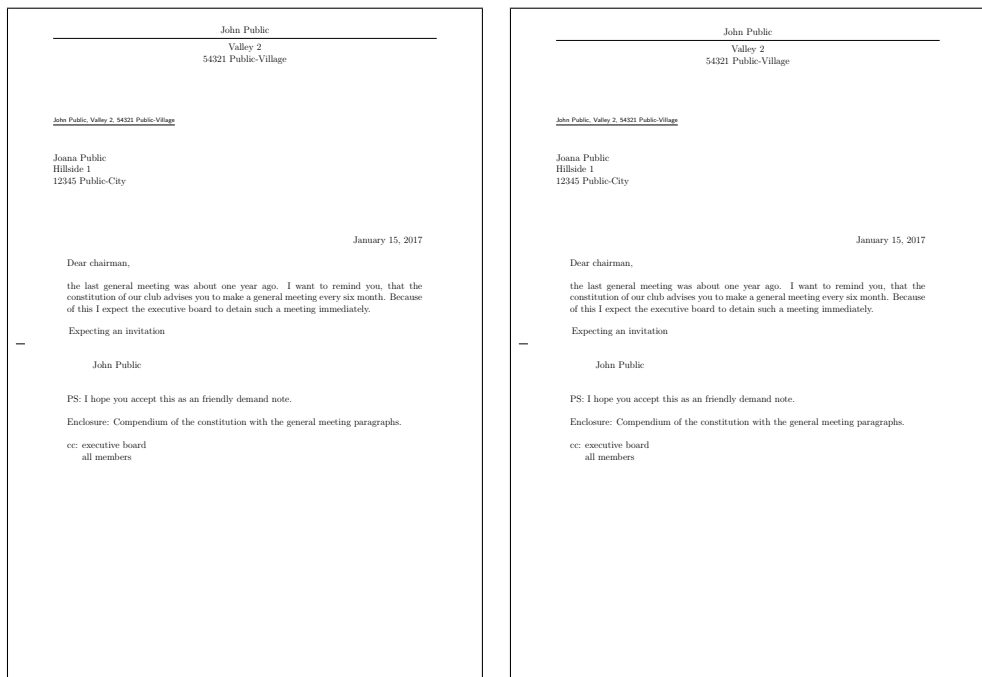


Figure 4.12.: result of a small letter with sender, separation rule, addressee, opening, text, closing, signature, postscript, distribution list, enclosure and puncher hole mark (the date is a default of DIN-letters): at left one the standard letterhead using `fromalign=false`, at right one the extended letterhead using `fromalign=center`

```

symbolicnames=simple switch
fromphone=simple switch
frommobilephone=simple switch
fromfax=simple switch
fromemail=simple switch
fromurl=simple switch
\setkomavar{fromphone}[description]{contents}
\setkomavar{frommobilephone}[description]{contents}
\setkomavar{fromfax}[description]{contents}
\setkomavar{fromemail}[description]{contents}
\setkomavar{fromurl}[description]{contents}
\setkomavar{phoneseparator}[description]{contents}
\setkomavar{mobilephoneseparator}[description]{contents}
\setkomavar{faxseparator}[description]{contents}
\setkomavar{emailseparator}[description]{contents}
\setkomavar{urlseparator}[description]{contents}

```

Whether or not a telephone number, a mobile telephone number, a fax number, an e-mail

Table 4.6.: Predefined descriptions of the variables of the letterhead (the description and contents of the separator variables may be found at [table 4.7](#))

fromemail	<code>\usekomavar*{emailseparator}\usekomavar{emailseparator}</code>
fromfax	<code>\usekomavar*{faxseparator}\usekomavar{faxseparator}</code>
frommobilephone	<code>\usekomavar*{mobilephoneseparator}\usekomavar{mobilephoneseparator}</code>
fromname	<code>\headfromname</code>
fromphone	<code>\usekomavar*{phoneseparator}\usekomavar{phoneseparator}</code>
fromurl	<code>\usekomavar*{urlseparator}\usekomavar{urlseparator}</code>

v3.12

the options `fromphone`, `fromfax`, `fromemail`, and `fromurl`. Any standard value for simple switches from [table 2.5, page 39](#) may be assigned to these options. Default is `false` for all of them. The *contents* depends on the corresponding variable. The default of the *description* or title of each variable may be found in [table 4.6](#). The separators, that will be inserted between *description* and *content*, may be found in [table 4.7](#).

v3.12

You may change the defaults for the description of the separator variables at once using option `symbolicnames`. The option understands the values for simple switches from [table 2.5, page 39](#). Switching on the option replaces the descriptions from the language depending terms `\emailname`, `\faxname`, `\mobilephonenumber`, and `\phonenumber` into symbols of the package `marvosym`. Also the colon will be removed from the content of the separator variables. And in this case the description and the content of the URL separator will be empty. Note, that you have to load package `marvosym` on your own, if you switch on `symbolicnames` first time after `\begin{document}`.

Example: Mr Public from the example letter has telephone and e-mail. He wants to show this also in the letterhead. Furthermore the separation rule should be placed below the letterhead. So he uses the corresponding options and defines the content of the needed variables:

```
\documentclass[foldmarks=true,foldmarks=blmtP,
  fromalign=false,fromrule=afteraddress,
  fromphone,fromemail,
  version=last]{scrllttr2}
```

Table 4.7.: de-
scrip-
tion
and
con-
tent
of the
sepa-
rators
at the
letter-
head
with-
out
option
symbolicnames

Predefined variable name	description	content
emailseparator	<code>\emailname</code>	<code>:~</code>
faxseparator	<code>\faxname</code>	<code>:~</code>
mobilephoneseparator	<code>\mobilephonenumber</code>	<code>\usekomavaer{phoneseparator}</code>
phoneseparator	<code>\phonenumber</code>	<code>:~</code>
urlseparator	<code>\wwwname</code>	<code>:~</code>

```
\usepackage[english]{babel}
\begin{document}
\setkomavar{fromname}{John Public}
\setkomavar{fromaddress}{Valley 2\\
                        54321 Public-Village}
\setkomavar{fromphone}{0\,12\,34-56\,78}
\setkomavar{fromemail}{Peter@Public.invalid}
\begin{letter}{%
    Joana Public\\
    Hillside 1\\
    12345 Public-City%
}
\opening{Dear chairman,}
the last general meeting was about one year ago.
I want to remind you, that the constitution of our
club advises you to make a general meeting every
six month. Because of this I expect the executive
board to detain such a meeting immediately.
\closing{Expecting an invitation}
\ps PS: I hope you accept this as an friendly
demand note.
\setkomavar*{enclseparator}{Enclosure}
\encl{Compendium of the constitution with the
general meeting paragraphs.}
\cc{executive board\\all members}
\end{letter}
\end{document}
```

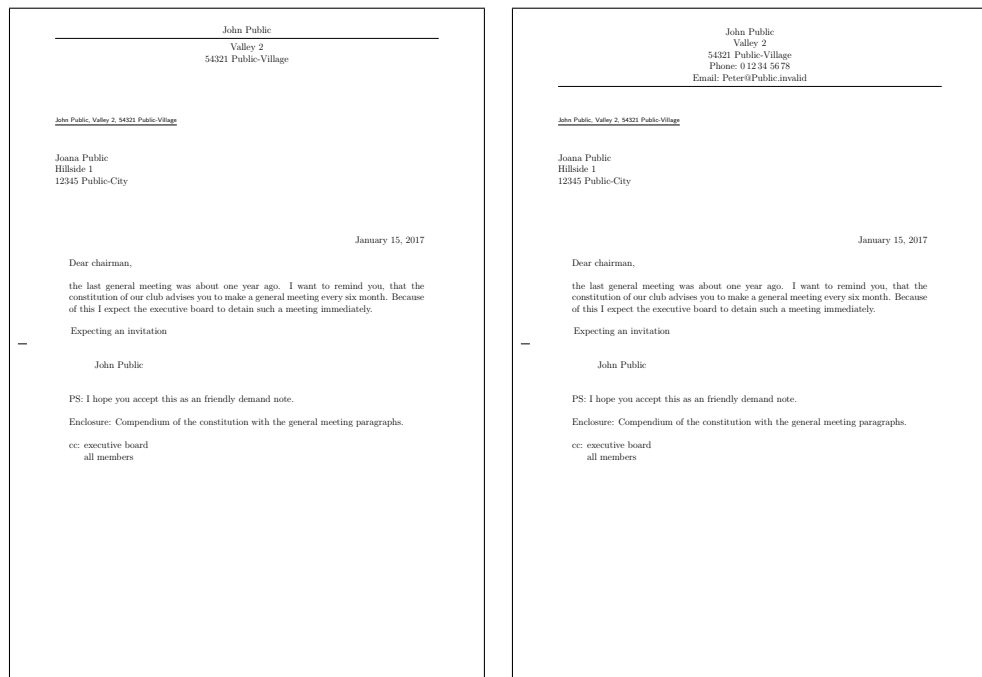


Figure 4.13.: result of a small letter with sender, separation rule, addressee, opening, text, closing, signature, postscript, distribution list, enclosure and puncher hole mark (the date is a default of DIN-letters): at left one the standard letterhead using `fromalign=false`, at right one the extended letterhead using `fromalign=center`

Nevertheless the result at the left side of figure 4.13 is not disillusioning: the options are ignored. That's the case because the additional variables and options will be used at the extended letterhead only. So option `fromalign` has to be used, like done at the right letter of figure 4.13.

```
\documentclass[foldmarks=true,foldmarks=blmtP,
  fromalign=center,fromrule=afteraddress,
  fromphone,fromemail,
  version=last]{scrllttr2}
\usepackage[english]{babel}
\begin{document}
\setkomavar{fromname}{John Public}
\setkomavar{fromaddress}{Valley 2\\
  54321 Public-Village}
\setkomavar{fromphone}{0\,12\,34-56\,78}
\setkomavar{fromemail}{Peter@Public.invalid}
\begin{letter}{%
  Joana Public\\
```

```

        Hillside 1\\
        12345 Public-City%
    }
\opening{Dear chairman,}
the last general meeting was about one year ago.
I want to remind you, that the constitution of our
club advises you to make a general meeting every
six month. Because of this I expect the executive
board to detain such a meeting immediately.
\closing{Expecting an invitation}
\ps PS: I hope you accept this as an friendly
        demand note.
\setkomavar*{enclseparator}{Enclosure}
\encl{Compendium of the constitution with the
        general meeting paragraphs.}
\cc{executive board\\all members}
\end{letter}
\end{document}

```

An arrangement of alternatives with left aligned using `fromalign=left` and right aligned sender using `fromalign=right` may be found in [figure 4.14](#).

```

fromlogo=simple switch
\setkomavar{fromlogo}[description]{contents}

```

With option `fromlogo` you may configure whether or not to use a logo at the letterhead. The option value may be any *simple switch* from [table 2.5, page 39](#). Default is `false`, that means no logo. The logo itself is defined by the *content* of variable `fromlogo`. The *description* of the logo is empty by default and KOMA-Script does not use it anywhere at the predefined note papers (see also [table 4.6](#)).

Example: Mr Public likes to use a letterhead with logo. He generated a graphics file with the logo and would like to include this using `\includegraphics`. Therefor he uses the additional package `graphics` (see [\[Car05\]](#)).

```

\documentclass[foldmarks=true,foldmarks=blmtP,
        fromrule=afteraddress,
        fromphone,fromemail,fromlogo,
        version=last]{scrlltr2}
\usepackage[english]{babel}
\usepackage{graphics}
\begin{document}
\setkomavar{fromname}{John Public}
\setkomavar{fromaddress}{Valley 2\\
        54321 Public-Village}

```

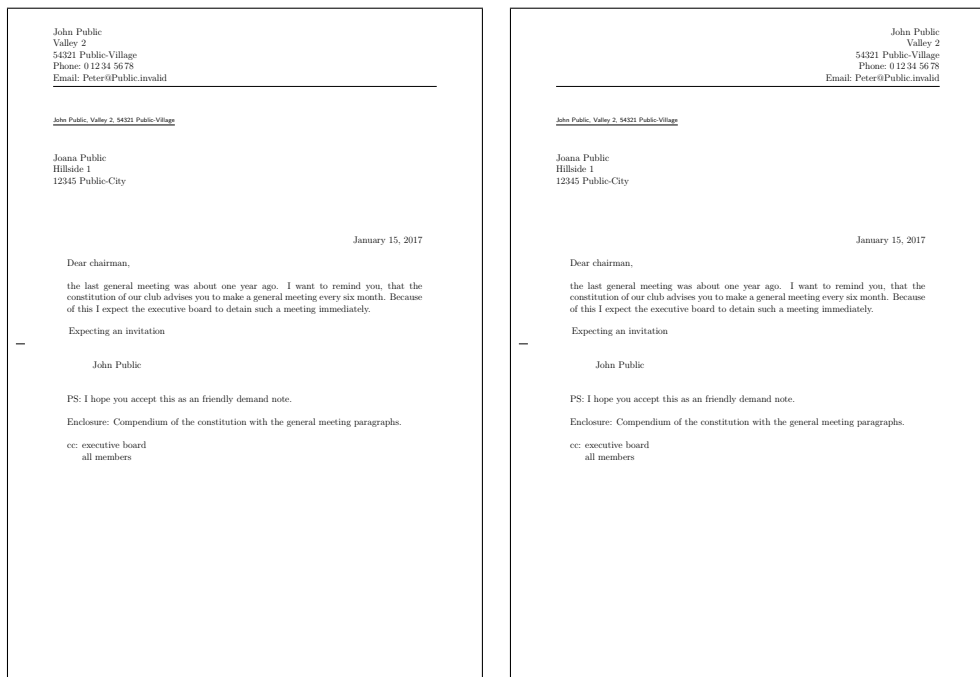


Figure 4.14.: result of a small letter with sender, separation rule, addressee, opening, text, closing, signature, postscript, distribution list, enclosure and puncher hole mark (the date is a default of DIN-letters): at left one left aligned letterhead using `fromalign=left`, at right one right aligned letterhead using `fromalign=right`

```
\setkomavar{fromphone}{0\,12\,34-56\,78}
\setkomavar{fromemail}{Peter@Public.invalid}
\setkomavar{fromlogo}{\includegraphics{musterlogo}}
\begin{letter}{%
  Joana Public\\
  Hillside 1\\
  12345 Public-City%
}
\opening{Dear chairman,}
the last general meeting was about one year ago.
I want to remind you, that the constitution of our
club advises you to make a general meeting every
six month. Because of this I expect the executive
board to detain such a meeting immediately.
\closing{Expecting an invitation}
\ps PS: I hope you accept this as an friendly
demand note.
\setkomavar*{enclseparator}{Enclosure}
```

```

\encl{Compendium of the constitution with the
      general meeting paragraphs.}
\cc{executive board\all members}
\end{letter}
\end{document}

```

The result may be seen at the left top position of [figure 4.15](#). The additional letter prints at this figure shows the result with right aligned or centered sender.

```
\setkomavar{firsthead}[description]{contents}
```

For many 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 as content of variable `firsthead`. Within `\firsthead`, and with the help of the `\parbox` command (see [\[Tea05b\]](#)), 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 and should use other variables with the help of `\usekomavar`. KOMA-Script does not use the *description* of variable `firsthead`.

The command `\firsthead` exists only for reason of compatibility to former `scrlltr2` versions. However it is deprecated and you should not use it anymore.

```

addrfield=mode
backaddress=value
priority=priority
\setkomavar{toname}[description]{contents}
\setkomavar{toaddress}[description]{contents}
\setkomavar{backaddress}[description]{contents}
\setkomavar{backaddressseparator}[description]{contents}
\setkomavar{specialmail}[description]{contents}
\setkomavar{fromzipcode}[description]{contents}
\setkomavar{zipcodeseparator}[description]{contents}
\setkomavar{place}[description]{contents}
\setkomavar{PPcode}[description]{contents}
\setkomavar{PPdatamatrix}[description]{contents}
\setkomavar{addresseeimage}[description]{contents}

```

Option `addrfield` defines whether or not to set an address field. Default is to use an address field. This option can take the mode values from [table 4.8](#). Default is `true`. With values `true`, `topaligned`, `PP`, and `backgroundimage` name and address of the addressee will be defined by the mandatory argument of the `letter` environment (see [section 4.7, page 153](#)). These

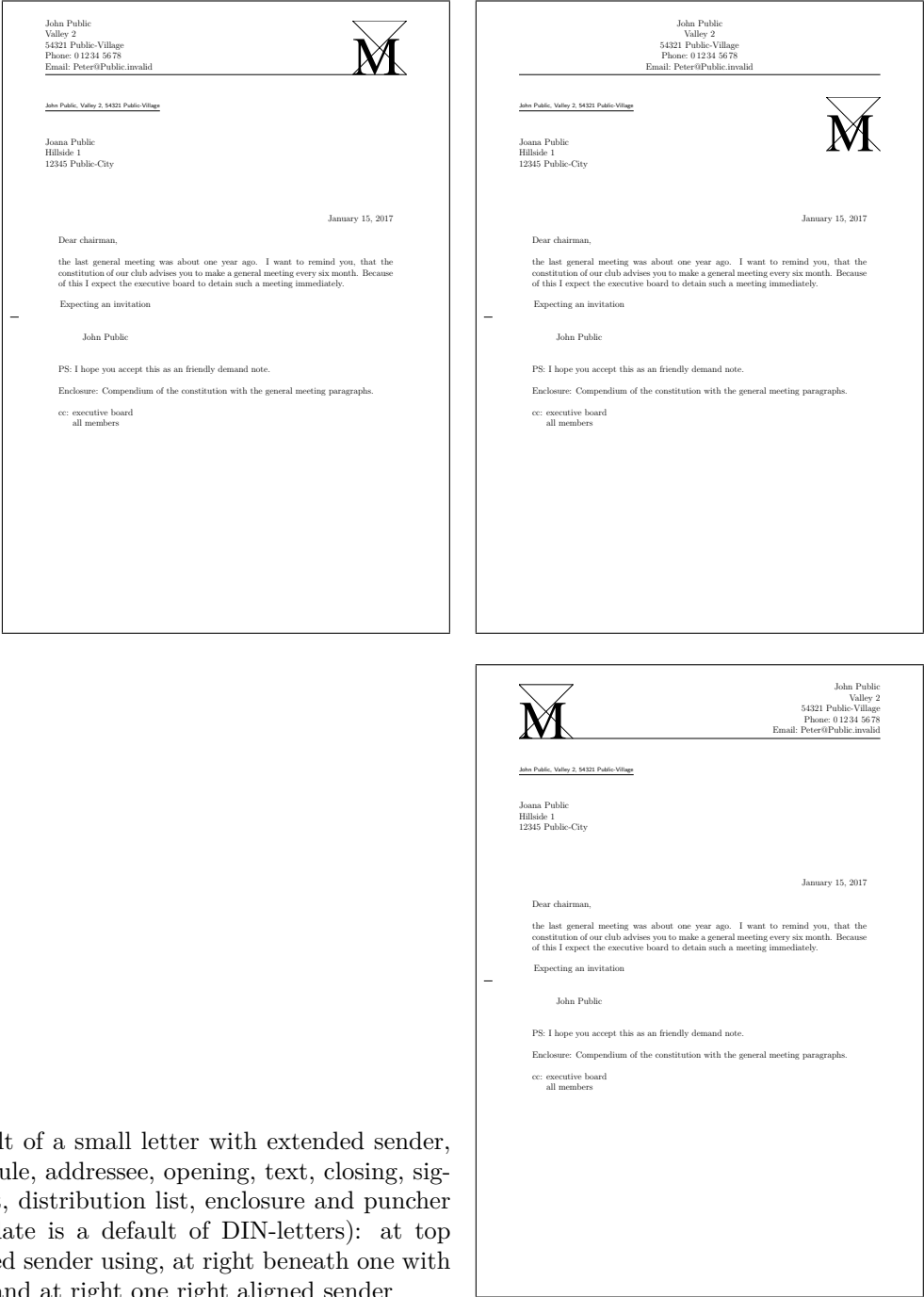


Figure 4.15.: result of a small letter with extended sender, logo, separation rule, addressee, opening, text, closing, signature, postscript, distribution list, enclosure and puncher hole mark (the date is a default of DIN-letters): at top left one left aligned sender using, at right beneath one with centered sender, and at right one right aligned sender

Table 4.8.: available values for option `addrfield` to change the addressee mode of `scrLtr2`

<hr/>	
<code>backgroundimage</code> , <code>PPbackgroundimage</code> , <code>PPBackgroundImage</code> , <code>PPBackGroundImage</code> , <code>ppbackgroundimage</code> , <code>ppBackgroundImage</code> , <code>ppBackGroundImage</code>	Prints an address field with Port-Payé head, in variable <code>addresseimage</code> defined background graphics, but without return address or mode of dispatch.
<code>false</code> , <code>off</code> , <code>no</code>	Omits the address field.
<code>image</code> , <code>Image</code> , <code>PPimage</code> , <code>PPImage</code> , <code>ppimage</code> , <code>ppImage</code>	Prints variable <code>addresseeimage</code> as addressee with Port-Payé, but ignores addressee information and definitions for return address, mode of dispatch or priority.
<code>PP</code> , <code>pp</code> , <code>PPexplicite</code> , <code>PPExplicite</code> , <code>ppexplicite</code> , <code>ppExplicite</code>	Prints an address field with Port-Payé head, defined by the variables <code>fromzipcode</code> , <code>place</code> , and <code>PPcode</code> and when indicated with priority and data array defined by <code>PPdatamatrix</code> but without return address and mode of dispatch.
<div>v3.17</div> <code>topaligned</code> , <code>alignedtop</code>	Prints an address field including a return address and a mode of dispatch or priority without centring the address vertically in the available space.
<code>true</code> , <code>on</code> , <code>yes</code>	Prints an address field including a return address and a mode of dispatch or priority.
<hr/>	

elements will additionally be copied into the variables `toname` and `toaddress`. The predefined font styles may be changed by execution of command `\setkomafont` or `\addtokomafont` (siehe [section 4.9, page 164](#)). Thereby three elements exist. First of all element `addressee`, that is responsible for the addressee overall. The additional elements `toname` and `toaddress` are responsible only either for the name or the address of the addressee. They may be used to define modifications from the `addressee` configuration.

With the default `addrfield=true` an additional return address will be printed. Option `backaddress` defines whether a return address for window envelopes will be set. This option can take the standard values for simple switches, as listed in [table 2.5, page 39](#). These values do not change style of the return address. On the other hand, additionally to switching on the return address, the style of the return address may be selected. Option value `underlined` selects an underlined return address. In opposite to this value `plain` selects a style without underline. Default is `underlined` and therefore printing an underlined return address.

The return address itself is defined by the *content* of variable `backaddress`. Default is a combination of variable `toname` and `toaddress` with redefinition of the double backslash to set the *content* of variable `backaddresseseparator`. The predefined separator *content* is a

Table 4.9.: Predefined font style for the elements of the address field.

element	font style
addressee	
backaddress	\sffamily
PPdata	\sffamily
PPlogo	\sffamily\bfseries
priority	\fontsize{10pt}{10pt}\sffamily\bfseries
prioritykey	\fontsize{24.88pt}{24.88pt}\selectfont
specialmail	
toaddress	
toname	

comma followed by a non-breakable white space. The *description* of variable `backaddress` is not used by KOMA-Script. The font style of the return address is configurable via element `backaddress`. Default for this is `\sffamily` (see table 4.9). Before execution of the element’s font style KOMA-Script switches to `\scriptsize`.

By default, `addrfield=true`, the address will be vertically centred in the available space below the mode of dispatch or priority. In opposite to this, `addrfield=topaligned` will not centre the address but follow it aligned top at the available space. This is the only difference to `addrfield=true`.

At the default `addrfield=true` an optional dispatch type can be output within the address field between the return address and the addressee address, This will be done only if variable `specialmail` is not empty and `priority>manual` has been selected, which is also the default. Class `scrlltr2` itself does not use the *description* of variable `specialmail`. The alignment is defined by the pseudo-lengths `PLengthspecialmailindent` and `specialmailrightindent` (siehe page 475). The predefined font style of element `specialmail`, that has been listed in table 4.9, may be changed using commands `\setkomafont` and `\addtokomafont` (see section 4.9, page 164).

On the other hand, using an international priority with `priority=A` or `priority=B` (see table 4.10) together with `addrfield=true` will print the priority as mode of dispatch. Using it together with `addrfield=PP` will print the priority at the corresponding position in the Port-Payé head. Thereby the element `priority` defines the basic font style and `prioritykey` the modification of the basic font style for the priority key, “A” or “B”. The Port-Payé logo “P. P.” uses the font style of element `PPlogo`. The default font styles are listed in table 4.9 and may be changed using commands `\setkomafont` und `\addtokomafont` (siehe section 4.9, page 164).

With `addrfield=PP` also the zip-code of variable `fromzipcode` and the place from *content* of variable `place` will be set in the Port-Payé head. Thereby the *content* of variable `fromzipcode` will be preceded by the *description* of variable `fromzipcode` and the *content* of variable `zipcodeseparator`. The predefined *description* depends on the used lco-file

Table 4.10.: available values for option `priority` to select the international priority at the address field of `scrlltr2`

<code>false, off, no, manual</code> Priority will not be printed.
<code>B, b, economy, Economy, ECONOMY, B-ECONOMY, B-Economy, b-economy</code> Use international priority B-Economy. With <code>addrfield=true</code> this will be printed instead of the mode of dispatch.
<code>A, a, priority, Priority, PRIORITY, A-PRIORITY, A-Priority, a-priority</code> Use international priority A-Priority. With <code>addrfield=true</code> this will be printed instead of the mode of dispatch.

(see [section 4.21](#) from [page 214](#) onward). On the other hand the default of the *content* of variable `zipcodeseparator` is a small distance followed by an endash followed by one more small distance (`\,--\,`).

v3.03

Beyond that, with `addrfield=PP` and sender’s identification code will be used at the Port-Payé head. This is the *content* of variable `PPcode`. Right beside the address of the addressee an additional data array may be printed. The *content* of variable `PPdatamatrix` will be used for this.

v3.03

Zip-code, place and code will be printed with default font size 8pt. Thereby the font style of element `PPdata` will be used. The default of the element is listed in [table 4.9](#) and may be changed using commands `\setkomafont` and `\addtokomafont` (see [section 4.9, page 164](#)).

v3.03

With options `addrfield=backgroundimage` or `addrfield=image` a picture will be print inside the address field. The *content* of variable `addresseeimage` will be used for this. KOMA-Script does not use the *description* of that variable. Nothing else but the picture will be printed with option `addrfield=image`. But with option `addrfield=backgroundimage` the addressee’s name and address from the mandatory argument of the `letter` environment will be output.

The alignment of the Port-Payé head as long as the alignment of the Port-Payé address is defined by pseudo-length `toaddrindent` (see [page 474](#)) as well as `PPheadwidth` and `PPheadheight` (siehe [page 475](#)). The alignment of the data array is defined by pseudo-length `PPdatamatrixvskip` (see [page 475](#)).

Please note that KOMA-Script itself cannot set any external graphics or pictures. So, if you want to put external picture files into variables like `addresseeimage` or `PPdatamatrix`, you have to use an additional graphics package like `graphics` or `graphicx` to use, i. e., the command `\includegraphics` at the *content* of the variables.

Table 4.11.: Possible values of option `locfield` for setting the width of the field with additional sender attributes with `scrlltr2`

narrow	narrow sender supplement field
wide	wide sender supplement field

`locfield=selection`

`scrlltr2` 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 4.21](#)). 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 22.1.4, page 476](#). Possible values for this option are shown in [table 4.11](#). Default is `narrow`.

`\setkomavar{location}[description]{contents}`

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`. KOMA-Script does not use the *description* of the variable.

Example: Mr Public likes to show some additional information about his membership. Therefore he uses the `location` variable:

```
\documentclass[foldmarks=true,foldmarks=blmtP,
  fromphone,fromemail,fromlogo,
  version=last]{scrlltr2}
\usepackage[english]{babel}
\usepackage{graphics}
\begin{document}
\setkomavar{fromname}{John Public}
\setkomavar{fromaddress}{Valley 2\\
  54321 Public-Village}
\setkomavar{fromphone}{0\,12\,34-56\,78}
\setkomavar{fromemail}{Peter@Public.invalid}
\setkomavar{fromlogo}{\includegraphics{musterlogo}}
\setkomavar{location}{\raggedright
  Club number no.~4711\\
  since 11.09.2001\\
  chairman 2003--2005}
\begin{letter}{%
  Joana Public\\
  Hillside 1\\
  12345 Public-City%
```

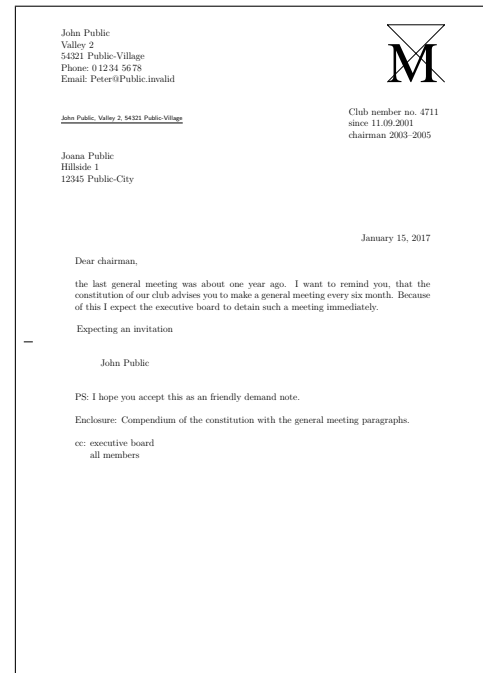


Figure 4.16.: result of a small letter with extended sender, logo, addressee, additional sender information, opening, text, closing, signature, postscript, distribution list, enclosure and puncher hole mark (the date is a default of DIN-letters)

```

    }
\opening{Dear chairman,}
the last general meeting was about one year ago.
I want to remind you, that the constitution of our
club advises you to make a general meeting every
six month. Because of this I expect the executive
board to detain such a meeting immediately.
\closing{Expecting an invitation}
\ps PS: I hope you accept this as an friendly
demand note.
\setkomavar*{enclseparator}{Enclosure}
\encl{Compendium of the constitution with the
general meeting paragraphs.}
\cc{executive board\\all members}
\end{letter}
\end{document}

```

This will define the field beside the addressee's address like shown in [figure 4.16](#).

```
numericaldate=simple switch
```

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 `ngerman`, `babel`, or `isodate`. The short, numerical presentation, on the other hand, is produced by `scrlltr2` itself. This option can take the standard values for simple switches, as listed in [table 2.5](#), [page 39](#). Default is `false`, which results in standard date presentation.

```
\setkomavar{date}[description]{contents}
```

The date in the selected presentation will become the *content* of variable `date`. The selection of option `numericaldate` does not influence the date any longer, if the *content* of this variable will be changed by the user. Normally the date will be part of the reference line. If all other elements of the reference line will be empty and therefore unused a date line instead of a reference line will be printed. See the description of variable `place` on [page 190](#) for more information about the date line. You should note, that you can change the automatic printing of the date using option `refline`, that will be described next.

```
refline=selection
```

Especially in business letters a line with information like identification code, direct dial, customer's number, invoice's number, or references to previous letters may be found usually. This line will be named *reference line* in this manual.

With the `scrlltr2` class, the header, footer, address, and sender attributes may extend beyond the normal type area to the left and to the right. Option `refline=wide` defines that this 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 4.12](#).

Default is `narrow` and `dateright`.

v3.09

```
\setkomavar{yourref}[description]{contents}
\setkomavar{yourmail}[description]{contents}
\setkomavar{myref}[description]{contents}
\setkomavar{customer}[description]{contents}
\setkomavar{invoice}[description]{contents}
```

These five variables represent typical fields of the reference line. Their meanings are given in [table 4.1](#) on [page 142](#). Each variable has also a predefined *description*, shown in [table 4.13](#). Additional information about adding other variables to the reference line may be found in [section 22.2](#) from [page 480](#) onward.

v2.97c

Font style and color of the *description* and *content* of the fields of the reference line may be changed with elements `refname` and `refvalue`. Therefor the commands `\setkomafont`

Table 4.12.: Possible value of option `refline` for setting the width of the reference fields line with `scrlltr2`

	<code>dateleft</code>	The date will be placed leftmost at the reference line.
v3.09	<code>dateright</code>	The date will be placed rightmost at the reference line.
	<code>narrow</code>	The reference line will be restricted to type area.
	<code>nodate</code>	The date is not placed automatically into the reference line.
v3.09	<code>wide</code>	The with of the reference line corresponds to address and sender's additional information.

and `\addtokomafont` (see [section 4.9, page 164](#)) should be used. The default configuration of both elements is listed in [table 4.14](#).

`\setkomavar{placeseparator}[description]{contents}`

If all variables of the reference line are empty, the line will be omitted. 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 *content* value then the hyphen remains unset also. The predefined *content* of `date` is `\today` and depends on the setting of the option `numericaldate` (see [page 189](#)).

Since version 3.09 place and date the alignment of place and date is given by option `refline`. The available alignment values for this option are listed in [table 4.12](#).

The font setting of place and date in the date line are given by font element `placeanddate`

Table 4.13.: predefined descriptions of typical variables of the reference fields line using macros depending on the current language

variable name	description	e. g., 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

Table 4.14.: default font style configuration of the elements of the reference line

element	default configuration
<code>refname</code>	<code>\sffamily\scriptsize</code>
<code>refvalue</code>	

instead of element `refvalue`, which would be used for general reference lines. You can change the empty default of the font element using commands `\setkomafont` and `\addtokomafont` (see [section 4.9](#), [page 164](#)).

Example: Now Mr Public also sets the place:

```
\documentclass[foldmarks=true,foldmarks=blmtP,
  fromphone,fromemail,fromlogo,
  version=last]{scrllttr2}
\usepackage[english]{babel}
\usepackage{graphics}
\begin{document}
\setkomavar{fromname}{John Public}
\setkomavar{fromaddress}{Valley 2\\
  54321 Public-Village}
\setkomavar{fromphone}{0\,12\,34-56\,78}
\setkomavar{fromemail}{Peter@Public.invalid}
\setkomavar{fromlogo}{\includegraphics{musterlogo}}
\setkomavar{location}{\raggedright
  Club number no.~4711\\
  since 11.09.2001\\
  chairman 2003--2005}
\setkomavar{date}{29th February 2011}
\setkomavar{place}{Public-Village}
\begin{letter}{%
  Joana Public\\
  Hillside 1\\
  12345 Public-City%
}
\opening{Dear chairman,}
the last general meeting was about one year ago.
I want to remind you, that the constitution of our
club advises you to make a general meeting every
six month. Because of this I expect the executive
board to detain such a meeting immediately.
\closing{Expecting an invitation}
\ps PS: I hope you accept this as an friendly
demand note.
\setkomavar*{enclseparator}{Enclosure}
\encl{Compendium of the constitution with the
```

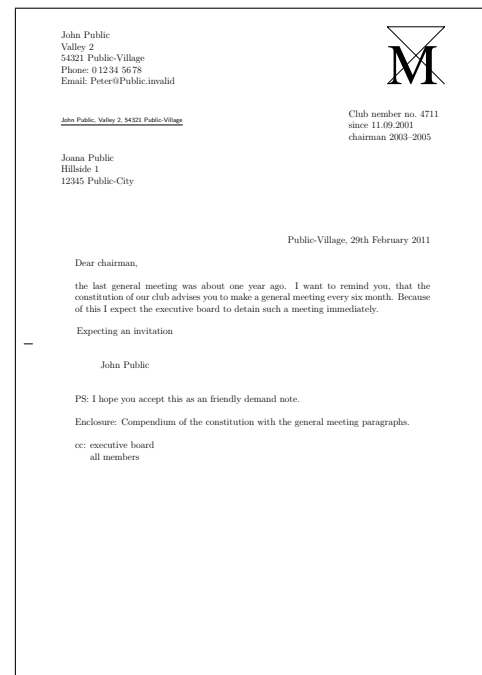


Figure 4.17.: result of a small letter with extended sender, logo, addressee, additional sender information, place, date, opening, text, closing, signature, postscript, distribution list, enclosure and puncher hole mark

```

        general meeting paragraphs.}
\cc{executive board\all members}
\end{letter}
\end{document}

```

In this case [figure 4.17](#) shows the place and the automatic separator in front of the date. The date has been defined explicitly to make the printed date independent from the date of the \LaTeX run.

```
\setkomavar{title}[description]{contents}
```

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 the element `lettertitle` can be changed with commands `\setkomafont` und `\addtokomafont` (see [section 4.9, page 164](#)). Font size declarations are allowed. Font size `\LARGE` is not part of the predefined default `\normalcolor\sffamily\bfseries` but nevertheless will be used before the font style of the element. With `scrlltr2` you can also use `title` as an alias of `lettertitle`. This is not provided using `scrletter` with a KOMA-Script class, because these classes already define an element `title` with different meaning for the document title.

Example: Assume that you are to write a reminder. Thus you put as title:

Table 4.15.: predefined descriptions of subject-related variables

variable name	description
subject	<code>\usekomavar*{subjectseparator}% \usekomavar{subjectseparator}</code>
subjectseparator	<code>\subjectname</code>

```
\setkomavar{title}{Reminder}
```

This way the addressee will recognize a reminder as such.

Like shown in the example, the *content* of the variable defines the title. KOMA-Script will not use the *description*.

```
subject=selection
\setkomavar{subject}[description]{contents}
\setkomavar{subjectseparator}[description]{contents}
```

In case a subject should be set, the *content* of the variable `subject` need to be defined. First of all with option `subject=true` the usage of the *description* before the output of the subject may be configured. See [table 4.15](#) for the predefined *description*. In case of using the *description* the *content* of variable `subjectseparator` will be set between the *description* and *content* of the subject. The predefined *content* of *subjectseparator* is a colon followed by a white space.

On the other hand, `subject=afteropening` may be used to place the subject below instead of before the letter opening. Furthermore, the formatting of the subject may be changed using either `subject=underlined`, `subject=centered`, or `subject=right`. All available values are listed in [table 4.16](#). Please note, that with option `subject=underlined` the while subject must fit at one line! Defaults are `subject=left`, `subject=beforeopening`, and `subject=untitled`.

The subject line is set in a separate font. To change this use the commands `\setkomafont` and `\addtokomafont` (siehe [section 4.9, page 164](#)). For element `letterssubject` the predeterminded font is `\normalcolor\bfseries`. With `scrlltr2` you can also use `subject` as an alias of `letterssubject`. This is not provided using `scrletter` with a KOMA-Script class, because these classes already define an element `subject` with different meaning for the document title.

Example: Now, Mr Public sets a subject. He’s a more traditional person, so he likes to have a kind of heading to the subject and therefor sets the corresponding option:

```
\documentclass[foldmarks=true,foldmarks=blmtP,
  fromphone,fromemail,fromlogo,
  subject=titled,
  version=last]{scrlltr2}
\usepackage[english]{babel}
```

v2.97c

Table 4.16.: available values of option `subject` for the position and formatting of the subject with `scrlltr2`

<code>afteropening</code>	subject after opening
<code>beforeopening</code>	subject before opening
<code>centered</code>	subject centered
<code>left</code>	subject left-justified
<code>right</code>	subject right-justified
<code>titled</code>	add title/description to subject
<code>underlined</code>	set subject underlined (see note in text please)
<code>untitled</code>	do not add title/description to subject

```
\usepackage{graphics}
\begin{document}
\setkomavar{fromname}{John Public}
\setkomavar{fromaddress}{Valley 2\\
                        54321 Public-Village}
\setkomavar{fromphone}{0\,12\,34-56\,78}
\setkomavar{fromemail}{Peter@Public.invalid}
\setkomavar{fromlogo}{\includegraphics{musterlogo}}
\setkomavar{location}{\raggedright
  Club member no.~4711\\
  since 11.09.2001\\
  chairman 2003--2005}
\setkomavar{date}{29th February 2011}
\setkomavar{place}{Public-Village}
\setkomavar{subject}{Missing general meeting}
\begin{letter}{%
  Joana Public\\
  Hillside 1\\
  12345 Public-City%
}
\opening{Dear chairman,}
```

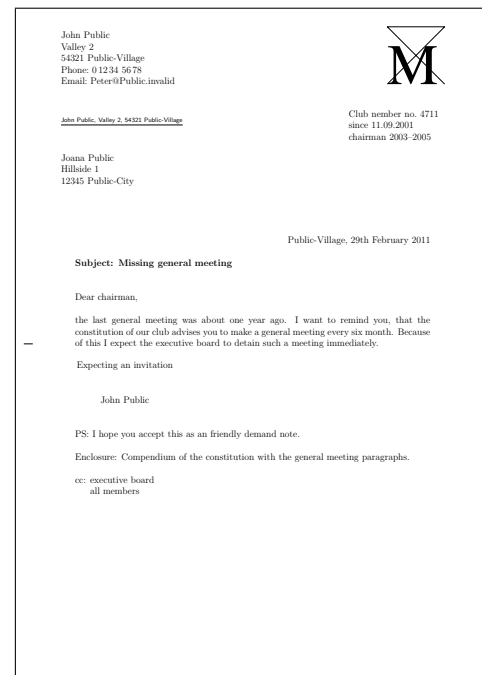


Figure 4.18.: result of a small letter with extended sender, logo, addressee, additional sender information, place, date, subject opening, text, closing, signature, postscript, distribution list, enclosure and puncher hole mark

```

the last general meeting was about one year ago.
I want to remind you, that the constitution of our
club advises you to make a general meeting every
six month. Because of this I expect the executive
board to detain such a meeting immediately.
\closing{Expecting an invitation}
\ps PS: I hope you accept this as an friendly
demand note.
\setkomavar*{enclseparator}{Enclosure}
\encl{Compendium of the constitution with the
general meeting paragraphs.}
\cc{executive board\\all members}
\end{letter}
\end{document}

```

The result is shown by [figure 4.18](#).

`firstfoot=simple switch`

v2.97e

This option determines whether the letterfoot is set or not. Switching off the letterfoot, e. g., using `firstfoot=false`, has an effect when the option `enlargefirstpage` (see [page 170](#)) is used concurrently. In this case the text area of the page will be enlarged down to the bottom.

Then the normal distance between typing area and the letter foot will become the only distance remaining between the end of the typing area and the end of the page.

The option understands the standard values for simple switches, as given in [table 2.5](#), [page 39](#). Default is the setting of the letter foot.

```
\setkomavar{firstfoot}[description]{contents}
```

v3.08

The first page's footer is preset to empty. However, a new construction may be made at the *content* of variable `firstfoot`. KOMA-Script does not use the *description* of the variable. For more information see the example following the next description. Only for compatibility reason the deprecated command `\firstfoot` of `scrlltr2` before release 3.08 still exists. Nevertheless it should not be used any longer.

```
\setkomavar{frombank}[description]{contents}
```

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 additional information (see variable [location](#), [page 187](#)) or the footer.

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:

```
\setkomavar{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:

```
\setkomavar{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.

In the previous example a multi-line footer was set. With a compatibility setting to version 2.9u (see [version](#) in [section 4.4](#), [page 149](#)) the space will in general not suffice. In that case, you may need to reduce `firstfootvpos` (see [page 479](#)) appropriately.

4.11. Paragraph Markup

In the preliminaries of [section 3.10](#), [page 73](#) it was argued why paragraph indent is preferred over paragraph spacing. But the elements the argumentation depends on, i. e., figures, tables, lists, equations, and even new pages, are rare. Often letters are not so long that an oversighted paragraph will have serious consequences to the readability of the document. Because of this, the arguments are less serious for standard letters. This may be one reason that in letters you often encounter paragraphs marked not with indentation of the first line, but with a vertical skip between them. But there may be still some advantages of the paragraph indent. One may be that such a letter is highlighted in the mass of letters. Another may be that the *corporate identity* need not be broken for letters only. Apart from these suggestions, everything that has been written at [section 3.10](#) for the other KOMA-Script classes is valid for letters too. So if you have already read and understood [section 3.10](#) you can jump to [section 4.12](#) on [page 198](#).

`parskip=manner`

Especially in letters you often encounter paragraphs marked not with indentation of the first line, but with a vertical skip between them. KOMA-Script class `scrlltr2` provides several capabilities for this. The *manner* consists of two elements. The first element is either `full` or `half`, meaning the space amount of one line or only half of a line. The second element is `*`,

“+”, or “-”, and may be omitted. Without the second element the last line of a paragraph will end with white space of at least 1em. With the plus character as second element the white space amount will be a third, and with the asterisk a fourth, of the width of a normal line. The minus variant does not take care about the white space at the end of the last line of a paragraph.

The setting may be changed at any place inside the document. In this case the command `\selectfont` will be called implicitly. The change will be valid and may be seen from the next paragraph.

v3.08

Besides the resulting eight possible combinations for *manner*, the values for simple switches shown at [table 2.5](#), [page 39](#) may be used. Switching on the option would be the same as using `full` without `annex` and therefore will result in inter-paragraph spacing of one line with at least 1em white space at the end of the last line of each paragraph. Switching off the options would reactivate the default of 1em indent at the first line of the paragraph instead of paragraph spacing. All the possible values of option `parskip` are shown in [table 3.7](#) at [page 74](#).

All eight `full` and `half` option values 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. Several element of the first letter page will be set without inter-paragraph spacing always.

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

4.12. Detection of Odd and Even Pages

What is described in [section 3.11](#) applies, *mutatis mutandis*. So if you have already read and understood [section 3.11](#) you can switch to [page 199](#), [page 199](#).

In double-sided documents we distinguish left and right pages. Left pages always have an even page number, right pages always have an odd page number. Thus, they are most often referred to as even and odd pages in this guide. Letters will be set single-side mostly. Nevertheless, printing letters with single-side layout using both paper sides or exceptionally generating real double-side letters it may be useful to know whether producing currently an even or odd page.

```
\ifthispageodd{true part}{false part}
```

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 part* argument is executed only if the command falls on an odd page. Otherwise the *false part* argument is executed.

Example: Assume that you simply want to show whether a text will be placed onto an even or odd page. You may achieve that using

```
This page has an \ifthispageodd{odd}{even}
page number.
```

which will result in the output

```
This page has 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.

At [section 21.1, page 441](#) experts may find more information about the problems detecting left and right pages or even and odd page number.

4.13. Head and Foot Using Predefined Page Style

One of the general properties of a document is the page style. In \LaTeX this means mostly the contents of headers and footers. Like already mentioned in [section 4.10](#), the head and foot of the note paper are treated as elements of the note paper. Therefore they do not depend on the settings of the page style. So this section describes the page styles of the consecutive pages of letter after the note paper. At single-side letters this is the page style of the secondary letter sheet. At double-side letters this is also the page style of all backsides.

`\letterpagestyle`

v3.19

The default page style used for letters is specified by the contents of command `\letterpagestyle`. With `scrlettr2` an empty value has been predefined that does not change the page style of the letter from the page style of the document. This has been done, because `scrlettr2` has been made for letter-only-documents. So it is much easier to define the page style of all letters with the page style of the document using `\pagestyle` inside the preamble or before starting the first letter.

The plain page style of letters differs from the plain page style of most classes. Therefore package `scrletter` defines `\letterpagestyle` with the contents `plain.letter`. So all letters will use the plain style of page style pair `letter` by default and independent from the page style of the rest of the document. See [section 22.3](#) for more information about the characteristics of the page style of package `scrletter`.

Example: You are using package `scrletter` and want to have the same page style inside letters as for the rest of the document. So you are using

```
\renewcommand*{\letterpagestyle}{}
```

in your document preamble. Take care about the star of `\renewcommand*`!

Nevertheless, if you use `\pagestyle` or `\thispagestyle` inside a letter, this will overwrite the setting of `\letterpagestyle` that is only the default for the initialisation inside `\begin{document}`.

```
headsepline=simple switch
footsepline=simple switch
```

These two options select whether or not to insert a separator line below the header or above the footer, respectively, on consecutive pages. This option can take the standard values for simple switches, as listed in [table 2.5](#), [page 39](#). Activation of option `headsepline` switches on a rule below the page head. Activation of option `footsepline` switches on a rule above the page foot. Deactivation of the options switches off the corresponding rules.

Obviously option `headsepline` does not influence page style `empty` (see afterwards at this section). This page style explicitly does not use any page head.

Typographically such a rule makes a hard optical connection of head or foot and the text area. This would not mean, that the distance between head and text or text and foot should be increased. Instead of this the head or foot should be seen as parts of the typing area, while calculation of margins and typing area. To achieve this KOMA-Script will pass option `headinclude` or `footinclude`, respectively, to the `typearea` package, if option `headsepline` or `footsepline`, respectively, are used as a class option. In opposite to `headsepline` option `footsepline` does influence page style `plain` also, because `plain` uses a page number at the foot. Package `scllayer-scrpage` (see [chapter 5](#)) provides additional features for rules at head and foot and may be combined with `sclrttr2`.

```
pagenumber=position
```

This option defines if and where a page number will be placed on consecutive pages. This option affects the page styles `headings`, `myheadings`, and `plain`. It also affects the default page styles of the `scllayer-scrpage` package, if set before loading the package (see [chapter 5](#)). It can take values only influencing horizontal, only vertical, or both positions. Available values are shown in [table 4.17](#). Default is `botcenter`.

```
\pagestyle{page style}
\thispagestyle{local page style}
```

In letters written with `sclrttr2` there are four different page styles.

empty is the page style, in which the header and footer of consecutive pages 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 4.10](#), [page 155](#)).

headings is the page style for running (automatic) headings on consecutive pages. The inserted marks are the sender's name from the variable `fromname` and the subject from the variable `subject` (see [section 4.10](#), [page 172](#) and [page 193](#)). At which position these

Table 4.17.: available values of option `pagenumber` for the position of the page number in page styles `headings`, `myheadings`, and `plain` with `scrLtr2`

<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

marks and the page numbers are placed, depends on the previously described option `pagenumber` and the *content* of the variables `nexthead` and `nextfoot`. 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 `sclayer-scrpage` also `\markleft` (see [section 5.5, page 241](#)) is available.

myheadings is the page style for manual page headings on consecutive 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 `sclayer-scrpage` also `\markleft` can be utilized.

plain is the page style with only page numbers in the header or footer on consecutive pages. The placement of these page numbers is influenced by the previously described option `pagenumber`.

Page styles are also influenced by the previously described options `headsepline` and `footsepline`. 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 3.6](#). For header and footer the same element is used, which you can name either `pageheadfoot` or `pagehead`. There is an additional element `pagefoot` for the page foot. This will be used after `pageheadfoot` at page foots, that has been defined either with variable `nextfoot` or Package `sclayer-scrpage` (see [chapter 5, page 234](#)). The element for the page number within the header or footer is named `pagenumber`. Default settings are listed in [table 3.8, page 78](#). Please have also a look at the example in [section 3.12, page 78](#).

v3.00

```
\markboth{left mark}{right mark}
\markright{right mark}
```

The possibilities that are offered with variables and options in `sclrttr2` should be good enough in most cases to create letterheads and footers for the consecutive pages that follow the first letter page. Even more so since you can additionally change with `\markboth` and `\markright` the sender's statements that `sclrttr2` uses to create the letterhead. The commands `\markboth` and `\markright` can in particular be used together with `pagestyle myheadings`. If the package `sclayer-scrpage` is used then this, of course, is valid also for `pagestyle scrheadings`. There the command `\markleft` is furthermore available.

```
\setkomavar{nexthead}[description]{contents}
\setkomavar{nextfoot}[description]{contents}
```

At times one wants to have more freedom with creating the letterhead or footer of subsequent pages. Then one has to give up the previously described possibilities of predefined letterheads or footers that could have been chosen via the option `pagenumber`. Instead one is free to create the letterhead and footer of consecutive pages just the way one wants to have them set with page style `headings` or `myheadings`. For that, one creates the desired letterhead or footer construction using the *content* of variable `nexthead` or `nextfoot`, respectively. Within the *content* of `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 [Tea05b]). A more advanced user should have no problems creating letterheads or footers of his own. Within *content* you can and should of course also make use of other variables by using `\usekomavar`. KOMA-Script does not use the *description* of both variables.

Only for compatibility reason the deprecated commands `\nexthead` and `\nextfoot` of former scrllttr2 releases before 3.08 are also implemented. Nevertheless, you should not use those.

4.14. Interleaf Pages

What is described in [section 3.13](#) applies, *mutatis mutandis*. So if you have already read and understood [section 3.13](#) you can switch to [section 4.15](#), [page 205](#).

Interleaf pages are pages that are intended to stay blank. Originally these pages were really completely white. L^AT_EX, on the other hand, by default sets those pages with the current valid page style. So those pages may have a head and a pagination. KOMA-Script provides several extensions to this.

Interleaf pages may be found in books mostly. Because chapters in books commonly start on odd pages, sometimes a left page without contents has to be added before. This is also the reason that interleaf pages only exist in double-sided printing. The unused back sides of the one-sided printings are not interleaf pages, really, although they may seem to be such pages.

At letters interleaf pages are unusual. This may be benefited by the case, that real two-sided letters are seldom, because binding of letters is not done often. Nevertheless scrllttr2 supports interleaf pages in the case of two-sided letters. Because the following described commands are seldom used in letters no examples are shown. If you need examples, please note them at [section 3.13](#) from [page 81](#) upward.

```
cleardoublepage=page style
cleardoublepage=current
```

v3.00 With this option, you may define the page style of the interleaf pages created by the `\cleardoublepage`, `\cleardoubleoddpager`, or `\cleardoubleevenpage` to break until the wanted page. Every already defined *page style* (see [section 4.13](#) from [page 199](#) and [chapter 5](#) from [page 225](#)) may be used. Besides this, `cleardoublepage=current` is valid. This case is the default until KOMA-Script 2.98c and results in interleaf page without changing the page style. Since KOMA-Script 3.00 the default follows the recommendation of most typographers and has been changed to blank interleaf pages with page style `empty` unless you switch compatibility to an earlier version (see option [version](#), [section 4.4](#), [page 149](#)).

```
\clearpage
\cleardoublepage
\cleardoublepageusingstyle{page style}
\cleardoubleemptypage
\cleardoubleplainpage
\cleardoublestandardpage
\cleardoubleoddpager
\cleardoubleoddpagerusingstyle{page style}
\cleardoubleoddpageremptypage
\cleardoubleoddpagerplainpage
\cleardoubleoddpagerstandardpage
\cleardoubleevenpage
\cleardoubleevenpageusingstyle{page style}
\cleardoubleevenemptypage
\cleardoubleevenplainpage
\cleardoubleevenstandardpage
```

The L^AT_EX kernel 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 2.4](#), [page 38](#)) starts a new right-hand page. An empty left page in the current page style is output if necessary.

v3.00 With `\cleardoubleoddpagerstandardpage`, KOMA-Script works as described above. The `\cleardoubleoddpagerplainpage` command changes the page style of the empty left page to `plain` in order to suppress the page head. Analogously, the page style `empty` is applied to the empty page with `\cleardoubleoddpageremptypage`, suppressing the page number as well as the page head. The page is thus entirely empty. If another *page style* is wanted for the interleaf page is may be set with the argument of `\cleardoubleoddpagerusingpagestyle`. Every already defined *page style* (see [chapter 5](#)) may be used.

However, the approach used by the KOMA-Script commands `\cleardoublestandardpage`, `\cleardoubleemptypage`, `\cleardoubleplainpage`, and `\cleardoublepageusingstyle` is dependent on the option `cleardoublepage` described above and is similar to one of the corresponding commands above. The same is valid for the standard command `\cleardoublepage`, that may be either `\cleardoubleoddpaper` or `\cleardoubleevenpage`.

In `scrlltr2` the other commands are there only for completeness. More information about them may be found at [section 3.13, page 83](#) if needed.

The commands `\cleardoubleoddpaper` respective `\cleardoubleevenpage` leads to the next odd respectively even page. The page style of an interleaf page will be set depending on option `cleardoublepage`.

4.15. Footnotes

All of what is described in [section 3.14](#) is generally applicable. So if you have already read and understood [section 3.14](#) you can switch to [section 4.16, page 208](#).

The commands for setting footnotes may be found at each introduction into L^AT_EX, e. g., at [\[OPHS11\]](#). KOMA-Script provides additional features to change the footnote block format.

`footnotes=setting`

v3.00

Footnotes will be marked with a tiny superscript number in text by default. If more than one footnote falls at the same place, one may think that it is only one footnote with a very large number instead of multiple footnotes (i. e., footnote 12 instead of footnotes 1 and 2). Using `footnotes=multiple` will separate multiple footnotes immediately next to each other by a separator string. The predefined separator at `\multfootsep` is a single comma without space. The whole mechanism is compatible with package `footmisc`, Version 5.3d (see [\[Fai11\]](#)). It is related not only to footnotes placed using `\footnote`, but `\footnotemark` too.

Command `\KOMAOPTIONS` or `\KOMAOPTION` may be used to switch back to the default `footnotes=nomultiple` at any time. If any problems using another package that influences footnotes occur, it is recommended not to use the option anywhere and not to change the *setting* anywhere inside the document.

A summary of the available *setting* values of footnotes may be found at [table 3.11, page 85](#).

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

Similar to the standard classes, footnotes in KOMA-Script are produced with the `\footnote` command, or alternatively the paired usage 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 L^AT_EX no choice but to break the footnote onto the next page. KOMA-Script, unlike the standard classes, can recognize and separate consecutive footnotes automatically. See the previously documented option `footnotes` for this.

If you want to set the separator manually, you may use `\multiplefootnoteseparator`. Note that this command should not be redefined, because it has been defined not only to be the separator string but also the type style, i. e., font size and superscript. The separator string without type style may be found at `\multfootsep`. The predefined default is

```
\newcommand*{\multfootsep}{,}
```

and may be changed by redefining the command.

Examples and additional information may be found at [section 3.14](#) from [page 86](#) onward.

```
\footref{reference}
```

Sometimes there are single footnotes to multiple text passages. The least sensible way to typeset this would be to repeatedly use `\footnotemark` with the same manually set number. The disadvantages of this method would be that you have to know the number and manually fix all the `\footnotemark` commands, and if the number changes because of adding or removing a footnote before, each `\footnotemark` would have to be changed. Because of this, KOMA-Script provides the use of the `\label` mechanism in such cases. After simply setting a `\label` inside the footnote, `\footref` may be used to mark all the other text passages with the same footnote mark. Because of setting the additional footnote marks using the `\label` mechanism, changes of the footnote numbers will need at least two L^AT_EX runs to ensure correct numbers for all `\footref` marks. An example for the usage `\footref` can be found in [section 3.14](#) on [page 86](#).

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

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. The default configuration of the KOMA-Script classes is:

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

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

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 `\setkomafont` and `\addtokomafont` (see section 4.9, page 164). Please refer also to table 4.2, page 165. 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}}
```

In the above the font for the element `footnotereference` is applied (see table 4.2, page 165). Thus the footnote marks in the text and in the footnote itself are identical. The font can be changed with the commands `\setkomafont` and `\addtokomafont` (see section 4.9, page 164).

Examples may be found at section 3.14, from page 87 onwards.

```
\setfootnoterule[thickness]{length}
```

v3.06

Generally a horizontal rule will be placed between the text area and the footnote area. But normally this rule is not as long as the width of the typing area. With Command `\setfootnoterule` you may change the thickness and the width of that rule. Thereby the parameters *thickness* and *length* will be evaluated not at definition time but when setting the rule itself. If optional argument *thickness* has been omitted the thickness of the rule will not be changed. Empty arguments *thickness* or *length* are also allowed and do not change

the corresponding parameter. Using implausible values may result in warning messages not only setting the arguments but also when KOMA-Script uses the parameters.

v3.07 With element `footnoterule` the color of the rule may be changed using the commands `\setkomafont` and `\addtokomafont` for element `footnoterule` (see [section 4.9](#), [page 164](#)). Default is no change of font or color. For color changes a color package like `xcolor` would be needed.

4.16. Lists

All of what is described in [section 3.18](#) is generally applicable. So if you have already read and understood [section 3.18](#) you can switch to [section 4.17](#), [page 211](#).

Both \LaTeX and the standard classes offer different environments for lists. Though slightly changed or extended all these lists 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.

Because lists are standard elements of \LaTeX this section abandons on examples. Nevertheless, you may find examples either in [section 3.18](#) from [page 111](#) or in almost every introduction to \LaTeX .

```
\begin{itemize}
  \item ...
  :
\end{itemize}
\labelitemi
\labelitemii
\labelitemiii
\labelitemiv
```

The simplest form of a list is an `itemize` list. 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`.


```

\begin{enumerate}
  \item ...
  :
\end{enumerate}
\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 L^AT_EX 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`.

```

\begin{description}
  \item[keyword] ...
  :
\end{description}

```

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 `\setkomafont` and `\addtokomafont` (see [section 4.9, page 164](#)) for the element `descriptionlabel` (see [table 4.2, page 165](#)). Default setting is `\sffamily\bfseries`.

```

\begin{labeling}[delimiter]{widest pattern}
  \item[keyword]...
  :
\end{labeling}

```

An additional form of a description list is only available in the KOMA-Script classes: the `labeling` environment. Unlike the `description` environment, 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. The font which is responsible for emphasizing the item and the separator can be changed with the commands `\setkomafont` and `\addtokomafont` (see [section 4.9, page 164](#)) for the element `labelinglabel` and `labelingseparator` (see [table 4.2, page 165](#)).

v2.8p

v3.02

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.

```
\begin{verse}...\end{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 could, as usual, insert `*` instead of `\\`.

```
\begin{quote}...\end{quote}
```

```
\begin{quotation}...\end{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.

```
\begin{addmargin}[left indentation]{indentation}...\end{addmargin}
```

```
\begin{addmargin*}[inner indentation]{indentation}...\end{addmargin*}
```

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.

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 \LaTeX run. For details please refer to the explanation of the commands `\ifthispageodd` (section 4.12, page 198) and `\ifthispagewasodd` (section 21.1, page 441).

There may be several questions about coexistence of lists and paragraphs. Because of this additional information may be found at the description of option `parskip` in section 21.1, page 441. Also at the expert part, in section 21.1, page 441, you may find additional information about page breaks inside of `addmargin*`.

4.17. Math

There are not math environments implemented at the KOMA-Script classes. Instead of this, the math features of the \LaTeX kernel have been supported. Furthermore regular math with numbered equations or formulas is very unusual at letters. Because of this `scrllttr2` does not actively support numbered equations. Therefore options `leqno` and `fleqn`, that has been described for `scrbook`, `scrreprt`, and `scrartcl` at section 3.19 are not available from `scrllttr2`.

You will not find a description of the math environments of the \LaTeX kernel here. If you want to use `displaymath`, `equation` and `eqnarray` you should read a short introduction into \LaTeX like [OPHS11]. But if you want more than very simple mathematics, usage of package `amsmath` would be recommended (see [Ame02]).

4.18. Floating Environments of Tables and Figures

Floating environments for tables or figures are very unusual in letters. Therefore `scrllttr2` does not provide them. If someone nevertheless needs floating environments, then this is often points out a malpractice of the letter class. In such cases you may try to define the floating environments with help of packages like `tocbasic` (siehe chapter 15). Nevertheless, tabulars and pictures or graphics without floating environment may still be done with the letter class `scrllttr2`.

4.19. Margin Notes

All of what is described in section 3.21 is generally applicable. So if you have already read and understood section 3.21 you can switch to section 4.20, page 212.

Aside from the text area, that normally fills the typing area, usually a marginalia column may be found. Margin notes will be printed at this area. Nevertheless, margin notes are unusual at letters and should be used seldomly.

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

Usually margin notes in L^AT_EX 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`.

An example for this may be found in [section 3.21](#) at [page 136](#).

Experts and advanced users may find information about problems using `\marginpar` at [section 21.1](#), [page 441](#). These are valid for `\marginline` also.

4.20. Closing

That the letter closing will be set by `\closing` has been explained already in [section 4.7](#), [page 156](#). A kind of annotation to the signature is often placed below the signature of the letter. The signing or hand-written inscription itself is placed between this signature annotation and the closing phrase.

```
\setkomavar{signature}[description]{contents}
```

The variable `signature` holds an explanation or annotation for the inscription, the signing of the letter. The `content` is predefined as `\usekomavar{fromname}`. The annotation may consist of multiple lines. The lines should then be separated by a double backslash. Paragraph breaks in the annotation are however not permitted.

```
\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's `content`.

Where to place this box is determined by the pseudo-lengths `sigindent` and `sigbeforevskip` (see [section 22.1.7](#), [page 478](#)). 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 3.16](#), [page 102](#)).

Example: Now, Mr Public really wants to aggrandize himself. Therefor he uses the signature to show again, that he himself was formerly chairman. Because of this he changes `contents` of variable `signature`. Additionally he wants the signature be flush-left aligned and so he also redefines `\raggedsignature`:

```

\documentclass[foldmarks=true,foldmarks=blmtP,
  fromphone,fromemail,fromlogo,
  subject=titled,
  version=last]{scrLtr2}
\usepackage[english]{babel}
\usepackage{graphics}
\begin{document}
\setkomavar{fromname}{John Public}
\setkomavar{signature}{John Public\\
  (former chairman)}
\renewcommand*{\raggedsignature}{\raggedright}
\setkomavar{fromaddress}{Valley 2\\
  54321 Public-Village}
\setkomavar{fromphone}{0\,12\,34-56\,78}
\setkomavar{fromemail}{Peter@Public.invalid}
\setkomavar{fromlogo}{\includegraphics{musterlogo}}
\setkomavar{location}{\raggedright
  Club member no.~4711\\
  since 11.09.2001\\
  chairman 2003--2005}
\setkomavar{date}{29th February 2011}
\setkomavar{place}{Public-Village}
\setkomavar{subject}{Missing general meeting}
\begin{letter}{%
  Joana Public\\
  Hillside 1\\
  12345 Public-City%
}
\opening{Dear chairman,}
the last general meeting was about one year ago.
I want to remind you, that the constitution of our
club advises you to make a general meeting every
six month. Because of this I expect the executive
board to detain such a meeting immediately.
\closing{Expecting an invitation}
\ps PS: I hope you accept this as an friendly
demand note.
\setkomavar*{enclseparator}{Enclosure}
\encl{Compendium of the constitution with the
  general meeting paragraphs.}
\cc{executive board\\all members}
\end{letter}
\end{document}

```

See [figure 4.19](#) for the result.

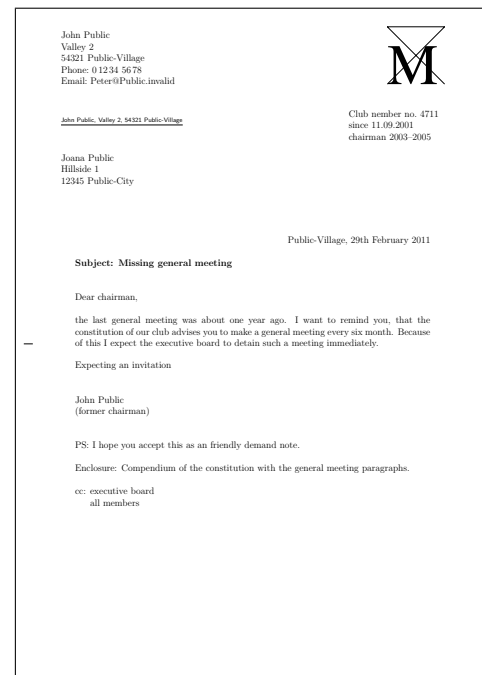


Figure 4.19.: result of a small letter with extended sender, logo, addressee, additional sender information, place, date, subject opening, text, closing, modified signature, postscript, distribution list, enclosure and puncher hole mark

4.21. Letter Class Option Files

Normally, you would not redefine selections like the sender's information 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 `scrlltr2` 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 `scrlltr2`, these are in particular the commands `\@newplength`, `\@setplength`, and `\@setplength` (see [section 22.1](#)).

There are already some `lco` files included in the KOMA-Script distribution. The `DIN.lco`, `DINmtext.lco`, `SNleft.lco`, `SN.lco`, `UScommercial9`, `UScommercial9DW`, `NF.lco` files serve to adjust KOMA-Script to different layout standards. They are well suited as templates for your own parameter sets, while you become a KOMA-Script expert. 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 4.18, page 218](#).

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. More particular complex examples of `lco`-files are shown at [\[KDP\]](#) or in [\[Koh03\]](#). Both locations are mainly in German.

```
\LoadLetterOption{name}
\LoadLetterOptions{list of names}
```

With `scr1ttr2` `lco`-files can be loaded by the `\documentclass` command. You enter the name of the `lco`-file without suffix as an option. In this case, the `lco`-file will be loaded right after the class file.

However, it is recommended to load an `lco`-file using `\LoadLetterOption` or `\LoadLetterOptions`. You can do this even from within another `lco`-file. Both commands take the *name* of the `lco`-file without suffix, `\LoadLetterOption` as a single parameter, `\LoadLetterOptions` as one member of a comma-separated *list of names*. The corresponding `lco`-files will be loaded in the order given by the list.

Example: Mr Public also writes a document containing several letters. Most of them should comply with the German DIN standard. So he starts with:

```
\documentclass{scr1ttr2}
```

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}{%
  Joana Public\\
  Hillside 1\\
  12345 Public-City}
\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`, then it may no longer have the same name as an option.

Example: Since Mr Public often writes letters with the same options and parameters, he does not like to copy all these to each new letter. To simplify the effort of writing a new letter, he therefore makes a `lco`-file:

```

\ProvidesFile{ich.lco}[2008/06/11 lco
  (John Public)]
\KOMAOptions{foldmarks=true,foldmarks=blmtP,
  fromphone,fromemail,fromlogo,subject=titled}
\setkomavar{fromname}{John Public}
\setkomavar{signature}{John Public\\
  (former chairman)}
\renewcommand*{\raggedsignature}{\raggedright}
\setkomavar{fromaddress}{Valley 2\\
  54321 Public-Village}
\setkomavar{fromphone}{0\,12\,34-56\,78}
\setkomavar{fromemail}{Peter@Public.invalid}
\setkomavar{fromlogo}{%
  \includegraphics{musterlogo}}
\setkomavar{location}{\raggedright
  Club member no.~4711\\
  since 11.09.2001\\
  chairman 2003--2005}
\setkomavar{place}{Public-Village}
\setkomavar{frombank}{Bank of Friendly Greetings}

```

With this the size of the previous letter decreases to:

```

\documentclass[version=last,ich]{scrLtr2}
\usepackage[english]{babel}
\usepackage{graphics}
\begin{document}
\setkomavar{date}{29th February 2011}
\setkomavar{subject}{Missing general meeting}
\begin{letter}{%
  Joana Public\\
  Hillside 1\\
  12345 Public-City%
}
\opening{Dear chairman,}
the last general meeting was about one year ago.
I want to remind you, that the constitution of our
club advises you to make a general meeting every
six month. Because of this I expect the executive
board to detain such a meeting immediately.
\closing{Expecting an invitation}
\ps PS: I hope you accept this as an friendly
demand note.
\setkomavar*{enclseparator}{Enclosure}
\encl{Compendium of the constitution with the
  general meeting paragraphs.}

```

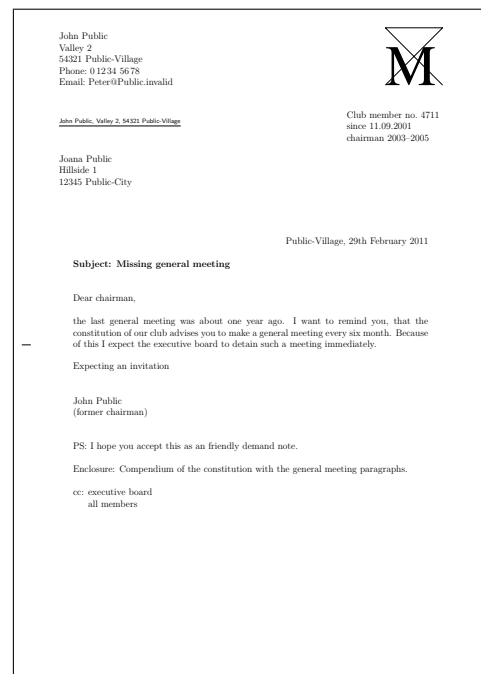



Figure 4.20.: result of a small letter with extended sender, logo, addressee, additional sender information, place, date, subject opening, text, closing, modified signature, postscript, distribution list, enclosure and puncher hole mark using a lco-file

```
\cc{executive board\all members}
\end{letter}
\end{document}
```

Nevertheless, as shown in [figure 4.20](#), the result does not change.

Please note that immediately after loading the document class normally neither a package for the input encoding nor a language package has been loaded. Because of this, you should use T_EX's 7-bit encoding for all characters, e. g., use “\ss” to produce a German “ß”.

In [table 4.18](#), [page 218](#) 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 line when using the `refline=wide` option (see [section 4.10](#), [page 189](#)). If you have this kind of problem, create your own lco file that loads SN first and then changes `toaddrhpos` (see [section 22.1.3](#), [page 473](#)) to a smaller value. Additionally, also reduce `toaddrwidth` accordingly.

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. Because of this, it is not necessary to load this default file on your own.

Please note that it is not possible to use `\PassOptionsToPackage` to pass options to pack-

ages from within an `lco`-file that have already been loaded by the class. Normally, this only applies to the `typearea`, `scrfile`, `scrbase`, and `keyval` packages.

Table 4.18.: predefined `lco`-files

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).
KakuLL	parameter set for Japanese letters in A4 format; suitable for Japanese window envelopes of type Kaku A4, in which the windows is approximately 90 mm wide, 45 mm high, and positioned 25 mm from the left and 24 mm from the top edge (see appendix A).
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 <code>lco</code> -file than with the other <code>lco</code> -files.
NF	parameter set for French letters, according to NF Z 11-001; suitable for window envelopes of type DL (110 mm to 220 mm) with a window of about 20 mm from right and bottom with width 45 mm and height 100 mm; this file was originally developed by Jean-Marie Pacquet, who provides an extended version and additional information on [Pac] .
NipponEH	parameter set for Japanese letters in A4 format; suitable for Japanese window envelopes of types Chou or You 3 or 4, in which the windows is approximately 90 mm wide, 55 mm high, and positioned 22 mm from the left and 12 mm from the top edge (see appendix A).

Table 4.18.: predefined lco-files (*continuation*)

NipponEL

parameter set for Japanese letters in A4 format; suitable for Japanese window envelopes of types Chou or You 3 or 4, in which the windows is approximately 90 mm wide, 45 mm high, and positioned 22 mm from the left and 12 mm from the top edge (see [appendix A](#)).

NipponLH

parameter set for Japanese letters in A4 format; suitable for Japanese window envelopes of types Chou or You 3 or 4, in which the windows is approximately 90 mm wide, 55 mm high, and positioned 25 mm from the left and 12 mm from the top edge (see [appendix A](#)).

NipponLL

parameter set for Japanese letters in A4 format; suitable for Japanese window envelopes of types Chou or You 3 or 4, in which the windows is approximately 90 mm wide, 45 mm high, and positioned 25 mm from the left and 12 mm from the top edge (see [appendix A](#)).

NipponRL

parameter set for Japanese letters in A4 format; suitable for Japanese window envelopes of types Chou or You 3 or 4, in which the windows is approximately 90 mm wide, 45 mm high, and positioned 25 mm from the left and 24 mm from the top edge (see [appendix A](#)).

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).

Table 4.18.: predefined lco-files (*continuation*)

UScommercial9

parameter set for US-American letters with paper size letter; suitable for US-American window envelopes of size *commercial No. 9* with window width of 4 1/2 in, height of 1 1/8 in, and position of 7/8 in from the left and 1/2 in from the bottom, without sender's return address inside of the window; with folding it first at the puncher mark then at the top folder mark also legal paper may be used but would result in a page size warning

UScommercial9DW

parameter set for US-American letters with paper size letter; suitable for US-American window envelopes of size *commercial No. 9* with addressee's window width of 3 5/8 in, height of 1 1/8 in, and position of 3/4 in from the left and 1/2 in from the bottom, and with a sender's window width of 3 1/2 in, height of 7/8 in, and position of 5/16 in from the left and 2 1/2 in from the bottom, but without a sender's return address at any of the windows; with folding it first at the puncher mark then at the top folder mark also legal paper may be used but would result in a page size warning

4.22. 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 `scrlltr2` provides basic support for this task.

```
\adrentry{last-name}{first-name}{address}{phone}{F1}{F2}{comment}{key}
```

The class `scrlltr2` 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 more than one uppercase letters in order to not interfere with existing `TEX` or `LATEX` 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 T_EX 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 L^AT_EX, or configure your system to find the file in a special directory.

```
\addrentry{last-name}{first-name}{address}{phone}{F1}{F2}{F3}{F4}{key}
```

Over the years people have objected that the `\adrentry` has only two free parameters. To cater to this demand, there now exists a new command called `\addrentry`—note the additional “d”—which supports four freely-definable parameters. Since T_EX supports maximally nine parameters per command, the comment parameter has fallen away. Other than this difference, the use is the same as that of `\adrentry`.

Both `\adrentry` and `\addrentry` 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 `\addrentry`. 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 `\addrentry`. 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 T_EX.

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*{\adrentry}[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 `\addentry` commands too, than an additional definition for `\addentry` is required before loading the address file:

```

\renewcommand*{\addentry}[9]{%
  \adentry{#1}{#2}{#3}{#4}{#5}{#6}{#7}{#9}%
}

```

In this simple example the extra freely-definable parameter is not used, and therefore `\addentry` is defined with the help of `\adentry`.

With some additional programming one can let the content of the letters depend on the address data. For this the free parameters of the `\adentry` and `\addentry` commands can be used.

Example: Suppose the 5th parameter of the `\adentry` 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*{\adentry}[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 `\adentry` command, or four free parameters of the `\addentry` 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 [Kie10]) is needed. This package contains interactive \LaTeX 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 *last name*. As a separator, the commands `\adrchar` and `\addrchar` can be used. These commands will be ignored if the address files are utilized in `scrlettr2`.

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 [Kie10]. 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 `\addrchar` 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 `\addreentry` and `\addrchar` commands. Former versions only know the commands `\adreentry` and `\addrchar`.

v3.12

Adapting Page Headers and Footers with `scrlayer-scrpage`

Until version 3.11b of KOMA-Script, package `scrpage2` has been the recommended way to customise page headers and footers beyond the options provided by the page styles `headings`, `myheadings`, `plain`, and `empty` of the standard KOMA-Script classes. Until 2001 there was also package `scrpage` as a supported solution for the same purpose. It was then made obsolete and in 2013, more than ten years later, it was finally removed from the regular KOMA-Script distribution.

In 2013 package `scrlayer` became a basic module of KOMA-Script. That package provides a layer scheme and a new page style scheme based upon the layer scheme. Nevertheless, the flexibility it provides and the resulting complexity may be too demanding for the average user to handle. More about `scrlayer` may be found in [chapter 17 of part II](#). Potential problems with the controllability of `scrlayer` apart, there are lots of users who are already familiar with the user interface of package `scrpage2`.

As a consequence the additional package `scrlayer-scrpage` provides a user interface, which is largely compatible with `scrpage2` and based on `scrlayer`. Thus, if you are already familiar with `scrpage2` and refrain from using dirty tricks, like calling internal commands of `scrpage2` directly, it should be easy for you to use `scrlayer-scrpage` as a drop-in replacement. Most examples covering `scrpage2` in L^AT_EX books or online resources should also work with `scrlayer-scrpage` either directly or with only minor code changes provided that they stick to the standard interfaces.

Apart from the aforementioned KOMA-Script packages, you could in principle also use `fancyhdr` (see [\[vO04\]](#)) in conjunction with a KOMA-Script class. However, `fancyhdr` has no support for several KOMA-Script features, e.g., the element scheme (see `\setkomafont`, `\addtokomafont`, and `\usekomafont` in [section 3.6](#), from [page 57](#)) or the configurable numbering format for dynamic headers (see option `numbers` and, e.g., `\chaptermarkformat` in [section 3.16](#), [page 93](#) and [page 105](#)). Hence, if you are using a KOMA-Script class, the usage of package `scrlayer-scrpage` is recommended. Of course you can use `scrlayer-scrpage` with other classes, namely the L^AT_EX standard classes, too.

Besides the features described in this chapter, `scrlayer-scrpage` provides several more that are likely only of minor interest to the average user and for this reason are described from [page 413](#) onwards in [chapter 18 of part II](#). Nevertheless, should the options described in [part I](#) be insufficient for your purposes you are encouraged to examine [chapter 18](#).

5.1. Early or late Selection of Options

All of what is described in [section 2.4](#) is generally applicable. So if you have already read and understood [section 2.4](#) you can switch to [section 5.2](#), [page 227](#).

In this section a peculiarity of KOMA-Script is presented, which, apart from the

scrlayer-scrpage package, is also relevant to other KOMA-Script packages and classes. Such that the user can find all information corresponding to a single package or a single class in the relevant chapter, this section is found almost identically in several chapters. Users who are not only interested in a particular package or class, but wish to gain an overview of KOMA-Script as a whole, may read the section in one chapter and may thereafter skip it wherever coming across it in the document.

```
\documentclass[option list]{KOMA-Script class}
\usepackage[option list]{package list}
```

In L^AT_EX, provision is made for the user to pass class options as a comma-separated list of keywords as optional arguments to `\documentclass`. Apart from being passed to the class, these options are also passed on to all packages which can understand the options. Provision is also made for the user to pass optional arguments as a comma-separated list of keywords as optional arguments to `\usepackage`. KOMA-Script expands the option mechanism for the KOMA-Script classes and various packages to use further possibilities. Thus, most KOMA-Script options can also take a value. An option may have not only the form *Option*, but may also have the form *option=value*. Apart from this difference `\documentclass` and `\usepackage` function the same in KOMA-Script as described in [Tea05b] or any introduction to L^AT_EX, for example [OPHS11].

You should note, that in opposite to the interface described below the options interface of `\documentclass` and `\usepackage` is not robust. So commands, lengths, counters and such constructs may break inside the optional argument of these commands. Because of this, the usage of a L^AT_EX length inside the value of an option would cause an error before KOMA-Script can get the control over the option execution. So, if you want to use a L^AT_EX length, counter or command a part of the value of an option, you have to use `\KOMAOPTIONS` or `\KOMAOPTION`. These commands will be described next.

```
\KOMAOPTIONS{option list}
\KOMAOPTION{option}{value list}
```

KOMA-Script offers most class and package options the opportunity to change the value of options even after loading of the class or package. One may then change the values of a list of options at will with the `\KOMAOPTIONS` command. Each option in the *option list* has the form *option=value*.

Some options also have a default value. If one does not give a value, i. e., gives the option simply in the form *option*, then the default value will be used.

Some options can assume several values simultaneously. For such options there exists the possibility, with the help of `\KOMAOPTION`, to pass a single *option* a list of values. The individual values are given as a comma-separated *value list*.

To implement this possibility KOMA-Script uses the commands `\FamilyOptions` and `\FamilyOption` with the family “KOMA”. For more information in these commands see [part II](#),

v3.00

v3.00

section 12.2, page 313.

5.2. Head and Foot Height

The \LaTeX standard classes do not use the page footer a lot and if they do use it, they put the contents into a `\mbox` which results in the footer being a single text line. This is probably the reason that \LaTeX itself does not have a well-defined foot height. Actually there is `\footskip` giving the distance between the last base line of the text area and the base line of the footer. However, if the footer consists of more than one text line, there is no definite statement whether this length should be the distance to the first or the last base line of the footer.

Despite the fact that the page header of the standard classes will also be put into a horizontal box and therefore is a single text line too, \LaTeX indeed has a length to setup the height of the page header. The reason for this may be that the height will be needed to determine the start of the text area.

```
\footheight
\headheight
```

The package `scrlayer` introduces `\footheight` as a new length similar to `\headheight` of the \LaTeX kernel. Additionally `scrlayer-scrpage` interprets `\footskip` to be the distance from the last possible base line of the text area to the first normal base line of the footer. Package `typearea` interprets `footheight` in the same way. So `typearea`'s foot height options may also be used to setup the values for packages `scrlayer` and `scrlayer-scrpage`. See option **footheight** and **footlines** in section 2.6, page 43) and option **footinclude** at page 40 of the same section.

If you do not use package `typearea`, you should setup the head and foot height using the lengths directly where necessary. At least for the head package `geometry` provides similar settings. If you setup a head or foot height that is too small for the effective content, `scrlayer-scrpage` will try to adjust the corresponding lengths properly. Furthermore, it will warn you and give you additional information about the changes and proper settings you may use yourself. The automatic changes will become valid immediately after the need for them has been detected. They will never be removed automatically, however, even if content with a lower height requirement should be detected at a later point in time.

5.3. Text Markup

What is described in section 3.6 applies, *mutatis mutandis*. So if you have already read and understood section 3.7 you can switch to page 230. But you should have a look at table 5.1, page 228.

\LaTeX offers different possibilities for logical and direct markup of text. Selection of the font family commands, as well as choosing the font size and width is supported. More information about the standard font facilities may be found at [OPHS11], [Tea05b], and [Tea05a].

```
\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`, and `\scshape`, as well as the font size commands `\Huge`, `\huge`, `\LARGE`, `\Large`, `\large`, `\normalsize`, `\small`, `\footnotesize`, `\scriptsize`, and `\tiny`. The description of these commands can be found in [OPHS11], [Tea05b], or [Tea05a]. Color switching commands like `\normalcolor` (see [Car05] and [Ker07]) are also acceptable. The behavior when using other commands, especially those that make redefinitions 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 inside the document body, but only in the document preamble. Usage examples can be found in the paragraphs on the corresponding element. Names, defaults, and meanings of the individual items are listed in [table 5.1](#).

With command `\usekomafont` the current font style may be changed into the font style of the selected *element*.

Table 5.1.: Elements of `scrlayer-scrpage` whose type style can be changed with KOMA-Script command `\setkomafont` or `\addtokomafont` and the default of those, if they have not been defined before loading `scrlayer-scrpage`

footbotline

Horizontal line below the footer of a page style defined using `scrlayer-scrpage`. The font will be used after `\normalfont` and the fonts of elements `pageheadfoot` and `pagefoot`. It is recommended to use this element for colour changes only.

Default: *empty*

footsepline

Horizontal line above the footer of a page style defined using `scrlayer-scrpage`. The font will be used after `\normalfont` and the fonts of elements `pageheadfoot` and `pagefoot`. It is recommended to use this element for colour changes only.

Default: *empty*

Table 5.1.: Elements whose type style can be changed (*continuation*)

headsepline

Horizontal line below the header of a page style defined using `scrlayer-scrpage`. The font will be used after `\normalfont` and the fonts of elements `pageheadfoot` and `pagehead`. It is recommended to use this element for colour changes only.

Default: *empty*

headtopline

Horizontal line above the header of a page style defined using `scrlayer-scrpage`. The font will be used after `\normalfont` and the fonts of elements `pageheadfoot` and `pagehead`. It is recommended to use this element for colour changes only.

Default: *empty*

pagefoot

Contents of the page footer of a page style defined using `scrlayer-scrpage`. The font will be used after `\normalfont` and the font of element `pageheadfoot`.

Default: *empty*

pagehead

Contents of the page header of a page style defined using `scrlayer-scrpage`. The font will be used after `\normalfont` and the font of element `pageheadfoot`.

Default: *empty*

pageheadfoot

Contents of the page header or footer of a page style defined using `scrlayer-scrpage`. The font will be used after `\normalfont`.

Default: `\normalcolor\slshape`

pagenumber

Pagination set with `\pagemark`. If you redefine `\pagemark`, you have to take care that your redefinition also uses `\usekomafont{pagenumber}`!

Default: `\normalfont`

```
\usefontofkomafont{element}
\useencodingofkomafont{element}
\usesizeofkomafont{element}
\usefamilyofkomafont{element}
\useseriesofkomafont{element}
\useshapeofkomafont{element}
```

v3.12

Sometimes and despite the recommendation users use the font setting feature of elements not only for font settings but for other settings too. In this case it may be useful to switch only to the font setting of an element but not to those other settings. You may use `\usefontofkomafont` in such cases. This will activate the font size and baseline skip, the font encoding, the font family, the font series, and the font shape of an element, but no further settings as long as those further settings are local.

You may also switch to one of those attributes only using one of the other commands. Note, that `\usesizeofkomafont` will activate both, the font size and the baseline skip.

You should not misunderstand these commands as a legitimization of using all kind of commands at the font setting of an element. Hence this would result in errors sooner or later (see [section 21.3, page 444](#)).

5.4. Usage of Predefined Page Styles

The easiest way to your desired design for page header and footer with `scrlayer-scrpage` is to use one of the predefined page styles.

```
\pagestyle{scrheadings}
\pagestyle{plain.scrheadings}
```

Package `scrlayer-scrpage` provides two page styles, which may be reconfigured to meet your individual requirements. Let's first of all discuss page style `scrheadings` which has been designed as a style using running heads. Its defaults are similar to the page style `headings` of the `LATEX` standard classes or the `KOMA-Script` classes. What exactly gets printed in the header or footer can be configured via the commands and options described hereafter.

The second page style to be mentioned here is `plain.scrheadings`, which has been designed to be a style with no running head. Its defaults are very similar to page style `plain` of `LATEX`'s standard classes or the `KOMA-Script` classes. The following will describe the commands and options you may use to adjust the contents of the header and footer.

You could of course configure `scrheadings` to be a page style without a running head and `plain.scrheadings` to be a page style using running heads. It is, however, far more expedient to adhere to the conventions mentioned above, if for the only reason that both page styles mutually influence one another. Once you have opted to apply one of these page styles, `scrheadings` will become accessible as `headings` and the page style

`plain.scrheadings` will become accessible as `plain`. Thus, if you use a class or package that automatically switches between `headings` and `plain`, you only need to select `scrheadings` or `plain.scrheadings` once and the switching class or package will then switch between `scrheadings` and `plain.scrheadings` without even being aware of these page styles. Patches or other adaptations of classes (or packages) will not be necessary. This pair of page styles may thus serve as a drop-in replacement for `headings` and `plain`. Should additional similar pairs be required I'd like to point you to [section 18.2](#) in [part II](#) for further reference.

For users of the older `scrpage2`, I'd like to mention that, for compatibility with `scrpage2`, page style `plain.scrheadings` may also be used under its alias name of `scrplain`.

```
\lehead[plain.scrheadings's content]{scrheadings's content}
\cehead[plain.scrheadings's content]{scrheadings's content}
\rehead[plain.scrheadings's content]{scrheadings's content}
\lohead[plain.scrheadings's content]{scrheadings's content}
\cohead[plain.scrheadings's content]{scrheadings's content}
\rohead[plain.scrheadings's content]{scrheadings's content}
```

The contents of the header of page style `plain.scrheadings` and `scrheadings` can be defined using these commands. Thereby the optional argument defines the content of an element of page style `plain.scrheadings`, while the mandatory argument sets the content of the corresponding element of page style `scrheadings`.

Contents of left — so called even — pages can be set with `\lehead`, `\cehead`, and `\rohead`. Remark: The “e” at the second position of the commands’ names means “*even*”.

Contents of right — so called odd — pages can be set with `\lohead`, `\cohead`, and `\rohead`. Remark: The “o” at the second position of the commands’ names means “*odd*”.

Please note that there are only odd pages within single side layouts independent of whether or not they have an odd page number.

Each header consists of a left aligned element that will be defined by `\lehead` respectively `\lohead`. Remark: The “l” at the first position of the commands’ names means “*left aligned*”.

Similarly each header has a centred element that will be defined by `\cehead` respectively `\cohead`. Remark: The “c” at the first position of the command’ names means “*centred*”.

Similarly each header has a right aligned element that will be defined by `\rehead` respectively `\rohead`. Remark: The “r” at the first position of the commands’ names means “*right aligned*”.

However, these elements do not have their own font attributes that may be changed using commands `\setkomafont` and `\addtokomafont` (see [section 3.6](#), [page 57](#)), but are grouped in an element named `pagehead`. And before the font of that element additionally the font of element `pageheadfoot` will be used. See [table 5.1](#) for the font default of these elements.

The semantics of the described commands within two-sided layouts are also sketched in [figure 5.1](#).

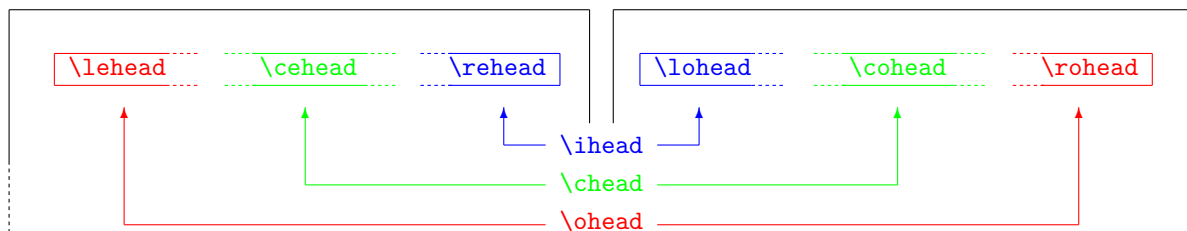


Figure 5.1.: The meaning of the commands to define the contents of the page head of the page styles sketched on a schematic double page

Example: Assume you're writing a short article and you want the title of that article to be shown left aligned and the author's name to be shown right aligned at the page head. You may for example use:

```
\documentclass{scrartcl}
\usepackage{scrlayer-scrpage}
\lohead{John Doe}
\rohead{Page style with \KOMAScript}
\pagestyle{scrheadings}
\begin{document}
\title{Page styles with \KOMAScript}
\author{John Doe}
\maketitle
\end{document}
```

But what happens: On the first page there's only a page number at the page foot, but the header is empty!

The explanation is very easy. Document class `scrartcl` switches to page style `plain` for the page with the title head. After command `\pagestyle{scrheadings}` in the preamble of the short document this will actually result in page style `plain.scrheadings`. Using a KOMA-Script class the default of this page style is an empty page header and a page number in the footer. In the example code the optional arguments of `\lohead` and `\rohead` are omitted. So page style `plain.scrheadings` remains unchanged as default and the result for the first page is indeed correct.

Please add some text below `\maketitle` until a second page will be printed. Alternatively you may just add `\usepackage{lipsum}` into the document preamble and `\lipsum` below `\maketitle`. You will see that the head of the second page will show the author and the document title as we wanted.

To see the difference you should also add an optional argument to `\lohead` and `\rohead` containing some content. To do so, change the example above:


```

\documentclass{scrartcl}
\usepackage{scrlayer-scrpage}
\lohead[John Doe]
    {John Doe}
\rohead[Page style with \KOMAScript]
    {Page style with \KOMAScript}
\pagestyle{scrheadings}
\begin{document}
\title{Page styles with \KOMAScript}
\author{John Doe}
\maketitle
\end{document}

```

Now, you also get a page header above the title head of the first page. That is because you have reconfigured page style `plain.scrheadings` with the two optional arguments. Most of you will also recognise that it would be better to leave this page style unchanged, because the running head above the document title is certainly annoying.

Allow me an important note: You should never put a section heading or section number directly into the page head using a new declaration by one of these commands. This could result in a wrong number or heading text in the running head, because of the asynchronous page generation and output of `TeX`. Instead you should use the mark mechanism and ideally you should use it together with the automatism described in the following section.

```

\lehead*[plain.scrheadings's content]{scrheadings's content}
\cehead*[plain.scrheadings's content]{scrheadings's content}
\rehead*[plain.scrheadings's content]{scrheadings's content}
\lohead*[plain.scrheadings's content]{scrheadings's content}
\cohead*[plain.scrheadings's content]{scrheadings's content}
\rohead*[plain.scrheadings's content]{scrheadings's content}

```

v3.14

The previously described commands have also a version with star that differs only if you omit the optional argument `plain.scrheadings's content`. In this case the version without star does not change the content of `plain.scrheadings`. The version with star on the other hand uses the obligatory argument `scrheading's content` also as default for `plain.scrheadings`. So, if both arguments should be the same, you can simply use the star version with the obligatory argument only.

Example: You can shorten the previous example using the star version of `\lohead` and `\rohead`:

```

\documentclass{scrartcl}
\usepackage{scrlayer-scrpage}

```

```

\lohead*{John Doe}
\rohead*{Page style with \KOMAScript}
\pagestyle{scrheadings}
\begin{document}
\title{Page styles with \KOMAScript}
\author{John Doe}
\maketitle
\end{document}

```

The obsolete package `scrpage2` does not provide this feature.

```

\leftfoot[plain.scrheadings's content]{scrheadings's content}
\rightfoot[plain.scrheadings's content]{scrheadings's content}
\leftfoot[plain.scrheadings's content]{scrheadings's content}
\leftfoot[plain.scrheadings's content]{scrheadings's content}
\rightfoot[plain.scrheadings's content]{scrheadings's content}
\rightfoot[plain.scrheadings's content]{scrheadings's content}

```

The contents of the footer of page style `plain.scrheadings` and `scrheadings` can be defined using these commands. Thereby the optional argument defines the content of an element of page style `plain.scrheadings`, while the mandatory argument sets the content of the corresponding element of page style `scrheadings`.

Contents of left — so called even — pages can be set with `\leftfoot`, `\rightfoot`, and `\rohead`. Remark: The “e” at the second position of the commands’ names means “*even*”.

Contents of odd pages can be set with `\lofoot`, `\cofoot`, and `\rofoot`. Remark: The “o” at the second position of the commands’ names means “*odd*”.

Please note that there are only odd pages within single side layouts independent of whether or not they have an odd page number.

Each footer consists of a left aligned element that will be defined by `\leftfoot` respectively `\lofoot`. Remark: The “l” at the first position of the commands’ names means “*left aligned*”.

Similarly each footer has a centred element that will be defined by `\rightfoot` respectively `\cofoot`. Remark: The “c” at the first position of the command’ names means “*centred*”.

Similarly each footer has a right aligned element that will be defined by `\leftfoot` respectively `\rofoot`. Remark: The “r” at the first position of the commands’ names means “*right aligned*”.

However, these elements do not have their own font attributes that may be changed using commands `\setkomafont` and `\addtokomafont` (see [section 3.6, page 57](#)), but are grouped in an element named `pagefoot`. And before the font of that element additionally the font of element `pageheadfoot` will be used. See [table 5.1](#) for the defaults of the fonts of these elements.

The semantics of the described commands within two-sided layouts are also sketched in [figure 5.2](#).

Example: Let's return to the example of the short article. Assuming you want to print the publisher at the left side of the page footer, you would change the example above into:

```
\documentclass{scrartcl}
\usepackage{scrlayer-scrpage}
\lohead{John Doe}
\rohead{Page style with \KOMAScript}
\lofoot{Smart Alec Publishing}
\pagestyle{scrheadings}
\usepackage{lipsum}
\begin{document}
\title{Page styles with \KOMAScript}
\author{John Doe}
\maketitle
\lipsum
\end{document}
```

Once again the publisher is not printed on the first page with the title head. For the reason see the explanation about `\lohead` in the example above. And again the solution to print the publisher on the first page would be similar:

```
\lofoot[Smart Alec Publishing]
      {Smart Alec Publishing}
```

But now you also want to replace the slanted font used in page head and footer by a upright smaller font. This may be done using:

```
\setkomafont{pageheadfoot}{\small}
```

Furthermore, the head but not the footer should be bold:

```
\setkomafont{pagehead}{\bfseries}
```

For the last command it is important to have it just after `scrpage-scrlayer` has been loaded, because the KOMA-Script class already defines `pagehead` and `pageheadfoot` but with the same meaning. Only loading `scrpage-scrlayer` changes

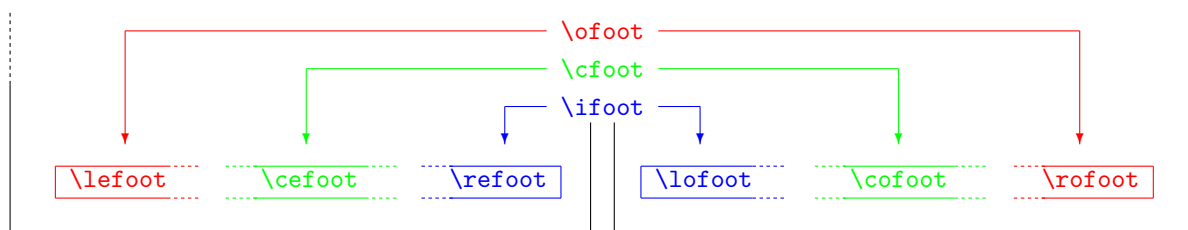


Figure 5.2.: The meaning of the commands to define the contents of the page footer of the page styles sketched on a schematic double page

the meaning of `pagehead` and makes it an element independent of `pageheadfoot`.

Now, please add one more `\lipsum` and add option `twoside` to the loading of `scrartcl`. First of all, you will see the page number moving from the middle of the page footer to the outer margin, due to the changed defaults of `scrheadings` and `plain.scrheadings` using double-sided layout and a KOMA-Script class.

Simultaneously the author, document title and publisher will vanish from page 2. It only appears on page 3. This is a consequence of using only commands for odd pages. You can recognise this by the “o” on the second position of the commands’ names.

Now, we could simply copy those commands and replace the “o” by an “e” to define the contents of *even* pages. But with double sided layout it makes more sense to use mirror-inverted elements. So the left element of an odd page should become the right element of the even page and visa versa. To achieve this, we also replace the first letter “l” by “r”:

```
\documentclass[twoside]{scrartcl}
\usepackage{scrlayer-scrpage}
\lohead{John Doe}
\rohead{Page style with \KOMAScript}
\lofoot[Smart Alec Publishing]
      {Smart Alec Publishing}
\rehead{John Doe}
\lohead{Page style with \KOMAScript}
\refoot[Smart Alec Publishing]
      {Smart Alec Publishing}
\pagestyle{scrheadings}
\usepackage{lipsum}
\begin{document}
\title{Page styles with \KOMAScript}
\author{John Doe}
\maketitle
\lipsum\lipsum
\end{document}
```

After reading the example it may appear to you that it is somehow uncomfortable to duplicate commands to have the same contents on mirror positions of the page header or footer of a double page. Therefore you will learn to know an easier solution for this standard case next.

Before allow me an important note: You should never put a section heading or section number directly into the page’s footer using a new declaration by one of these commands. This could result in a wrong number or heading text in the running footer, because of the asynchronous page generation and output of T_EX. Instead you should use the mark mechanism ideally together with the automatism described in the following section.

```

\lefoot*[plain.scrheadings's content]{scrheadings's content}
\cefoot*[plain.scrheadings's content]{scrheadings's content}
\refoot*[plain.scrheadings's content]{scrheadings's content}
\lofoot*[plain.scrheadings's content]{scrheadings's content}
\cofoot*[plain.scrheadings's content]{scrheadings's content}
\rofoot*[plain.scrheadings's content]{scrheadings's content}

```

v3.14

The previously described commands have also a version with star that differs only if you omit the optional argument *plain.scrheadings's content*. In this case the version without star does not change the content of `plain.scrheadings`. The version with star on the other hand uses the obligatory argument *scrheading's content* also as default for `plain.scrheadings`. So, if both arguments should be the same, you can simply use the star version with the obligatory argument only.

Example: You can shorten the previous example using the star version of `\lofoot` and `\refoot`:

```

\documentclass[twoside]{scrartcl}
\usepackage{scrlayer-scrpage}
\lohead{John Doe}
\rohead{Page style with \KOMAScript}
\lofoot*{Smart Alec Publishing}
\rehead{John Doe}
\lohead{Page style with \KOMAScript}
\refoot*{Smart Alec Publishing}
\pagestyle{scrheadings}
\usepackage{lipsum}
\begin{document}
\title{Page styles with \KOMAScript}
\author{John Doe}
\maketitle
\lipsum\lipsum
\end{document}

```

The obsolete package `scrpage2` does not provide this feature.

```

\ohead[plain.scrheadings's content]{scrheadings's content}
\chead[plain.scrheadings's content]{scrheadings's content}
\ihead[plain.scrheadings's content]{scrheadings's content}
\ofoot[plain.scrheadings's content]{scrheadings's content}
\cfoot[plain.scrheadings's content]{scrheadings's content}
\ifoot[plain.scrheadings's content]{scrheadings's content}

```

To define the contents of page headers and footers of odd and the even pages of a double-sided layout using the commands described before, you would have to define the contents of the even page different from the contents of the odd page. But in general the pages should be symmetric. An element, that should be printed left aligned on an even page, should be right aligned on an odd page and vice versa. Elements, that are centred on odd pages, should be centred on even pages too.

To simplify the definition of such symmetric page styles, `scrlayer-scrpage` provides a kind of abbreviation. Command `\ohead` is same like usage of both `\lehead` and `\rohead`. Command `\chead` is same like `\cehead` and `\cohead`. And command `\ihead` is same like `\rehead` and `\lohead`. The corresponding commands for the page footer are defined accordingly. A sketch of these commands can be found also in [figure 5.1](#) on [page 232](#) and [figure 5.2](#) on [page 235](#) together with the relationships of all the page header and footer commands.

Example: You can simplify the example before using the new commands:

```

\documentclass[twoside]{scrartcl}
\usepackage{scrlayer-scrpage}
\ihead{John Doe}
\ohead{Page style with \KOMAScript}
\ifoot[Smart Alec Publishing]
    {Smart Alec Publishing}
\pagestyle{scrheadings}
\usepackage{lipsum}
\begin{document}
\title{Page styles with \KOMAScript}
\author{John Doe}
\maketitle
\lipsum\lipsum
\end{document}

```

As you can see, you can spare half of the commands but get the same result.

In single-sided layouts all pages are odd pages. So in LaTeX's single-sided mode these commands are synonymous for the odd page commands. Therefore in most cases you will only need these six commands instead of the twelve described before.

Once again, allow me an important note: You should never put a section heading or section number directly into the page head or page foot using a new declaration by one of these commands. This could result in a wrong number or heading text in the running header or

footer, because of the asynchronous page generation and output of \TeX . Instead you should use the mark mechanism ideally together with the automatism described in the following section.

```
\ohead*[plain.scrheadings's content]{scrheadings's content}
\chead*[plain.scrheadings's content]{scrheadings's content}
\ihead*[plain.scrheadings's content]{scrheadings's content}
\ofoot*[plain.scrheadings's content]{scrheadings's content}
\cfoot*[plain.scrheadings's content]{scrheadings's content}
\ifoot*[plain.scrheadings's content]{scrheadings's content}
```

v3.14

The previously described commands have also a version with star that differs only if you omit the optional argument `plain.scrheadings's content`. In this case the version without star does not change the content of `plain.scrheadings`. The version with star on the other hand uses the obligatory argument `scrheading's content` also as default for `plain.scrheadings`. So, if both arguments should be the same, you can simply use the star version with the obligatory argument only.

Example: You can shorten the previous example using the star version of `\ifoot`:

```
\documentclass[twoside]{scrartcl}
\usepackage{scrlayer-scrpage}
\ihead{John Doe}
\ohead{Page style with \KOMAScript}
\ifoot*[Smart Alec Publishing]
\pagestyle{scrheadings}
\usepackage{lipsum}
\begin{document}
\title{Page styles with \KOMAScript}
\author{John Doe}
\maketitle
\lipsum\lipsum
\end{document}
```

The obsolete package `scrpage2` does not provide this feature.

`pagestyleset=setting`

In the examples above you can already find some information about the defaults of `scrheadings` and `plain.scrheadings`. Indeed `scrlayer-scrpage` provides two different defaults yet. You may select one of those defaults manually using option `pagestyleset`.

If `setting` is KOMA-Script the defaults will be used that would also be activated automatically if a KOMA-Script class has been detected. In this case and within double-sided layout `scrheadings` uses running heads outer aligned in the page head. The page number will be

printed outer aligned in the page footer. Within single-sided layout the running head will be printed in the middle of the page head and the page number in the middle of the page footer. Upper and lower case will be used for the automatic running head as given by the words you have typed. This would be same like using Option `markcase=used`. Page style `plain.scrheadings` has not got running heads, but the page numbers will be printed in the same manner.

If *setting* is `standard` the defaults will be used that are similar to page style `headings` and `plain` of the standard classes. This *setting* will also be activated automatically if the option has not been used and KOMA-Script class cannot be detected. Within double-sided layout thereby `scrheadings` uses running heads aligned inner in the page head and the page numbers will be printed — also in the page head — aligned outer. Within single-sided layout `scrheadings` is the same. But because of single side layout knows only odd pages, the running head will be aligned left always and the page number will be aligned right. In spite of typographic objection, the automatic running head will be converted into upper cases like they would using `markcase=upper`. Within single side layout page style `plain.scrheadings` differs a lot from `scrheading`, because the page number will be printed in the middle of the page footer. Using double side layout page style `plain.scrheadings` differs from standard classes' `plain`. The standard classes would print the page number in the middle of the page footer. But this would not harmonise with the `scrheadings`, so `plain.scrheadings` does not print a page number. But like `plain` it does not print a running head.

Please note that together with this option page style `scrheadings` will be activated. This will be also the case, if you use the option inside the document.

Options `komastyle` and `standardstyle`, provided by `scrpage`, are defined only for compatibility reasons in `scrlayer-scrpage`. But they are implemented using option `pagestyleset`. They are deprecated and you should not use them.

5.5. Manipulation of Defined Page Styles

Section 5.4 states the predefined defaults for the page style `scrheadings` and `plain.scrheadings` and how they can be adapted. But information about the generation of, i. e., the running heads, the manipulation of the widths of page head and footer, and how you can get horizontal lines printed above or below the page head or footer are still missing. Even though all these are features of package `scrlayer`, they will be described consecutively, because these basic features of `scrlayer` are also a fundamental part of the features of `scrlayer-scrpage`.


```

\automark[section level of the right mark]{section level of the left mark}
\automark*[section level of the right mark]{section level of the left mark}
\manualmark
automark
autooneside=simple switch
manualmark

```

With the L^AT_EX standard classes or the KOMA-Script classes the decision whether using automatic running heads, or static or manual running heads would be done using either page style headings or myheadings. Automatic running heads are replications of a significant, characterizing text snippet of the page mostly inside the page head, sometimes in the page footer.

article, The article classes article or scrartcl with page style headings use the section heading, which
scrartcl is either the mandatory or the optional argument of `\section`, for the automatic running head of single side documents. Double side documents use this section heading as the *left mark* and at once use the subsection heading as the *right mark*. The left mark will be printed on left pages, which founds the name *left mark*. The right mark will be printed on right—in single side mode this means every—page. The classes by default also remove the right mark whenever they put the section heading into the left mark.

report, The report and book classes start one level higher. So they use the chapter heading to be the
scrreprt, right mark in single side layout. In double side layout they use the chapter heading to be the left
book, mark and the section heading to be the right mark.
scrbook

If you are using myheadings instead of headings, the marks in the page header still exists and would be printed same like using headings. But section commands will not automatically set the marks any longer. So you can fill them only using the commands `\markright` and `\markboth`, which will be described later in this section.

This difference between those two page styles has been abolished by `scrpage2` and also by `scrlayer`. Instead of distinguishing between automatic and manual running head by the selection of a page style, two new commands, `\automark` and `\manualmark`, are provided.

The command `\manualmark` switches to manual marks and deactivates the automatic filling of the marks. In contrast to this `\automark` and `\automark*` can be used to define, which section levels should be used for the automatic mark filling. The optional argument is the *section level of the right mark*, the mandatory argument the *section level of the left mark*. The arguments always should be the name of a section level like `part`, `chapter`, `section`, `subsection`, `subsubsection`, `paragraph`, or `subparagraph`.

Generally the higher level should be used for the left mark and the lower level for the right mark. This is only a convention and not a need, but it makes sense.

Please note that not every class provides running heads for every section level. For example the standard classes never setup the running head of `\part`. The KOMA-Script classes provide running heads for every section level.

The difference in `\automark` and `\automark*` is, that `\automark` deletes all prior usages of `\automark` or `\automark*`, while `\automark*` changes only the behaviour of the section levels

of its arguments. So you can even build more complex cases.

Example: Assume you want the chapter heading to be the running head of even pages and the section heading to be the running head of odd pages like this is usual for books. But on odd pages you also want the chapter headings as a running head as long as the first section appears. To do so, you first have to load `scrlayer-scrpage` and select page style `scrheadings`. So your document starts with:

```
\documentclass{scrbook}
\usepackage{scrlayer-scrpage}
\pagestyle{scrheadings}
```

Next you will setup the chapter heading to be not only the left but also the right mark:

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

Then the section heading should also become a right mark:

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

Here the star version is used, because the prior `\automark` command should be still valid. Additionally the mandatory argument for the *section level of the left mark* is empty, because this mark should be unchanged.

Now you just need some document contents to see a result:

```
\usepackage{lipsum}
\begin{document}
\chapter{Chapter Heading}
\lipsum[1-20]
\section{Section Heading}
\lipsum[21-40]
\end{document}
```

We once again use package `lipsum` to generate some dummy text with command `\lipsum`. The package is really useful.

If you'd test the example, you'd see, that the chapter start page does not have a running head as usual. This is, because it automatically uses the `plain` page style `plain.scrheadings`. Pages 2–4 have the chapter headings in the running head. After the section heading on page 4 the running head of page 5 changes into this section heading. From this page to the end the running head alternates between chapter and section heading from page to page.

Instead of the commands you may also use the options `manualmark` and `automark` to switch between manual and automatic running heads. Thereby `automark` always uses the default

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

for classes with `\chapter` and

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

for classes without `\chapter`.

But normally in single side mode you do not want that the lower level influences the right mark, you want the higher level, that will fill only the left mark in double side layout, to be the running head of all pages. The default option `autooneside` corresponds to this behaviour. The option understands the values for simple switches, that can be found in [table 2.5](#) on [page 39](#). If you'd deactivate the option, in single side layout the optional and the obligatory arguments of `\automark` and `\automark*` will influence the running head again.

Example: Assume you have a single sided report but want similar running heads as in the book example before. The chapter headings should be used as a running head as long as the first section appears. From the first section on, the section heading should be used. So we modify the previous example a little bit:

```
\documentclass{scrreprt}
\usepackage[autooneside=false]{scrlayer-scrpage}
\pagestyle{scrheadings}
\automark[section]{chapter}
\usepackage{lipsum}
\begin{document}
\chapter{Chapter Heading}
\lipsum[1-20]
\section{Section Heading}
\lipsum[21-40]
\end{document}
```

You can see, that we do not need a `\automark*` command in this case. Please try the example also with `autooneside` set to `true` or remove the option and its value. You should find a difference at the running head in the pages' head from page 4 on.

Please note, only loading the package does not have any effect on the fact whether automatic or manual running heads are used or what kind of section headings do fill up the marks. Only using an explicit option `automark` or `manualmark` or one of the commands `\automark` or `\manualmark` can reach a well defined state.

draft=simple switch

This KOMA-Script option understands the values for simple switches, that are shown in [table 2.5](#) on [page 39](#). If the option is active, all elements of the page styles will also show dimension lines. This might be useful while draft editing. If the option has been set as a global option (see the optional argument of `\documentclass`), but you do not want the dimension

lines, you can deactivate them for the package only using `draft=false` as an optional argument of `\usepackage` while loading the package.

```
\MakeMarkcase{string}
```

The automatic running head and only this uses `\MakeMarkcase` for its output. If the command has not been defined, e.g., by the class, while loading `sclayer`, it would be defined with the default of outputting the argument `string` without changes. But the default can be change either redefining `\MakeMarkcase` or using option `markcase`, that will be described next. Depending on the setting the argument could, e.g., be converted into upper or lower cases.

```
markcase=Wert
```

As already mentioned with `sclayer` you may switch between manual and automatic running heads. Using automatic running heads the corresponding marks will be filled by the section heading commands. Some culture areas do convert the running heads into upper case letters in opposite to the German typographic habit. The \LaTeX standard classes do so by default. Package `sclayer` optionally provides this too. Therefore you'd use option `markcase=upper` which results in a redefinition of `\MakeMarkcase`, a command, that is used by `sclayer` for automatic running heads.

Unfortunately \LaTeX 's command for upper case letter typesetting, `\MakeUppercase` results in an very inadequate typesetting, because it neither uses letter spacing nor does it space balancing. One reason for this behaviour might be, that a glyph analyzing would be needed, to incorporate the letter shapes and their combination while space balancing. Because of this KOMA-Script author suggests to abstain from upper case letter typesetting for running heads. This could be achieved using `markcase=used`. Nevertheless, some classes would add `\MarkUppercase` or even \TeX command `\uppercase` into the running heads. For such cases option `markcase=noupper` can be used. This will also deactivate `\MakeUppercase` and `\uppercase` inside the running heads.

You can find all valid values for `markcase` in [table 5.2](#).

```
\leftmark
\rightmark
\headmark
\pagemark
```

If you want to differ from the predefined page styles, usually you need to decide, where to place the marks' contents. With `\leftmark` you can state the contents of the left mark.

Similar you can use `\rightmark` to state the contents of the right mark. For more information about some intricacies of this see the further description of [21.1](#) in [section 21.1](#), [page 441](#).

Table 5.2.: Possible values for option `markcase` to select upper/lower case letter typesetting in automatic running heads

<code>lower</code>	redefines <code>\MakeMarkcase</code> to convert the automatic running heads into lower case letters using <code>\MakeLowercase</code> (lower case typesetting).
<code>upper</code>	redefines <code>\MakeMarkcase</code> to convert the automatic running heads into upper case letters using <code>\MakeUppercase</code> (upper case typesetting).
<code>used</code>	redefines <code>\MakeMarkcase</code> to use automatic running heads without any case changes.
<code>ignoreuppercase, nouppercase, ignoreupper, noupper</code>	redefines not only <code>\MakeMarkcase</code> but also locally to the running heads <code>\MakeUppercase</code> and <code>\uppercase</code> to leave the automatic running heads unchanged.

Somehow easier would be usage of `\headmark`. This extension of `scrlayer` aliases either `\leftmark` or `\rightmark` depending on whether the current page is even or odd.

Strictly thought command `\pagemark` is not involved by \TeX 's mark mechanism. It is only used to output a formatted page number. The font of element `pagenumber` will be used for the output. This can be changed using command `\setkomafont` or `\addtokomafont` (see also [section 3.6, page 57](#)).

Example: Assume you want to use a left aligned running head and right aligned page number in the head of the pages of a document in single side layout. The following minimal but working examples would do so:

```
\documentclass{scrreprt}
\usepackage{blindtext}
\usepackage[automark]{scrlayer-scrpage}
\pagestyle{scrheadings}
\ihead{\headmark}
\ohead[\pagemark]{\pagemark}
\chead{}
\cfoot[]{}
\begin{document}
\blinddocument
\end{document}
```

Here package `blindtext` with its command `\blinddocument` has been used to easily generate content for an example document.

Commands `\ihead` and `\ohead` have been used to place the wanted marks. Al-

though the page mark will be place not only to pages in page style `scrheadings` but using the optional argument also to pages in page style `plain.scrheadings`.

Because of both page styles already have marks in the middle of the page head and page foot, those elements will be cleared using `\chead` and `\cfoot` with empty arguments. Alternatively you'd use `\clearpairofpagestyles` *before* `\ihead`. You will find a description of this command in [section 18.2](#).

Please note, that the empty optional argument of `\cfoot` in the example above is not the same like omitting the optional argument. Please try it out and have a look at the difference in the footer of the first page.

If all the mark features described above are not sufficient, experienced users can find more of them on [page 415](#). For example, you can find there `\leftfirstmark` and `\rightbotmark`, which seem to be useful for lexicon like documents.

```
\partmarkformat
\chaptermarkformat
\sectionmarkformat
\subsectionmarkformat
\subsubsectionmarkformat
\paragraphmarkformat
\subparagraphmarkformat
```

Usually the KOMA-Script classes and package `scrlayer` use these commands internally to bring the section numbers into wanted form and additionally they support the `\autodot` mechanism of the KOMA-Script classes. If wanted these commands may be redefined to get another form of section numbers.

Example: Assume you do not want section numbers but section heading text only in the running head. So:

```
\renewcommand*{\sectionmarkformat}{}%
```

would be a simple solution for this.

```

\partmark{Text}
\chaptermark{Text}
\sectionmark{Text}
\subsectionmark{Text}
\subsubsectionmark{Text}
\paragraphmark{Text}
\subparagraphmark{Text}

```

Most classes use these commands to setup marks corresponding to the section commands. Thereby the only argument is the text without the number of the section heading, that should be used for the running head. For the number simply the number of the current section level will be used, if the current level uses numbers.

Unfortunately, not all classes use such a command for every section level. The standard classes for example do not call `\partmark`. The KOMA-Script classes support such commands for all section levels and therefore also use `\partmark`.

If users redefine these commands, they should take care to also use the counter `secnumdepth` for the test whether or not the section level should output a number, even in the case the user does not change counter `secnumdepth` himself, because packages and classes may do so locally and rely on correct handling of `secnumdepth`.

However, package `scrlayer` redefines these commands whenever you use `\automark` or `\manualmark` or the corresponding options, to activate or deactivate the wanted running heads.

```

\markleft{left mark}
\markright{right mark}
\markboth{left mark}{right mark}

```

Independent of whether currently manual or automatic running heads are active, you may change the contents of the *left mark* or the *right mark* at any time using these commands. You should note, that the resulting contents of `\leftmark` is the *left mark* of the last `\markleft` or `\markboth` command of the current page. In opposite to this the resulting contents of `\rightmark` is the *right mark* of the first `\rightmark` or `\markboth` command of the current page.

If you are using manual running heads, the marks will stay valid until one of the corresponding commands will be used again. If you are using automatic running heads the marks can loose their validity with the next section heading depending on the configuration of the automatism.

You may also use these commands together with the star versions of the section commands.

Example: Assume you are using a preface with several pages just before the table of contents, that should not have an entry at the table of contents itself. Because of using a separation line at the page head, you want also a running head for this preface:

```

\documentclass[headsepline]{book}

```

```

\usepackage{scrlayer-scrpage}
\pagestyle{scrheadings}
\usepackage{blindtext}
\begin{document}
\chapter*{Preface}
\markboth{Preface}{Preface}
\blindtext[20]
\tableofcontents
\blindeddocument
\end{document}

```

First of all this seems to produce the wanted result. But have a closer look: In difference to the other running heads “**Preface**” is not in upper case letters. But you can fix this easily:

```

\documentclass[headsepline]{book}
\usepackage{scrlayer-scrpage}
\pagestyle{scrheadings}
\usepackage{blindtext}
\begin{document}
\chapter*{Preface}
\markboth{\MakeMarkcase{Preface}}{\MakeMarkcase{Preface}}
\blindtext[20]
\tableofcontents
\blindeddocument
\end{document}

```

Using command `\MakeMarkcase` results in getting the same letter case as for automatic running heads.

Now, let’s move command `\tableofcontents` in front of the preface. Let’s remove the `\markboth` command too. You’ll realise, that the preface now has the running head “CONTENTS”. This is because of a quirk of `\chapter*` (see also [section 3.16](#) on [page 99](#)). If in such cases the running head should be removed, `\markboth` with two empty arguments would be useful:

```

\documentclass[headsepline]{book}
\usepackage{scrlayer-scrpage}
\pagestyle{scrheadings}
\usepackage{blindtext}
\begin{document}
\tableofcontents
\chapter*{Preface}
\markboth{}{}
\blindtext[20]
\blindeddocument
\end{document}

```



```
headwidth=width:offset:offset
footwidth=width:offset:offset
```

By default the page head and foot are as wide as the type area. This can be changed using these KOMA-Script options. The value *width* is the wanted width of the head respective foot. The *offset* defines how much the head or foot should be moved towards the outer—in single side layout to the right—margin. All three values are optional and can be omitted. If you omit a value, you can also omit the corresponding colon left beside. If there is only one *offset* it is used for both, odd and even pages. Otherwise, the first *offset* is used for odd and the second *offset* for even pages in two-side mode. If you only use one value without colon, this will be the *width*.

For the *width* as well as the *offset* you can use any valid length value, L^AT_EX length, T_EX dimension or T_EX skip. In addition you may use an ε -T_EX dimension expression with basic arithmetic operations +, −, *, /, and parenthesis. See [Tea98, section 3.5] for more information on such expressions. See [section 5.1](#) for more information on using, e. g., a L^AT_EX length as an option value. The *width* can alternatively be one of the symbolic values shown in [table 5.3](#).

By default the head and the foot are as wide as the text area. The default *offset* depends on the used *width*. In single side layout generally the half of the difference of *width* and the width of the text area will be used. This results in horizontal centring the page head above or the page footer below the text area. In difference to this, in double side layout generally a third of the difference of *offset* and the width of the text area will be used. But if *width* is the width of the whole text area plus the marginal note column, default *offset* will be zero. If you think, this is complicated, you should simply use an explicit *offset*.

```
headtopline=thickness:length
headsepline=thickness:length
footsepline=thickness:length
footbotline=thickness:length
```

The KOMA-Script classes provide only a separation line below the page head and above the page head, and you may only switch each of these lines on or off. But package `scrlayer-scrpage` provides four such horizontal lines: one above the head, one below the head, one above the foot, and one below the foot. And you can not only switch them on an off, but also configure the *length* and *thickness* of each of these lines.

Both values are optional. If you omit the *thickness*, a default value of 0.4pt will be used, a so called *hairline*. If you omit the *length*, the width of the head respective the foot will be used. If you omit both, you can also omit the colon. If you use only one value without colon, this will be the *thickness*.

For sure, the *length* can be not only shorter than the current width of the page head respectively the page foot, but also longer. See additionally options [ilines](#), [clines](#), and [olines](#) later in this section.

Table 5.3.: Possible symbolic values for the *width* value of options `headwidth` and `footwidth`

<code>foot</code>	the current width of the page foot
<code>footbotline</code>	the current length of the horizontal line below the page foot
<code>footsepline</code>	the current length of the horizontal line above the page foot
<code>head</code>	the current width of the page head
<code>headsepline</code>	the current length of the horizontal line below the page head
<code>headtopline</code>	the current length of the horizontal line above the page head
<code>marginpar</code>	the current width of the marginal note column including the distance between the text area and the marginal note column
<code>page</code>	the current width of the page considering a binding correction of package <code>typearea</code> (see option <code>BCOR</code> in section 2.6, page 31)
<code>paper</code>	the current width of the paper without considering a binding correction
<code>text</code>	the current width of the text area
<code>textwithmarginpar</code>	the current width of the text area plus the marginal note column including the distance between them (note: in this case and only in this case the default of <i>offset</i> would be zero)

Beside the length and thickness also the colour of the lines can be changed. First of all the colour depends on the colour of the head or foot. But independent from those or additional to them the settings of the corresponding elements `headtopline`, `headsepline`, `footsepline`, and `footbotline` will be used. You may change these using command `\setkomafont` or `\addtokomafont` (see [section 3.6](#) from [page 57](#)). By default those settings are empty, which means no change of the current font or colour. Change of font in opposite to colour would not make sense and is not recommended for these elements.

Package `scrpage2` has additionally to the options that do not take any values, also four commands `\setheadtopline`, `\setheadsepline`, `\setfootsepline`, and `\setfootbotline`. These have a first optional argument for the *length*, a second mandatory argument for the *thickness*, and a third optional argument for the setting of font or colour. Package `scrlayer-scrpage` does also provide those commands. Nevertheless, these commands are deprecated and should not be used any longer. To get it clear: These commands have never been made to switch the lines on or off. They have been made to configure already switched on lines. Users often ignored this!

```
plainheadtopline=simple switch
plainheadsepline=simple switch
plainfootsepline=simple switch
plainfootbotline=simple switch
```

These options can be used to inherit the settings of the lines also for the `plain` page style. Possible values for *simple switch* can be found in [table 2.5](#) on [page 39](#). If a option is activated, the `plain` page style will use the line settings given by the options and commands described above. If the option is deactivated, the `plain` will not show the corresponding line.

```
ilines
clines
olines
```

You have already been told that the horizontal lines above or below the page head or foot can be shorter or longer than the page head or page foot itself. Only the answer to the question about the alignment of those lines is still missing. By default all lines are left aligned at single side layout and aligned to the inner margin of the head or foot at double side layout. This is same like using option `ilines`. Alternatively, you can use option `clines` to centre the lines in the head or foot, or option `olines` to align them right respectively to the outer margin.

The Day of the Week Using `scrdate`

With version 3.05a the functionality of this package enhanced a lot. Beside of the current day of the week this package provides the day of the week of every date of the Gregorian calendar now.

```
\CenturyPart{year}
\DecadePart{year}
```

v3.05a

The command `\CenturyPart` offers the value of the century digits—hundreds and thousands—of a *year*. The command `\DecadePart` in difference offers the other digits which are the units and tens. The number of digits of *year* does not care. The value may be assigned to a counter or may be used for calculations, i.e., using `\numexpr`. For output of an Arabic number of the value prefix it with `\the`.

Example: You want to calculate and output the century of the current year.

```
The year \the\year\ is the year \the\DecadePart{\year}
of the \engord{\numexpr\CenturyPart{\year}+1\relax} century.
```

The result would be:

The year 2017 is the year 17 of the 21st century.

Package `engord` has been used for this example. See [\[Obe10\]](#) for more information.

Please note, that within used method of counting the year 2000 is the year 0—and therefore the first year—of the 21st century.

```
\DayNumber{year}{month}{day}
\ISODayNumber{ISO-date}
```

v3.05a

These two commands offers the value of the number of the day of the week of any date. The differ only in the kind of date declaration. Command `\DayNumber` needs *year*, *month*, and *day* as separate parameters. Command `\ISODayNumber` expects an *ISO-date* as a single argument. The expected format of the *ISO-date* is: *year-month-day*. It does not matter whether *month* or *day* have one or two digits. The result of both commands may be assigned to a counter or used for calculations, i.e., using `\numexpr`. For output of an Arabic number of the value prefix it with `\the`.

Example: You want to know the number of the day of the week of the 1st May 2027.

```
The 1st-May-2027 has \the\ISODayNumber{2027-5-1}
as the number of the day of the week.
```

The result will be:

The 1st May 2027 has 6 as the number of the day of the week.

A special feature is to walk a number of days into future or past from a given date.

Example: You want to know the number of the day of the week, that will be in 12 days and that will be 24 days before the 24th December 2027.

In 12~days the number of the day of the week will be `\the\DayNumber{\year}{\month}{\day+12}` and 24~days before the 24th~December~2027 it will be `\the\ISODayNumber{2027-12-24-24}`.

The result may be, e.g.:

In 12 days the number of the day of the week will be 5 and 24 days before the 24th December 2027 it will be 2.

The days of the week are numbered: Sunday = 0, Monday = 1, Tuesday = 2, Wednesday = 3, Thursday = 4, Friday = 5, and Saturday = 6.

```
\DayNameByNumber{number of the day of the week}
\DayName{year}{month}{day}
\ISODayName{ISO-date}
```

v3.05a

Usually you do not want to know the number of the day of the week, but the name of the day of the week. Because of this, the command `\DayNameByNumber` offers the name of the day of the week corresponding with a number. The number may be the result of `\DayNumber` or `\ISODayNumber`. The two commands `\DayName` and `\ISODayName` directly offer the name of the day of the week of a given date.

Example: You want to know the name of the day of the week of the 24th December 2027.

Please pay you bill until `\ISODayName{2027-12-24}`,
24th~December~2027.

The result will be:

Please pay you bill until Friday, 24th December 2027.

Again a special feature is to make some calculations inside the argument of `\DayName`.

Example: You want to know the names of the days of the week, that will be in 12 days and that will be 24 days before the 24th December 2027.

In 12~days the name of the day of the week will be `\DayName{\year}{\month}{\day+12}` and 24~days before the 24th~December~2027 it will be `\ISODayName{2027-12-24-24}`. Nevertheless two weeks and three days after a Wednesday a `\DayNameByNumber{3+2*7+3}` will follow.

The result may be, e. g.:

In 12 days the name of the day of the week will be Friday and 24 days before the 24th December 2027 it will be Tuesday. Nevertheless two weeks and three days after a Wednesday a Saturday will follow.

```
\ISOToday
\IsoToday
\todayname
\todaynumber
```

v3.05a

In the prior examples the current date have been given clumsily and explicitly using the \TeX registers `\year`, `\month`, and `\day`. The commands `\ISOToday` and `\IsoToday` offers the current date in ISO-notation directly. These commands differ in the number of digits for numbers less than 10 only. `\ISOToday` prefixes numbers less than 10 for the month and day with a 0. In opposite to this `\IsoToday` will show numbers less than 10 for the month and day with one digit only. Command `\todayname` directly offers the name of the current day of the week. Command `\todaynumber` offers the number of that name instead. More information about usability of this value may be found at previous description of `\DayNumber` and `\ISODayNumber`.

Example: I want to show you the name of the weekday in which this document has been type-set:

```
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 Sunday.
```

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.

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`, e. g.:

```
\MakeLowercase{\todayname}
```

This converts the whole argument into lower case letters. Of course, this may be done also using previous described commands `\DayNameByNumber`, `\DayName` and `\ISODayName`.

```
\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.

There's no such command for changing the result of `\ISOToday` or `\IsoToday`.

```
\newdaylanguage{language}{Monday}{Tuesday}{Wednesday}{Thursday}{Friday}{Saturday}
{Sunday}
```

Currently the package scrdate knows the following languages:

- Croatian (`croatian`),
- Czech (`czech`),
- Danish (`danish`),
- Dutch (`dutch`),
- English (`american`, `australian`, `british`, `canadian`, `english`, `UKenglish`, and `USenglish`),
- Finnish (`finnish`),
- French (`acadian`, `canadien`, `francais`, and `french`),
- German (`austrian`, `german`, `naustrian`, `ngerman`, `nswissgerman`, and `swissgerman`),
- Italian (`italian`),
- Norwegian (`norsk`),
- Polish (`polish`),
- Spanish (`spanish`),
- Swedish (`swedish`).

v3.13

v3.13

v3.13

v3.38B

v3.13

If you want to configure it for other languages, you may use this command. The first argument is the name of the language and the other arguments are the names of the corresponding days.

In the current implementation it does not matter whether you load `scrdate` before or after `german`, `ngerman`, `babel` or similar packages. In both cases the correct language will be used.

To explain a little bit more exactly: while you are using a language selection which works in a compatible way to `babel` or `ngerman`, the correct language will be used by `scrdate`. If you are using another language selection you will get (US-)English names.

By the way: If you found a suitable definition for a language, that has not been defined before, please mail it to the KOMA-Script author. There is a good chance, that he will add support for the language to `scrdate`.

Getting the Time with Package `scrtime`

This package provides an answer to the question of the current time. Since version 3.05 this package uses the option interface already known from the KOMA-Script classes and several other KOMA-Script packages. See for example [section 2.4](#) for more information.

```
\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 11:59.
```

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 59 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=simple-switch`

v3.05a

With option `12h` one can select whether the result of `\thistime` and `\thistime*` is in 12- or in 24-hour format. The option understands the values for simple-switch listed in [table 2.5, page 39](#). The option without a value is same like `12h=true` and therefore 12-hour-format will be used. The default is `24h`.

You may use this option either as a global option in the optional argument of `\documentclass`, as a package option in the optional argument of `\usepackage` or after loading the package using `\KOMAOPTIONS` or `\KOMAOPTION` (see, e.g., [section 2.4, page 30](#)). The option has no effect on the results of `\thistime` and `\thistime*` if `\settime` is used.

Only for compatibility with former releases of `scrttime` also option `24h` will switch to 24-hour format if used in the optional argument of `\documentclass` or `\usepackage`. Nevertheless, you should not use this option any longer.

Access to Address Files with scraddr

8.1. 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 [chapter 4](#) from [page 220](#).

```
\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 4.22](#) from [page 220](#) 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 `\adrenentry` entry and parameter 9 for the `\addrenentry` entry, is the identifier of an entry, thus the *key* has to be unique and non-blank. The *key* should only be composed of multiple uppercase letters out of the namespace of T_EX macro names.

If the file contains more than one entry with the same *key* value, the last occurrence will be used.

8.2. 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.

```

\addrenentry{Baggins}{Frodo}%
    {The Hill\\ Bag End/Hobbiton in the Shire}{}%
    {Bilbo Baggins}{pipe-weed}%
    {the Ring-bearer}{Bilbo's heir}{FRODO}
\adrenentry{Gamgee}{Samwise}%
    {Bagshot Row 3\\Hobbiton in the Shire}{}%
    {Rosie Cotton}{taters}%
    {the Ring-bearer's faithful servant}{SAM}
\adrenentry{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{'O spring-time and summer-time
        and spring again after!\\
        O wind on the waterfall,
        and the leaves' laughter!''}
\end{letter}
```

In the address of letters often both firstname and lastname are required, als shown above in **\opening**. Thus, the command **\Name{key}** is an abridgement for **\FirstName{key}** **\LastName{key}**.

The 5th and 6th parameters of the **\adrentry** or **\adrentry** 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 **\addrentry** entries; for **\adrentry** entries it results in an error. More on this is covered in the next section.

8.3. Package Warning Options

As mentioned above, the command **\FreeIV** leads to an error if it is used for **\adrentry** 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 L^AT_EX 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 L^AT_EX 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 an 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 L^AT_EX 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 `adrconv`, with a single *d*, entirely replaces `addrconv`. If it is not included in your T_EX distribution then it can be downloaded from [Kie10] and you can install it separately.

Making Basic Feature of the KOMA-Script Classes Available with Package `scrextend` while Using Other Classes

There are several features, that are shared by all KOMA-Script classes. This means not only the classes `scrbook`, `scrreprt`, and `scrartcl`, that has been made as a drop-in replacement for the standard classes `book`, `report`, and `article`, but also for several features of the KOMA-Script class `scrletter2`, the successor of `scrletter`, that may be used for letters. These basic features, that may be found in the above-named classes, are also provided by package `scrextend` since KOMA-Script release 3.00. This package should not be used together with a KOMA-Script class, but may be used together with many other classes. Package `scrextend` would recognize, if it would be used with a KOMA-Script class, and would terminate with a warning message in that case.

There is no warranty for compatibility of `scrextend` with every class. The package has been designed primary to extend the standard classes and derived classed. Anyway, before using `scrextend` you should make sure that the used class does not already provide the feature you need.

Beside the features from this chapter, there are additional common features, that are mainly provides for authors of classes and packages. These may be found in [chapter 12](#) from [page 308](#). The package `scrbase`, that has been described at that chapter, was designed to be used mainly by authors of classes and packages. Package `scrextend` and all KOMA-Script classes also use that package.

KOMA-Script classes and package `scrextend` also load package `scrfile` described in [chapter 13](#) from [page 331](#). Because of this the features of that package are also available when using `scrextend`.

In difference to the above, only the KOMA-Script classes `scrbook`, `scrreprt`, and `scrartcl` load package `tocbasic` (see [chapter 15](#) from [page 346](#)), that has been designed to be used by authors of classes and packages too. Because of this `scrextend` does not provide the features of this package. Nevertheless you may use `tocbasic` together with `scrextend`.

10.1. Early or late Selection of Options

All of what is described in [section 2.4](#) is generally applicable. So if you have already read and understood [section 2.4](#) you can switch to [section 10.2](#), [page 266](#).

In this section a peculiarity of KOMA-Script is presented, which, apart from the classes and the `scrextend` package is also relevant to several other KOMA-Script packages. Such that the user can find all information corresponding to a single package or a single class in the relevant chapter, this section is found almost identically in several chapters. Users who are not only interested in a particular package or class, but wish to gain an overview of KOMA-Script as a whole, may read the section in one chapter and may thereafter skip it wherever coming across

it in the document.

```
\documentclass[option list]{KOMA-Script class}
\usepackage[option list]{package list}
```

In L^AT_EX, provision is made for the user to pass class options as a comma-separated list of keywords as optional arguments to `\documentclass`. Apart from being passed to the class, these options are also passed on to all packages which can understand the options. Provision is also made for the user to pass optional arguments as a comma-separated list of keywords as optional arguments to `\usepackage`. KOMA-Script expands the option mechanism for the KOMA-Script classes and various packages to use further possibilities. Thus, most KOMA-Script options can also take a value. An option may have not only the form *Option*, but may also have the form *option=value*. Apart from this difference `\documentclass` and `\usepackage` function the same in KOMA-Script as described in [Tea05b] or any introduction to L^AT_EX, for example [OPHS11].

You should note, that in opposite to the interface described below the options interface of `\documentclass` and `\usepackage` is not robust. So commands, lengths, counters and such constructs may break inside the optional argument of these commands. Because of this, the usage of a L^AT_EX length inside the value of an option would cause an error before KOMA-Script can get the control over the option execution. So, if you want to use a L^AT_EX length, counter or command a part of the value of an option, you have to use `\KOMAOPTIONS` or `\KOMAOPTION`. These commands will be described next.

```
\KOMAOPTIONS{option list}
\KOMAOPTION{option}{value list}
```

KOMA-Script offers most class and package options the opportunity to change the value of options even after loading of the class or package. One may then change the values of a list of options at will with the `\KOMAOPTIONS` command. Each option in the *option list* has the form *option=value*.

Some options also have a default value. If one does not give a value, i. e., gives the option simply in the form *option*, then the default value will be used.

Some options can assume several values simultaneously. For such options there exists the possibility, with the help of `\KOMAOPTION`, to pass a single *option* a list of values. The individual values are given as a comma-separated *value list*.

To implement this possibility KOMA-Script uses the commands `\FamilyOptions` and `\FamilyOption` with the family “KOMA”. For more information in these commands see [part II, section 12.2, page 313](#).

v3.00

v3.00

10.2. Compatibility with Earlier Versions of KOMA-Script

It applies, *mutatis mutandis*, what is written in [section 2.5](#). So if you have already read and understood [section 2.5](#) you can switch to [page 266](#), [page 266](#).

Those who achieve their documents in source code set utmost value to the fact that future L^AT_EX runs will yield exactly the same result. In some cases improvement and bug corrections of packages will result in changes of the behaviour and make-up. But sometimes this is not wanted.

```
version=value
version=first
version=last
```

v3.20a

At scrextend it's your choice if your source code should result in the same make-up at future L^AT_EX runs or if you like to participate in all improvements of new releases. You may select the compatible version of KOMA-Script with option `version`. Compatibility to the lowest supported KOMA-Script release may be achieved by `version=first` or `version=2.9` or `version=2.9t`. Setting *value* to an unknown release number will result in a warning message and selects `version=first` for safety.

With `version=last` the most recent version will be selected at every L^AT_EX run. Be warned, though, that using `version=last` poses possibilities of compatibility issues for future L^AT_EX runs. Option `version` without any *value* means the same. This is the default behaviour as long as you do not use any deprecated options.

v3.01a

If you use a deprecated option of KOMA-Script 2, KOMA-Script 3 will switch to `version=first` automatically. This will also result in a warning message that explains how to prevent this switching. Alternatively you may select another adjustment using option `version` with the wanted compatibility after the deprecated option.

Compatibility is primarily make-up compatibility. New features not related to the mark-up will be available even if you switch compatibility to a version before first implementation of the feature. Option `version` does not influence make-up changes that are motivated by bug fixes. If you need bug compatibility you should physically save the used KOMA-Script version together with your document.

Please note that you cannot change option `version` anymore after loading the package scrextend. Therefore, the usage of option `version` within the argument of `\KOMAOPTIONS` or `\KOMAOPTION` is not recommended and will cause an error.

10.3. Optional, Extended Features

Package scrextend provides some optional, extended features. Such features are not available by default, but may be activated additionally. These features are optional, i.e., because they conflict with features of the standard classes of often used packages.

Table 10.1.: overview of the optional available extended features of scrextend

<i>title</i>	extends the title pages to the features of the KOMA-Script classes; this means not only the commands for the title page but also option titlepage (see section 10.7 , from page 270)
--------------	--

`extendedfeature=feature`

With this option an extended *feature* of scrextend may be activated. Option `extendedfeature` is available only while loading the package scrextend. User have to set the option in the optional argument of `\usepackage[optional argument]{scrextend}`. An overview of all available optional features is shown in [table 10.1](#).

10.4. Draft Mode

What is written in [section 3.3](#) applies, mutatis mutandis. So if you have already read and understood [section 3.3](#) you can jump to [section 10.5](#) on [page 267](#).

Many classes and packages provide a draft mode aside from the final typesetting mode. The difference of draft and final mode may be as manifold as the classes and package that support these modes. For instance, the `graphics` and the `graphicx` packages do not actually output the graphics in their own draft mode. Instead they output a framed box of the appropriate size containing only the graphic's file name (see [[Car05](#)]).

`draft=simple switch`

v3.00

This option is normally used to distinguish between the draft and final versions of a document. *simple switch* value may be any standard value from [table 2.5](#), [page 39](#). In particular, switching on the option 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 default `draft=false` option no such boxes are shown. Such overly long lines often vanish using package microtype [[Sch13](#)].

10.5. Selection of the Document Font Size

What is described in [section 3.5](#) applies, mutatis mutandis. So if you have already read and understood [section 3.5](#) you can jump to the end of this section on [page 268](#).

The main document font size is one of the basic decisions for the document layout. The maximum width of the text area, and therefore splitting the page into text area and margins, depends on the font size as stated in [chapter 2](#). The main document font will be used for most

of the text. All font variations either in mode, weight, declination, or size should relate to the main document font.

```
fontsize=size
```

In contrast to the standard classes and most other classes that provide only a very limited number of font sizes, the KOMA-Script classes offer the feature of selection of any desired *size* for the main document font. In this context, any well known T_EX unit of measure may be used and using a number without unit of measure means pt.

If you use this option inside the document, the main document font size and all dependent sizes will change from this point. This may be useful, e. g., if the appendix should be set using smaller fonts on the whole. It should be noted that changing the main font size does not result in an automatic recalculation of type area and margins (see `\recalcctypearea`, [section 2.4, page 37](#)). On the other hand, each recalculation of type area and margins will be done on the basis of the current main font size. The effects of changing the main font size to other additionally loaded packages or the used document class depend on those packages and the class. This may even result in error messages or typesetting errors, which cannot be considered a fault of KOMA-Script, and even the KOMA-Script classes do not change all lengths if the main font size changes after loading the class.

This option is not intended to be a substitution for `\fontsize` (see [\[Tea05a\]](#)). Also, you should not use it instead of one of the main font depending font size commands `\tiny` up to `\Huge`!

10.6. Text Markup

What is described in [section 3.6](#) applies, mutatis mutandis. So if you have already read and understood [section 3.7](#) you can switch to [page 270](#).

L^AT_EX offers different possibilities for logical and direct markup of text. Selection of the font family commands, as well as choosing the font size and width is supported. More information about the standard font facilities may be found at [\[OPHS11\]](#), [\[Tea05b\]](#), and [\[Tea05a\]](#).

```
\textsuperscript{Text}  
\textsubscript{Text}
```

The L^AT_EX-Kern already defines the command `\textsuperscript` to superscript text. Unfortunately, until release 2015/01/01 L^AT_EX itself does not offer a command to produce text in subscript instead of superscript. KOMA-Script defines `\textsubscript` for this purpose. You may find an example of usage at [section 3.6, page 56](#).

```
\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`, and `\scshape`, as well as the font size commands `\Huge`, `\huge`, `\LARGE`, `\Large`, `\large`, `\normalsize`, `\small`, `\footnotesize`, `\scriptsize`, and `\tiny`. The description of these commands can be found in [OPHS11], [Tea05b], or [Tea05a]. Color switching commands like `\normalcolor` (see [Car05] and [Ker07]) are also acceptable. The behavior when using other commands, especially those that make redefinitions 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 inside the document body, but only in the document preamble. Usage examples can be found in the paragraphs on the corresponding element. Names and meanings of the individual items are listed in [table 3.2, page 58](#). However only the listed elements for the document title, dicta, footnotes, and the `labeling` environment are supported. Though element `disposition` exists, it will also be used for the document title only. This has been done for compatibility with the KOMA-Script classes. The default values are shown in the corresponding paragraphs.

With command `\usekomafont` the current font style may be changed into the font style of the selected *element*.

Example: Assumed, you want to print the document title in a serif font and with red colour. You may do this using:

```
\setkomafont{title}{\color{red}}
```

Package `color` or `xcolor` will be needed for command `\color{red}`. The additional usage of `\normalfont` is not needed in this case, because it is already part of the definition of the title itself. This example also needs option `extendedfeature=title` (see [section 10.3, page 267](#)).

```

\usefontofkomafont{element}
\useencodingofkomafont{element}
\usesizeofkomafont{element}
\usefamilyofkomafont{element}
\useseriesofkomafont{element}
\useshapeofkomafont{element}

```

v3.12

Sometimes and despite the recommendation users use the font setting feature of elements not only for font settings but for other settings too. In this case it may be useful to switch only to the font setting of an element but not to those other settings. You may use `\usefontofkomafont` in such cases. This will activate the font size and baseline skip, the font encoding, the font family, the font series, and the font shape of an element, but no further settings as long as those further settings are local.

You may also switch to one of those attributes only using one of the other commands. Note, that `\usesizeofkomafont` will activate both, the font size and the baseline skip.

You should not misunderstand these commands as a legitimization of using all kind of commands at the font setting of an element. Hence this would result in errors sooner or later (see [section 21.3, page 444](#)).

10.7. Document Titles

What is written in [section 3.7](#) applies, *mutatis mutandis*. So if you have already read and understood [section 3.7](#) you can jump to [section 10.8](#) on [page 274](#). But there's a difference: The document title capabilities of `scrextend` are part of the optional, advanced features. Therefore they are only available, if `extendedfeature=title` has been selected while loading the package (see [section 10.3, extendedfeature](#)).

Beyond that `scrextend` cannot be used with a KOMA-Script class together. Because of this

```
\documentclass{scrbook}
```

must be replaced by

```

\documentclass{book}
\usepackage[extendedfeature=title]{scrextend}

```

at all examples from [section 3.7](#), if `scrextend` should be used.

In general we distinguish two kinds of document titles. First known are title pages. In this case the document title will be placed together with additional document information, like the author, on a page of its own. Besides the main title page, several further title pages may exist, like the half-title or bastard title, publisher data, dedication, or similar. The second known kind of document title is the in-page title. In this case, the document title is placed at the top of a page and specially emphasized, and may be accompanied by some additional information too, but it will be followed by more material in the same page, for instance by an abstract, or the table of contents, or even a section.

```
titlepage=simple switch
titlepage=firstiscover
```

Using `\maketitle` (see [page 271](#)), this option switches between document title pages and in-page title. For *simple switch*, any value from [table 2.5, page 39](#) may be used.

The option `titlepage=true` makes L^AT_EX use separate pages for the titles. Command `\maketitle` sets these pages inside a `titlepage` environment and the pages normally have neither header nor footer. In comparison with standard L^AT_EX, KOMA-Script expands the handling of the titles significantly.

The option `titlepage=false` 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.

v3.12

The third choice, `titlepage=firstiscover` does not only select title pages. It additionally prints the first title page of `\maketitle`, this is either the extra title or the main title, as a cover page. Every other setting of option `titlepage` would cancel this setting. The margins of the cover page are given by `\coverpagetopmargin`, `\coverpagebottommargin`, `\coverpageleftmargin` und `\coverpagerightmargin`. The defaults of these depend on the lengths `\topmargin` and `\evensidemargin` and can be changed using `\renewcommand`.

The default depends on the used class and `scrextend` handles it compatible to the standard class. If a class does not set up a comparable default, in-page title will be used.

```
\begin{titlepage}...\end{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—and in single column mode. 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 pages provided by `\maketitle`, that will be described next; it is advisable to design a new one with the help of this environment.

A simple example for a title page with `titlepage` is shown in [section 3.7 on page 63](#)

```
\maketitle[page number]
```

While the the standard classes produce at least one 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 count 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 3.12, page 79](#).

The following commands do not lead immediately to the ship-out of the titles. The typesetting and ship-out of the title pages are always done by `\maketitle`. By the way, you should note that `\maketitle` should not be used inside a `\titlepage` environment. Like shown in the examples, one should use either `\maketitle` or `\titlepage` only, but not both.

The commands explained below only define the contents of the title pages. Because of this, they have to be used before `\maketitle`. It is, however, not necessary and, when using, e.g., the `babel` package, not recommended to use these in the preamble before `\begin{document}` (see [\[BB13\]](#)). 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`.

A simple example for a title page with extra title and main title is shown in [section 3.7 on page 64](#)

```
\titlehead{title head}
\subject{subject}
\title{title}
\subtitle{subtitle}
\author{author}
\date{date}
\publishers{publisher}
\and
\thanks{footnote}
```

The contents of the full title page are defined by seven elements. The output of the full title page occurs as part of the title pages of `\maketitle`, whereas the now listed elements only define the corresponding elements.

The *title head* is defined with the command `\titlehead`. It is typeset with the font of the homonymous element in regular justification and full width at the top of the page. It can be freely designed by the user.

The *subject* is output with the font of the homonymous element immediately above the *title*.

The *title* is output with a very large font size. Beside all other element the font size is, however, not affected by the font switching element `\title` (see [table 3.4, page 66](#)).

The *subtitle* is output with the font of the homonymous element just below the title.

Below the *subtitle* appears the *author*. Several authors can be specified in the argument of `\author`. They should be separated by `\and`. The font of element *author* is used for the output of the authors.

Below the author or authors appears the date in the font of the homonymous element. The default value is the present date, as produced by `\today`. The `\date` command accepts arbitrary information—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 instead of centering it. Then it is to be considered equivalent to the title head. However, note that this field is put above any existing footnotes. The font of element *publishers* is used for the output.

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. Note, that `\thanks` has to be used inside the argument of another command, e.g., at the argument *author* of the command `\author`.

v3.12

While printing the title elements the equal named font switching elements will be used for all them. The defaults, that may be found in [table 3.3, page 66](#), may be changed using the commands `\setkomafont` and `\addtokomafont` (see [section 10.6, page 269](#)).

With the exception of *titlehead* and possible footnotes, all the items are centered horizontally. The information is summarised in [table 3.4, page 66](#). Please note, that for the main title `\huge` will be used after the font switching element `\title`. So you cannot change the size of the main title using `\setkomafont` or `\addtokomafont`.

An example for a title page with all elements provided by KOMA-Script for the main title page is shown in [section 3.7 on page 66](#).

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.

Nevertheless, since KOMA-Script 3.12 the first title page of `\maketitle` can be printed as a cover page with different margins. For more information about this see the description of option `titlepage=firstiscover` on page 271.

```
\uppertitleback{titlebackhead}
\lowertitleback{titlebackfoot}
```

With the standard classes, the back of the title page of a double-side print 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 given by the font of the homonymous element. The font can be changed using command `\setkomafont` or `\addtokomafont` (see section 10.6, page 268). 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.

An example with all title pages provided by KOMA-Script is shown in section 3.7 on page 68.

10.8. Detection of Odd and Even Pages

What is described in section 3.11 applies, *mutatis mutandis*. So if you have already read and understood section 3.11 you can switch to page 275, page 275.

In double-sided documents we distinguish left and right pages. Left pages always have an even page number, right pages always have an odd page number. Thus, they are most often referred to as even and odd pages in this guide.

```
\ifthispageodd{true part}{false part}
```

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 part* argument is executed only if the command falls on an odd page. Otherwise the *false part* argument is executed.

Example: Assume that you simply want to show whether a text will be placed onto an even or odd page. You may achieve that using

```
This page has an \ifthispageodd{odd}{even}
page number.
```

which will result in the output

This page has 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.

At [section 21.1](#), [page 441](#) experts may find more information about the problems detecting left and right pages or even and odd page number.

10.9. Head and Foot Using Predefined Page Styles

One of the basic features of a document is the page style. Page style in \LaTeX means mainly header and footer of the page. Package `scrextend` does not define any page style, but it uses and expects the definition some page styles.

`\titlepagestyle`

Some pages have a different page style automatically selected using `\thispagestyle`. With `scrextend` this will be used currently for the page with the in-page title if and only if option `extendedfeature=title` has been used (see [section 10.3](#), [page 267](#)). In this case the page style stored at `\thispagestyle` will be used. Default for `\thispagestyle` is `plain`. This page style is predefined by the \LaTeX kernel. So it should be available always.

10.10. Interleaf Pages

What is described in [section 3.13](#) applies, *mutatis mutandis*. So if you have already read and understood [section 3.13](#) you can switch to [section 10.11](#), [page 278](#).

Interleaf pages are pages that are intended to stay blank. Originally these pages were really completely white. \LaTeX , on the other hand, by default sets those pages with the current valid page style. So those pages may have a head and a pagination. KOMA-Script provides several extensions to this.

Interleaf pages may be found in books mostly. Because chapters in books commonly start on odd pages, sometimes a left page without contents has to be added before. This is also the reason that interleaf pages only exist in double-sided printing. The unused back sides of the one-sided printings are not interleaf pages, really, although they may seem to be such pages.

```
cleardoublepage=page style
cleardoublepage=current
```

With this option, you may define the page style of the interleaved pages created by the `\cleardoublepage`, `\cleardoubleoddpaper`, or `\cleardoubleevenpage` to break until the wanted page. Every already defined *page style* (see [section 10.9](#) from [page 275](#) and [chapter 5](#) from [page 225](#)) may be used. Besides this, `cleardoublepage=current` is valid. This case is the default until KOMA-Script 2.98c and results in interleaved page without changing the page style. Since KOMA-Script 3.00 the default follows the recommendation of most typographers and has been changed to blank interleaved pages with page style `empty` unless you switch compatibility to an earlier version (see option `version`, [section 10.2](#), [page 266](#)).

Example: Assume you want interleaved pages almost empty but with pagination. This means you want to use page style `plain`. You may use following to achieve this:

```
\KOMAoptions{cleardoublepage=plain}
```

More information about page style `plain` may be found at [section 3.12](#), [page 78](#).

```
\clearpage
\cleardoublepage
\cleardoublepageusingstyle{page style}
\cleardoubleemptypage
\cleardoubleplainpage
\cleardoublestandardpage
\cleardoubleoddpaper
\cleardoubleoddpaperusingstyle{page style}
\cleardoubleoddpaperemptypage
\cleardoubleoddpaperplainpage
\cleardoubleoddpaperstandardpage
\cleardoubleevenpage
\cleardoubleevenpageusingstyle{page style}
\cleardoubleevenemptypage
\cleardoubleevenplainpage
\cleardoubleevenstandardpage
```

The L^AT_EX kernel 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 2.4](#), [page 38](#)) starts a new right-hand page. An empty left page in the current page style is output if necessary.

With `\cleardoubleoddpaperstandardpage`, KOMA-Script works as described above. The `\cleardoubleoddpaperplainpage` command changes the page style of the empty left page to `plain`.

in order to suppress the running head. Analogously, the page style `empty` is applied to the empty page with `\cleardoubleoddeemptypage`, suppressing the page number as well as the running head. The page is thus entirely empty. If another *page style* is wanted for the interleaved page it may be set with the argument of `\cleardoubleoddusingpagestyle`. Every already defined *page style* (see [chapter 5](#)) may be used.

Sometimes chapters should not start on the right-hand page but the left-hand page. This is in contradiction to the classic typography; nevertheless, it may be suitable, e.g., if the double-page spread of the chapter start is of special contents. KOMA-Script therefore provides the commands `\cleardoubleevenstandardpage`, `\cleardoubleevenplainpage`, `\cleardoubleevenemptypage`, and `\cleardoubleevenpageusingstyle`, which are equivalent to the odd-page commands.

However, the approach used by the KOMA-Script commands `\cleardoublestandardpage`, `\cleardoubleemptypage`, `\cleardoubleplainpage`, and `\cleardoublepageusingstyle` is dependent on the option `cleardoublepage` described above and is similar to one of the corresponding commands above. The same is valid for the standard command `\cleardoublepage`, that may be either `\cleardoubleoddpager` or `\cleardoubleevenpage`.

Example: Assume you want to set next in your document a double-page spread with a picture at the left-hand page and a chapter start at the right-hand page. The picture should have the same size as the text area without any head line or pagination. If the last chapter ends with a left-hand page, an interleaved page has to be added, which should be completely empty.

First you will use

```
\KOMAoptions{cleardoublepage=empty}
```

to make interleaved pages empty. You may use this setting at the document preamble already. As an alternative you may set it as the optional argument of `\documentclass`.

At the relevant place in your document, you'll write:

```
\cleardoubleevenemptypage
\thispagestyle{empty}
\includegraphics[width=\textwidth,%
                  height=\textheight,%
                  keepaspectratio]%
                  {picture}
\chapter{Chapter Headline}
```

The first of these lines switches to the next left page. If needed it also adds a completely blank right-hand page. The second line makes sure that the following left-hand page will be set using page style `empty` too. From third down to sixth line, an external picture of wanted size will be loaded without deformation. Package

`graphicx` will be needed for this command. The last line starts a new chapter on the next page which will be a right-hand one.

The commands `\cleardoubleoddpage` respective `\cleardoubleevenpage` leads to the next odd respectively even page. The page style of an interleaved page will be set depending on option `cleardoublepage`.

10.11. Footnotes

All of what is described in [section 3.14](#) is generally applicable. So if you have already read and understood [section 3.14](#) you can switch to [section 10.12](#), [page 282](#).

Package `scrextend` supports all the footnote features of the KOMA-Script classes. Nevertheless, by default the footnotes are under full control of the used class. This changes as soon as command `\deffootnote` (see [page 280](#)) has been used.

Package `scrextend` does not provide settings for the separation line above the footnotes.

`footnotes=setting`

At several classes footnotes will be marked with a tiny superscript number in text by default. If more than one footnote falls at the same place, one may think that it is only one footnote with a very large number instead of multiple footnotes (i. e., footnote 12 instead of footnotes 1 and 2). Using `footnotes=multiple` will separate multiple footnotes immediately next to each other by a separator string. The predefined separator at `\multfootsep` is a single comma without space. The whole mechanism is compatible with package `footmisc`, Version 5.3d (see [\[Fail1\]](#)). It is related not only to footnotes placed using `\footnote`, but `\footnotemark` too.

Command `\KOMAOPTIONS` or `\KOMAoption` may be used to switch back to the default `footnotes=nomultiple` at any time. If any problems using another package that influences footnotes occur, it is recommended not to use the option anywhere and not to change the *setting* anywhere inside the document.

A summary of the available *setting* values of footnotes may be found at [table 3.11](#), [page 85](#).

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

Similar to the standard classes, footnotes in KOMA-Script are produced with the `\footnote` command, or alternatively the paired usage 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 L^AT_EX no choice but to break the footnote onto the next page. KOMA-Script, unlike the standard classes, can recognize and separate consecutive footnotes automatically. See the previously documented option `footnotes` for this.

If you want to set the separator manually, you may use `\multiplefootnoteseparator`. Note that this command should not be redefined, because it has been defined not only to be the separator string but also the type style, i. e., font size and superscript. The separator string without type style may be found at `\multfootsep`. The predefined default is

```
\newcommand*{\multfootsep}{,}
```

and may be changed by redefining the command.

Example: Assume you want to place two footnotes following a single word. First you may try

```
Word\footnote{1st footnote}\footnote{2nd footnote}
```

for this. Assume that the footnotes will be numbered with 1 and 2. Now the reader may think it's a single footnote 12, because the 2 immediately follows the 1. You may change this using

```
\KOMAoptions{footnotes=multiple}
```

which would switch on the automatic recognition of footnote sequences. As an alternative you may use

```
Word\footnote{1st footnote}%
\multiplefootnoteseparator
\footnote{2nd footnote}
```

This should give you the wanted result even if the automatic solution would fail or could not be used.

Further, assume you want the footnotes separated not only by a single comma, but by a comma and a white space. In this case you may redefine

```
\renewcommand*{\multfootsep}{,\nobreakspace}
```

at the document preamble. `\nobreakspace` instead of a usual space character has been used in this case to avoid paragraph or at least page breaks within footnote sequences.

`\footref{reference}`

Sometimes there are single footnotes to multiple text passages. The least sensible way to typeset this would be to repeatedly use `\footnotemark` with the same manually set number. The disadvantages of this method would be that you have to know the number and manually fix all the `\footnotemark` commands, and if the number changes because of adding or removing a footnote before, each `\footnotemark` would have to be changed. Because of this, KOMA-Script provides the use of the `\label` mechanism in such cases. After simply setting a `\label` inside the footnote, `\footref` may be used to mark all the other text passages with the same footnote mark.

Example: Maybe you have to mark each trade name with a footnote which states that it is a registered trade name. You may write, e.g.,

```
Company SplishSplash\footnote{This is a registered trade name.
  All rights are reserved.\label{refnote}}
produces not only SplishPlump\footref{refnote}
but also SplishSplash\footref{refnote}.
```

This will produce the same footnote mark three times, but only one footnote text. The first footnote mark is produced by `\footnote` itself, and the following two footnote marks are produced by the additional `\footref` commands. The footnote text will be produced by `\footnote`.

Because of setting the additional footnote marks using the `\label` mechanism, changes of the footnote numbers will need at least two L^AT_EX runs to ensure correct numbers for all `\footref` marks.

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

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. The default configuration of the KOMA-Script classes is:

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

`\textsuperscript` controls both the superscript and the smaller font size. Command `\thefootnotemark` is the current footnote mark without any formatting. Package `scrextend` in contrast to this does not change the default footnote settings of the used class. Loading the package does not change any type style of footnote marks or footnote text in general. You have to copy the above shown source to use the default settings of the KOMA-Script classes with `scrextend`. This may be done immediately after loading package `scrextend`.

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 `\setkomafont` and `\addtokomafont` (see section 10.6, page 269). Please refer also to table 3.2, page 58. Default setting is no change in the font. With `scrextend` the elements may change the fonts only if the footnotes are handled by the package, i. g., after using `\deffootnote`.

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}}
```

In the above the font for the element `footnotereference` is applied (see table 3.2, page 58). Thus the footnote marks in the text and in the footnote itself are identical. The font can be changed with the commands `\setkomafont` and `\addtokomafont` (see section 10.6, page 269) after usage of `\deffootnotemark`.

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 `\setkomafont` and `\addtokomafont` (see [section 10.6](#), [page 269](#)):

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

```
\setfootnoterule[thickness]{length}
```

v3.06

Generally a horizontal rule will be placed between the text area and the footnote area. But normally this rule is not as long as the width of the typing area. With Command `\setfootnoterule` you may change the thickness and the width of that rule. Thereby the parameters *thickness* and *length* will be evaluated not at definition time but when setting the rule itself. If optional argument *thickness* has been omitted the thickness of the rule will not be changed. Empty arguments *thickness* or *length* are also allowed and do not change the corresponding parameter. Using implausible values may result in warning messages not only setting the arguments but also when KOMA-Script uses the parameters.

v3.07

With element `footnoterule` the color of the rule may be changed using the commands `\setkomafont` and `\addtokomafont` for element `footnoterule` (see [section 10.6](#), [page 269](#)). Default is no change of font or color. For color changes a color package like `xcolor` would be needed.

10.12. Dicta

What is written in [section 3.17](#) applies, mutatis mutandis. However, `scrextend` does not support the commands `\setchapterpreamble` and `\setpartpreamble`. You should read the manual of the used class, if you want to know, if that class does support similar commands. So if you have already read and understood [section 3.17](#) you can jump to [section 10.13](#) on [page 283](#).

Sometimes you may find a dictum, a kind of smart slogan or excerpt, often ragged left above or below the heading of a chapter or section. The text and the source of the slogan often use special styles.

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

The command `\dictum` inserts such a dictum. The dictum together with an optional *author* is inserted in a `\parbox` (see [Tea05b]) 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 `\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 *dictum*, which can be changed with the commands `\setkomafont` and `\addtokomafont` (see section 10.6, page 269). Default settings are listed in table 3.16, page 109.

If there is an *author* name, it is separated from the *dictum* by a rule to the full width of the `\parbox`. This rule is defined as vertical object to command `\dictumrule`:

```
\newcommand*{\dictumrule}{\vskip-1ex\hrulefill\par}
```

The alignment is defined with `\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 parentheses. For the element *dictumauthor*, a different font than for the element *dictum* can be defined. Default settings are listed in table 3.16. Changes can be made using the commands `\setkomafont` and `\addtokomafont` (see section 10.6, page 269).

10.13. Lists

All of what is described in section 3.18 is generally applicable. So if you have already read and understood section 3.18 you can switch to section 10.14, page 285. However, `scrextend` does support only the environments `labeling`, `addmargin` and `addmargin*`. All the other list environments may be supported and controlled by the used class.

Both L^AT_EX and the standard classes offer different environments for lists. Though slightly changed or extended all these lists 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.

Because lists are standard elements of L^AT_EX this section abandons on examples. Nevertheless, you may find examples either in [section 3.18](#) from [page 111](#) or in almost every introduction to L^AT_EX.

```
\begin{labeling}[delimiter]{widest pattern}
  \item[keyword]...
  :
\end{labeling}
```

An additional form of a description list is only available in the KOMA-Script classes and the package scrextend: the `labeling` environment. Unlike the `description` environment, 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. The font which is responsible for emphasizing the item and the separator can be changed with the commands `\setkomafont` and `\addtokomafont` (see [section 10.6](#), [page 269](#)) for the element `labelinglabel` and `labelingseparator` (see [table 3.2](#), [page 58](#)).

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.

```
\begin{addmargin}[left indentation]{indentation}...\end{addmargin}
\begin{addmargin*}[inner indentation]{indentation}...\end{addmargin*}
```

Similar to `quote` and `quotation` which are available at the standard classes and also the KOMA-Script classes 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.

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` (section 10.8, page 274) and `\ifthispagewasodd` (section 21.1, page 441).

There may be several questions about coexistence of lists and paragraphs. Because of this additional information may be found at the description of option `parskip` in section 21.1, page 441. Also at the expert part, in section 21.1, page 441, you may find additional information about page breaks inside of `addmargin*`.

10.14. Margin Notes

All of what is described in section 3.21 is generally applicable. So if you have already read and understood section 3.21 you can switch to chapter 11, page 286.

Aside from the text area, that normally fills the typing area, usually a marginalia column may be found. Margin notes will be printed at this area. A lot of them may be found in this manual.

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

Usually margin notes in L^AT_EX 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`.

An example for this may be found in section 3.21 at page 136.

Experts and advanced users may find information about problems using `\marginpar` at section 21.1, page 441. These are valid for `\marginline` also.

Support for the Law Office by scrjura

In case you'd like to write a contract, the bylaws of a company or of the club, an act of law or a whole commentary, the package `scrjura` will provide typographical support. Despite the fact that `scrjura` is intended as a broad help for juridical documents, the contract is the central element of the package. Particular attention is being paid to the clause with numbered title and numbered paragraphs — if a clause consists of more than one paragraph —, even numbered sentences, entries in the table of contents and cross references according to German standards.

The package has been developed in cooperation with Dr Alexander Willand, lawyer in Karlsruhe.

Note that the package cooperates with `hyperref`. Nevertheless, `hyperref` has to be loaded after `scrjura` as usual.

11.1. Early or late Selection of Options

All of what is described in [section 2.4](#) is generally applicable. So if you have already read and understood [section 2.4](#) you can switch to [section 11.2](#), [page 287](#).

In this section a peculiarity of KOMA-Script is presented, which, apart from the `scrjura` package, is also relevant to other KOMA-Script packages and classes. Such that the user can find all information corresponding to a single package or a single class in the relevant chapter, this section is found almost identically in several chapters. Users who are not only interested in a particular package or class, but wish to gain an overview of KOMA-Script as a whole, may read the section in one chapter and may thereafter skip it wherever coming across it in the document.

```
\documentclass[option list]{KOMA-Script class}
\usepackage[option list]{package list}
```

In L^AT_EX, provision is made for the user to pass class options as a comma-separated list of keywords as optional arguments to `\documentclass`. Apart from being passed to the class, these options are also passed on to all packages which can understand the options. Provision is also made for the user to pass optional arguments as a comma-separated list of keywords as optional arguments to `\usepackage`. KOMA-Script expands the option mechanism for the KOMA-Script classes and various packages to use further possibilities. Thus, most KOMA-Script options can also take a value. An option may have not only the form *Option*, but may also have the form *option=value*. Apart from this difference `\documentclass` and `\usepackage` function the same in KOMA-Script as described in [\[Tea05b\]](#) or any introduction to L^AT_EX, for example [\[OPHS11\]](#).

You should note, that in opposite to the interface described below the options interface of `\documentclass` and `\usepackage` is not robust. So commands, lengths, counters and such

constructs may break inside the optional argument of these commands. Because of this, the usage of a \LaTeX length inside the value of an option would cause an error before KOMA-Script can get the control over the option execution. So, if you want to use a \LaTeX length, counter or command a part of the value of an option, you have to use `\KOMAOPTIONS` or `\KOMAOPTION`. These commands will be described next.

```
\KOMAOPTIONS{option list}
\KOMAOPTION{option}{value list}
```

v3.00

KOMA-Script offers most class and package options the opportunity to change the value of options even after loading of the class or package. One may then change the values of a list of options at will with the `\KOMAOPTIONS` command. Each option in the *option list* has the form *option=value*.

Some options also have a default value. If one does not give a value, i. e., gives the option simply in the form *option*, then the default value will be used.

Some options can assume several values simultaneously. For such options there exists the possibility, with the help of `\KOMAOPTION`, to pass a single *option* a list of values. The individual values are given as a comma-separated *value list*.

To implement this possibility KOMA-Script uses the commands `\FamilyOptions` and `\FamilyOption` with the family “KOMA”. For more information in these commands see [part II, section 12.2, page 313](#).

11.2. Text Markup

What is described in [section 3.6](#) applies, mutatis mutandis. So if you have already read and understood [section 3.7](#) you can switch to [page 308](#).

\LaTeX offers different possibilities for logical and direct markup of text. Selection of the font family commands, as well as choosing the font size and width is supported. More information about the standard font facilities may be found at [\[OPHS11\]](#), [\[Tea05b\]](#), and [\[Tea05a\]](#).

```
\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`, and `\scshape`, as well as the font size commands `\Huge`, `\huge`, `\LARGE`, `\Large`, `\large`, `\normalsize`, `\small`, `\footnotesize`, `\scriptsize`, and `\tiny`. The description of these commands can be found in [\[OPHS11\]](#), [\[Tea05b\]](#), or

[Tea05a]. Color switching commands like `\normalcolor` (see [Car05] and [Ker07]) are also acceptable. The behavior when using other commands, especially those that make redefinitions 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 inside the document body, but only in the document preamble. Usage examples can be found in the paragraphs on the corresponding element.

With command `\usekomafont` the current font style may be changed into the font style of the selected *element*.

```
\usefontofkomafont{element}
\useencodingofkomafont{element}
\usesizeofkomafont{element}
\usefamilyofkomafont{element}
\useseriesofkomafont{element}
\useshapeofkomafont{element}
```

v3.12

Sometimes and despite the recommendation users use the font setting feature of elements not only for font settings but for other settings too. In this case it may be useful to switch only to the font setting of an element but not to those other settings. You may use `\usefontofkomafont` in such cases. This will activate the font size and baseline skip, the font encoding, the font family, the font series, and the font shape of an element, but no further settings as long as those further settings are local.

You may also switch to one of those attributes only using one of the other commands. Note, that `\usesizeofkomafont` will activate both, the font size and the baseline skip.

You should not misunderstand these commands as a legitimization of using all kind of commands at the font setting of an element. Hence this would result in errors sooner or later (see [section 21.3, page 444](#)).

11.3. Table of Contents

The package `scrjura` provides entries into the table of contents.

```
juratotoc=simple switch
juratotoc=level number
```

A clause is being shown in the table of contents only, if its *level number* is smaller or equal to the counter `tocdepth` (see [section 3.9, page 72](#)). Default for the *level number* is 10000, which as well will be used, if the option is switched off by the *simple switch* (see [table 2.5, page 39](#)). Because the counter `tocdepth` usually has a one digit value, clause entries are not shown in the table of contents.

If you switch on the option using the *simple switch*, as a default *level number* 2 is used, so that clauses are shown in the table of contents on the same level as subsections. If the counter `tocdepth` has default values as well, clauses are shown with all KOMA-Script classes.

```
juratocindent=indent
juratocnumberwidth=number width
```

These two options can be used to determine the indentation in the table of contents as well as the reserved space for the numbers there. Defaults are the values for the subsection entries in `scrartcl`.

11.4. Environment for Contracts

The main mechanism of `scrjura` only work inside of the contract environment.

```
\begin{contract}...\end{contract}
```

Till this date, this is the one and only environment for legal practitioner provided by `scrjura`. Using it will activate the automatic numbering of paragraphs and the commands `\Clause` and `\SubClause` will become a form, which will be documented below.

The environment `contract` must not be nested in itself. Within the document the environment may be used several times. In this case the clauses within the environment are treated as if they were within the same environment. Ending the environment means just a break and with beginning a new environment in the same document the former environment is continued. A break inside a clause is not possible.

```
contract
```

The whole document becomes a contract if you use this option while loading the package with `\usepackage` or as a global option with `\documentclass`. The document behaves as if you started the contract environment right after the beginning of the document.

Neither you can use this option with `\KOMAoption` nor with `\KOMAOPTIONS`, so that it is not possible to switch the option off in this way. Please use the `contract` environment directly.

11.4.1. Clauses

With `scrjura` clauses¹ in a legal sense only exist inside of contracts, meaning inside of the environment `contract`.

¹In English, the word “section” also is used in an act of law or in an agreement. To distinguish `\section` of most document classes from the section in `scrjura`, we decided to call the section in the latter simply “clause”.

Table 11.1.: Possible properties for the optional argument of `\Clause` and `\SubClause`

<code>dummy</code>	The heading will not be printed but counted in the automatic numbering.
<code>head=<i>running head</i></code>	If running heads are available, <i>running head</i> is used instead of the clause <i>title</i> .
<code>nohead</code>	The running head stays unchanged.
<code>notocentry</code>	Does not make an entry into the table of contents.
<code>number=<i>number</i></code>	Uses <i>number</i> for the output of the clause number.
<code>preskip=<i>skip</i></code>	Changes the vertical <i>skip</i> before the clause heading.
<code>postskip=<i>skip</i></code>	Changes the vertical <i>skip</i> after the clause heading.
<code>title=<i>title</i></code>	Additional to the clause number a clause <i>title</i> will be printed. This is also used as default for the <i>running head</i> and the <i>entry</i> in the table of contents.
<code>tocentry=<i>entry</i></code>	Independent from the clause <i>title</i> , an <i>entry</i> into the table of contents will be made, if such entries are activated.

`\Clause[options]`
`\SubClause[options]`

These are the most important commands inside of a contract. Without using further *options* the command `\Clause` creates the heading of a clause, which consists only of the sign »§«, followed by its number. In contrast to this the command `\SubClause` creates the heading of a clause with the last number used by `\Clause` and adds a lowercase letter. `\SubClause` mainly is intended for cases where an act or a contract is amended and not only clauses are changed or deleted, but between existing clauses new ones are inserted without changing the numbering.

Both commands except as *options* a list separated by commas of properties. An overview over possible properties is provided by [table 11.1](#). For the most important of them we will go into the details.

A skip of two lines is inserted before the heading and afterwards a skip of one line as a default. Via the options `preskip` and `postskip` these skips can be changed. The new values are not only valid for the current clause but beginning with the actual clause until the end of the current contract environment. It is possible as well to set these keys in advance by writing

```
\setkeys{contract}{preskip=skip,
                    postskip=skip}
```

independently from a clause and outside of a contract environment as well. Also it is possible to set the keys inside of the preamble after loading `scrjura`. But it is not possible to set these options while loading the package or by using `\KOMAOPTIONS` or `\KOMAOPTION`.

The headings use as a default the fonts `\sffamily\bfseries\large`. The fonts can be changed using the element `contract.Clause` with the help of `\setkomafont` and `\addtokomafont` (see [section 3.6, page 287](#)). Inside the `contract` environment instead of `contract.Clause` simply `Clause` may be used.

With the options `title`, `head`, and `tocentry` a clause may get in addition to the number a title. It is recommended to put the value of these options inside brackets. Failing this, e.g., commas may lead to confusion between different options. Empty values for `head` and `tocentry` cause an empty entry. If one would like to avoid an entry, the options `nohead` and `notocentry` have to be used.

Instead of the running numbers the option `number` manually sets the number of a clause. This will have no impact on the numbers of the subsequent clauses. Empty numbers are not possible. Fragile commands inside `number` have to be protected with `\protect`. It is recommended only to use numbers and letters as assignment to `number`.

With the option `dummy` the output of the whole heading of a clause can be suppressed. The automatic numbering will count this clause nonetheless. Thus you can skip a clause in the automatic numbering with

```
\Clause{dummy}
```

in case the clause has been deleted in a later version of a contract.

Note, only the values `true` and `false` may be used in combination with `dummy`. All other values will be ignored, but may throw up an error.

```
\Clauseformat{number}
```

As already mentioned the clauses and subclauses usually are being numbered. The format of the number is done with the help of the command `\Clauseformat`, which expects as one and only argument the number. The default is defined as

```
\newcommand*{\Clauseformat}[1]{\S~#1}
```

and, as you see, it is only the `\S` followed by a non-breaking space and the number. In case of redefinition take care to keep it expandable.

Table 11.2.: Possible values for option `clausemark` for activation of running heads

<code>both</code>	Clauses generate left and right marks for running heads, if the document provides automatic running heads.
<code>false</code> , <code>off</code> , <code>no</code>	Clauses do not generate marks for running heads and therefore do not change running heads.
<code>forceboth</code>	Clauses use <code>\markboth</code> to generate left and right marks for running heads even if the document does not provide automatic running heads for the current page style.
<code>forceright</code>	Clauses use <code>\markright</code> to generate right marks for running heads even if the document does not provide automatic running heads for the current page style.
<code>right</code>	Clauses generate right marks for running heads, if the document provides automatic running heads.

`juratitlepagebreak=simple switch`

Usually a page break inside a heading is prohibited. Some jurists may need page breaks inside of clause headings. Such a break may be allowed using `juratitlepagebreak`. The possible values for `simple switch` are printed in [table 2.5, page 39](#).

`clausemark=value`

Clauses are a kind of inferior structure with an independent numbering they will not have running headlines as a default. Running headlines are possible and may be created with alternative properties. The possible *values* and their meaning are listed in [table 11.2](#).

11.4.2. Paragraphs

Within clauses the paragraphs are being numbered automatically. With this, the paragraphs are a strongly structuring element, similar to `\paragraph` or `\subparagraph` known, e.g., from article classes. Contracts usually use a vertical skip between paragraphs. The package `scrjura` does not provide its own mechanism for this. Instead, it uses the option `parskip` of the KOMA-Script classes (see [section 3.10, page 74](#)).

```
parnumber=value
```

The default numbering of paragraphs is `parnumber=auto` and `parnumber=true`. Once in a while it might be necessary to switch off the automatic numbering. This is done with `parnumber=false`. In this case the numbering of sentences will be reseted.

To realise this way of numbering paragraphs it has been necessary to gear into the paragraph building mechanism of L^AT_EX. In rare cases there is a negative impact, which can be avoided by switching to `parnumber>manual`. On the other hand L^AT_EX itself sometimes undoes the change. In those cases one has to activate it again with `parnumber=auto`.

In a clause which consist of only one paragraph, the paragraph usually has no number. This does only work if there are not two clauses with an identical number in a document. Note that the number of paragraphs in a clause is not available before the end of the clause. Therefore you need a least two L^AT_EX runs to get the correct, automatic paragraph numbering.

```
par
\thepar
\parformat
\parformatseparation
```

For numbering the paragraphs inside a clause we use the counter `par`. The output of `\thepar` will display an arabic number, because the default is `\arabic{par}`. `\parformat` provides the format, which is `\thepar` in rounded brackets. For numbering a paragraph manually, use `\parformat` as well. It makes sense to call `\parformat` with a following `\nobreakspace` or a tilde.

v0.7

The output of `\parformat` is followed by one or more delimiter(s). These are provided by `\parformatseparation`, which currently consists of `\nonbreakspace`, the non-breakable inter word distance.

Package `scrjura` assumes internally, that `\thepar` is an arabic number. Don't redefine!

```
\ellipsispar[number]
\parellipsis
```

v0.7

Sometimes — particularly in comparisons — it is desirable to omit paragraphs, but to mark the omission. Those omitted paragraphs shall be included in the counter of the paragraphs. The package `scrjura` provides the command `\ellipsispar` to do this.

By default `\ellipsispar` omits precisely one paragraph. Using the optional argument multiple paragraphs may be omitted. In any case the output shows just one not numbered paragraph, which only consists of the ellipsis defined by `\parellipsis`. The automatic numbering of paragraphs takes the *number* of omitted paragraphs into account.

Example: Supposed you are writing a »comment« of the German² penal code, but only

²We have decided to translate it into English. But please remember, it is only an example not of existing law but of a technical realisation with `scrjura`.

paragraph 3 of § 2. Nevertheless you'd like to hint at the omission. This can be done this way:

```

\documentclass[parskip=half]{scrartcl}
\usepackage{scrjura}
\begin{document}
\begin{contract}
  \Clause{title={Temporal application},number=2}
  \ellipsispar[2]

  If, subsequent to the commission of a criminal
  offence, the law provides for a lighter penalty,
  that penalty shall be applicable.

  \ellipsispar[3]
\end{contract}
\end{document}

```

To see the result, just give it a try.

The ellipsis is by default `\textellipsis`, if such a command is defined. If not, `\dots` will be used. `\parellipsis` may be redefined with `\renewcommand`.

11.4.3. Sentences

In a contract the paragraphs consist of one or more sentences. In German acts of law it is common to number the sentences as well. Regarding `scrjura`, an automatic numbering is cumbersome and error-prone and has not been implemented yet.

```

sentence
\thesentence
\Sentence

```

Manual numbering of sentences is done with the command `\Sentence`. It adds one to the counter `sentence`. As a default, `\thesentence` is printed as an arabic number.

Using `babel` offers an easy way to define a short hand for `\Sentence`:

```

\usesshorthands{' }
\defineshorthand{'S}{\Sentence\ignorespaces}

```

With this definition any space after 'S will be ignored. It is even possible to use the dot as abbreviation for dot and new number of the following sentence:

```

\defineshorthand{'.'}{. \Sentence\ignorespaces}

```

These abbreviations have been tried and tested. For details regarding `\usesshorthands` and `\defineshorthands` please consult the manual of the package `babel` (see [BB13]).

11.5. Cross References

The conventional mechanism to set cross references using `\label`, `\ref`, and `\pageref` does not suffice. `scrjura` provides more commands.

```
\ref{label}
\refL{label}
\refS{label}
\refN{label}
```

The commands `\refL`, `\refS`, and `\refN` give a full reference to clause, paragraph and sentence. `\refL` is a long text, `\refS` a short text and `\refN` an abbreviated, numeric form. `\ref` defaults to `\refL`.

```
\refClause{label}
\refClauseN{label}
```

For a cross reference to to a clause without displaying paragraph and sentences as well. `\refClause` puts a section symbol (§) in front of the reference, while `\refClauseN` does not.

```
\refPar{label}
\refParL{label}
\refParS{label}
\refParN[number format]{label}
```

You can make a cross reference to a paragraph using `\refParL`, `\refParS` and `\refParN`. The differences between the forms correspond to the differences between `\refL`, `\refN` and `\refS`. A special feature is the optional argument of `\refParN`. Usually the numeric reference to a paragraph uses a roman number. The optional argument allows to set a different *number format*, it may make sense to use arabic numbers. `\refPar` defaults to `\refParL`.

```
\refSentence{label}
\refSentenceL{label}
\refSentenceS{label}
\refSentenceN{label}
```

You can make a cross reference to a sentence with `\refSentenceL`, `\refSentenceS`, or `\refSentenceN`. Again we have a long text form, a short text form and a numerical form. `\refSentence` defaults to `\refSentenceL`.

Table 11.3.: Possible values for option `ref` to configure the cross reference format of `\ref`, `\refPar`, and `\refSentence`

<code>long</code>	Combination of <code>parlong</code> and <code>sentencelong</code> .
<code>numeric</code>	Combination of <code>parnumeric</code> and <code>sentencenumeric</code> .
<code>clauseonly</code> , <code>onlyclause</code> , <code>ClauseOnly</code> , <code>OnlyClause</code>	Combination of <code>paroff</code> and <code>sentenceoff</code> ; note that <code>\refPar</code> and <code>\refSentence</code> have empty results!
<code>parlong</code> , <code>longpar</code> , <code>ParL</code>	Paragraphs are referenced in long textual form.
<code>parnumeric</code> , <code>numericpar</code> , <code>ParN</code>	Paragraphs are referenced in simple numerical form.
<code>paroff</code> , <code>nopar</code>	Paragraphs have not reference. Note that <code>\refPar</code> has an empty result!
<code>parshort</code> , <code>shortpar</code> , <code>ParS</code>	Paragraphs are referenced in short textual form.
<code>sentencelong</code> , <code>longsentence</code> , <code>SentenceL</code>	Sentences are referenced in long textual form.
<code>sentencenumeric</code> , <code>numeralsentence</code> , <code>SentenceN</code>	Sentences are referenced in simple numeric form.
<code>sentenceoff</code> , <code>nosentence</code>	Sentences have no reference. Note that <code>\refSentence</code> has an empty result!
<code>sentenceshort</code> , <code>shortsentence</code> , <code>SentenceS</code>	Sentences are referenced in short textual form.
<code>short</code>	Combination if <code>parshort</code> and <code>sentenceshort</code> .

Table 11.4.: Example outputs of the `ref`-independent cross reference commands

Command	Example output
<code>\refL{label}</code>	§ 1 paragraph 1 sentence 1
<code>\refS{label}</code>	§ 1 par. 1 sent. 1
<code>\refN{label}</code>	§ 1 I 1.
<code>\refClause{label}</code>	§ 1
<code>\refClauseN{label}</code>	1
<code>\refParL{label}</code>	paragraph 1
<code>\refParS{label}</code>	par. 1
<code>\refParN{label}</code>	I
<code>\refParN[arabic]{label}</code>	1
<code>\refParN[roman]{label}</code>	i
<code>\refSentenceL{label}</code>	sentence 1
<code>\refSentenceS{label}</code>	sent. 1
<code>\refSentenceN{label}</code>	1.

ref=value

The results of `\ref`, `\refPar`, and `\refSentence` depend on the *value* of option `ref`. Defaults are `\refL`, `\refParL` and `\refSentenceL`. For possible values and their meaning see [table 11.3](#).

Example: Supposed you would like to reference to paragraphs in the form “paragraph 1 in clause 1”. As `scrjura` lacks of a predefined command for this, you have to build your own definitions out of what’s given:

```
\newcommand*{\refParM}[1]{%
  paragraph~\refParN[arabic]{#1}
  in clause~\refClauseN{#1}%
}
```

This new command can be used in the same way as `\refParL`.

[Table 11.4](#) provides examples of the output of the fundamental commands, not configured.

11.6. Additional Environments

There are users of `scrjura` who use it to draft neither contracts nor commentaries on certain acts of law, but a compilation of, e. g., different laws, which does not necessarily use the section prefix (§) before the title of each clause. Maybe somebody would like to write something like “Art. XY” and needed an independent counter for each contract environment.

```
\DeclareNewJuraEnvironment{name}[options]{start commands}{end commands}
```

v0.9

This command allows to define new and independent environments of the type `contract`. The argument *name* is the name of the new environment, of course. *start commands* are commands which will be executed at the beginning of the environment, as if they were added directly after `\begin{name}`. Correspondingly *end commands* will be executed at the end of the environment, as if added directly before `\end{name}`. Without any *options* the new environment behaves similar to the `contract` environment, but with its own counters. It is possible to set *options* in a comma separated list. Currently the following *options* are supported:

Clause=instruction: Defines on which *instruction* inside the environment the command `\Clause` is being mapped. The *instruction* expects exactly one argument. To use it correctly, a deeper knowledge and understanding of the internal functions of `scrjura` is needed. Furthermore the requirements for *instruction* may change in future versions. It is recommended not to use this option.

SubClause=instruction: See explanation for `Clause` above.

Sentence=instruction: Defines on which *instruction* inside the environment the command `\Sentence` is being mapped. The *instruction* must not have an argument. Typically it should add one to the counter `sentence` (using `\refstepcounter`) and display it somehow. Please avoid undesirable spaces.

ClauseNumberFormat=instruction: Formats the numbers of clauses. Expects exactly one argument, the number of the clause. If the *instruction* implements a series of instructions and the number is the last argument of a that series, you may directly use the series of instructions as *instruction*.

Example: To define the environment we mentioned in the preface of this section, for articles, it is sufficient to write:

```
\DeclareNewJuraEnvironment{Artikel}[ClauseNumberFormat=Art.]{\{}}
```

In case paragraphs have to be separated by space between the paragraphs, `scrjura` together with a KOMA-Script document class allows to write:

```
\DeclareNewJuraEnvironment{Artikel}[ClauseNumberFormat=Art.~]
{\KOMAOPTIONS{parskip}}{\}
```

In cross references will “Art.” be used instead of “§”, of course.

Table 11.5.: Meanings and English defaults of language dependent terms if not already defined

Command	Meaning	Default
<code>\parname</code>	long form “paragraph”	paragraph
<code>\parshortname</code>	short form “paragraph”	par.
<code>\sentencename</code>	long form “sentence”	sentence
<code>\sentenceshortname</code>	short form “sentence”	sent.

11.7. Support for Different Languages

The package `scrjura` has been developed in cooperation with an German lawyer. Therefore primarily it has provided the languages `german`, `ngerman`, `austrian`, and `naustrian`. Nevertheless, it has been designed to support common language packages like `babel`. Users can make language adjustments simply using `\providecaptionname` (see [section 12.4, page 325](#)). But if you have definite information about the correct juristic terms and conventions of a language, it is recommended to contact the KOMA-Script author. This has been happened for English and therefore in the meantime `scrjura` also provides terms for languages `english`, `american`, `british`, `canadian`, `USenglish`, and `UKenglish`.

```
\parname
\parshortname
\sentencename
\sentenceshortname
```

These are the language-dependent terms used by `scrjura`. The meaning of the terms and the English defaults are shown in [table 11.5](#). The package itself uses `\providecaptionname` inside `\begin{document}` to define them. So pre-definitions for the same language but before loading `scrjura` will not be changed. If you use `scrjura` with a language setting currently not supported, the package throws an error.

11.8. A Detailed Example

Remember the letter from [chapter 4](#). A club member has written to the board because of the general meeting that is regulated by the club statutes. Such club statutes are a kind of contract and you can realise them using `scrjura`.³

```
\documentclass[fontsize=12pt,pagesize,parskip=half]
{scrartcl}
```

³Please note, the example is still in German and has to be translated. Currently we have had not enough manpower to do the translation. Nevertheless, the contents of the example does not matter but the technical realisation.

We use class `scrartcl`. Because paragraph distance instead of paragraph indentation is usual in club statutes we load the class with option `parskip=half` (see [section 3.10, page 74](#)).

```
\usepackage[ngerman]{babel}
```

The club rules are in German. Therefore package `babel` with option `ngerman` is used, too.

```
\usepackage[T1]{fontenc}
\usepackage{lmodern}
\usepackage{textcomp}
```

Default settings for the fonts are made. Additionally package `textcomp` is loaded, because it does not only provide a usable Euro sign but also an improved section sign (§).

```
\usepackage{selenium}
\SelectInputMappings{
  adieresis={ä},
  germandbls={ß},
}
```

We want to input special characters and umlauts without commands. To do so, we let `LATEX` detect the input encoding. Alternatively we could use package `inputenc`.

```
\usepackage{enumerate}
```

Later in the document we want lists numbered not with arabic numbers but with lower letters. We can do this easily with package `enumerate`.

```
\usepackage[clausemark=forceboth,
  juratotoc,
  juratocnumberwidth=2.5em]
{scrjura}
\usesorthands{' }
\defineshorthand{'S}{\Sentence\ignorespaces}
\defineshorthand{'.'}{. \Sentence\ignorespaces}
```

```
\pagestyle{myheadings}
```

Now, it is time for `scrjura`. With option `clausemark=forceboth` we enforce left and right clause marks for the running head. On the other hand we do not want `\section` to change the marks for the running head. Therefore we use page style `myheadings`. This page style generally does not provide automatic running heads.

Later we also want a table of contents with the clauses. This can be achieved with option `juratotoc`. Doing so we will see, that the clause number width is too large for default of the entry into the table of contents. With `juratocnumberwidth=2.5em` we declare a larger reserved width.

The definition of short hands has already been shown in [section 11.4.3](#). We also use this to make the input easier and more readable.

```
\begin{document}
```

It is time to begin the document.

```
\subject{Satzung}
\title{VfVmai}
\subtitle{Verein für Vereinsmaierei mit ai n.e.V.}
\date{11.\,11.\,2011}
\maketitle
```

Like other documents the statutes have a title. We make it with the usual KOMA-Script commands from [section 3.7](#) (see down from [page 62](#)).

```
\tableofcontents
```

As already stated, we want to have a table of contents.

```
\addsec{Präambel}
```

```
Die Vereinslandschaft in Deutschland ist vielfältig.
Doch leider mussten wir feststellen, dass es dabei oft
am ernsthaften Umgang mit der Ernsthaftigkeit krankt.
```

Preambles are not unusual in club statutes. Here we use `\addsec` to realise it, because we want to have an entry into the table of contents.

```
\appendix
```

Here we use a tiny trick. The articles of the club statutes should be numbered with uppercase letters instead of arabic numbers. This is the same like the numbering of appendix sections of an article with `scrartcl`.

```
\section{Allgemeines}
```

```
\begin{contract}
```

We begin the contract with the first article.

```
\Clause{title={Name, Rechtsform, Sitz des Vereins}}
```

```
Der Verein führt den Namen »Verein für Vereinsmaierei mit
ai n.e.V.« und ist in keinem Vereinsregister eingetragen.
```

```
'S Der Verein ist ein nichtwirtschaftlicher, unnützer
Verein'. Er hat keinen Sitz und muss daher stehen.
```

```
Geschäftsjahr ist vom 31.-März bis zum 1.-April.
```

The first clause has a number and a title. We will do the same with all following clauses.

The first paragraph of the clause is very usual. Because it is not the only paragraph, the output is done with a number before the text. Note that the numbering of the first paragraph needs at least two \LaTeX runs like you will for the table of contents.

In the second paragraph we have two sentences. Here we can see the short hands `'S` and `'`.

in action. The first one does only generate the sentence number. The second one does not only print the full stop but also the sentence number. With this, both sentences are numbered.

```
\Clause{title={Zweck des Vereins}}

'S Der Verein ist zwar sinnlos, aber nicht zwecklos'.
Vielmehr soll er den ernsthaften Umgang mit der
Ernsthaftigkeit auf eine gesunde Basis stellen.

Zu diesem Zweck kann der Verein
\begin{enumerate}[\quad a)]
\item in der Nase bohren,
\item Nüsse knacken,
\item am Daumen lutschen.
\end{enumerate}
```

Der Verein ist selbstsüchtig und steht dazu.

Der Verein verfügt über keinerlei Mittel.\label{a:mittel}

The second clause: also with several paragraphs with one or more sentences. The second paragraph additionally has a numbered list. At the last paragraph we used a label, because we want to reference it later.

```
\Clause{title={Vereinsämter}}

Die Vereinsämter sind Ehrenämter.

'S Würde der Verein über Mittel verfügen
(siehe \ref{a:mittel}), so könnte er einen
hauptamtlichen Geschäftsführer bestellen'. Ohne
die notwendigen Mittel ist dies nicht möglich.
```

The third clause has something special: a cross reference. Here we use the long form with clause, paragraph and sentence. If we would decide, that sentences should not be referenced, we could use option `ref=nosentence` globally.

```
\Clause{title={Vereinsmaier},dummy}
\label{p.maier}
```

Here we have a special kind of clauses. In prior versions of the club statutes this was a real clause. But then it has been removed. Nevertheless, the numbering of the following clauses should not be changed by removing this one. Therefore the `\Clause` statement has not been removed but supplemented by option `dummy`. With this, we also can set a label despite the clause will not be printed.

```
\end{contract}
```

```
\section{Mitgliedschaft}
```

```
\begin{contract}
```

Another article starts. To avoid problems with the paragraph numbering we interrupt the `contract` environment.

```
\Clause{title={Mitgliedsarten},dummy}
```

The first clause of the next article also has been removed.

```
\Clause{title={Erwerb der Mitgliedschaft}}
```

```
Die Mitgliedschaft kann jeder zu einem angemessenen
Preis von einem der in \refClause{p.maier}
genannten Vereinsmaier erwerben.\label{a.preis}
```

```
'S Zum Erwerb der Mitgliedschaft ist ein formloser
Antrag erforderlich'. Dieser Antrag ist in grüner
Tinte auf rosa Papier einzureichen.
```

We have a real clause again. We cross reference one of the removed clauses and also use a label.

```
\SubClause{title={Ergänzung zu vorstehendem
Paragraphen}}
```

```
'S Mit Abschaffung von \refClause{p.maier} verliert
\ref{a.preis} seine Umsetzbarkeit'. Mitgliedschaften
können ersatzweise vererbt werden.
```

Once more, this is a special kind of clause. This time we have not removed a clause but added one without renumbering of the following clauses. To do so, we use `\SubClause`. Therefore the clause number is the same like the previous one but with an appended “a”.

```
\Clause{title={Ende der Mitgliedschaft}}
```

```
'S Die Mitgliedschaft endet mit dem Leben'. Bei nicht
lebenden Mitgliedern endet die Mitgliedschaft nicht.
```

```
\Clause{title={Mitgliederversammlung}}
```

```
Zweimal jährlich findet eine Mitgliederversammlung statt.
```

```
Der Abstand zwischen zwei Mitgliederversammlungen
beträgt höchstens 6~Monate, 1~Woche und 2~Tage.
```

```
Frühestens 6~Monate nach der letzten Mitgliederversammlung
hat die Einladung zur nächsten Mitgliederversammlung zu
```

erfolgen.

```
\SubClause{title={Ergänzung zur Mitgliederversammlung}}
```

Die Mitgliederversammlung darf frühestens 2~Wochen nach
letztem Eingang der Einladung abgehalten werden.

```
\end{contract}
```

The other clauses of this article are very usual. You already know all the features used for them.

```
\section{Gültigkeit}
```

```
\begin{contract}
```

```
\Clause{title={In Kraft treten}}
```

Diese Satzung tritt am 11.\,11.\,2011 um 11:11~Uhr
in Kraft.

'S Sollten irgendwelche Bestimmungen dieser Satzung im
Widerspruch zu einander stehen, tritt die Satzung am
11.\,11.\,2011 um 11:11~Uhr und 11~Sekunden wieder
außer Kraft'. Der Verein ist in diesem Fall als
aufgelöst zu betrachten.

```
\end{contract}
```

More articles with known features are following.

```
\end{document}
```

Then the L^AT_EX document ends. You can see three pages from the front of the document in [figure 11.1](#).

11.9. State of Development

The package is part of KOMA-Script for several years and has been used by lawyers even longer. Nevertheless, it has a version number less than 1. So you should still regard it as work in progress. Here are the three reasons for this:

The package has been designed much more general than it can be used currently. For example several environments beside **contract** has been expected. Later we find that this one and only environment could be used very general. Nevertheless we also find that it could be useful to be able to define additional **contract** environments, e. g., for articles of constitutional law. This has been implemented now.

Neither the cooperation with other packages nor the cooperation of the **contract** environment with all kind of L^AT_EX environments or document classes has been tested. The main

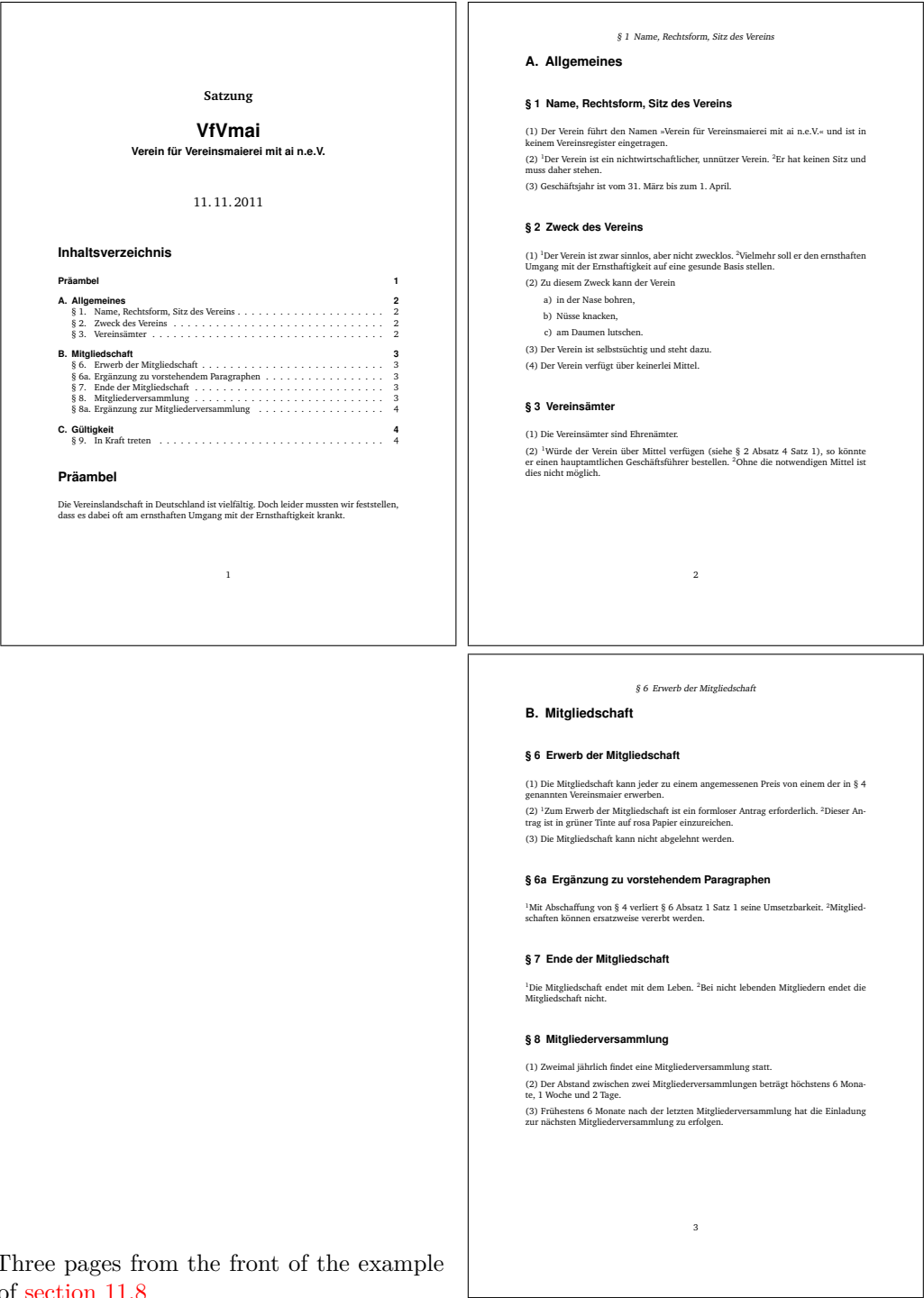


Figure 11.1.: Three pages from the front of the example club statutes of [section 11.8](#)

reason for this is that the package is very special and the package author does not have any requirement to use it. So all changes, all features, all improvement can only base on detailed user feedback and only about two and a half users are willing to send such feedback.

The low version number should state that things could change. The author endeavours to preserve compatibility to prior versions. Nevertheless, sometimes compatibility is less important than usability. So compatibility cannot be guaranteed.

Part II.

KOMA-Script for Advanced Users and Experts

In this part information for the authors of LaTeX packages and classes can be found. This applies not only instructions that are useful for implementation of new packages and classes, but also interfaces to allow further intervention in KOMA-Script. Moreover, in this part, information on obsolete options and instructions are provided as well as background information on the implementation of KOMAScript.

This part is intended to supplement the information for authors of articles, reports, books and letters in [part I](#). More information and examples for those users can be found in that part.

Basic Functions of Package `scrbase`

The package `scrbase` provides basic features designed and implemented for use by authors of packages and classes. However, `scrbase` cannot only be used for wrapper classes related to KOMA-Script class. Authors of classes that have nothing to do with KOMA-Script can benefit from `scrbase` functionality.

12.1. Loading the Package

Whereas users load packages using `\usepackage`, authors of packages or classes should use `\RequirePackage`. Authors of wrapper packages may also use `\RequirePackageWithOptions`. Command `\RequirePackage` has the same optional argument for package options like `\usepackage`. In contrast, `\RequirePackageWithOptions` does not have an optional argument but passes all options given when loading the wrapper package to the required package. See [Tea06] for more information about these commands.

The package `scrbase` needs the functionality of package `keyval` internally. This may be provided by package `xkeyval` alternatively. Package `scrbase` loads `keyval` as needed.

The package `keyval` provides definition of keys and assignment of values to these keys. The options provided by `scrbase` also use `keyval` syntax: *key=value*.

internalonly=value

Package `scrbase` provides some commands for conditional execution. The primary names for these are builds like `\scr@name`, which are internal commands. KOMA-Script only uses these internal commands internally. Authors of packages and classes may use these internal commands, too, but should not redefine them. Because some of these commands are useful for users, too, they are provided as `\name` normally. But eventually, other packages may provide commands with the same name but different syntax or different functionality. As this would result in a conflict, `scrbase` can suppress the definition of the user commands `\name`. Using option `internalonly` without *value* will define only the internal commands and suppress definition of all the user commands for conditional execution. Alternatively, the user may give all the commands that should not be defined as *value*, but replaces “\” by “/”.

Authors of packages and classes normally should not use this option. Users may use it with or without *value* either as a global option with `\documentclass` or using `\PassOptionsToPackage`.

Example: The user does not want `scrbase` to define commands `\ifVTeX` and `\ifundefinedorrelax`. Because of this, to load the class, the user writes:

```
\documentclass%
[internalonly=/ifVTeX/ifundefinedorrelax]%
{foo}
```

Class name `foo` is, therefore, an placeholder for any class in this example. The meanings of commands `\ifVTeX` and `\ifundefinedorrelax` and many more commands for conditional execution is located in [section 12.3](#).

12.2. Keys as Attributes of Families and their Members

As already mentioned in [section 12.1](#), `scrbase` uses package `keyval` for keys and values of keys. Nevertheless `scrbase` extends the functionality of `keyval`. Whereas only one family owns all keys of `keyval`, `scrbase` recognises also family members. Therefore, a key may be owned by a family or by one or more family members. Additionally, a value may be assigned to the key of a family member, to the key of a family, or to the keys of all family members.

```
\DefineFamily{family}
\DefineFamilyMember[family member]{family}
```

`scrbase` needs to know the members of a family for different reasons. First, you have to define a new family using `\DefineFamily`, which produces an empty member list. If the family has already been defined nothing would happen. Nothing also means that an already existing member list would not be overwritten.

Next, a new member can be added to the family using `\DefineFamilyMember`. If the family does not exist, this would result in an error message. If the member already exists, nothing happens. If the optional *family member* is omitted, the default value “`.\@currname.\@currentext`” is used. During class or package loading `\@currname` and `\@currentext` together represent the file name of the class or package.

Theoretically, it is possible, to define a member without a name using an empty optional *family member* argument. But this is the same as the family itself. It is recommended that only letters and digits be used for *family* and the first character of *family member* should not be a letter or digit. Otherwise, it could happen that members of one family are the same as members of another family.

`scrbase` assigns family “`KOMA`” to itself and adds member “`.scrbase.sty`” to it. Family “`KOMA`” is reserved to KOMA-Script. For your own packages, use the name of the bundle as *family* and the name of the package as *family member* of that *family*.

Example: Assume you are writing a bundle called “`master butcher`”. Within that bundle you have packages `salami.sty`, `liversausage.sty`, and `kielbasa.sty`. Therefore, you decide to use family name “`butcher`” and, to each of the package file, you add the lines

```
\DefineFamily{butcher}
\DefineFamilyMember{butcher}
```

When loading the three packages, this will add the members “.salami.sty”, “.liversausage.sty”, and “.kielbasa.sty” to the family “butcher”. After loading all three packages, all three member will be defined.

```
\DefineFamilyKey[family member]{family}{key}[default]{action}
\FamilyKeyState
\FamilyKeyStateUnknown
\FamilyKeyStateProcessed
\FamilyKeyStateUnknownValue
\FamilyKeyStateNeedValue
```

The command `\DefineFamilyKey` defines a *key*. If a *family member* is given, the *key* becomes an attribute of that member in the given *family*. If a *family member* is not given, the member “.\@currname.\@currentx” is assumed. If, later, a value is assigned to the *key*, the *action* will be executed and the value made an argument of *action*. So inside *action* “#1” would be that value. If the value is omitted, the *default* is used instead. If there is no *default*, the *key* can be used only with a value being defined.

At least

```
\DefineFamilyKey[member]{family}{key}
[default]{action}
```

will result in a call of

```
\define@key{family member}{key}
[default]{extended action}
```

with `\define@key` provided by package `keyval` (see [Car99a]). However, the call of `\define@key` and the *action* is, in fact, extended by additional arrangements.

v3.12

Success or failure of the execution of the *action* should be reported back to `scrbase` by `\FamilyKeyState`. The package itself will take care of additional procedures if needed. You should not report errors by yourself! The default state before execution of *action* is `\FamilyKeyStateUnknown`. This signals that it is not known whether or not the execution is successful. If this state does not change until end of execution of the *action*, `scrbase` will write a message into the log file and assumes state `\FamilyKeyStateProcessed` during the further procedure.

State `\FamilyKeyStateProcessed` signals that the option and the value assignment to the option are completely and successfully finished. You may switch to this state by using `\FamilyKeyStateProcessed` itself.

State `\FamilyKeyStateUnknownValue` indicates that the option was handled, but the value, that should be assigned to the key, was unknown or not allowed. You should use `\FamilyKeyStateUnknownValue` to switch to this state.

State `\FamilyKeyStateNeedValue` signals that the option could not be set because it needs a value, but no value was assigned to the key. This state is used automatically, whenever an

option has been defined without *default* value and is used without value assignment. You should not set the state using `\FamilyKeyStateNeedValue` yourself.

Last but not least you may switch to additional failure states, simply re-defining `\FamilyKeyState` with a very short text message. Generally, the four predefined states should be sufficient.

Example: Assume each of the three packages from the previous example should get a key named `coldcuts`. When used, a switch is set at each of the packages. For package `salami` this may be:

```
\newif\if@Salami@Aufschnitt
\DefineFamilyKey{butcher}%
    {coldcuts}[true]{%
    \expandafter\let\expandafter\if@salami@coldcuts
    \csname if#1\endcsname
    \FamilyKeyStateProcessed
}
```

Available values for the key are `true` or `false` in this case. Instead of testing on inappropriate values, success will be signalled for any case in this example. If the key is used later, it is executed with one of the allowed values or without assignment. In the second case, the default *true* will be used.

The definitions in the other packages are similar. Only “`salami`” has to be replaced by the corresponding names.

```
\RelaxFamilyKey[family member]{family}{key}
```

v3.15

If the *key* of the *family member* of *family* has been defined before, that definition will be cancelled. Afterwards the *key* will not be defined for the *family member* of *family* any longer. Usage of `\RelaxFamilyKey` for a not yet defined *key* of the *family member* of the *family* is also allowed.

```
\FamilyProcessOptions[family member]{family}
```

Generally the extension of keys of families to keys of families and family members, as mentioned earlier, uses keys or key-value settings as class or package options. The command `\FamilyProcessOptions` is an extension of `\ProcessOption*` from L^AT_EX kernel (see [Tea06], which processes not only options that has been declared using `\DeclareOption`, it processes all keys of the given family member. If the optional argument *family member* is omitted, family member “`.\@currname.\@current`” is used.

Somewhat special are keys that are not attached to a family member, but to a family. These are keys with an empty family member. Such keys are set before the keys of the family members.

Example: If a package in the previous example would be extended by the line

```
\FamilyProcessOptions{butcher}
```

then the user may select the option `coldcuts` when loading the package. If the option is used globally, this means at the optional argument of `\documentclass`, then the option would be passed automatically to all three packages, if all three packages are loaded later.

Please note that packages always process global options before local options. When processing unknown options initiate an entry in the `log`-file and the option is otherwise ignored. By contrast, unknown options assigned to the package locally leads to an error message.

`\FamilyProcessOptions` may be interpreted either as an extension of `\ProcessOption*` or as an extension of the *key=value* mechanism of `keyval`. Ultimately, with the help of `\FamilyProcessOptions`, *key=value* pairs become options.

```
\BeforeFamilyProcessOptions[member]{family}{code}
```

v3.18

Especially authors of wrapper classes or wrapper packages sometimes need a hook to execute *code* just before `\FamilyProcessOptions`. The package `scrbase` provides such a hook and you can add *code* to it using `\BeforeFamilyProcessOptions`. The parameters *member* and *family* are same to `\FamilyProcessOptions`. In difference to `\FamilyProcessOptions` you can add *code* also to the hook of not yet defined *family* or *member*.

Note, the hook of a family member will be cleaned after execution. But if you use an empty *member* the *code* will be executed before the `\FamilyProcessOptions` of every member of the *family* and will never be deleted.

Example: Now, you are writing a package that loads `liversausage`. But you do not provide option `coldcut` with this package. So you use `\BeforeFamilyProcessOptions` to deactivate that option before loading the package:

```
\RequirePackage{scrbase}
\BeforeFamilyProcessOptions[.liversausage.sty]{butcher}{%%
  \RelaxFamilyKey[.liversausage.sty]{butcher}{coldcut}%
}
\RequirePackageWithOptions{liversausage}
```

If after this a user tries to load your package with option `coldcut`, package `liversausage` will throw an undefined option error. If `coldcut` is used as a global option, package `liversausage` will ignore it. But default settings inside `liversausage`, e.g., using `\FamilyExecuteOptions` before `\FamilyProcessOptions` are not effected.


```
\FamilyExecuteOptions[family member]{family}{options list}
```

This command is an extension of `\ExecuteOptions` from the L^AT_EX kernel (see [Tea06]). The command processes not only options that are defined using `\DeclareOption`, but also processes all keys of the given *family member*. If the optional argument `\family member` is omitted, then “`.\@currname.\@currentx`” is used.

Somehow special are keys of empty family members, which are not attached to a family member, but to a family. Such keys are set before the keys of family members.

Example: Assume option `coldcuts` should be set by default in the previous example. In this case only line

```
\FamilyExecuteOptions{butcher}{coldcuts}
```

has to be added.

v3.20

If you call `\FamilyExecuteOptions` with an unknown option inside the *options list*, you will get an error. But you will not get an error, if the *family member* has an option `@else@`. In this case option `@else@` will be used instead of the unknown option. KOMA-Script itself uses this feature, e. g., inside the definition of section like commands to prior the style attribute above all other attributes.

Usage of the command inside the execution of an option is provided.

```
\FamilyOptions{family}{options list}
```

Hence *options list* is like:

```
key=value, key=value...
```

after which the value assignment may be omitted for *keys* that have a defined default.

In contrast to average options that are defined using `\DeclareOption`, the *keys* also may be set after loading a class or package. For this, the user calls `\FamilyOptions`. Thereafter, the *keys* of all members of the specified family are set. If a *key* also exists as a family attribute, then the family key is set first. After this, the member keys follow in the order in which the members have been defined. If a given *key* does not exist, either for the family or for any member of the family, then `\FamilyOptions` will result in an error. Package `scrbase` reports an error also if there are members with key *key*, but all those members signal failure via `\FamilyKeyState`.

Example: You extend your `butcher` project by a package `sausagesalad`. If this package has been loaded, all `sausage` package should generate cold cuts:

```
\ProvidesPackage{sausagesalad}%
    [2008/05/06 nonsense package]
\DefineFamily{butcher}
\DefineFamilyMember{butcher}
```

```
\FamilyProcessOptions{butcher}\relax
\FamilyOptions{butcher}{coldcuts}
```

If currently none of the sausage packages are loaded, the undefined option `coldcuts` leads to an error message. This is avoided by adding before the last line of the code above:

```
\DefineFamilyKey{butcher}%
    {coldcuts}[true]{}%
```

However, sausage packages loaded after `sausagesalad` still do not produce cold cuts. This may be corrected by editing the last line of the code again to:

```
\AtBeginDocument{%
    \DefineFamilyKey[.sausagesalad.sty]%
        {butcher}%
        {coldcuts}[true]{}%
}
\DefineFamilyKey{butcher}%
    {coldcuts}[true]{%
    \AtBeginDocument{\FamilyOptions{butcher}%
        {coldcuts=#1}}%
}%
```

This code does following: First the option will be defined while `\begin{document}` to do nothing for package `sausagesalad`. Because `\@currname` and `\@currentx` are not longer valid at this time, the optional argument on `\DefineFamilyKey` has to be used. But until this re-definition of the option, a second definition is made, that calls the option again while `\begin{document}` for the whole family and so also for other sausage salad packages.

```
\FamilyOption{family}{option}{values list}
```

Besides options that have concurrently excluding values, there may be options that produce several values at the same time. Using `\FamilyOptions` for that type of option would result in using the same option several times with different value assignments. Instead of this, `\FamilyOption` may be used to assign a whole *values list* to the same *option*. The *values list* is a comma separated list of values, also known as *csv*:

```
value,value...
```

By the way, please note that usage of a comma inside a value may be done only if the value is put inside braces. The general functionality of this command is the same as that of the previous command `\FamilyOptions`.

Example: Package `sausagesalad` should have one more option, to add additional ingredients. Each of the ingredients sets a switch as was done previously for the cold cuts.

```

\newif\if@saladwith@onions
\newif\if@saladwith@gherkins
\newif\if@saladwith@chillies
\DefineFamilyKey{butcher}{ingredient}{%
  \csname @saladwith@#1true\endcsname
}

```

Here the three ingredients “onions”, “gherkins”, and “chillies” have been defined. An error message for “not defined” ingredients does not exist.

For a salad with onions and gherkins the user may use

```

\FamilyOptions{butcher}{%
  ingredient=onions,ingredient=gherkins}

```

or shorter

```

\FamilyOption{butcher}
  {ingredient}{onions,gherkins}

```

```

\AtEndOfFamilyOptions{action}

```

v3.12

Sometimes it is useful to delay the execution of an *action* that is part of a value assignment to a key until all assignments inside one `\FamilyProcessOptions`, `\FamilyExecuteOptions`, `\FamilyOptions`, or `\FamilyOption` is finished. This may be done using `\AtEndOfFamilyOptions` inside an option definition. Reporting failure states of *action* is not possible in this case. Furthermore, the command should not be used outside an option definition.

```

\FamilyBoolKey[family member]{family}{key}{switch name}
\FamilySetBool{family}{key}{switch name}{value}

```

In the previous examples, boolean switches often have been used. In the example with option `coldcuts`, it is necessary that the user assigns either value `true` or value `false`. There is no error message for wrong value assignment. Because of this, boolean switches are a necessary feature. Package `scrbase` provides `\FamilyBoolKey` for definition of such options. Therefore, the arguments *family member*, *family*, and *key* are the same as that used by `\DefineFamilyKey` (see page 310). Argument *switch name* is the name of the switch without the prefix `\if`. If a switch with this name does not exist already, `\FamilyBoolKey` will define it and initialize it to *false*. Internally `\FamilyBoolKey` uses `\FamilySetBool` as *action* of `\DefineFamilyKey`. The *default* for those options is always `true`.

On the other hand, `\FamilySetBool` understands *value on* and *yes* beside *true* for switching on and *off* and *no* beside *false* for switching off. Unknown values will result in a call of `\FamilyUnknownKeyValue` with the arguments *family*, *key*, and *value*, which results in setting of `\FamilyKeyState`. Depending on the command used and other family members,

this may result in an error message about unknown value assignment (see also [page 319](#) and [page 310](#)).

Example: The key `coldcuts` should be declared somehow more robust. Additionally, all sausage packages should use the same key. So either all or none of them will produce cold cuts.

```
\FamilyBoolKey{butcher}{coldcuts}%
    {@coldcuts}
```

A test, whether or not to produce cold cuts, may be:

```
\if@coldcuts
...
\else
...
\fi
```

This would be the same in each of the three sausage packages, thereby defining the attribute “coldcuts” as a family option:

```
\@ifundefined{if@coldcuts}{%
    \expandafter\newif\csname if@coldcuts\endcsname
}{}%
\DefineFamilyKey[] {butcher}{coldcuts}[true]{%
    \FamilySetBool{butcher}{coldcuts}%
        {@coldcuts}%
        {#1}%
}
```

or shorter:

```
\FamilyBoolKey[] {butcher}{coldcuts}%
    {@coldcuts}
```

using the additional information at [page 310](#), this is not only valid for `\DefineFamilyKey` but for `\FamilyBoolKey` too.

```
\FamilyNumericalKey[family member]{family}{key}[default]{command}{values list}
\FamilySetNumerical{family}{key}{command}{values list}{value}
```

In contrast to switches that may be either true or false, a key exists that accept several values. For example an alignment may not only be left or not left, but left, centered, or right. Internally such differentiation often is made using `\ifcase`. This T_EX command expects a numerical value. Because of this the command to define a macro by a *key* has been named `\FamilyNumericalKey` in scrbase. The *values list* thereby has the form:

```
{value}{definition},{value}{definition},...
```

Therefore, the *values list* does not solely define the supported values to the *key*. For each of the *values*, the *definition* of macro `\command` also is given. Usually, *definition* is just a numerical value. Nevertheless, other content is possible and allowed. Currently, the only limitation is that *definition* has to be fully expandable and will be expanded during the assignment.

Example: The sausage may be cut different. For example the cold cuts may stay uncut or will be cut roughly or may be cut thinly. This information should be stored in command `\cuthow`.

```
\FamilyNumericalKey{butcher}%
    {saladcut}{cuthow}{%
        {none}{none},{no}{none},{not}{none}%
        {rough}{rough},%
        {thin}{thin}%
    }
```

Not cutting anything may be selected either by

```
\FamilyOptions{butcher}{saladcut=none}
```

or

```
\FamilyOptions{butcher}{saladcut=no}
```

or

```
\FamilyOptions{butcher}{saladcut=not}
```

In all three cases `\cuthow` would be defined with content `none`. It may be very useful to provide several values for the same result as shown in this example.

Now, it's most likely, that the kind of cutting will not be printed, but should be evaluated later. In this case a textual definition would not be useful. If the key is defined like this:

```
\FamilyNumericalKey{butcher}%
    {saladcut}{cuthow}{%
        {none}{0},{no}{0},{not}{0}%
        {rough}{1},%
        {thin}{2}%
    }
```

then a condition like the following may be used:

```
\ifcase\cuthow
    % no cut
\or
    % rough cut
\else
    % thin cut
```

`\fi`

Internally, `\FamilyNumericalKey` uses `\FamilySetNumerical` as *action* of `\DefineFamilyKey`. If a unknown value is assigned to such a key, `\FamilySetNumerical` will call `\FamilyUnknownKeyValue` with the arguments *family*, *key* and *value*. This will normally result in an error message about assigning an unknown value.

```
\FamilyCounterKey[family member]{family}{key}[default]{ATEX counter}
\FamilySetCounter{family}{key}{ATEX counter}{value}
```

v3.12 Beside `\FamilyNumericalKey` that defines a macro to a numeric value depending on a symbolic value, there are circumstances when a *key* directly represents a *AT_EX counter*, that should be assigned directly to a numeric value. In those cases, `\FamilyCounterKey` can be used that internally will call `\FamilySetCounter`. A primary test of the *value* argument is done that only detects whether or not usage of *value* as a number is plausible. The assignment is done, only if this plausibility test is successful. Nevertheless the assignment can still fail and result in a *TeX* error. But if the plausibility test fails already, this is signalled by `\FamilyKeyStateUnknownValue`.

v3.15 If the value is omitted, the *default* is used instead. If there is no *default*, the *key* can be used only with a value being defined.

```
\FamilyCounterMacroKey[family member]{family}{key}[default]{macro}
\FamilySetCounterMacro{family}{key}{macro}{value}
```

v3.12 These two commands differ from the previously described `\FamilyCounterKey` and `\FamilySetCounter` only by the fact, that they do not assign a *value* to a *AT_EX counter*, but define a `\macro` with the *value*.

```
\FamilyLengthKey[family member]{family}{key}[default]{length}
\FamilySetLength{family}{key}{length}{value}
\FamilyLengthMacroKey[family member]{family}{key}[default]{Makro}
\FamilySetLengthMacro{family}{key}{Makro}{value}
```

v3.12 With `\FamilyLengthKey` you can define a *key* that represents a *length*. It does not matter whether the *length* is a *LaTeX* length, a *TeX* skip or a *TeX* dimension. Internally the *length* will be set to the *value* using `\FamilySetLength`. Thereby a primary test is used to decide, whether or not it is plausible to assign *length* to *value*. The assignment is done only if this plausibility test is successful. Nevertheless, not all assignment errors can be recognised, so an inaccurate *value* can still result in *TeX* error. Recognised errors however will be signalled by `\FamilyKeyStateUnknownValue`.

v3.15 If the value is omitted, the *default* is used instead. If there is no *default*, the *key* can be used only with a value being defined.

In difference to this, `\FamilyLengthMacroKey` and `\FamilySetLengthMacro` do not assign *value* to a *length*, but define a `\macro` with this *value*.

```
\FamilyStringKey[family member]{family}{key}[default]{command}
```

v3.08

This defines a *key* that accepts every value. The value will be stored into the given `\command`. If there is no optional argument for the *default*, `\FamilyStringKey` is the same as:

```
\DefineFamilyKey[family member]{family}{key}
{\def command{#1}\FamilyKeyStateProcessed}.
```

If an optional argument *default* is used, `\FamilyStringKey` is the same as:

```
\DefineFamilyKey[family member]{family}{key}
[default]{\def command{#1}}.
```

If *command* is not been defined, an empty macro will be defined.

Example: By default an amount of 250 g sausage salad should be produced. The amount should be configurable by an option. The wanted amount will be stored in the macro `\saladweight`. The option should be named `saladweight`, too:

```
\newcommand*{\saladweight}{250g}
\FamilyStringKey{butcher}%
{saladweight}[250g]{\saladweight}
```

To switch back to the default weight after changing it, the user may simply use the option without weight:

```
\FamilyOptions{butcher}{saladweight}
```

This may be done, because the default weight has been set as default at the definition of the option above.

In this case there are not unknown values because all values are simply used for a macro definition. Nevertheless, you should note that paragraph breaks at the value assignment to the key are not allowed.

```
\FamilyUnknownKeyValue{family}{key}{value}{values list}
```

The command `\FamilyUnknownKeyValue` throws an error message about unknown values assigned to a known key. The *values list* is a comma separated list of allowed values in the form:

```
'value', 'value' ...
```

v3.12

Currently, *values list* is not used by scrbase.

Example: Now, for the cold cuts, the choices should be cut or no-cut and in case of cut rough or thin. Rough should be the default for cutting.

```
\@ifundefined{if@thincut}{%
  \expandafter
  \newif\csname if@thincut\endcsname}{}%
\@ifundefined{if@coldcuts}{%
  \expandafter
  \newif\csname if@coldcuts\endcsname}{%
\DefineFamilyKey{butcher}%
  {coldcuts}[true]{%
  \FamilySetBool{butcher}{coldcuts}%
    {coldcuts}%
    {#1}%
  \ifx\FamilyKeyState\FamilyKeyStateProcessed
    \@thincutfalse
  \else
    \ifstr{#1}{thin}{%
      \@coldcutstrue
      \@thincuttrue
      \FamilyKeyStateProcessed
    }{}%
  \fi
}%
```

Let's have a look at the definition of `butcher`: First of all, we try to set the boolean switch of cold cuts using `\FamilySetBool`. After this command, we test whether or not `\FamilyKeyState` signals the success of the command with state `\FamilyKeyStateProcessed`. If so, only the thin cut has to be deactivated.

In the other case, the value will be tested to be equal to `thin`. In that case, cold cuts and thin cut are activated and the state will be switched to `\FamilyKeyStateProcessed`. If the last test failed also, the failure state of `\FamilySetBool` is still valid at the end of the execution.

Command `\ifstr` is used for the thin test. It is described on [page 322](#) in [section 12.3](#).

`\FamilyElseValues`

v3.12

With former releases of `scrbase`, additional allowed values reported by `\FamilyUnknownKeyValue` can be set re-defining `\FamilyElseValues` in the form:

, *'value'*, *'value'* ...

Since release 3.12 `\FamilyUnknownKeyValue` does not report errors itself, but signals them using `\FamilyKeyState`. Therefore, `\FamilyElseValues` became deprecated. Nevertheless, its former usage is recognised by `scrbase` and results in a code change demand message.

12.3. Conditional Execution

The package `scrbase` provides several commands for conditional execution. other than the \TeX syntax of conditionals, e. g.,

```
\iftrue
...
\else
...
\fi
```

yet the \LaTeX syntax also known from \LaTeX commands like `\IfFileExists`, `\@ifundefined`, `\@ifpackageloaded`, and many others, is used. Nevertheless, some package authors prefer to use the \TeX syntax even for users at the \LaTeX interface level that could inevitably lead to naming conflicts with the `scrbase` conditionals. In fact, the conditionals of `scrbase` are very basic. Because of this `scrbase` firstly defines these conditionals as internal commands with prefix `\scr@`. Additional user commands without this prefix are subsequently defined. But the definition of the user commands may be suppressed with option `internalonly` (see [section 12.1, page 308](#)).

Authors of packages and classes should use the internal commands as KOMA-Script itself does. Nevertheless, for completeness the user commands are described separately.

```
\scr@ifundefinedorrelax{name}{then instructions}{else instructions}
\ifundefinedorrelax{name}{then instructions}{else instructions}
```

This command has almost the same functionality as `\@ifundefined` from the \LaTeX kernel (see [\[BCJ+05\]](#)). So the *then instructions* will be executed if *name* is the name of a command that is currently either not defined or `\relax`. Otherwise, the *else instructions* will be executed. In contrast to `\@ifundefined`, *name* is not defined to be `\relax` in the event it was not defined before. Moreover, using ε - \TeX this case will not consume any hash memory.

```
\scr@ifpdfptex{then instructions}{else instructions}
\ifpdfptex{then instructions}{else instructions}
```

If pdf \TeX has been used, the *then instructions* will be executed, otherwise the *else instructions*. Whether or not a PDF-file is generated does not matter, and the pdf \TeX test is rarely useful. In general, test for the desired command instead (see previous `\scr@ifundefinedorrelax` and `\ifundefinedorrelax`).

```
\scr@ifVTeX{then instructions}{else instructions}
\ifVTeX{then instructions}{else instructions}
```

If $\text{V}\text{T}_{\text{E}}\text{X}$ has been used, the *then instructions* will be executed, otherwise the *else instructions*. This test is seldom useful. Test for the desired command instead (see previous `\scr@ifundefinedorrelax` and `\ifundefinedorrelax`).

```
\scr@ifpdfoutput{then instructions}{else instructions}
\ifpdfoutput{then instructions}{else instructions}
```

When generating a PDF-file the *then instructions* will be executed, otherwise the *else instructions*. Whether $\text{pdf}\text{T}_{\text{E}}\text{X}$ or $\text{V}\text{T}_{\text{E}}\text{X}$ or another $\text{T}_{\text{E}}\text{X}$ engine is used to generate the PDF-file does not matter.

```
\scr@ifpsoutput{then instructions}{else instructions}
\ifpsoutput{then instructions}{else instructions}
```

When generating a PostScript-file the *then instructions* will be executed, otherwise the *else instructions*. $\text{V}\text{T}_{\text{E}}\text{X}$ provides direct PostScript generation that will be recognised here. If $\text{V}\text{T}_{\text{E}}\text{X}$ is not used, but a switch `\if@dvi` has been defined, the decision depends on that switch. KOMA-Script provides `\if@dvi` in `typearea`.

```
\scr@ifdvioutput{then instructions}{else instructions}
\ifdvioutput{then instructions}{else instructions}
```

When generating a DVI-file the *then instructions* will be executed, otherwise the *else instructions*. If neither a direct PDF-file generation nor a direct PostScript-file generation has been detected, DVI-file generation is assumed.

```
\ifnotundefined{name}{then instructions}{else instructions}
```

$\varepsilon\text{-T}_{\text{E}}\text{X}$ will be used to test, whether or not a command with the given *name* has already been defined. The *then instructions* will be executed if the command is defined, otherwise the *else instructions*. There is no corresponding internal command.

```
\ifstr{string}{string}{then instructions}{else instructions}
```

Both *string* arguments are expanded and afterwards compared. If the expansions are the same, the *then instructions* will be executed, otherwise the *else instructions*. There is no corresponding internal command.

```
\ifstrstart{string}{string}{then instructions}{else instructions}
```

- v3.12 Both *string* arguments are expanded and afterwards compared. If aside from white spaces the first string starts with the second one, the *then instructions* will be executed, otherwise the *else instructions*. The command is not completely expandable and there is no corresponding internal command.

```
\ifisdimen{code}{then instructions}{else instructions}
```

- v3.12 If the expansion of *code* results in a `\dimen`, which is also known as T_EX length register, the *then instructions* will be executed, otherwise the *else instructions*. The command is not completely expandable and there is no corresponding internal command.

```
\ifisdimension{code}{then instructions}{else instructions}
```

- v3.12 If *code* expands to something with the syntax of the value of a length, the *then instructions* will be executed, otherwise the *else instructions*. Please note that currently an invalid unit will result in an error message. The command is not completely expandable and there is no corresponding internal command.

```
\ifisdimexpr{code}{then instructions}{else instructions}
```

- v3.12 If the expansion of *code* results in a `\dimexpr`, which is also known as T_EX length expression, the *then instructions* will be executed, otherwise the *else instructions*. Note that illegal expressions will result in error messages. The command is not completely expandable and there is no corresponding internal command.

```
\ifisskip{code}{then instructions}{else instructions}
```

- v3.12 If the expansion of *code* results in a `\skip`, which is also known as T_EX distance register, the *then instructions* will be executed, otherwise the *else instructions*. The command is not completely expandable and there is no corresponding internal command.

```
\ifisglue{code}{then instructions}{else instructions}
```

- v3.12 If *code* expands to something with the syntax of the value of a skip, the *then instructions* will be executed, otherwise the *else instructions*. Please note that currently an invalid unit will result in an error message. The command is not completely expandable and there is no corresponding internal command.

```
\ifisglueexpr{code}{then instructions}{else instructions}
```

- v3.12 If the expansion of `code` results in a `\glueexpr`, which is also known as T_EX distance expression, the *then instructions* will be executed, otherwise the *else instructions*. Note, that illegal expressions will result in error messages. The command is not completely expandable and there is no corresponding internal command.

```
\ifiscount{code}{then instructions}{else instructions}
```

- v3.12 If the expansion of `code` results in a `\count`, which is also known as T_EX counter register, the *then instructions* will be executed, otherwise the *else instructions*. The command is not completely expandable and there is no corresponding internal command. For test with L^AT_EX counters, see `\ifiscouter`.

```
\ifisinteger{code}{then instructions}{else instructions}
```

- v3.12 If `code` expands to something with the syntax of the value of a counter, which would be a negative or positive integer, the *then instructions* will be executed, otherwise the *else instructions*. The command is not completely expandable and there is no corresponding internal command.

```
\ifisnumexpr{code}{then instructions}{else instructions}
```

- v3.12 If the expansion of `code` results in a `\numexpr`, which is also known as T_EX number expression, the *then instructions* will be executed, otherwise the *else instructions*. Note that illegal expressions will result in error messages. The command is not completely expandable and there is no corresponding internal command.

```
\ifiscounter{counter}{then instructions}{else instructions}
```

- v3.12 If *counter* is an already defined L^AT_EX counter, the *then instructions* will be executed, otherwise the *else instructions*. The command is not completely expandable and there is no corresponding internal command.

```
\ifnumber{string}{then instructions}{else instructions}
```

Note that this does not compare numbers. The *then instructions* will be executed if the one step expansion of *string* consists of digits only. Otherwise the *else instructions* will be used. There is no corresponding internal command.

```
\ifdimen{string}{then instructions}{else instructions}
```

Note that this does not compare dimensions. The *then instructions* will be executed, if the one step expansion of *string* consists of digits and a valid unit of length. Otherwise, the *else instructions* will be used. There is no corresponding internal command.

```
\if@atdocument the instructions \else else instructions \fi
```

This command exists intentionally as internal command only. In the document preamble `\if@atdocument` is same as `\iffalse`. After `\begin{document}` it's the same as `\iftrue`. Authors of classes and packages may use this if a command should work somehow different depending on whether it has been used in the preamble or inside the documents body. Please note that this is a condition in T_EX syntax not in L^AT_EX syntax!

12.4. Definition of Language-Dependent Terms

Normally, one has to change or define language-dependent terms like `\captionseenglish` 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. These definitions unfortunately occur in such a manner as to destroy all previous private settings. That is also the reason why it makes sense to delay 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 package `scrbase` provides some additional commands for defining language-dependent terms.

```
\defcaptionname{language list}{term}{definition}
\providecaptionname{language list}{term}{definition}
\newcaptionname{language list}{term}{definition}
\renewcaptionname{language list}{term}{definition}
\defcaptionname*{language list}{term}{definition}
\providecaptionname*{language list}{term}{definition}
\newcaptionname*{language list}{term}{definition}
\renewcaptionname*{language list}{term}{definition}
```

Using one of these commands, the user can assign a *definition* for a particular language to a *term*. Several languages can be concatenated with comma to a *language list*. The *term* is always a macro. The commands differ depending 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 a language is not defined, then `\providecaptionname` does nothing other than write a message in the log file. This happens only once for each language. If a 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 a language is not yet defined, then a new language command will be created and a message written to the log file. For language `USenglish`, for example, this would be the language command `\captionsUSenglish`. If *term* is not yet defined in a language, then it will be defined using *definition*. If *term* already exists in a language, then this results in an error message.

The command `\renewcaptionname` again behaves differently. It requires an existing definition of *term* in the languages. If neither a language nor *term* exist or *term* is unknown in a defined language, then an error message is given. Otherwise, the *term* for the language will be redefined according to *definition*.

v3.12

The command `\defcaptionname` always defines the *term*. So previous definitions of *term* for the given *language* will be overwritten. You may use this command even for undefined languages.

KOMA-Script employs `\providcaptionname` in order to define the commands in [section 22.6](#).

Example: If you prefer “fig.” instead of “figure” in `USenglish`, you may achieve this using:

```
\renewcaptionname{USenglish}{\figurename}{fig.}
```

If you want the same change not only in `USenglish` but also in `UKenglish`, you do not need an additional:

```
\renewcaptionname{UKenglish}{\figurename}{fig.}
```

but can simply extend the *language list*:

```
\renewcaptionname{USenglish,UKenglish}{\figurename}{fig.}
```

You can extend the *language list* in the same manner by `american`, `australian`, `british`, `canadian`, and `newzealand`.

v3.12

Since KOMA-Script 3.12 you do not need to delay the definition or redefinition until `\begin{document}` using `\AtBeginDocument` any longer because `scrbase` does the delay automatically for (re)definitions in the document’s preamble. Additionally, `scrbase` tests if a redefinition should be made in `\extraslanguage` instead of `\captionslanguage` and does so automatically. The new star variants of the commands always use `\extraslanguage`. So redefinition of language dependent terms for packages like `hyperref` that use `\extraslanguage` should work as expected by the user.

Language dependent terms usually defined by classes and language packages are listed and described in [table 12.1](#).

Table 12.1.: Overview of language dependent terms of usual language packages

<code>\abstractname</code>	heading of the abstract
<code>\alsoname</code>	“see also” in additional cross references of the index
<code>\appendixname</code>	“appendix” in the heading of an appendix chapter
<code>\bibname</code>	heading of the bibliography
<code>\ccname</code>	prefix heading for the distribution list of a letter
<code>\chaptername</code>	“chapter” in the heading of a chapter
<code>\contentsname</code>	heading of the table of contents
<code>\enclname</code>	prefix heading for the enclosure of a letter
<code>\figurename</code>	prefix heading of figure captions
<code>\glossaryname</code>	heading of the glossary
<code>\headtoname</code>	“to” in header of letter pages
<code>\indexname</code>	heading of the index
<code>\listfigurename</code>	heading of the list of figures
<code>\listtablename</code>	heading of the list of tables

Table 12.1.: Overview of usual language dependent terms (*continuation*)

<code>\pagename</code>	“page” in the pagination of letters
<code>\partname</code>	“part” in the heading of a part
<code>\prefacename</code>	heading of the preface
<code>\proofname</code>	prefix heading of mathematical proofs
<code>\refname</code>	heading of the list of references
<code>\seename</code>	“see” in cross references of the index
<code>\tablename</code>	prefix heading at table captions

12.5. Identification of KOMA-Script

Package `scrbase` may be used independent of KOMA-Script with other packages and classes. Nevertheless, it is a KOMA-Script package. For this, `scrbase` also provides commands to identify KOMA-Script and to identify itself as a KOMA-Script package.

`\KOMAScript`

This command only sets the word “KOMA-Script” with sans-serif font and a little bit tracking for the capitals. By the way: All KOMA-Script classes and packages define this command, if it has not been defined already. The definition is robust using `\DeclareRobustCommand`.

`\KOMAScriptVersion`

KOMA-Script defines the main version of KOMA-Script in this command. It has the form “*date version* KOMA-Script”. This main version is same for all KOMA-Script classes and the KOMA-Script packages that are essential for the classes. Because of this, it may be inquired after loading `scrbase`, too. For example, this document has been made using KOMA-Script version “2017/01/03 v3.22 KOMA-Script”.

12.6. Extension of the \LaTeX Kernel

Sometimes the \LaTeX kernel itself provides commands, but lacks other, similar commands that would often be useful, too. Some of those commands for authors of packages and classes are provided by `scrbase`.

```
\ClassInfoNoLine{class name}{information}
\PackageInfoNoLine{package name}{information}
```

For authors of classes and package the \LaTeX kernel already provides commands like `\ClassInfo` and `\PackageInfo` to write information. together with the current line number, into the log-file. Besides `\PackageWarning` and `\ClassWarning` to throw warning messages with line numbers, it also provides `\PackageWarningNoLine` and `\ClassWarningNoLine` for warning messages without line numbers. Nevertheless, the commands `\ClassInfoNoLine` and `\PackageInfoNoLine` for writing information without line numbers into the log-file are missing. Package `scrbase` provides them.

```
\l@addto@macro{command}{extension}
```

The \LaTeX kernel already provides an internal command `\g@addto@macro` to extend the definition of macro `\command` by *extension* globally. This may be used only for macros that have no arguments. Nevertheless, sometimes a command like this, that works locally to a group instead of globally, could be useful. Package `scrbase` provides such a command with `\l@addto@macro`. An alternative may be usage of package `etoolbox`, which provides several of such commands for different purposes (see [Leh11]).

12.7. Extension of the Mathematical Features of $\varepsilon\text{-TeX}$

$\varepsilon\text{-TeX}$, that is meanwhile used by \LaTeX and needed by KOMA-Script, provided with `\numexpr`, an extended feature for calculation of simple arithmetic with \TeX counters and integers. The four basic arithmetic operations and brackets are supported. Correct rounding is done while division. Sometimes additional operators would be useful.

```
\XdivY{dividend}{divisor}
\XmodY{dividend}{divisor}
```

v3.05a

Having a division with remainder command `\XdivY` gives the value of the integer quotient, with command `\XmodY` giving the value of the remainder. This kind of division is defined:

$$dividend = divisor \cdot integer\ quotient + remainder$$

with *dividend* and *remainder* are integer, *remainder* is greater or equal to *divisor*, and *divisor* is a natural number greater than 0.

The value may be used for assignment to a counter or directly in the expression of `\numexpr`. For output the value as an Arabic number has to be prefixed by `\the`.

Control Package Dependencies with `scrfile`

The introduction of L^AT_EX 2_ε in 1994 brought many changes in the handling of L^AT_EX 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.

13.1. 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 L^AT_EX 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 L^AT_EX users, too. Therefore, this chapter gives and explains examples for both groups of users.

13.2. 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 L^AT_EX command `\InputIfFileExists`. If this macro does not have the expected definition then scrfile issues a warning. This is for the case that in future L^AT_EX versions the macro can have a different definition, or that another package has already redefined it.

The command `\InputIfFileExists` is used by L^AT_EX 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}
\AfterClass+{class}{instructions}
\AfterClass!{class}{instructions}
\AfterAtEndOfClass{class}{instructions}
\AfterPackage{package}{instructions}
\AfterPackage*{package}{instructions}
\AfterPackage+{package}{instructions}
\AfterPackage!{package}{instructions}
\AfterAtEndOfPackage{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 are a little bit different. They 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.

v3.09 The plussed version executes the *instructions* after loading of the class or package has been finished. The difference to the starred version is only valid, if loading of the class or package already started but has not been finished yet. Nevertheless, *instructions* will be executed before the instructions of `\AtEndOfClass` or `\AtEndOfPackage` when loading of the class or package has not been finished already.

If a class uses `\AtEndOfClass` or a package uses `\AtEndOfPackage` to execute instructions after the class or package file has been loaded completely, and if you want to execute *instructions* after the instructions of these commands, you may use the exclamation mark version, `\AfterClass!` respectively `\AfterPackage!`.

v3.09 If you want to do this only in the case the class or package will be loaded later, and if you want to execute *instructions* outside the context of the class or package, that will be loaded, you may use `\AfterAtEndOfClass` for classes and `\AfterAtEndOfPackage` for packages.

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 L^AT_EX user. Suppose a document that should be available as a PS file, using L^AT_EX and dvips, as well as a PDF file, using pdfL^AT_EX. 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 pdfL^AT_EX method, since here hyperlinks can be broken across multiple lines. However, if a `hyperref` driver for dvips or hyperT_EX 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 [Tea06]). The new lines have to be inserted directly after the `\ProvidesFile` line, thus immediately prior to the execution of the options `dvips` or `pdftex`.

```

\BeforeClosingMainAux{instructions}
\AfterReadingMainAux{instructions}

```

Package authors often want to write something into the `aux`-file after the last document page have been shipped out. To do so, often

```

\AtEndDocument{%
  \if@filesw
    \write\@auxout{%
      \protect\writethistoaux%
    }%
  \fi
}

```

is used. Nevertheless this is not a real solution of the problem. If the last page of the document already have been shipped out before `\end{document}`, the code above will not result in any writing into the `aux`-file. If someone would try to fix this new problem using `\immediate` just before `\write`, the inverse problem would occur: If the last page was not shipped out before `\end{document}` the `\writethistoaux` would be written into `aux`-file before ship-out the last page. Another often seen suggestion for this problem therefore is:

```

\AtEndDocument{%
  \if@filesw
    \clearpage
    \immediate\write\@auxout{%
      \protect\writethistoaux%
    }%
  \fi
}

```

```
}
```

This suggestion has a disadvantage again: The ship-out of the last page has been enforced by the `\clearpage`. After that, instructions like

```
\AtEndDocument{%
  \par\vspace*{\fill}%
  Note at the end of the document.\par
}
```

would not any longer output the note at the end of the last page of the document but at the end of one more page. Additionally `\writethistoaux` would be written one page too early into the aux-file again.

The best solution for this problem would be, to write to the aux-file immediately after the final `\clearpage`, that is part of `\end{document}`, but just before closing the aux-file. This is the purpose of `\BeforeClosingMainAux`:

```
\BeforeClosingMainAux{%
  \if@files
    \immediate\write\@auxout{%
      \protect\writethistoaux%
    }%
  \fi
}
```

This would be successful even if the final `\clearpage` inside of `\end{document}` would not really ship-out any page or if someone have had used `\clearpage` in the argument of `\AtEndDocument`.

Nevertheless there one important limitation using `\BeforeClosingMainAux`: You should not use a typeset instruction inside the *instructions* of `\BeforeClosingMainAux`! If you miss this limitation the result would be as unpredictable as the results of the problematic suggestions using `\AtEndDocument` upward.

v3.03

Command `\AfterReadingMainAux` actually executes the *instructions* just after closing and input of the aux-file inside of `\end{document}`. This will make sense only in some cases, e.g., to show statistic information, that will be valid only after input of the aux-file, or to write such information into the log-file, or to implement additional *rerun* requests. Typeset instructions are even more critical inside these *instructions* that inside the argument of `\BeforeClosingMainAux`.

13.3. Replacing Files at Input

All previous sections in this chapter describe commands to execute instructions before or after input of a file, class, or package. Package `scrfile` also provides commands to input another file, class, or package instead of the one, that has been declared.


```
\ReplaceInput{source file name}{replacement file name}
```

v2.96

This command defines a replacement for the file of the first argument: *source file name*, by the file of the second argument: *replacement file name*. If L^AT_EX will be instructed to input the file with *source file name* at any time afterward, the file with the *replacement file name* will be input instead. The replacement definition will be valid for all files, that the user will input with `\InputIfFileExists` and for all files, that will be input with a command, that uses `\InputIfFileExists` internally. To do so, scrfile redefined `\InputIfFileExists`.

Example: You want L^AT_EX to input file `\jobname.xua` instead of file `\jobname.aux`. This may be done using

```
\ReplaceInput{\jobname.aux}{\jobname.xua}
```

Additionally you may replace `\jobname.xua` by `\jobname.uxa` using:

```
\ReplaceInput{\jobname.xua}{\jobname.uxa}
```

This will also replace input of `\jobname.aux`, i.e., while `\end{document}`, by `\jobname.uxa`. As you see, the whole replacement chain will be executed.

Nevertheless a round robin replacement like

```
\ReplaceInput{\jobname.aux}{\jobname.xua}
\ReplaceInput{\jobname.xua}{\jobname.aux}
```

would result in a *stack size error*. So it is not possible to define a replacement of a file by itself directly or indirectly.

In theory it would also be possible to replace a package or class by another one. But L^AT_EX would recognize the usage of the wrong file name in this case. A solution for this problem will be shown next.

```
\ReplaceClass{source class}{replacement package}
```

```
\ReplacePackage{source package}{replacement package}
```

v2.96

Classes or packages should never be replaced using previously described command **\ReplaceInput**. Using this command would result in a L^AT_EX warning because of class or package name not according the file name.

Example: You replace package `fancyhdr` by package `scrpage2` inconsiderately using

```
\ReplaceInput{fancyhdr.sty}{scrpage2.sty}
```

Loading `fancyhdr`, would result in

```
LaTeX warning: You have requested 'scrpage2',
                but the package provides 'fancyhdr'.
```

after this. Users may be confused by such a warning, because they've used, e.g., `\usepackage{fancyhdr}` and never requested package `scrpage2` on their own. But `scrfile` replaced the input of `fancyhdr.sty` by `scrpage2.sty` because of your replacement definition.

A solution for this problem would be, to use `\ReplaceClass` or `\ReplacePackage` instead of `\ReplaceInput`. Please note, that in this case you have to use the names of the classes or packages only instead of the whole file name. This is similar to usage of `\documentclass` and `\usepackage`.

The class replacement would perform for all classes, that will be loaded using `\documentclass`, `\LoadClassWithOptions`, or `\LoadClass`. The package replacement would perform for all packages, that will be loaded using `\usepackage`, `\RequirePackageWithOptions`, or `\RequirePackage`.

Please note, that the *replacement class* or the *replacement package* will be loaded with the same options, the *source class* or *replacement class* would until it has been replaced. Replacement of a class or package by a class or package, that does not support a requested option, would result in a warning or even an error message. But you may declare such missing options using `\BeforeClass` or `\BeforePackage`.

Example: Assumed, package `oldfoo` should be replaced by `newfoo`. This may be done using:

```
\ReplacePackage{oldfoo}{newfoo}
```

Assumed the old package provides an option `oldopt`, but the new package does not. Using

```
\BeforePackage{newfoo}{%
  \DeclareOption{oldopt}{%
    \PackageInfo{newfoo}%
      {option 'oldopt' not supported}%
  }}%
```

additionally, would declare this missing option for package `newfoo`. This would avoid warning message about unsupported options.

However, if package `newfoo` supports an option `newopt`, that should be used instead of option `oldopt` of old package `oldfoo`, this may achieved using:

```
\BeforePackage{newfoo}{%
  \DeclareOption{oldopt}{%
    \ExecuteOptions{newopt}%
  }}%
```

Last but not least different default options may be selected, that should be valid while package replacement:

```
\BeforePackage{newfoo}{%
```

```

\DeclareOption{oldopt}{%
  \ExecuteOptions{newopt}%
}%
\PassOptionsToPackage{newdefoptA,newdefoptB}%
{newfoo}%
}

```

or somehow more directly:

```

\BeforePackage{newfoo}{%
  \DeclareOption{oldopt}{%
    \ExecuteOptions{newopt}%
  }%
}%
\PassOptionsToPackage{newdefoptA,newdefoptB}%
{newfoo}%

```

To replace classes package scrfile has to be loaded before the class using `\RequirePackage` instead of `\usepackage`.

```

\UnReplaceInput{file name}
\UnReplacePackage{package}
\UnReplaceClass{class}

```

v3.12 A replacement definition can be removed using one of these commands. The replacement definition of a input file should be removed using `\UnReplaceInput`, the replacement definition of a package should be removed using `\UnReplacePackage`, and the replacement definition of a class should be removed using `\UnReplaceClass`.

13.4. Prevent File Loading

v3.08 Especially classes or packages, that have been made for companies or institutes, often load a lot of packages not needed by the classes or packages itself but only because the users often use them. Now, if such a not essential package causes any kind of problem, loading of that package has to prevented. For this purpose scrfile again provides a solution.

```

\PreventPackageFromLoading[instead code]{package list}
\PreventPackageFromLoading*[instead code]{package list}

```

v3.08 Calling this command before loading a package using `\usepackage`, `\RequirePackage`, or `\RequirePackageWithOptions` will prevent the package from being loaded effectively if the package is part of the *package list*.

Example: Assumed you're working in a company, that uses font Latin-Modern for all kind of documents. Because of this the company class, `compycls` contains the lines:

```
\RequirePackage[T1]{fontenc}
\RequirePackage{lmodern}
```

But now, you want to use X_YLaTeX oder LuaLaTeX the first time. In this case loading of fontenc would not be a good suggestion and Latin-Modern would be the default font of the recommended package fontspec. Because of this you want to prevent both packages from being loaded. This may be done, loading the class like this:

```
\RequirePackage{scrfile}
\PreventPackageFromLoading{fontenc,lmodern}
\documentclass{firmenci}
```

The example above also shows, that package scrfile may be loaded before the class. In this case \RequirePackage has to be used, because \usepackage before \documentclass is not permitted.

If *package list* is empty or contains a package, that already has been loaded, \PreventPackageFromLoading will warn. If you'd prefer an info at the log-file only, you may use \PreventPackageFromLoading* instead.

The optional argument may be used to execute code instead of loading the package. But you must not load another packages or files inside *instead code*. See \ReplacePackage in section 13.2 on page 337 for information about replacing a package by another one. Note also, that the *instead code* will be executed several times, if you try to load the package more than once!

```
\StorePreventPackageFromLoading{\command}
\ResetPreventPackageFromLoading
```

\StorePreventPackageFromLoad defines \command to be the current list of packages, that should be prevented from being loaded. In opposite to this, \ResetPreventPackageFromLoading resets the list of packages, that should be prevented from being loaded. After \ResetPreventPackageFromLoading all packages may be loaded again.

Example: Assumed, you really need a package inside your own package and you want the user inhibit to prevent loading of that package with \PreventPackageFromLoading. Because of this, you reset the package preventing list before loading the package:

```
\ResetPreventPackageFromLoading
\RequirePackage{foo}
```

Unfortunately the complete prevention list of the user would be lost after that. To avoid this, you first store the list and restore it at the end:

```
\newcommand*{\Users@PreventList}{}%
\StorePreventPackageFromLoading\Users@PreventList
```

```

\ResetPreventPackageFromLoading
\RequirePackage{foo}
\PreventPackageFromLoading{\Users@PreventList}

```

Please note, that `\StorePreventPackageFromLoading` would define `\Users@PreventList` even if it already has been defined before. In other words: `\StorePreventPackageFromLoading` overwrites existing `\command` definitions without care. Because of this, `\newcommand*` has been used in the example to get an error message, if `\Users@PreventList` has already been defined.

At this point please note, that everybody who manipulates the list, that has been stored using `\StorePreventPackageFromLoading` is responsible for the correct restorability. For example the list elements must be separated by comma, must not contain white space or group braces, and must be fully expandable.

Please note, that `\ResetPreventPackageFromLoading` does not clean the *instead code* of a package. Only the execution is not done as long as the prevention is not reactivated.

```

\UnPreventPackageFromLoading{package list}
\UnPreventPackageFromLoading*{package list}

```

v3.12

Instead of resetting the whole list of packages, that should be prevented from being loaded, you may also remove some packages from that list. The star version of the command does also clean the *instead code*. So reactivation of the prevent package list, e. g., from a stored one, will not reactivate the *instead code* of the packages.

Example: Assuming, you want to prevent a package `foo` from being loaded, but you do not want an already stored *instead code* to be executed. Instead of that code, you're own *instead code* should be executed. You can do this:

```

\UnPreventPackageFromLoading*{foo}
\PreventPackageFromLoading[\typeout{Stattdessencode}]{foo}

```

For `\UnPreventPackageFromLoading` it does not matter whether or not the package has been prevented from being loaded before.

Surely you can use the command also to remove only the *instead code* of all packages:

```

\StorePreventPackageFromLoading\TheWholePreventList
\UnPreventPackageFromLoading*{\TheWholePreventList}
\PreventPackageFromLoading{\TheWholePreventList}

```

In this case the packages, that has been prevented from being loaded, are still prevented from being loaded, but their *instead code* has been cleaned and will not be executed any longer.

Economise and Replace Files Using `scrwfile`

\TeX supports 18 write handles only. Handle 0 is used by \TeX itself (log file). \LaTeX needs at least handle 1 for `\@mainaux`, handle 2 for `\@partaux`, one handle for `\tableofcontents`, one handle for `\listoffigures`, one handle for `\listoftables`, one handle for `\makeindex`. So there are 11 left. Seems a lot and enough. But every new type of float, every new index and several other packages, e.g., `hyperref` need write handles, too.

The bottom line is, that this eventually will result in the error message:

```
! No room for a new \write .
\check ... \else \errmessage {No room for a new #3}
\fi
```

There is an additional disadvantage of immediately opening a new write handle for every table of contents, list of figures, list of tables etc. These are not only set by the corresponding commands, they also could not be set once more, because their helper files are empty after the corresponding commands until the end of the document.

Package `scrwfile` provides an amendment of the \LaTeX kernel, that solves both problems.

14.1. General Modifications of the \LaTeX Kernel

To allocate a new file handle eg. for `\listoffigures` or `\listoftables` \LaTeX classes use the \LaTeX kernel command `\@starttoc`. This command not only inputs the associated helper file but also allocates a new write handle for the associated helper file and opens it for writing. Nevertheless, if afterwards new entries to these lists of floats are added using `\addcontentsline`, then these file handles are not used immediately, instead \LaTeX writes `\@writefile` commands into the `aux`-file. Only while reading the `aux`-file while the end of the document, those `\@writefile` commands become real write operations on the helper files. Additionally \LaTeX does not close the helper files explicitly. Instead \LaTeX relies on \TeX to close all open files at the end.

Thus all the helper files are open throughout the entire process. However the content is written at `\end{document}`. The basic idea of `scrwfile` is tackling this contradiction by redefining `\@starttoc` and `\@writefile`.

Surely, changes of the \LaTeX kernel may result in incompatibilities with other packages. In case of `scrwfile`, clashes may arise with all packages also redefining `\@starttoc` or `\@writefile`. Sometimes changing the order of loading the packages may help.

However, only few such problems have been reported eventhough several users have tested the package for one year before its first release. See [section 14.5](#) for more information about known incompatibilities. If you find such a problem, please contact the KOMA-Script author.

14.2. The Single File Feature

At the point the package is loaded using, e.g.,

```
\usepackage{scrwfile}
```

`scrwfile` will redefine `\@starttoc` to not longer allocate a write handle or open any file for writing. Immediately before closing the `aux`-file in `\end{document}` it will redefine `\@writefile` to no longer write into the usual helper files but into one single new file with file extension `wrt`. After reading the `aux`-file this `wrt`-file will be processed once per helper file. This means, that not all of the helper file have to be open at the same time, but only one at a time. And this single file will be closed afterwards and the write handle is not longer needed after it is closed. An internal write handle of L^AT_EX is used for this. So `scrwfile` doesn't need any own write handle.

Because of this, even if only one table of contents should be generated, loading `scrwfile` gives one extra write file handle, e.g., for bibliographies, indexes, glossaries and similar, that are not using `\@starttoc`. Additionally the number of tables of contents and lists of whatever, that use `\@starttoc` is not limited any longer.

14.3. The Clone File Write Feature

Sometimes it is useful to input a file not only once but several times. As `\@starttoc` does not open files for writing any more, this can be done by simply using `\@starttoc` several times with the same extension. But sometimes you may have additional entries in only some of the content directories. `scrwfile` allows to copy all entries of a file to another file, too. We call this cloning.

```
\TOCclone[heading]{source}{destination}
```

activates the clone feature for files with extensions *source* and *destination*. All entries to the file `\jobname.source` will be added to `\jobname.destination`.

If extension *destination* is a new one, *destination* will be added to the list of known extensions using the KOMA-Script package `tocbasic`.

If the optional argument *heading* is given, a new list-of macro `\listofdestination` is defined. *heading* will be used as section (or chapter) heading of this list. In this case several `tocbasic` features of the *source* will be copied to *destination*, if and only if they have been set up when `\TOCclone` was used. Feature *nobabel* will always be set, because the language selection commands are part of the helper file and would be cloned, anyway.

Example: Assumed, you want a short table of contents with only the chapter level but an additional entry with the table of contents:

```
\usepackage{scrwfile}
\TOCclone[Short \contentsname]{toc}{stoc}
```

This would create a new table of contents with the heading “Short Contents”. The new table of contents uses a helper file with extension `stoc`. All entries to the helper file with extension `toc` will also be copied to this new helper file.

The new short table of contents should only have the chapter entries. This may be done using:

```
\addtocontents{stoc}{\protect\value{tocdepth}=0}
```

Normally you cannot write into a helper file before `\begin{document}`. But using `scrwfile` changes this. So the code above is correct already after loading `scrwfile`.

To show the new short contents of helper file extension `stoc` we use

```
\listofstoc
```

somewhere after `\begin{document}`.

If we also want an entry for the table of contents at the short contents, we cannot use

```
\addtocontents{toc}{% write to the Contents
\protect\addcontentslinedefault{stoc}% write to Short Contents
{chapter}% a chapter entry with
{\contentsname}% the Contents' name
}
```

because the `\addcontentsline` command would be copied to `stoc` too. So we cannot add the command to the `toc`-file. Package `toctbasic` may be used to solve this:

```
\BeforeStartingTOC[toc]{%
\addcontentslinedefault{stoc}{chapter}
{\protect\contentsname}%
}
```

However, this needs, that the file with extension `toc` is under control of package `toctbasic`, which is indeed the case within all KOMA-Script classes. See [section 15.2](#) on [page 353](#) for more information about [page 353](#).

14.4. Note on State of Development

Eventhough this package has been tested by several users and even is in productivity usage several times it is still under construction. Therefore, there might be amendments especially to the internal functionality. Most likely the package will be extended. Some code for extensions is already in the package. However, there is currently no user documentation available, as up to now nobody has requested any of these extensions.

14.5. Known Package Incompatibilities

As mentioned in [section 14.1](#), `scrwfile` redefines some commands of the \LaTeX kernel. This happens not only while loading the package, but indeed at different times of processing a document, e. g., just before reading the `aux`-file. This results in incompatibility with packages that also redefine these commands at run-time.

The `titletoc` package is an example for such an incompatibility. That package redefines `\@writefile` under some conditions at run-time. If you use both, `scrwfile` and `titletoc`, there is no warranty for the correct behaviour of neither of them. This is neither an error of `titletoc` nor of `scrwfile`.

Management of Tables and Lists of Contents Using tocbasic¹

The main purpose of package `tocbasic` is to provide features for authors of classes and packages to create own tables or lists of contents like the list of figures and the list of tables and thereby allow other classes or packages some types of control over these. For examples package `tocbasic` delegates language control of all these tables and lists of contents to package `babel` (see [BB13]). So automatic change of language will be provided inside all these tables and lists of contents. Using `tocbasic` will exculpate authors of classes and packages from implementation of such features.

KOMA-Script itself uses `tocbasic` not only for the table of contents but also for the already mentioned lists of figures and tables. In this chapter we call all kinds of tables of contents or lists of contents simply TOC.

15.1. Basic Commands

Basic commands are used to handle a list of all file name extensions known for files representing a table or list of contents. We call these auxiliary files² TOC-files independent from the file name extension that is used. Entries to such files are typically written using `\addtocontents` or `\addcontentsline`. Later in this chapter you will learn to know recommended, additional commands. There are also commands to do something for all known extensions. Additionally, there are commands to set or unset features of a file name extension or the file represented by the extension. Typically an extension also has an owner. This owner may be a class or package or a term decided by the author of the class or package using `tocbasic`, e.g., KOMA-Script uses the owner `float` for list of figures and list of tables, and the file name of the class file as owner for the table of contents.

```
\ifattoclist{extension}{truepart}{false part}
```

This command may be used to ask, whether or not a file name *extension* is already a known extension. If the *extension* is already known the *true instructions* will be used, otherwise the *false instructions* will be used.

Example: Maybe you want to know if the file name extension “foo” is already in use to report an error, if you can not use it:

```
\ifattoclist{foo}{%
  \PackageError{bar}{%
    extension ‘foo’ already in use%
  }{%
```

¹Translation of this chapter has been made by the package author and needs editing!

²Here we do not talk about the `aux`-file but the auxiliary files used indirect via the `aux`-file, e.g., the `toc`-file, the `lof`-file, or the `lot`-file.

```

Each extension may be used only
once.\MessageBreak
The class or another package already
uses extension 'foo'.\MessageBreak
This error is fatal!\MessageBreak
You should not continue!}%
}{%
\PackageInfo{bar}{using extension 'foo'}%
}

```

```
\addtotoclist[owner]{extension}
```

This command adds the *extension* to the list of known extensions. But if the *extension* is a known one already, then an error will be reported to avoid double usage of the same *extension*.

If the optional argument, *[owner]*, was given, this *owner* will be stored to be the owner of the *extension*. If the optional argument has been omitted, tocbasic tries to find out the file name of the current processed class or package and stores this as owner. This will fail if `\addtotoclist` was not used, loading a class or package but using a command of a class or package after loading this class or package. In this case the owner will be set to “.”.

Please note that an empty *owner* is not the same like omitting the optional argument with the braces. An empty argument would result in an empty owner.

Example: You want to add the extension “foo” to the list of known extension, while loading your package with file name “bar.sty”:

```
\addtotoclist{foo}
```

This will add the extension “foo” with owner “bar.sty” to the list of known extensions, if it was not already in the list of known extensions. If the class or another package already added the extension you will get the error:

```
Package tocbasic Error: file extension 'foo' cannot be used twice
```

```
See the tocbasic package documentation for explanation.
```

```
Type H <return> for immediate help.
```

and after typing h and pressing the return key you will get the help:

```
File extension 'foo' is already used by a toc-file, while bar.sty
tried to use it again for a toc-file.
```

```
This may be either an incompatibility of packages, an error at a ↵
package,
or a mistake by the user.
```

Maybe your package has a command, that creates list of files dynamically. In this case you should use the optional argument of `\addtotoclist` to set the owner.

```

\newcommand*{\createnewlistofsomething}[1]{%
  \addtotoclist[bar.sty]{#1}%
  % Do something more to make this list of something available
}

```

If the user calls now, e.g.,

```
\createnewlistofsomething{foo}
```

this would add the extension “foo” with the owner “bar.sty” to the list of known extension or report an error, if the extension is already in use.

You may use any owner you want. But it should be unique! So, if you would be, e.g., the author of package float you could use for example owner “float” instead of owner “float.sty”, so the KOMA-Script options for the list of figures and the list of tables will also handle the lists of this package. Those are already added to the known extensions when the option is used. This is because KOMA-Script already registers file name extension “lof” for the list of figures and file name extension “lot” for the list of tables with owner “float” and sets options for this owner. Package `scrhack` redefines some of package float’s commands to do this.

```
\AtAddToTocList[owner]{instructions}
```

This command adds the *instructions* to an internal list of instructions that will be processed whenever a file name extension with the given *owner* will be added to the list of known extensions using `\addtotoclist`. The optional argument is handled in the same way as with the command `\addtotoclist`. With an empty *owner* you may add `{instructions}`, that will be processed at every successful `\addtotoclist`, after processing the instructions for the individual owner. While processing the *instructions*, `\@currentx` will be set to the extension of the currently added extension.

Example: tocbasic itself uses

```

\AtAddToTocList[]{%
  \expandafter\tocbasic@extend@babel
  \expandafter{\@currentx}%
}

```

to add every extension to the tocbasic-internal babel handling of files.

The two `\expandafter` commands are needed, because the argument of `\tocbasic@extend@babel` has to be expanded! See the description of `\tocbasic@extend@babel` at [section 15.4, page 368](#) for more information.

```
\removefromtoclist[owner]{extension}
```

This command removes the *extension* from the list of known extensions. If the optional argument, *[owner]*, was given, the *extension* will only be removed if it was added by this *owner*. See description of `\addtotoclist` for information of omitting optional argument. Note that an empty *owner* is not the same like omitting the optional argument, but removes the *extension* without any owner test.

```
\doforeachtocfile[owner]{instructions}
```

Until now you’ve learned to know commands that result in more safety in handling file name extensions, but also needs some additional effort. With `\doforeachtocfile` you will win for this. The command provides to processes *instructions* for every known file name extension of the given *owner*. While processing the *instructions* `\@currentx` is the extension of the current file. If you omit the optional argument, *[owner]*, every known file name extension independent from the owner will be used. If the optional argument is empty, only file name extensions with an empty owner will be processed.

Example: If you want to type out all known extensions, you may simply write:

```
\doforeachtocfile{\typeout{\@currentx}}
```

and if only the extensions of owner “foo” should be typed out:

```
\doforeachtocfile[foo]{\typeout{\@currentx}}
```

```
\tocbasicautomode
```

This command redefines L^AT_EX kernel macro `\@starttoc` to add all not yet added extensions to the list of known extensions and use `\tocbasic@starttoc` instead of `\@starttoc`. See [section 15.4, page 368](#) for more information about `\tocbasic@starttoc` and `\@starttoc`.

This means that after using `\tocbasicautomode` every table of contents or list of something, that will be generated using `\@starttoc` will be at least partially under control of `tocbasic`. Whether or not this will make the wanted result, depends on the individual TOC. At least the `babel` control extension for all those TOCs will work. Nevertheless, it would be better if the author of the corresponding class or package will use `tocbasic` explicitly. In that case additional advantages of `tocbasic` may be used that will be described at the following sections.

15.2. Creating a Table or List of Contents

In the previous section you’ve seen commands to handle a list of known file name extensions and to trigger commands while adding a new extension to this list. You’ve also seen a command to do something for all known extensions or all known extensions of one owner. In this section you will learn to know commands to handle the file corresponding with an extension or the list of known extensions.

```
\addtoeachtocfile[owner]{content}
```

This command writes *content* to the TOC-files of every known file name extension of *owner* using L^AT_EX kernel command `\addtocontents`. If you omit the optional argument, *content* is written to the files of every known file name extension. Furthermore, the practical file name is built from `\jobname` and the file name extension. While writing the *content*, `\@currentx` is the extension of the currently handled file.

Example: You may add a vertical space of one text line to all toc-files.

```
\addtoeachtocfile{%
  \protect\addvspace{\protect\baselineskip}%
}
```

And if you want to do this, only for the TOC-files of owner “foo”:

```
\addtoeachtocfile[foo]{%
  \protect\addvspace{\protect\baselineskip}%
}
```

Commands, that shouldn’t be expanded while writing, should be prefixed by `\protect` in the same way like they should be in the argument of `\addtocontents`.

```
\addxcontentsline{extension}{level}[number]{text}
```

The command `\addxcontentsline` adds an entry of given *level* to TOC-file with file name *extension*. If the `{number}` is empty or omitted the entry will not have number for the entry with the given *text*. Entries without number may be left aligned to the number of the numbered entries of the same *level* or indented like the text of the numbered entries of the same *level*, depending on the `numberline` feature.

Example: Maybe you are not using a KOMA-Script class but need a not numbered chapter with entry to the table of contents. This may be done using

```
\cleardoublepage
\csname phantomsection\endcsname
\addxcontentsline{toc}{chapter}
  {Chapters without Numbers}
\chapter*{Chapters without Numbers}
\markboth{Chapters without Numbers}{}

```

As you can see, you simply have to replace usual `\addcontentsline` by `\addxcontentsline` to support the tocbasic feature `numberline`.

Note that `\addxcontentsline` uses `\addlevelextension` entry if such a macro exists and `\tocbasic@addxcontentsline` otherwise. Therefore you cannot define a macro `\addlevelextension` entry using `\addxcontentsline` but `\tocbasic@addxcontentsline`.

It is recommended to use `\addxcontentsline` instead of `\addcontentsline` whenever possible.

```
\addcontentslinetoeachtocfile[owner]{level}{contentsline}
\addxcontentslinetoeachtocfile[owner]{level}[number]{text}
```

The first command is something like `\addcontentsline` from L^AT_EX kernel. In difference to that it writes the *contentsline* not only into one file, but into all files of all known file name extensions or of all known file name extensions of a given owner.

The command `\addxcontentslinetoeachtocfile` is similar but uses `\addxcontentsline` instead of `\addcontentsline` and therefore supports tocbasic feature `numberline`.

Example: You are a class author and want to write the chapter entry not only to the table of contents TOC-file but to all TOC-files, while `#1` is the title, that should be written to the files.

```
\addxcontentslinetoeachtocfile
      {chapter}[\thechapter]{#1}%
```

In this case the current chapter number should be expanded while writing into the file. So it isn't protected from expansion using `\protect`.

While writing `\@currentx` is the file name extension of the file into which *contentsline* will be written.

It is recommended to use `\addxcontentslinetoeachtocfile` instead of `\addcontentslinetoeachtocfile` whenever possible.

```
\listoftoc[list of title]{extension}
\listoftoc*{extension}
\listofeachtoc[owner]
\listofextensionname
```

These commands may be used to print the TOC corresponding to file name *extension*. The star version `\listoftoc*` needs only one argument, the *extension* of the file. It does setup the vertical and horizontal spacing of paragraphs, calls before hooks, reads the file, and last but not least calls the after hooks. You may interpret it as direct replacement of the L^AT_EX kernel macro `\@starttoc`.

The version without star, prints the whole file with title, optional table of contents entry, and running heads. If the optional argument *[list of title]* was given, it will be used as title term, optional table of contents entry and running head. Please note: If the optional argument is empty, this term will be empty, too! If you omit the optional argument, but `\listofextensionname` was defined, that will be used. If that is also not defined, a standard replacement name will be used and reported by a warning message.

The command `\listofeachtoc` outputs all lists of something of the given *owner* or of all known file extensions. Thereby `\listofextensionname` should be defined to get the correct titles.

It is recommended to define `\listofextensionname` for all used file name extensions, because the user itself may use `\listofeachtoc`.

Example: Assumed, you have a new “list of algorithms” with extension `loa` and want to show it:

```
\listoftoc[List of Algorithms]{loa}
```

will do it for you. But maybe the “list of algorithms” should not be set with a title. So you may use

```
\listof*{loa}
```

Note that in this case no entry at the table of contents will be created, even if you’d used the setup command above. See command `\setuptoc` at [page 354](#) for more information about the attribute of generating entries into the table of contents using `\setuptoc`.

If you’ve defined

```
\newcommand*{\listofloaname}{%
  List of Algorithms%
}
```

before, then

```
\listoftoc{loa}
```

would be enough to print the list of algorithms with the wanted heading. For the user it may be easier to operate, if you’d define

```
\newcommand*{\listofalgorithms}{\listoftoc{loa}}
```

additionally.

Because L^AT_EX normally opens a new file for each of those lists of something immediately, the call of each of those commands may result in an error like:

```
! No room for a new \write .
\ch@ck ... \else \errmessage {No room for a new #3}
\fi
```

if there are no more write handles left. Loading package `scrwfile` (see [chapter 14](#)) may solve this problem.


```
\BeforeStartingTOC[extension]{instructions}
\AfterStartingTOC[extension]{instructions}
```

Sometimes it's useful, to process *instructions* immediately before reading the auxiliary file of a TOC. These commands may be used to process *instructions* before or after loading the file with given *extension* using `\listoftoc*`, `\listoftoc`, or `\listofeachtoc`. If you omit the optional argument (or set an empty one) the general hooks will be set. The general before hook will be called before the individual one and the general after hook will be called after the individual one. While calling the hooks `\@currentx` is the extension of the TOC-file and should not be changed.

An example for usage of `\BeforeStartingTOC` may be found in [section 14.3](#) at [page 344](#).

```
\BeforeTOCHead[extension]{instructions}
\AfterTOCHead[extension]{instructions}
```

This commands may be used to process *instructions* before or after setting the title of a TOC corresponding to given file name *extension* using `\listoftoc*` or `\listoftoc`. If you omit the optional argument (or set an empty one) the general hooks will be set. The general before hook will be called before the individual one and the general after hook will be called after the individual one. While calling the hooks `\@currentxIndexCmd@currentx` is the extension of the corresponding file and should not be changed.

```
\MakeMarkcase{text}
```

Whenever tocbasic sets a mark for a running head, The text of the mark will be an argument of `\MakeMarkcase`. This command may be used, to change the case of the letters at the running head if wanted. The default is, to use `\@firstofone` for KOMA-Script classes. This means the text of the running head will be set without change of case. `\MakeUppercase` will be used for all other classes. If you are the class author you may define `\MakeMarkcase` on your own. If `sclayer` or another package, that defines `\MakeMarkcase` will be used, tocbasic will not overwrite that definition.

Example: For incomprehensible reasons, you want to set the running heads in lower case letters only. To make this automatically for all running heads, that will be set by tocbasic, you define:

```
\let\MakeMarkcase\MakeLowercase
```

Please allow me some words about `\MakeUppercase`, First of all this command isn't fully expandable. This means, that problems may occur using it in the context of other commands. Beyond that typographers accord, that whenever setting whole words or phrases in capitals, letter spacing is absolutely necessary. But correct letter spacing of capitals should not be done with a fix white space between all letters. Different pairs of letters need different space between each other. Additional some letters build holes in the text, that have to be taken

into account. Packages like `ulem` or `soul` doesn't provide this and `\MakeUppercase` does not do anything like this. Also automatic letter spacing using package `microtype` is only one step to a less-than-ideal solution, because it cannot recognize and take into account the glyphs of the letters. Because of this typesetting whole words and phrases is expert work and almost ever must be hand made. So average users are recommended to not do that or to use it only spare and not at exposed places like running heads.

```
\defptoheading{extension}{definition}
```

The package `tocbasic` contains a standard definition for typesetting headings of TOCs. This standard definition is configurable by several features, described at `\setuptoc` next. But if all those features are not enough, an alternative heading command may be defined using `\defptoheading`. Thereby *extension* is the file name extension of the corresponding TOC-file. The *definition* of the heading command may use one single parameter `#1`. While calling the newly defined command inside of `\listoftoc` or `\listofeachtoc` that `#1` will be replaced by the corresponding heading term.

```
\setuptoc{extension}{feature list}
\unsettoc{extension}{feature list}
```

This commands set up and unset features bound to a file name *extension*. The *feature list* is a comma separated list of single features. `tocbasic` does know following features:

leveldown uses not the top level heading below `\part` — `\chapter` if available, `\section` otherwise — but the first sub level. This feature will be evaluated by the internal heading command. On the other hand, if an user defined heading command has been made with `\defptoheading`, that user is responsible for the evaluation of the feature. The KOMA-Script classes set this feature using option `listof=leveldownimportantlistof=leveldown` for all file name extensions of the owner `float`.

nobabel prevents usage of the language switch of `babel` at the TOC-file with the corresponding file name *extension*. This feature should be used only for auxiliary files, that contain text in one language only. Changes of the language inside of the document will not longer regarded at the TOC-file. Package `scrwfile` uses this feature also for clone destinations, because those will get the language change from the clone source already.

Please note, the feature is executed while adding a file extension to the list of known file extension. Changing the feature afterwards would not have any effect.

v3.17

noparskipfake prevents usage of an extra `\parskip` before switching `\parskip` off. In general, the consequence of this feature for documents using paragraph distance is less vertical space between heading and first entry than between normal headings and normal text.

v3.10 **noprotrusion** prevents disabling character protrusion at the TOC. Character protrusion at the TOCs will be disabled by default if package `microtype` or another package, that supports `\microtypesetup`, was loaded. So if you want protrusion at a TOC, you have to set this feature. But note, with character protrusion TOC-entries may be printed wrong. This is a known issue of character protrusion.

numbered uses a numbered heading for the TOC and because of this also generates an entry to the table of contents. This feature will be evaluated by the internal heading command. On the other hand, if an user defined heading command has been made with `\deftocheading`, that user is responsible for the evaluation of the feature. The KOMA-Script classes set this feature using option `listof=numbered` for all file name extensions of the owner `float`.

v3.12 **numberline** redefines `\nonumberline` to use `\numberline`. With this the not numbered entries generated by KOMA-Script or using `\nonumberline` at the very beginning of the last argument of `\addcontentline` will also be indented like numbered entries of the same type. Using `\numberline` for entries without number can have additional side effects if you use entry style `tocline`. See style attribute `breakafternumber` and `entrynumberformat` in [table 15.1](#) from [page 360](#) downwards.

v3.20 KOMA-Script classes set this feature for the file name extensions of the owner `float` if you use option `listof=numberline` and for file name extension `toc` if you use option `toc=numberline`. Analogous they reset this feature if you use `listof=nonumberline` or `toc=nonumberline`.

v3.01 **onecolumn** typesets the corresponding TOC with internal one column mode of `\onecolumn`. This will be done only, if the corresponding table of contents or list of something does not use feature `leveldown`. The KOMA-Script classes `scrbook` and `scrreprt` activate this feature with `\AtAddToTocList` (see [section 15.1](#), [page 348](#)) for all TOCs with owner `float` or with themselves as owner. With this, e. g., the table of contents, the list of figures and the list of tables of both classes will be single columned automatically. The multiple-column-mode of package `multicol` will not be recognised or changed by this option.

totoc writes the title of the corresponding TOC to the table of contents. This feature will be evaluated by the internal heading command. On the other hand, if an user defined heading command has been made with `\deftocheading`, that user is responsible for the evaluation of the feature. The KOMA-Script classes set this feature using option `listof=totoc` for all file name extensions of the owner `float`.

Classes and packages may know features, too, e. g., the KOMA-Script classes know following additional features:

`chapteratlist` activates special code to be put into the TOC at start of a new chapter. This code may either be vertical space or the heading of the chapter. See `listof` in [section 3.20, page 133](#) for more information about such features.

Example: Because KOMA-Script classes use `tocbasic` for the list of figures and list of tables, there’s one more way to remove chapter structuring at those:

```
\unsettoc{lof}{chapteratlist}
\unsettoc{lot}{chapteratlist}
```

And if you want to have the chapter structuring of the KOMA-Script classes at your own list of algorithms with file name *extension* “loa” from the previous examples, you may use

```
\setuptoc{loa}{chapteratlist}
```

And if classes with `\chapter` should also force single column mode for the list of algorithms you may use

```
\ifundefinedorrelax{chapter}{}{%
  \setuptoc{loa}{onecolumn}%
}
```

Usage of `\ifundefinedorrelax` presumes package `scrbase` (see [section 12.3, page 321](#)).

It doesn’t matter if you’re package would be used with another class. You should never the less set this feature. And if the other class would also recognise the feature your package would automatically use the feature of that class.

As you may see, packages that use `tocbasic`, already provide several interesting features, without the need of a lot of implementation effort. Such an effort would be needed only without `tocbasic` and because of this, most packages currently lack of such features.

```
\iftocfeature{extension}{feature}{true-instructions}{false-instructions}
```

This command may be used, to test, if a *feature* was set for file name *extension*. If so the *true-instructions* will be processed, otherwise the *false-instruction* will be. This may be useful, e. g., if you define your own heading command using `\defptocheading` but want to support the features `totoc`, `numbered` or `leveldown`.

15.3. Configuration of Entries to a Table or List of Contents

Since version 3.20 package `tocbasic` can not only configure the tables or lists of contents and their auxiliary files but also influence their entries. To do so, you can define new styles or you can use and configure one of the predefined styles. In the medium term `tocbasic` will replace the

experimental package `tocstyle` that never became an official part of the KOMA-Script bundle. The KOMA-Script classes intensively use the TOC-entry styles of `tocbasic` since version 3.20.

tocdepth

Entries to tables or lists of contents are in hierarchical order. Therefore each entry level has a numerical value. Higher values correspond with deeper levels. Within the standard classes, e.g., parts have the numerical entry level -1 in the table of contents and chapter entries have value 0. Counter `tocdepth` defines the deepest level that should be shown in the tables and lists of any contents.

Class `book` sets `tocdepth` to 2, so entries of the levels `part`, `chapter`, `section`, and `subsection` are printed. Deeper levels like `subsubsection`, which has the numerical value 3, are not printed. Nevertheless the entries are part of the auxiliary file for the table of contents.

Most entry styles of `tocbasic` also cares about `tocdepth`. Only style `gobble` (see `\DeclareTOCStyleEntry`) ignores `tocdepth`.

\numberline{entry number}
\usetocbasicnumberline[code]

Beta-Feature Though the L^AT_EX kernel already defines command `\numberline`, the kernel definition is not sufficient for `tocbasic`. Therefore `tocbasic` has its own definition of `\numberline`. The package uses `\usetocbasicnumberline` to activate this definition whenever a TOC-entry needs it. Because of this, re-defining `\numberline` often does not work and even may result in warnings if you use `tocbasic`.

You can use the definition of `tocbasic` putting `\usetocbasicnumberline` into your document's preamble. The command first of all checks, whether or not the current definition of `\numberline` uses essential, internal commands of `tocbasic`. Only if this is not the case `\usetocbasicnumberline` redefines `\numberline` and executes `code`. If you omit the optional argument, a `\PackageInfo` outputs a message about the successful re-definition. If you just do not want such a message, use an empty optional argument.

Please note, as a side effect `\usetocbasicnumberline` can globally change the internal switch `\@tempwa!`

\DeclareTOCStyleEntry[option list]{style}{entry level}

Beta-Feature This command is used to define or configure the TOC-entries of a given *entry level*. Argument *entry level* is a symbolic name, e.g., `section` for the entry to the table of contents of the section level with the same name or `figure` for an entry of a figure to the list of figures. A *style* is assigned to each *entry level*. The *style* has to be defined before using it as an argument of `\DeclareTOCStyleEntry`. The *option list* is used to configure several *style*-dependent attributes of the *entry level*.

Currently, `tocbasic` defines the following entry styles:

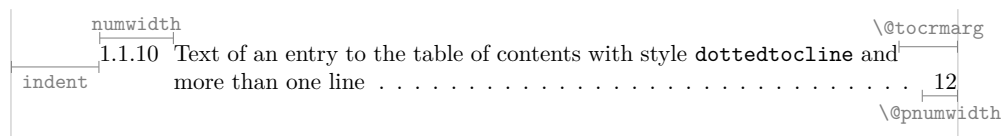


Figure 15.1.: Illustrations of some attributes of a TOC-entry with style `dottedtocline`

default defaults to a clone of style `dottedtocline`. It is recommended to class authors, who use `tocbasic`, to change this style into the default style of the class using `\CloneTOCStyle`. For example the KOMA-Script classes change **default** into a clone of `tocline`.

dottedtocline is similar to the style used by the standard classes `book` and `report` for the section down to `subparagraph` entry levels of the table of contents and for the entries at the list of figures or list of tables. It supports the attributes `level`, `indent`, and `numwidth`. The entries will be indented by the value of `indent` from the left. The width of the entry number is given by the value of `numwidth`. For multiline entries the indent will be increased by the value of `numwidth` for the second and following lines. The page number is printed using `\normalfont`. Entry text and page number are connected by a dotted line. [Figure 15.1](#) illustrates the attributes of this style.

gobble is the most ordinary style. Independently from the setting of `tocdepth`, entries with this style will never be printed. The style simply gobbles the entries. Nevertheless, it supports the standard attribute `level` but does ignore it.

targetocline is similar to the style used by the standard classes for the level `part`. It supports attributes `level` and `indent` only. The last one is already a variation of the standard classes that do not support an indent of the `part` entries.

Before an entry a page break will be made easier. The entries will be indented by the value of `indent` from the left. They are printed using `\large\bfseries`. If `\numberline` is used, the number width is 3em. `\numberline` is not redefined. The standard classes do not use `\numberline` for `part` entries. The value of `indent` even does not influence the indent from the second line of an entry.

[Figure 15.2](#) illustrates the attributes of this style. There you can also see that the style copies inconsistencies of the standard classes, e.g. the missing indent of the second and following lines of an entry or the different values of `\@pnumwidth` that results from the font size dependency. This can even result in a too small distance between the entry text and the page number. Please note, the entry number width shown in the figure is only valid if `\numberline` has been used. In contrast, the standard classes use a distance of 1em after the number.

tocline is a very flexible style. The KOMA-Script classes use this style by default for all

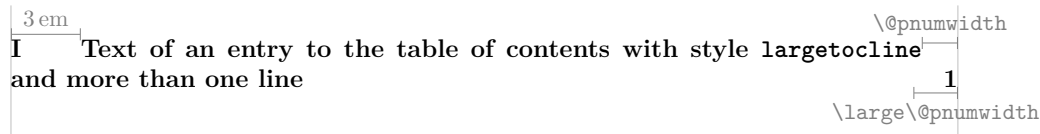


Figure 15.2.: Illustrations of some attributes of a TOC-entry with style `targetocline`

kinds of entries. Classes `scrbook` and `scrreprt` respectively `scrartcl` also define clones `part`, `chapter` and `section` respectively `section` and `subsection`, but add extra *initial code* to the clones to change their defaults.

The style supports attributes `level`, `beforeskip`, `dynnumwidth`, `entryformat`, `entrynumberformat`, `breakafternumber`, `indent`, `linefill`, `numsep`, `numwidth`, `onstarthigherlevel`, `onstartlowerlevel`, `onstartsamelevel`, `pagenumberbox`, `pagenumberformat`, `raggedentrytext`, and `raggedpagenumber`. The defaults of all these attributes depend on the name of the *entry level*. They correspond to the results of the standard classes. So after loading `tocbasic`, you can change the style of the standard classes entries to the table of contents into `tocline` using `\DeclareTOCEntryStyle` without obvious visual changes unless you change exactly these attributes that should induce such changes. Same is valid for list of figures or list of tables of the standard classes.

Because of the flexibility of this style it even could be used instead of the styles `dottedtocline`, `undottedtocline` or `targetocline`. However it needs more effort for configuration.

Figure 15.3 illustrates some of the length attributes of this style. All attributes are described in [table 15.1](#) from [page 360](#).

`undottedtocline` is similar to the style used by the standard classes `book` and `report` for the

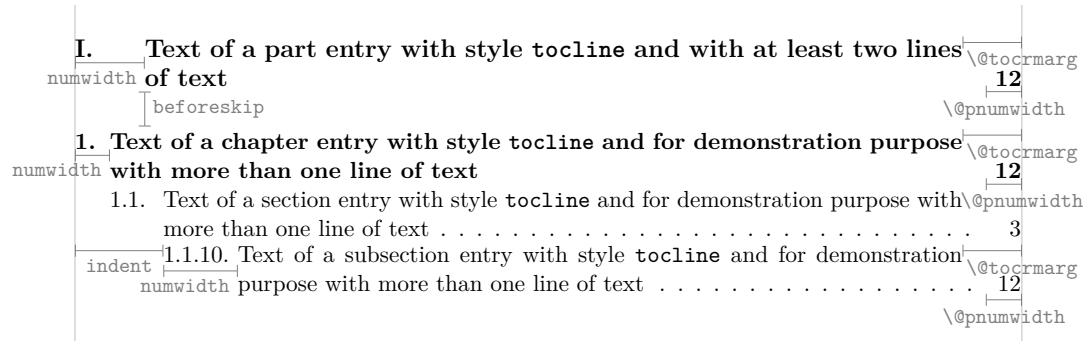


Figure 15.3.: Illustrations of some attributes of a TOC-entry with style `tocline`

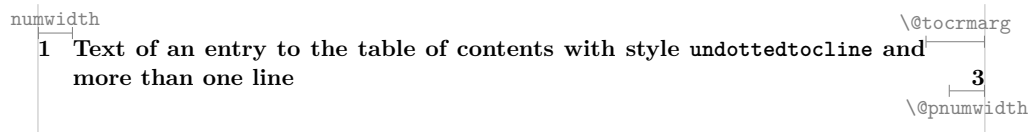


Figure 15.4.: Illustration of some attributes of style `undottedtocline` by the example of a chapter title

chapter entry level or by article for the section entry level of the table of contents. It supports the attributes `level`, `indent`, and `numwidth`. The last one is already a variation of the standard classes that do not support an indent of these entry levels.

Before an entry, a page break will be made easier. The entries will be indented by the value of `indent` from the left. They are printed using `\bfseries`. `\numberline` is used unchanged. The width of the entry number is given by the value of `numwidth`. For multiline entries the indent will be increased by the value of `numwidth` for the second and following lines. [Figure 15.4](#) illustrates the attributes of this style.

Beta-Feature

[Table 15.1](#) describes all attributes of all styles defined by `tocbasic`. If you want to use these attributes as options to `\DeclareNewTOC` (see [page 373](#)) you have to prefix the names of the attribute by `tocentry`, e.g., attribute `level` becomes option `tocentrylevel`. If you want to use these attributes as options to `\DeclareSectionCommand` (see [page 446](#)) and similar commands you have to prefix the names of the attributes by `toc`, e.g., attribute `level` becomes option `toclevel`.

Last but not least using `\DeclareTOCStyleEntry` will define internal command `\l@entry level`.

Table 15.1.: Attributes of the predefined TOC-entry styles of `tocbasic`

<code>beforeskip=length</code>
vertical distance, inserted before an entry of this level using style <code>tocline</code> (see figure 15.3). The distance is made using either <code>\vskip</code> or <code>\addvspace</code> depending on the <i>entry level</i> to adapt the differences of the standard classes and former versions of KOMA-Script.
At <i>entry level</i> part the attribute will be initialised with 2.25em plus 1pt, at chapter with 1em plus 1pt. If <i>entry level</i> currently is unknown, rather section is initialised with 1em plus 1pt. The initial value for all other levels is 0pt plus .2pt.

Table 15.1.: Attributes of the TOC-entry styles (*continuation*)

`breakafternumber=switch`

switch is one of the values for simple switches from [table 2.5, page 39](#). If the switch is active with style `tocline`, there will be a line break after the entry number of `\numberline`. The line after the entry number again starts left aligned with the number.

This switch is not active by default at style `tocline`.

If the feature `numberline` of a list of something has been activated (see `\setuptoc`, [section 15.2, page 354](#)), i. e., if a KOMA-Script class with option `toc=numberline` is used, then the not numbered entries will nevertheless have a (by default empty) number line using the format code of `entrynumberformat`.

`dynnumwidth=switch`

switch is one of the values for simple switches from [table 2.5, page 39](#). If the switch is active with style `tocline`, attribute `numwidth` is ignored. Instead of that the maximum number width detected at the previous L^AT_EX run increased by the value of `numsep` is used.

`entryformat=command`

This attributes makes the format of the entry. The value should be a *command* with exactly one argument. The *command* should not expect the argument to be fully expandable. Commands like `\MakeUppercase`, that need a fully expandable argument, must no be used here. Font changes are allowed and are relative to `\normalfont\normalsize`. Please note that the output of `linefill` and the page number are independent from `entryformat`. See also attribute `pagenumberformat`. The initial value of the attribute for *entry level* part is printing the argument in `\large\bfseries` and for `chapter` printing the argument in `\bfseries`. If currently no level `chapter` exists, `section` used `\bfseries`. All other levels print the argument unchanged.

`entrynumberformat=command`

This attribute makes the format of the entry number within `\numberline`. The value should be a *command* with exactly one argument. Font changes are relative to the one of attribute `entryformat`.

The initial *command* prints the argument unchanged. This means the entry number will be printed as it is.

If the feature `numberline` of a list of something has been activated (see `\setuptoc`, [section 15.2, page 354](#)), i. e., if a KOMA-Script class with option `toc=numberline` is used, then the not numbered entries will nevertheless execute the *command*.

Table 15.1.: Attributes of the TOC-entry styles (*continuation*)

`indent=length`

Horizontal distance of the entry relative to the left margin (siehe [figure 15.1](#) and [figure 15.3](#)).

At style `tocline` all entry levels with a name that starts with “sub” are initialised with the sum of the values of `indent` and `numwidth` of the entry level without this prefix. At styles `dottedtocline`, `undottedtocline` and `tocline` the initial values of levels `part` down to `subparagraph` and the levels `figure` and `table` are compatible with the standard classes. All other levels do not have an initial value. Therefore you have to set an explicit value for such levels when they are defined first time.

`level=integer`

The numerical value of the *entry level*. Only those entries are printed that have a numerical value less or equal to counter `tocdepth`.

This attribute is mandatory for all styles and will be defined automatically at the declaration of the style.

At style `tocline` all entry levels with a name starting with “sub”, the initial value is the level value of the entry level without this prefix increased by one. At the styles `dottedtocline`, `largetocline`, `tocline`, and `undottedtocline` entry levels `part` down to `subparagraph`, `figure`, and `table` are initialised compatible with the standard classes. For all other levels the initialisation is done with the value of `\entry levelnumdepth` if this is defined.

`linefill=code`

At style `tocline` there can be a filler between the end of the entry text and the page number. The value of attribute `linefill` is a *code* that prints this filler. For *entry level part* and *chapter* the attribute is initialised with `\hfill`. If currently no *entry level chapter* has been defined, `section` also uses `\hfill`. All other entry levels are initialised with `\TOCLineLeaderFill` (see [page 367](#)).

If *code* does not result in filling the distance, you should also activate attribute `raggedpagenumber`, to avoid “underfull \hbox” messages.

`numsep=length`

Style `tocline` tries to ensure a minimum distance of *length* between the entry number and the entry text. If `dynnumwidth` is active, it will correct the number width to achieve this. Otherwise it simply throws a warning, if the condition is missed.

The initial *length* is 0.4em.

Table 15.1.: Attributes of the TOC-entry styles (*continuation*)

`numwidth=length`

The reserved width for the entry number (see [figure 15.1](#) until [figure 15.4](#)). At the styles `dottedtocline`, `tocline`, and `undottedtocline` this *length* will be added to the *length* of attribute `indent` for the second and each following entry text line. At style `tocline` the initial *length* of all entries with a name starting with “sub” is the value of the level without this prefix plus 0.9em. At the styles `dottedtocline`, `undottedtocline` and `tocline` the initial *length* of levels `part` down to `subparagraph` and levels *figure* and *table* is compatible to the standard classes. All other levels do not have an initial value. Therefore you have to explicitly set `numwidth` at the first definition of the entry.

`onstarthigherlevel=code`

Style `tocline` can execute *code* at the start of an entry, if the numerical level is greater than the numerical level of the previous entry. Remember: The numerical level of, e.g., `section` is greater than the numerical level of `part`. Nevertheless `part` has the highest position in the entry hierarchy.

Please note that the detection of the level of the previous entry depends on a valid unchanged value of `\lastpenalty`.

The initial *code* is `\LastTOCLevelWasLower` (see [page 367](#)).

`onstartlowerlevel=code`

Style `tocline` can execute *code* at the start of an entry, if the numerical level is lower than the numerical level of the previous entry. Remember: The numerical level of, e.g., `part` is lower than the numerical level of `section`. Nevertheless `part` has the highest position in the entry hierarchy.

Please note that the detection of the level of the previous entry depends on a valid unchanged value of `\lastpenalty`.

The initial *code* is `\LastTOCLevelWasHigher` (see [page 367](#)).

`onstartsamelevel=code`

Style `tocline` can execute *code* at the start of an entry, if the level is same like the level of the previous entry.

Please note that the detection of the level of the previous entry depends on a valid unchanged value of `\lastpenalty`.

The initial *code* is `\LastTOCLevelWasSame` (see [page 367](#)).

Table 15.1.: Attributes of the TOC-entry styles (*continuation*)

`pagenumberbox=command`

By default the page number of an entry is printed right aligned in a box of width `\@pnumwidth`. At style `tocline` the *command* to print the number can be changed using this attribute. The *command* should have exactly one argument, the page number.

`pagenumberformat=command`

This attribute is the format of the page number of an entry. The *command* should have exactly one argument, the page number. Font changes are relative to the font of `entryformat` followed by `\normalfont\normalsize`.

The initial *command* of entry level `part` prints the argument in `\large\bfseries`. The initial *command* of all other levels prints the argument in `\normalfont\normalcolor`.

`raggedentrytext=switch`

v3.21

switch is one of the values for simple switches from [table 2.5](#), [page 39](#). If the switch is active, style `tocline` does print the text of an entry left-aligned instead of justified and only word, that are longer than a text line, are automatically hyphenated.

This switch is not active by default.

`raggedpagenumber=switch`

switch is one of the values for simple switches from [table 2.5](#), [page 39](#). If the switch is active, style `tocline` does not force the page number to be right aligned.

Depending on the value of `linefill`, the setting of this attribute could be needed for the wanted printing of the number, or only to avoid unwanted warning messages. So both attributes should correspond.

By default the switch is not activated and therefore corresponds with an initial value `\hfill` or `\TOCLineLeaderFill` of attribute `linefill`.

```

\DeclareTOCEntryStyle{style}[initial code]{command code}
\DefineTOCEntryOption{option}[default value]{code}
\DefineTOCEntryBooleanOption{option}[default value]{prefix}{postfix}{description}
\DefineTOCEntryCommandOption{option}[default value]{prefix}{postfix}{description}
\DefineTOCEntryIfOption{option}[default value]{prefix}{postfix}{description}
\DefineTOCEntryLengthOption{option}[default value]{prefix}{postfix}{description}
\DefineTOCEntryNumberOption{option}[default value]{prefix}{postfix}{description}

```

Beta-Feature

`\DeclareTOCEntryStyle` is one of the most complex commands in KOMA-Script. It is addressed to \LaTeX developers not the \LaTeX users. It provides the definition of a new TOC-entry *style*. Usually TOC-entries are made using `\addcontentsline`, or, if you use `tocbasic`, with recommended `\addxcontentsline` (see [section 15.1, page 350](#)). In both cases \LaTeX writes a corresponding `\contentsline` into the given auxiliary file. Reading this auxiliary file each `\contentsline` results in execution of a command `\l@entry level`.

Whenever you assign a *style* to a TOC-entry level using `\DeclareTOCStyleEntry`, first of all the *initial code* is executed and then `\l@entry level` is defined to be *command code*. So *command code* is the code that will be expanded and executed by `\l@entry level`. Inside *command code* #1 is the name of the TOC-entry level and ##1 and ##2 are the arguments of `\l@entry level`.

The *initial code* should initialise all attributes of the *style*. Developers are recommended to use *initial code* to initialise all internal macros of the *style* without the need of using an *option list*. The second task of the *initial code* is the definition of options to setup the attributes of the *style*. Option `level` is always defined automatically. The value of `level` can be got in *command code* using `\@nameuse{#1tocdepth}`, e.g., to compare it with the counter `tocdepth`.

Commands `\DefineTOCEntryBooleanOption`, `\DefineTOCEntryCommandOption`, `\DefineTOCEntryIfOption`, `\DefineTOCEntryLengthOption`, and `\DefineTOCEntryNumberOption` should be used to define options for the attributes of the *style* inside *initial code* only. If you use an *option* defined by one of these commands, a macro `\prefixentry levelpostfix` will be defined to be the assigned value or the *default value* of the option. Somehow special is `\DefineTOCEntryIfOption`. It defines `\prefixentry levelpostfix` as a command with two arguments. If the value to the option is an activation value of [table 2.5, page 39](#) the command expands to the first argument. If the value to the option is a deactivation value, the command expands to the second argument.

The *description* should be a real short text that describes the sense of the option with some catchwords. Package `tocbasic` uses this text in error messages, warnings or information output on the terminal and into the log-file.

The most simple style of `tocbasic`, `gobble`, is defined using:

```
\DeclareTOCEntryStyle{gobble}{}%
```

If you would define a entry level dummy using:

```
\DeclareTOCStyleEntry[level=1]{gobble}{dummy}
```

among others this would do something like:

```
\def\dummytocdepth{1}
\def\l@dummy#1#2{}
```

Inside style `tocline` for example

```
\DefineTOCEntryCommandOption[linefill]{\TOCLineLeaderFill}%
{scr@tso@}{@linefill}{filling between text and page number}%
```

is used to define option `linefill` with *default value* `\TOCLineLeaderFill`. A call like:

```
\RedeclareTOCStyleEntry[linefill]{tocline}{part}
```

would therefore result in a definition like:

```
\def\scr@tso@part@linefill{\TOCLineLeaderFill}
```

Whoever likes to define his own styles is recommended to first study the definition of style `dottedtocline`. If this definition is understood, the much more complex definition of style `tocline` gives a lot of hints of the correct usage of the described commands.

In many cases it will be enough to clone an existing style using `\CloneTOCEntryStyle` and to change the initial code of the new style using `\TOCEntryStyleInitCode` or `\TOCEntryStyleStartInitCode`.

`\DefineTOCEntryOption` is merely used to define the other commands. It is not recommended to define options directly using `\DefineTOCEntryOption`. Normally this is even not needed. It is alluded only for completeness.

```
\CloneTOCEntryStyle{style}{new style}
```

Beta-Feature

With this command you can clone an existing *style*. This defines a *new style* with the same attributes and settings like the existing *style*. The package itself uses `\CloneTOCEntryStyle` to declare style `default` as a clone of `dottedtocline`. The KOMA-Script classes use the command to declare the styles `part`, `section`, and `chapter` or `subsection` as a clone of `tocline` and the style `default new` as a clone of `section` or `subsection`.

```
\TOCEntryStyleInitCode{style}{initial code}
\TOCEntryStyleStartInitCode{style}{initial code}
```

Beta-Feature

Every TOC-entry style has an initialisation code. This is used whenever a *style* is assigned to an TOC-entry using `\DeclareTOCEntryStyle`. This *initial code* should never do anything global, because it is also used for local initialisation inside other commands like `\DeclareNewTOC`. The *initial code* not only defines all attributes of a *style*. It also should set the defaults for those attributes.

You can use `\TOCEntryStyleStartInitCode` and `\TOCEntryStyleInitCode` to extend the already existing initialisation code by *initial code*. `\TOCEntryStyleStartInitCode` adds *initial code* in front of the existing initialisation code. `\TOCEntryStyleInitCode` adds the *initial code* at the end of the existing initialisation code. The KOMA-Script classes, e.g., are using `\TOCEntryStyleStartInitCode` to change the filling, font and vertical distances of style `part` that is a clone of `tocline`. Class `scrbook` and `scrreprt` use

```
\CloneTOCEntryStyle{tocline}{section}
\TOCEntryStyleStartInitCode{section}{%
  \expandafter\providecommand%
  \csname scr@tso@#1@linefill\endcsname
  {\TOCLineLeaderFill\relax}%
}
```

to declare `section` as a modified clone of `tocline`.

```
\LastTOCLevelWasHigher
\LastTOCLevelWasSame
\LastTOCLevelWasLower
```

Beta-Feature At the very beginning entries with style `tocline` `tocbasic` executes one of these three commands depending on `\lastpenalty`. `\LastTOCLevelWasHigher` and `\LastTOCLevelWasSame` used in vertical mode add `\addpenalty{\@lowpenalty}` and therefore permit a page break before an entry with same or higher hierarchical position. `\LastTOCLevelWasLower` is an empty command. Therefore page break between an entry and its sub-entry is not permitted.

Users should not redefine these commands. Instead of a redefinition you should change the behaviour of single entry levels using attributes `onstartlowerlevel`, `onstartsamelevel`, and `onstarthigherlevel`.

```
\TOCLineLeaderFill[filling code]
```

Beta-Feature Command has been made to be used as value of option `linefill` of assigning style `tocline` to a TOC-entry. It is a line filler between the end of the entry text and the entry page number. The *filling code* will be repeated with constant distance. The default for this optional argument is a dot.

As implied by the name of the command it uses `\leaders` to put the *filling code*. The distance is defined analogous to the L^AT_EX kernel command `\@dottedtocline` by `\mkern\@dotsepmu`.

15.4. Internal Commands for Class and Package Authors

Commands with prefix `\tocbasic@` are internal but class and package authors may use them. But even if you are a class or package author you should not change them!

```
\tocbasic@extend@babel{extension}
```

The Package `babel` (see [BB13]), or more specifically a L^AT_EX kernel that has been extended by the language management of `babel` writes instructions to change the language inside of the files with the file name extensions `toc`, `lof`, and `lot` into those files at every change of the current language either at the begin of the document or inside the document. Package `tocbasic` extends this mechanism with `\tocbasic@extend@babel` to be used for other file name extensions too. Argument *extension* has to be expandable! Otherwise the meaning of the argument may change until it will be used really.

Normally this command will be used by default for every file name *extension* that will be added to the list of known extensions using `\addtotoclist`. This may be suppressed using feature `nobabel` (see `\setuptoc`, section 15.2, page 354). For the file name extensions `toc`, `lof`, and `lot` this will be done automatically by `tocbasic` to avoid double language switching in the corresponding files.

Normally there isn't any reason to call this command yourself. But there may be lists of something, that should not be under control of `tocbasic`, and so are not in `tocbasic`'s list of known file name extensions, but nevertheless should be handled by the language change mechanism of `babel`. The command may be used explicitly for those files. But please note that this should be done only once per file name extension!

```
\tocbasic@starttoc{extension}
```

This command is something like the L^AT_EX kernel macro `\@starttoc`. It is the command behind `\listoftoc*` (see section 15.2, page 351). Authors of classes or packages who want to participate from the advantages of `tocbasic` should at least use this command. Nevertheless it is recommended to use `\listoftoc`. Command `\tocbasic@starttoc` internally uses `\starttoc`, but sets `\parskip` and `\parindent` to 0 and `\parfillskip` to 0 until infinite before. Moreover, `\@currentx` will be set to the file name extension of the current TOC-file, so this will be available while the execution of the hooks, that will be done before and after reading the auxiliary files.

Because of L^AT_EX will immediately open a new TOC-file for writing after reading that file, the usage of `\tocbasic@starttoc` may result in an error message like

```
! No room for a new \write .
\ch@ck ... \else \errmessage {No room for a new #3}
\fi
```

if there are no more unused write handles. This may be solved, e.g., using package `scrwfile`. See chapter 14 for more information about that package.


```
\tocbasic@@before@hook
\tocbasic@@after@hook
```

The hook `\tocbasic@@before@hook` will be executed immediately before reading a auxiliary file for a TOC even before execution of the instructions of a `\BeforeStartingTOC` command. It is permitted to extend this hook using `\g@addto@macro`.

Similarly `\tocbasic@@after@hook` will be executed immediately after reading such an auxiliary file and before execution of instructions of `\AfterStartingTOC`. It is permitted to extend this hook using `\g@addto@macro`.

KOMA-Script uses these hooks, to provide the automatic width calculation of the place needed by heading numbers. Only classes and packages should use these hooks. Users should really use `\BeforeStartingTOC` and `\AfterStartingTOC` instead. Authors of packages should also favour those commands! These hooks should not be used to generate any output!

If neither `\listofeachtoc` nor `\listoftoc` nor `\listoftoc*` are used for the output of a TOC, the hooks should be executed explicitly.

```
\tocbasic@extension@before@hook
\tocbasic@extension@after@hook
```

These hooks are processed after `\tocbasic@@before@hook`, respectively before `\tocbasic@@after@hook` before and after loading the TOC-file with the corresponding file *extension*. Authors of classes and packages should never manipulate them! But if neither `\listofeachtoc` nor `\listoftoc` nor `\listoftoc*` are used for the output of a TOC, the hooks should be executed explicitly, if they are defined. Please note that they even can be undefined.

```
\tocbasic@listhead{title}
```

This command is used by `\listoftoc` to set the heading of the TOC, either the default heading or the individually defined heading. If you define your own TOC-command not using `\listoftoc` you may use `\tocbasic@listhead`. In this case you should define `\@currentx` to be the file extension of the corresponding TOC-file before using `\tocbasic@listhead`.

```
\tocbasic@listhead@extension{title}
```

This command is used in `\tocbasic@listhead` to set the individual headings, optional table of contents entry, and running head, if it was defined. If it was not defined it will be defined and used in `\tocbasic@listhead` automatically.

```
\tocbasic@addxcontentsline{extension}{level}{number}{text}
\nonumberline
```

v3.12

Command `\tocbasic@addxcontentsline` uses `\addcontentsline` to either create a numbered or not numbered text entry to the TOC-file with the given *extension*. Note, all parameters of `\tocbasic@addxcontentsline` are mandatory. But you may use an empty *number* argument, if you do not want a number. In this case the *text* will be prefixed by `\nonumberline` without any argument. In the other case, if *number* is not empty, `\numberline` with argument *number* will be used as usual.

Command `\nonumberline` is redefined inside `\listoftoc` (see [section 15.2, page 351](#)) depending on feature `numberline` (see [section 15.2, page 354](#)). This guarantees that changes of the feature results in changes of the corresponding TOC immediately at the next L^AT_EX run.

```
\tocbasic@DependOnPenaltyAndTOCLevel{entry level}
\tocbasic@SetPenaltyByTOCLevel{entry level}
```

Beta-Feature

At the end of TOC-entry style `tocline` (see [section 15.3](#)) `\penalty` is set to prohibit page breaks. The used penalty value depends on the *entry level*. This is done by `\tocbasic@SetPenaltyByTOCLevel`. At the very beginning of an entry `\tocbasic@DependOnPenaltyAndTOCLevel` is used to execute the value of either style option `onstartlowerlevel`, `onstartsamelevel`, or `onstarthigherlevel` depending on `\lastpenalty` and the current *entry level*. The default of the first and second option would be to permit a page break, if used in vertical mode.

Developers of `tocline`-compatible styles should adapt this. To do so, they are even allowed to copy the style option declarations of `onstartlowerlevel`, `onstartsamelevel`, and `onstarthigherlevel`. These options should even use the same internal macros `\scr@tso@entry level@LastTOCLevelWasHigher`, `\scr@tso@entry level@LastTOCLevelWasSame` and `\scr@tso@entry level@LastTOCLevelWasLower` to store the current values of the options.

15.5. A Complete Example

This section will show you a complete example of a user defined floating environment with list of that kind of floats and KOMA-Script integration using tocbasic. This example uses internal commands, that have a “@” in their name. This means, that the code has to be put into a package or class, or has to be placed between `\makeatletter` and `\makeatother`.

First of all, a new floating environment will be needed. This could simply be done using:

```
\newenvironment{remarkbox}{%
  \@float{remarkbox}%
}{%
  \end@float
}
```

To the new environment is named `remarkbox`.

Each floating environment has a default placement. This is build by some of the well known placement options:

```
\newcommand*{\fps@remarkbox}{tbp}
```

So, the new floating environment should be placed by default only either at the top of a page, at the bottom of a page, or on a page on its own.

Floating environments have a numerical floating type. Environments with the same active bit at the floating type cannot change their order. Figures and table normally use type 1 and 2. So a figure that comes later at the source code than a table, may be output earlier than the table and vica versa.

```
\newcommand*{\ftype@remarkbox}{4}
```

The new environment has floating type 4, so it may pass figures and floats and may be passed by those.

The captions of floating environment also have numbers.

```
\newcounter{remarkbox}
\newcommand*{\remarkboxformat}{%
  Remark~\theremarkbox\csname autodot\endcsname}
\newcommand*{\fnum@remarkbox}{\remarkboxformat}
```

Here first a new counter has been defined, that is independent from chapters or the counters of other structural levels. L^AT_EX itself also defines `\theremarkbox` with the default Arabic representation of the counter's value. Afterwards this has been used defining the formatted output of the counter. Last this formatted output has been used for the output of the environment number of the `\caption` command.

Floating environments have lists of themselves and those need a auxiliary file with name `\jobname` and a file name extension, the TOC-file:

```
\newcommand*{\ext@remarkbox}{lor}
```

The file name extension of the TOC-file for the list of `remarkboxes` is “lor”.

This was the definition of the floating environment. But the list of this new environment's captions is still missing. To reduce the implementation effort package `tocbasic` will be used for this. This will be loaded using

```
\usepackage{tocbasic}
```

inside of document preambles. Authors of classes or packages would use

```
\RequirePackage{tocbasic}
```

instead.

Now we register the file name extension for package `tocbasic`:

```
\addtotoclist[float]{lor}
```

Thereby the owner `float` has been used, to allude all further KOMA-Script options for list of figures and list of tables also to the new one.

Next we define a title or heading for the list of `remarkboxes`:

```
\newcommand*{\listoflorname}{List of Remarks}
```

You may use package `scrbase` to additionally support titles in other languages than English.

Also a command is needed to define the layout of the entries to the list of remarks:

```
\newcommand*{\l@remarkbox}{\l@figure}
```

Here simply the entries to the list of remarks get the same layout like the entries to the list of figures. This would be the easiest solution. A more explicit would be, e.g.,

```
\DeclareTOCStyleEntry[level=1,indent=1em,numwidth=1.5em]%
{tocline}{remarkbox}
```

Additionally you may want structure the list of remarks depending on chapters.

```
\setuptoc{lor}{chapteratlist}
```

The KOMA-Script classes provide that feature and maybe other classes do so too. Unfortunately the standard classes do not.

This would already be enough. Now, users may already select different kinds of headings either using the corresponding options of the KOMA-Script classes, or `\setuptoc`, e.g., with or without entry in the table of contents, with or without number. But a simple

```
\newcommand*{\listofremarkboxes}{\listoftoc{lor}}
```

may make the usage a little bit easier again.

As you've seen only five commands refers to the list of remarks. Only three of them are necessary. Nevertheless the new list of remarks already provides optional numbering of the heading and optional not numbered entry into the table of contents. Optional even a lower document structure level may be used for the heading. Running headers are provides with the KOMA-Script classes, the standard classes, and all classes that explicitly support `tocbasic`. Supporting classes even pay attention to this new list of remarks at every new `\chapter`. Even changes of the current language are handled inside the list of remarks like they will inside the list of figures or inside the list of tables.

Moreover, an author of a package may add more features. For example, options to hide `\setuptoc` from the users may be added. On the other hand, the `tocbasic` manual may be referenced to describe the corresponding features. The advantage of this would be that user would get information about new features provides by `tocbasic`. But if the user should be able to set the features of the remarks even without knowledge about the file name extension `lor` a simple

```
\newcommand*{\setupremarkboxes}{\setuptoc{lor}}
```

would be enough to use a list of features argument to `\setupremarkboxes` as list of features of file name extension `lor`.

15.6. Everything with One Command Only

The example from the previous section shows, that using `tocbasic` to define floating environments and lists with the captions of those floating environments is easy. The following example will show, that it may be even easier.

`\DeclareNewTOC[options]{extension}`

v3.06

This command declares in one step only a new TOC, the heading of that TOC, the term used for the TOC-entries, and to manage the file name *extension*. Additionally optional floating and non-floating environments may be defined, and inside of both such environments `\caption` may be used. The additional features `\captionabove`, `\captionbelow`, and `captionbeside` of the KOMA-Script classes (see [section 3.20](#)) may also be used inside of those environments.

Argument *extension* is the file name extension of the TOC-file, that represents the list of something. See [section 15.1](#) for more information about this. This argument is mandatory and must not be empty!

Argument *options* is a comma separated list, like you know it from, e.g., `\KOMAOPTIONS` (see [section 2.4](#)). Nevertheless, those options cannot be set using `\KOMAOPTIONS`! An overview of all available options may be found in [table 15.2](#).

v3.20

If option `tocentrystyle` is not used, style `default` will be used. For information about this style see [section 15.3](#). If you do not want to define a command for entries to the list of something, you can use an empty argument, i.e., `tocentrystyle=` or `tocentrystyle={}`. Nevertheless, this would contain the risk to get a lot of errors while printing that list.

Beta-Fe3t20e

Depending on the style of the entries to the list of something, you can setup all valid attributes of the selected style as part of the *options*. To do so you have to prefix the names of the attributes given in [table 15.1](#) from [page 360](#) by prefix `tocentry`. Later changes of the style of the entries can be made using `\DeclareTOCStyleEntry`. See [section 15.3](#), [page 357](#) for more information about the styles.

v3.06

Table 15.2.: Options for command `\DeclareNewTOC`

v3.09

`atbegin=instructions`

The *instructions* will be executed at the begin of the floating or non-floating environment.

v3.09

`atend=instructions`

The *instructions* will be executed at the end of the floating or non-floating environment.

Table 15.2.: Options for command `\DeclareNewTOC` (*continuation*)

`counterwithin=ATX counter`

If you define a float or non-float, the captions will be numbered and a counter *type* (see option *type*) will be defined. You may declare another counter to be the parent L^AT_EX counter. In this case, the parent counter will be set before the float counter and the float counter will be reset whenever the parent counter is increased using `\stepcounter` or `\refstepcounter`.

`float`

If set, float environments for that type will be defined. The names of the environments are the value of *type* and for double column floats the value of *type* with addendum “*”.

`floatpos=float positions`

The default floating position of the float. If no float position was given, “tbp” will be used like the standard classes do for figures and tables.

`floattype=number`

The numerical float type of the defined floats. Float types with common bits cannot be reordered. At the standard classes figures has float type 1 and tables has float type 2. If no float type was given, 16 will be used.

`forcenames`

If set, the names will be even defined, if they were already defined before.

`hang=length`

v3.20

This option is deprecated since KOMA-Script 3.20. Now, the amount of the hanging indent of the entries for that list depend on attributes of the TOC-entry style given by option `tocentrystyle`. The styles of KOMA-Script provide an attribute `numwidth`. If the used style has such an attribute, `\DeclareNewTOC` will initialise it with 1.5em. You can change the real *value* using `tocentrynumwidth=value`. The KOMA-Script classed for example use `tocentrynumwidth=2.3em`.

`indent=length`

v3.20

This option is deprecated since KOMA-Script 3.20. Now, the amount of indenting the entries of that list depend on attributes of the TOC-entry style given by option `tocentrystyle`. The styles of KOMA-Script provide an attribute `indent`. If the used style has such an attribute, `\DeclareNewTOC` will initialise it with 1em. You can change the real *value* using `tocentryindent=value`. The KOMA-Script classed for example use `tocentrynumwidth=1.5em`.

Table 15.2.: Options for command \DeclareNewTOC (*continuation*)

`level=number`

v3.20

This option is deprecated since KOMA-Script 3.20. Now, the level of the entries of that list depend on attributes of the TOC-entry style given by option `tocentrystyle`. Nevertheless all styles have an attribute `level` and `\DeclareNewTOC` initialises it with 1. You can change the real *value* using `tocentrylevel=value`.

`listname=string`

The name of the TOC. If not given the value of `types` with upper case first char using `\MakeUppercase` prefixed by “List of ” will be used.

`name=string`

The name of an element. If no name is given, the value of `type` with upper case first char will be used.

`nonfloat`

If set, a non floating environment will be defined. The name of the environment is the value of `type` with attached “-”.

`owner=string`

The owner as described in the sections before. If no owner was given `owner float` will be used.

`tocentrystyle=TOC-entry style`

v3.20

TOC-entry style is the style that should be used for all entries into the TOC corresponding to the *extension*. The name of the entry level is given by option `type`. Additional to the options of this table all attributes of the *TOC-entry style* can be used as options. To do so, you have to prefix the name of such an attribute by `toc`. For example, you can change the numerical level of the entries using option `tocentrylevel`. For more information about the styles and their attributes see [section 15.3](#) from [page 356](#).

`tocentrystyle-option=value`

v3.20

Additional options depending on the *TOC-entry style* given by `tocentrystyle`. See [section 15.3](#), [page 356](#) for additional information about TOC-entry styles. See [table 15.1](#), [page 360](#) for information about the attributes of the predefined TOC-entry styles of package `tocbasic`, that can be used as *style-option*.

Table 15.2.: Options for command `\DeclareNewTOC` (*continuation*)

<code>type=string</code>	sets the type of the new declared TOC. The type will be used e.g. to defined a <code>\listofstring</code> . If no type is set up the extension from the mandatory argument will be used.
<code>types=string</code>	the plural of the type. If no plural was given the value of <code>type</code> with attached “s” will be used.

Example: Using `\DeclareNewTOC` reduces the example from [section 15.5](#) to:

```
\DeclareNewTOC[%
  type=remarkbox,%
  types=remarkboxes,%
  float,% define a floating environment
  floattype=4,%
  name=Remark,%
  listname={List of Remarks}%
]{lor}
\setuptoc{lor}{chapteratlist}
```

Beside environments `remarkbox` and `remarkbox*` the counter `remarkbox`, the commands `\theremarkbox`, `\remarkboxname`, and `\remarkboxformat` that are used for captions; the commands `\listremarkboxnames` and `\listofremarkboxes` that are used at the list of remarks; and some internal commands that depends on the file name extension `lor` are defined. If the package should use a default for the floating type, option `Optionfloattype` may be omitted. If option `nonfloat` will be used additionally, then a non-floating environment `remarkbox-` will be also defined. You may use `\caption` inside of that non-floating environment as usual for floating environments. [Figure 15.3](#) shows a comparison of the commands, counters and environments of the example environment `remarkbox` and of the commands, counters and environments for figures.

And now a possible usage of the example environment:

```
\begin{remarkbox}
  \centering
  Equal should be typeset equally
  and with equal formatting.
  \caption{First theorem of typography}
  \label{rem:typo1}
\end{remarkbox}
```

A segment of an example page with this environment could be:

Table 15.3.: Comparison of example environment `remarkbox` and environment `figure`

remarkbox	figure	options of <code>\DeclareNewTOC</code>	short description
remarkbox	figure	type, float	floating environments of the respective types
remarkbox*	figure*	type, float	columns spanning floating environments of the respective types
remarkbox	figure	type, float	counter used by <code>\caption</code>
<code>\theremarkbox</code>	<code>\thefigure</code>	type, float	output command to the respective counters
<code>\remarkboxformat</code>	<code>\figureformat</code>	type, float	formatting command to the respective counters used by <code>\caption</code>
<code>\remarkboxname</code>	<code>\figurename</code>	type, float, name	names used in the label of <code>\caption</code>
<code>\listofremarkboxes</code>	<code>\listoffigures</code>	types, float	command to show the list of the respective environments
<code>\listremarboxname</code>	<code>\listfigurename</code>	type, float, listname	heading text of the respective list
<code>\fps@remarkbox</code>	<code>\fps@figure</code>	type, float, floattype	numeric float type for order perpetuation
lor	lof		file name extension of the TOC-file of the respective list

Equal should be typeset equally and with equal formatting.

Remark 1: First theorem of typography

Users of `hyperref` should always use option `listname`. Otherwise they may get an error message, because `hyperref` usually has a problem with the `\MakeUppercase` command that is used to change the case of the first letter of `types` in the name of the list.

Hacks for Third-Party Packages by Package scrhack

Some packages from other authors could have problems with KOMA-Script. In my opinion some packages could be improved. With some packages this makes only sense, if KOMA-Script was used. With some other packages the package author has another opinion. Sometimes proposals was never answered. Package `scrhack` contains all those improvement proposals for other packages. This means, `scrhack` redefines macros of packages from other authors! The redefinitions are only activated, if those packages were loaded. Users can prevent `scrhack` from redefining macros of individual packages.

16.1. State of Development Note

Though this package is part of KOMA-Script for long time and though it has been used by lot of users, there's one problem with it. While redefinition of macros of foreign packages, it depend on the exact definition an usage of those macros. This means additionally, that it depends on dedicated releases of those packages. If a unknown release of such a package will be used, `scrhack` eventually could not do the needed patch. Contrary, in extreme cases the patch can cause errors and fault.

So `scrhack` has to be continuously modified to fit new releases of foreign packages and will never be finished. Because of this `scrhack` will stay in beta state forever. Though the usage will generally be a benefit, the correct function could not be guaranteed forever.

16.2. Early or late Selection of Options

All of what is described in [section 2.4](#) is generally applicable. So if you have already read and understood [section 2.4](#) you can switch to [section 16.3, page 379](#).

In this section a peculiarity of KOMA-Script is presented, which, apart from the `scrhack` package, is also relevant to other KOMA-Script packages and classes. Such that the user can find all information corresponding to a single package or a single class in the relevant chapter, this section is found almost identically in several chapters. Users who are not only interested in a particular package or class, but wish to gain an overview of KOMA-Script as a whole, may read the section in one chapter and may thereafter skip it wherever coming across it in the document.

```
\documentclass[option list]{KOMA-Script class}
\usepackage[option list]{package list}
```

In L^AT_EX, provision is made for the user to pass class options as a comma-separated list of keywords as optional arguments to `\documentclass`. Apart from being passed to the class, these options are also passed on to all packages which can understand the options. Provision is also made for the user to pass optional arguments as a comma-separated list of keywords as optional arguments to `\usepackage`. KOMA-Script expands the option mechanism for the KOMA-Script classes and various packages to use further possibilities. Thus, most KOMA-Script options can also take a value. An option may have not only the form *Option*, but may also have the form *option=value*. Apart from this difference `\documentclass` and `\usepackage` function the same in KOMA-Script as described in [Tea05b] or any introduction to L^AT_EX, for example [OPHS11].

You should note, that in opposite to the interface described below the options interface of `\documentclass` and `\usepackage` is not robust. So commands, lengths, counters and such constructs may break inside the optional argument of these commands. Because of this, the usage of a L^AT_EX length inside the value of an option would cause an error before KOMA-Script can get the control over the option execution. So, if you want to use a L^AT_EX length, counter or command a part of the value of an option, you have to use `\KOMAOPTIONS` or `\KOMAOPTION`. These commands will be described next.

```
\KOMAOPTIONS{option list}
\KOMAOPTION{option}{value list}
```

KOMA-Script offers most class and package options the opportunity to change the value of options even after loading of the class or package. One may then change the values of a list of options at will with the `\KOMAOPTIONS` command. Each option in the *option list* has the form *option=value*.

Some options also have a default value. If one does not give a value, i. e., gives the option simply in the form *option*, then the default value will be used.

Some options can assume several values simultaneously. For such options there exists the possibility, with the help of `\KOMAOPTION`, to pass a single *option* a list of values. The individual values are given as a comma-separated *value list*.

To implement this possibility KOMA-Script uses the commands `\FamilyOptions` and `\FamilyOption` with the family “KOMA”. For more information in these commands see [part II, section 12.2, page 313](#).

16.3. Usage of tocbasic

In the early days of KOMA-Script users asked for handling lists of floats, that will be generated using package float, like list of figures and list of tables, that are generated by KOMA-Script

itself. At that time the KOMA-Script author contacted the author of `float`, to submit a proposal of an interface with support for such an extension. A somehow modified version of that interface has been implemented with commands `\float@listhead` and `\float@addtolists`.

Sometimes later it has appeared, that those two commands were not flexible enough to support all of the comprehensive features supported by KOMA-Script. Unfortunately the author of `float` has finalized the development already, so nobody should expect further changes of this package.

Other package authors have also inherited these commands. Thereby it appeared, that the implementation in some packages, even in package `float`, will need a certain package loading order, though all these packages are not related to each other. Wrong loading order could result in an error or break the functionality of the commands.

To clear all these disadvantages and problems, KOMA-Script officially does not support this old interface any more. Instead, KOMA-Script warns if the old interface is used. At the same time package `tocbasic` (see [chapter 15](#)) has been designed and implemented as a central interface for management of table of contents, lists of floats and similar lists. Usage of this package provides much more advantages and features than the two old commands that have been mentioned above.

Though the effort using that package is very small, the authors of most of the packages, that are using the old interface, have not done so currently. Because of this `scrhack` contains appropriate modifications of packages `float`, `floatrow`, and `listings`. Loading `scrhack` is enough to make these packages recognize not only setting of KOMA-Script option `listof`, but also language switching of package `babel`. More information about the features provided by the changeover to package `tocbasic` can be found in [section 15.2](#).

If the modification for any of the packages is not wanted or causes problems, then it can be deactivated selectively with option `float=false`, `floatrow=false`, or `listings=false`. Please note that changing these options after loading the corresponding package would not do it!

16.4. Incorrect Expectations to `\@ptsize`

Some packages always expect that the class-internal macro `\@ptsize` is not only defined but also expands to an integer. For compatibility, KOMA-Script defines `\@ptsize` even if the basic font size is neither 10pt nor 11pt nor 12pt. KOMA-Script also provides non-integer font sizes. So `\@ptsize` can expand to a non-integer number, too.

Package `setspace` is one of the packages that fail with non-integer number expansion of `\@ptsize`. Additionally the line stretching of that package always depends on the basic font size even if setting is made in the context of another font size. Package `scrhack` solves both problems by redefining `\onehalfspacing` and `\doublespacing` using always the current font size while setting the stretch.

If the modification for the package is not wanted or causes problems, then it can be deacti-

vated selectively with option `setspace=false`. Please note that changing these option after loading `setspace` would not do it! If you use `setspace` with either option `onehalfspacing` or `doublespacing` you have to load `scrhack` before it.

16.5. Special Case hyperref

Before version 6.79h package `hyperref` set the link anchors after instead of before the heading of star version commands like `\part*`, `\chapter*`, and so on. In the meantime this problem have been solved at the KOMA-Script author's suggestion. But because the KOMA-Script author was not patient enough to wait more than a year for the change of `hyperref`, a corresponding patch has been added to `scrhack`. This can be deactivated by `hyperref=false`. Nevertheless, it is recommended to use the current `hyperref` release. In this case `scrhack` does automatically deactivate the not longer needed patch.

16.6. Inconsistent Handling of `\textwidth` and `\textheight`

Package `lscape` defines an environment `landscape` to set the page contents but not head and foot landscape. Inside this environment it changes `\textheight` to the value of `\textwidth`, but it does not change `\textwidth` to the former value of `\textheight`. This is inconsistent. As far as I know, `\textwidth` is unchanged because setting it to `\textheight` could blame other packages or user commands. But changing `\textheight` could also blame other packages or user commands and indeed it breaks, e. g., `showframe` and `scrlayer`. So best would be, not to change `\textheight`, too. `scrhack` uses package `xpatch` (see [Gre12]) to modify the environment start macro `\landscape` appropriately.

If the modification for the package is not wanted or causes problems, then it can be deactivated selectively with option `lscape=false`. Please note that changing this option after loading `lscape` has an effect only, if it is not `false` while loading `lscape` respectively `scrhack`, if `scrhack` is loaded after `lscape`.

Please note, `pdfscape` also uses `lscape` and therefore is influenced by `scrhack`, too.

Defining Layers and Page Styles Using `sclayer`

Most users of graphics software already know layer models for pages or working sheets. \LaTeX itself does not know layers, but there are already packages like `eso-pic` or `textpos`, that provide a kind of background or foreground layer. `sclayer` is another package, that provides such background and foreground layers, but in difference to the other packages mentioned above these layers are part of the page style definition. With this you may simply switch between usage of layers by switching the page style.

To do so, the package also supports a low level interface to define page styles using a layer stack, to put layers onto a page style's layer stack, to put layers at the lowest position of a page style's layer stack, to put layers before or after a layer of a page style's layer stack, to remove a layer from a page style's layer stack and to remove doublets of layers of a page style's layer stack. In short words: The page style interface of `sclayer` provides commands to define layer-stack-based page styles and to manage those layer stacks.

Nevertheless, using the layers directly is recommended for advanced users only. End user interfaces for beginners or average users are provided by additional packages, that load `sclayer` on their own. See [chapter 5](#) in [part I](#) of this manual.

17.1. State of Development Note

Development of this package has not been finished yet. Parts of the package are even still experimental. Because of this, especially internal functionality may be changed in future. Most likely the package will be extended. And because of the early state of development, you should not expect a complete and finished user manual. Nevertheless, this manual, which is recommended for advanced users and developers, describes the current state of development and the released parts of `sclayer`. Everything, not documented here, should not be used for anything else but testing.

17.2. Early or late Selection of Options

All of what is described in [section 2.4](#) is generally applicable. So if you have already read and understood [section 2.4](#) you can switch to [section 17.3](#), [page 383](#).

In this section a peculiarity of KOMA-Script is presented, which, apart from the `sclayer` package, is also relevant to other KOMA-Script packages and classes. Such that the user can find all information corresponding to a single package or a single class in the relevant chapter, this section is found almost identically in several chapters. Users who are not only interested in a particular package or class, but wish to gain an overview of KOMA-Script as a whole, may read the section in one chapter and may thereafter skip it wherever coming across it in the document.

```
\documentclass[option list]{KOMA-Script class}
\usepackage[option list]{package list}
```

In L^AT_EX, provision is made for the user to pass class options as a comma-separated list of keywords as optional arguments to `\documentclass`. Apart from being passed to the class, these options are also passed on to all packages which can understand the options. Provision is also made for the user to pass optional arguments as a comma-separated list of keywords as optional arguments to `\usepackage`. KOMA-Script expands the option mechanism for the KOMA-Script classes and various packages to use further possibilities. Thus, most KOMA-Script options can also take a value. An option may have not only the form *Option*, but may also have the form *option=value*. Apart from this difference `\documentclass` and `\usepackage` function the same in KOMA-Script as described in [Tea05b] or any introduction to L^AT_EX, for example [OPHS11].

You should note, that in opposite to the interface described below the options interface of `\documentclass` and `\usepackage` is not robust. So commands, lengths, counters and such constructs may break inside the optional argument of these commands. Because of this, the usage of a L^AT_EX length inside the value of an option would cause an error before KOMA-Script can get the control over the option execution. So, if you want to use a L^AT_EX length, counter or command a part of the value of an option, you have to use `\KOMAOPTIONS` or `\KOMAOPTION`. These commands will be described next.

```
\KOMAOPTIONS{option list}
\KOMAOPTION{option}{value list}
```

KOMA-Script offers most class and package options the opportunity to change the value of options even after loading of the class or package. One may then change the values of a list of options at will with the `\KOMAOPTIONS` command. Each option in the *option list* has the form *option=value*.

Some options also have a default value. If one does not give a value, i. e., gives the option simply in the form *option*, then the default value will be used.

Some options can assume several values simultaneously. For such options there exists the possibility, with the help of `\KOMAOPTION`, to pass a single *option* a list of values. The individual values are given as a comma-separated *value list*.

To implement this possibility KOMA-Script uses the commands `\FamilyOptions` and `\FamilyOption` with the family “KOMA”. For more information in these commands see [part II, section 12.2, page 313](#).

17.3. Some Generic Information

The package needs some generic information about the class. Class authors may help `sclayer` by setting this information. Otherwise the package tries to detect the information itself. This

works, e.g., for the standard classes and the KOMA-Script classes. But it may or may not fail with other classes.

This section describes some of the information, that class authors may provide. Generally users need not to care about this.

```
\if@chapter then code \else else code \fi
```

If `\if@chapter` is `\iftrue`, `sclayer` will additionally consider the chapter level, e.g., processing option `automark`. If it is defined, but differs from `\iftrue`, only part, section, subsection, sub...subsection, paragraph, subparagraph, sub...subparagraph will be considered. If the macro is not defined, `sclayer` searches for `\chapter`. If `\chapter` is defined and not `\relax`, `sclayer` will define `\if@chapter` to `\iftrue`, otherwise `\if@chapter` will become `\iffalse`.

```
\if@mainmatter then code \else else code \fi
```

Classes like `book` define `\frontmatter`, `\mainmatter`, and `\backmatter`. They also use `\if@mainmatter` to distinguish whether or not the current matter is the main matter. Classes like `report` and `article` do not have `\frontmatter`, `\mainmatter`, or `\backmatter` and therefore also do not have `\if@mainmatter`.

For `sclayer` it's easier not to test always for the existence of the matter commands, but to use `\if@mainmatter` even with classes like `report` and `article`, simply set to `\iftrue`. So if `\if@mainmatter` is not defined, it will be defined to `\iftrue`.

Some classes have `\frontmatter`, `\mainmatter`, or `\backmatter` but not `\if@mainmatter`. In this case `sclayer` also defines `\if@mainmatter` to be `\iftrue` and it extends definition of `\frontmatter`, `\mainmatter`, and `\backmatter` to set `\if@mainmatter` properly. Other matter commands are not known, not tested, and not extended. So if there are other matter commands `sclayer` needs help of the class author to set `\if@mainmatter` correctly.

```
\DeclareSectionNumberDepth{level name}{level depth}
```

Generally each section level is related to an integer number indicating its depth in the document structure. \LaTeX needs this to manage hierarchic section levels. But normally only the document class, that defines the section commands, itself knows that *level depth* of a section level and uses these numerical values inside the corresponding commands, when needed.

But `sclayer` also needs information about the section hierarchy. With command `\DeclareSectionNumberDepth` you can map the name of a heading level to a *level depth*. With standard class `book`, e.g., the *level name* could be `part`, `chapter`, `section`, `subsection`, `subsubsection`, `paragraph`, or `subparagraph` and the corresponding *level depths* would be -1, 0, 1, 2, 3, 4, and 5.

Package `sclayer` tries to determine the *level depths* on its own while loading and again while `\begin{document}`. But, if it fails, i.e., if completely different section commands are used, it would be useful, to be able to define the relationship explicitly. For such cases

`\DeclareSectionNumberDepth` gives the class author the opportunity to define the relationship explicitly.

17.4. Declaration of Layers

A layer is a kind of virtual sheet of transparent paper (in opposite to a physical sheet of paper). One layer is stacked onto another layer and opaque material on one layer may hide material on the layers below. The stack of all layers together makes the physical page. Package `scrlayer` provides two such layer stacks for each page: a background layer stack and a foreground layer stack. The background layer stack is behind the normal page contents, the foreground layer stack is above the normal page contents. So the normal contents is a kind of a separating layer between the background layer stack and the foreground layer stack.

A layer has several attributes, which are the answers to some basic questions:

Is the layer part of the background or the foreground? During page building background layers will be printed first, followed by the main contents and the foreground layers. Therefore, in the output the background layers show up behind the main contents and the foreground layers in front of the main contents. By default, a layer is both, a background layer *and* a foreground layer and therefore will be printed twice. Normally, it makes sense to restrict the layer in this aspect.

What is the position of the layer? There are some attributes to define the horizontal and the vertical position of a layer.

What is the size of the layer? As for the position there are several attributes to define the width and height of a layer. So a layer can also be smaller or larger than the paper and it can be placed everywhere on the paper.

How is a layer aligned on the paper? This question is answered by the alignment attribute. The horizontal position can be relative from the left edge of the paper either to the left edge of the layer or to the centre of the layer or to the right edge of the layer. Also the vertical position can be relative from the top edge of the paper either to the top edge of the layer or to the centre of the layer or to the bottom edge of the layer.

Is the layer intended for text or picture output? This question is also related to the position. For picture output users often expect the origin at the bottom left corner of the layer. But this would not be suitable for text output. So the origin of a text layer is the height of a standard text line below the top left corner of the layer. Picture layers on the other hand span a `picture` environment and provide additional positioning commands.

Should the layer be printed on left or right pages? By default a layer will be printed on both, left and right pages. Note, that `LATEX` names left pages as even pages and right pages as odd pages and that there are no left or even pages in single-sided mode.

Should the layer be printed in single-side mode or two-side mode? By default a layer will be printed in both, single-side mode and two-side mode. Nevertheless, an even page layer will never be printed in single-side mode and therefore is not really a two-side mode layer.

Should the layer be printed on float pages or non-float pages? \LaTeX produces float pages for float environments like tables or figures, if they are allowed to be printed on a page without normal page contents (see option `p` for `figure` or `table`). So from some point of view a float page is a page, that may itself flow. Non-float pages are not pages without floats, but pages, that are not float pages. They may contain floats inside the text, on the top of the page, or on the bottom of the page. Very large floats may seem to be page floats, while in reality they are top floats.

What are the contents of the layer? The corresponding attribute are the commands that produce the output.

So we have eight questions that immediately result in attributes yet. Below in this manual we will describe additional attributes. However, they are just defined for convenience and can be expressed by a combination of these primary attributes.

```
\DeclareLayer[option list]{layer name}
\DeclareNewLayer[option list]{layer name}
\ProvideLayer[option list]{layer name}
\RedeclareLayer[option list]{layer name}
\ModifyLayer[option list]{layer name}
```

These commands can be used to define a layer. The *layer name* has to be fully expandable and should expand to letters only. Some additional characters are tolerated, but are not recommended.

Command `\DeclareLayer` does not care whether or not a layer with the given *layer name* already exists. It will under all circumstances define the layer with the attribute defined by the *option list*. An *option* can be either a key or a key followed by an equal sign followed by a value. Several options may be concatenated to a *option list* and have to be separated by comma. If you'd like to have a comma or a white space inside the value of an option, you should put the value inside curly brackets. See [table 17.1](#) for more information on keys, values, and the corresponding attributes.

In contrast to `\DeclareLayer` using `\DeclareNewLayer` results in an error, if a layer with the same *layer name* already exists. So you may prevent yourself using the same *layer name* more than once by mistake. This would be useful, e. g., if a class or package also defines layers internally.

If you use `\ProvideLayer` instead of `\DeclareLayer`, the declaration will be ignored in case a layer with the same layer name already exists. It could be paraphrased by: *declare the layer only if it has not been declared already*.

If an existing layer should be redefined, `\RedeclareLayer` or `\ModifyLayer` can be used. `\RedeclareLayer` would simply define the layer as if it would be defined newly. In difference to this, `\ModifyLayer` would change only those attributes, that are represented by an option of the *option list*. All other attributes will stay unchanged and will not be reset to the initial default value. Using either `\RedeclareLayer` or `\ModifyLayer` will result in an error, in case there has not been a layer with *layer name* defined before.

Table 17.1.: Options for the definition of page layers with description of the corresponding layer attribute

v3.16	<div><div><code>addcontents=Code</code></div><div>Value <i>code</i> will be added (or appended) at the very end of the current value of attribute <code>contents</code>. So the new content will be generated behind the end of the already existing content. For more information about the handling of <i>code</i> see option <code>contents</code>.</div></div>
v3.16	<div><div><code>addheight=additional height</code></div><div>The current value of attribute <code>height</code> will be increased by the value of this option. You can use the same kind of values as for <code>height</code>.</div></div>
v3.16	<div><div><code>addhoffset=additional horizontal offset</code></div><div>The current value of attribute <code>hoffset</code> will be increased by the value of this option. You can use the same kind of values as for <code>hoffset</code>.</div></div>
v3.16	<div><div><code>addvoffset=additional vertical offset</code></div><div>The current value of attribute <code>voffset</code> will be increased by the value of this option. You can use the same kind of values as for <code>voffset</code>.</div></div>
v3.16	<div><div><code>addwidth=additional width</code></div><div>The current value of attribute <code>width</code> will be increased by the value of this option. You can use the same kind of values as for <code>width</code>.</div></div>

...

Table 17.1.: Options for the definition of layers (*Continuation*)

`align=alignment characters`

The *alignment characters* define the desired alignment of the layer. Each *alignment character* influences either, how argument *length* of option `hoffset` or `voffset` will be used. Several *alignment characters* may be used together (without comma or space) and will be interpreted in the order of occurrence. No macros should be used here! Valid *alignment characters* are:

- `b` – align the layer at its bottom edge; the value of `voffset` is interpreted as the distance from the top edge of the paper to the bottom edge of the layer.
- `c` – align the layer at its centre; the values of `voffset` and `hoffset` are interpreted as the distance from the top left corner of the paper to the centre of the layer.
- `l` – align the layer at its left edge: the value of `hoffset` is interpreted as the distance from the left edge of the paper to the left edge of the layer.
- `r` – align the layer at its right edge; the value of `hoffset` is interpreted as the distance from the left edge of the paper to the right edge of the layer.
- `t` – align the layer at its top edge; the value of `voffset` is interpreted as the distance from the top edge of the paper to the top edge of the layer.

`area={hoffset}{voffset}{width}{height}`

The composed option results in `hoffset=hoffset`, `voffset=voffset`, `width=width`, `height=height`.

v3.18

`backandforeground`

This option removes any foreground or background restriction of the layer. Generally usage of this option makes no sense, but it is provided for completeness of the user interface. The option does not expect any value.

`background`

Print the layer only in the background, but not in the foreground. This makes a background-only layer in opposite to the default of layers which are both, background and foreground layers and therefore would be printed twice. The option does not expect any value. By the default the attribute is not set.

Table 17.1.: Options for the definition of layers (*Continuation*)

`bottommargin`

The composed option sets `hoffset`, `voffset`, `width`, `height`, and `align` to horizontally span the paper from the left edge to the right and vertically span the area below the footer down to the bottom edge of the paper.

`clone=layer name`

The composed option sets all primary attributes of the layer to the same values as the primary attributes of the layer with the given *layer name*. Note, that *layer name* has to be fully expandable and should expand to letters only. Some additional characters are tolerated, but are not recommended!

`contents=code`

The *code* will be expanded whenever the layer is printed. So the *code* is what you will see. Code validity is not checked. So errors in *code* may result in several failures on each page, that prints the layer.

`evenpage`

Print the layer on even pages only, but not on odd pages. The option does not expect any value. By the default the option is not set and therefore layers would be printed on odd pages and on even pages. Note, that this attribute subsumes `twoside`.

v3.18

`everypage`

This is a combination of `odddorevenpage` and `floatornonfloatpage`. The option does not expect any value.

`everyside`

This option removes any restriction of the layer to single side or double side printing. This is also the default of layers. The option does not expect any value.

v3.18

`floatornonfloatpage`

This option removes any restriction of the layer to float or non-float pages. So the layer will be printed independent of whether a page is a float page or not. This is also the default of layers. The option does not expect any value.

`floatpage`

Print the layer on float pages only, but not on other pages. The option does not expect any value. By the default the attribute is not set and therefore layers would be printed on float pages and on non-float pages.

Table 17.1.: Options for the definition of layers (*Continuation*)

<code>foot</code>	The composed option sets <code>hoffset</code> , <code>voffset</code> , <code>width</code> , <code>height</code> , and <code>align</code> to horizontally span the text area and vertically span the page footer defined by the new \LaTeX length <code>\footheight</code> .
<code>footskip</code>	The composed option sets <code>hoffset</code> , <code>voffset</code> , <code>width</code> , <code>height</code> , and <code>align</code> to horizontally span the text area and vertically span the distance between the text area and the page footer (note, that this is not the same like <code>\footskip</code>).
<code>foreground</code>	Print the layer only in the foreground, but not in the background. This makes a foreground-only layer in opposite to the default of layers which are both, background and foreground layers and therefore would be printed twice. The option does not expect any value. By the default the attribute is not set.
<code>head</code>	The composed option sets <code>hoffset</code> , <code>voffset</code> , <code>width</code> , <code>height</code> , and <code>align</code> to horizontally span the text area and vertically span the page head defined by usual \LaTeX length <code>\headheight</code> .
<code>headsep</code>	The composed option sets <code>hoffset</code> , <code>voffset</code> , <code>width</code> , <code>height</code> , and <code>align</code> to horizontally span the text area and vertically span the distance between the page head and the text area.
<code>height=length</code>	Sets the height of the layer. Note, that <i>length</i> can either be a \LaTeX length, declared using <code>\newlength</code> , or a \TeX length, declared using <code>\newdimen</code> or <code>\newskip</code> , a length value like 10pt, or a dimensional expression using +, -, /, *, (, and). For more information about valid dimensional expressions see [Tea98, section 3.5].
<code>hoffset=length</code>	Sets the offset of the layer (depending on <code>align</code> either left edge of the layer, middle of the layer or right edge of the layer) from the left edge of the paper. See <code>height</code> for more information about valid content of <i>length</i> .

...

Table 17.1.: Options for the definition of layers (*Continuation*)

`innermargin`

The composed option sets `hoffset`, `voffset`, `width`, `height`, and `align` to horizontally span the distance between the right edge of text area and the right edge of the paper on even pages or the distance between the left edge of the paper and the left edge of the text area on odd pages and vertically span the whole paper from the top edge to the bottom edge.

`leftmargin`

The composed option sets `hoffset`, `voffset`, `width`, `height`, and `align` to horizontally span the distance between the left edge of the paper and the left edge of the text area and vertically span the whole paper from the top edge to the bottom edge.

v3.19

`mode=mode`

The primary option defines the *mode* for the layer output. Default is `mode=text`. The Baseline of the first text line will be placed the height of one standard text line below the top edge of the layer. Therefore the text output will be top aligned in the layer. In `picture mode` the layer spans a `picture` environment with the origin in the bottom left corner of the layer. The also available `raw mode` current is the same like `text mode`.

Changing the *mode* of a layer generally can move the contents. Additionally, i.e., `picture mode` provides additional commands that result in errors with another *mode*. Therefore, generally it makes no sense to change the *mode* of a layer after it's initial declaration!

`nonfloatpage`

Restricts the layer to pages, that are not float pages. The option does not expect any value. By the default the attribute is not set and therefore layers would be printed on float pages and on non-float pages.

v3.18

`oddorevenpage`

Removes any restriction of the layer to odd or even pages. So it will be printed independently of whether a page is odd or even. This is also the default of layers. The option does not expect any value.

`oddpages`

Print the layer on odd pages only, but not on even pages. The option does not expect any value. By the default the option is not set and therefore layers would be printed on odd pages and on even pages.

Table 17.1.: Options for the definition of layers (*Continuation*)

oneside

Print the layer in single-side mode only, but not in two-side mode. The option does not expect any value. By the default the attribute is not set and therefore layers would be printed in single-side and two-side mode.

outermargin

The composed option sets `hoffset`, `voffset`, `width`, `height`, and `align` to horizontally span the distance between the left edge of the paper and the left edge of the text area on even pages or the distance between the right edge of the text area and the right edge of the paper on odd pages and vertically span the whole paper from the top edge to the bottom edge.

page

The composed option sets `hoffset`, `voffset`, `width`, `height`, and `align` to horizontally and vertically span the whole paper from the left edge to the right edge and the top edge to the bottom edge.

v3.16

pretoccontents=*code*

Value *code* will be added at the very beginning of the current value of attribute `contents`. So the new content will be generated in front of the already existing content. For more information about the handling of *code* see option `contents`.

rightmargin

The composed option sets `hoffset`, `voffset`, `width`, `height`, and `align` to horizontally span the distance between the right edge of text area and the right edge of the paper and vertically span the whole paper from the top edge to the bottom edge.

textarea

The composed option sets `hoffset`, `voffset`, `width`, `height`, and `align` to horizontally and vertically span the whole text area from the left edge to the right edge and the top edge to the bottom edge.

topmargin

The composed option sets `hoffset`, `voffset`, `width`, `height`, and `align` to horizontally span the whole page from the left edge to the right edge and vertically span the distance between the top edge of the paper and the page head.

Table 17.1.: Options for the definition of layers (*Continuation*)

twoside

Print the layer in two-side mode only, but not in single-side mode. The option does not expect any value. By the default the attribute is not set and therefore layers would be printed in single-side and two-side mode.

v3.18

unrestricted

This option removes all output restrictions of the layer. It is a combination of **backandforeground**, **everywhere**, and **floatornonfloatpage**. The option does not expect any value.

voffset=*length*

Sets the offset of the layer (depending on **align** either top edge of the layer, middle of the layer or bottom edge of the layer) from the top edge of the paper. See **height** for more information about valid content of *length*.

width=*length*

Sets the width of the layer. See **height** for more information about valid content of *length*.

```
\layerhalign
\layervalign
\layerxoffset
\layeryoffset
\layerwidth
\layerheight
```

These commands are valid during output of the layer’s contents only. So they can only be used inside the *code* of option **contents** of the previously described commands. In this case they give the effective alignment, position and dimension of the layer, that will be used for the output. However, the effective dimension of the layer’s contents may differ, i.e., if the contents are oversized or do not fill the layer completely.

v3.19

The primary layer attribute **align** is mapped to **\layerhalign** and **\layervalign**. The horizontal values **l** and **r** are copied to **\layerhalign**. The vertical values **r** and **b** are copied to **\layervalign**. The also horizontal and vertical value **c** is copied to both commands. If **align** has contradictory values, only the last one is copied. Therefore the resulting **\layerhalign** is either **l**, **c**, or **r** and the resulting **\layervalign** is either **r**, **c**, or **b**.

Redefinition of the commands to change the stored values is strictly forbidden and would result in unpredictable issues.

`\LenToUnit{length}`

v3.19 This command has been copied from `eso-pic` 2.0f. It converts lengths into multiples of `\unitlength`. It can be used everywhere \LaTeX expects `picture` coordinates or `\unitlength`-depending values. For more information see [Nie15] and the following descriptions of `\putUR`, `\putLL`, and `\putLR`. If the command is already defined, e. g., by loading `eso-pic` before `scrlayer`, the package does not define it again.

It should be noted at this point that using package `picture` (see [Obe09]) supersedes `\LenToUnit` more or less. The package extends environment `picture` and the `picture` commands to accept \LaTeX lengths directly.

```
\putUL{content}
\putUR{content}
\putLL{content}
\putLR{content}
\putC{content}
```

v3.19 You can use these commands inside the value of primary layer option `contents`, if the layer is declared with `mode=picture`. In that case, `\putUL` places the *content* relative to the upper left corner of the layer and therefore is the same like `\put(0,\LenToUnit{\layerheight})`. `\putUR` places the *content* relative to the upper right corner of the layer and therefore is the same like `\put(\LenToUnit{\layerwidth},\LenToUnit{\layerheight})`. `\putLL` places the *content* relative to the lower left corner of the layer and therefore is the same like `\put(0,0)`. `\putLR` places the *content* relative to the lower right corner and therefore is the same like `\put(\LenToUnit{\layerwidth},0)`. Last but not least `\putC` places the *content* relative to the centre of the layer.

Example: You want to investigate, how accurate `DIV=classic` sets the height of the text area to the width of the page size. So you declare a layer that not only border the text area, but also places a circle with paper width as diameter dimension into the centre of the text area:

```
\documentclass[DIV=classic]{scrartcl}
\usepackage{pict2e}
\usepackage{scrlayer}
\DeclareNewLayer[%
  textarea,background,mode=picture,
  contents={%
    \putLL{\line(1,0){\LenToUnit{\layerwidth}}}%
    \putLR{\line(0,1){\LenToUnit{\layerheight}}}%
    \putUR{\line(-1,0){\LenToUnit{\layerwidth}}}%
    \putUL{\line(0,-1){\LenToUnit{\layerheight}}}%
    \putC{\circle{\LenToUnit{\paperwidth}}}%
  }
}
```

```
]{showtextarea}
\DeclareNewPageStyleByLayers{test}{showtextarea}
\pagestyle{test}
\begin{document}
\null
\end{document}
```

You will see that `typearea`'s mapping to an integer *DIV* value is very accurate in this example.

For more information about the sketched Middle Age method of generating a text area see [section 2.3, page 29](#).

For more information about command `\DeclareNewPageStyleByLayers` see the description later in [section 17.5, page 398](#). In short: It defines a new page style using the newly declared layer.

`\GetLayerContents{layer name}`

v3.16

This command results in the whole content of a layer. You have to note that using this command inside the *code* of the layer attributes `contents`, `addcontents`, or `pretocontents` can result in an infinite recursion, if the content of the current layer is referenced. You yourself should avoid such situations!

`\IfLayerExists{string}{then-code}{else-code}`

This command may be used to execute code depending on whether or not a layer has been defined already. If the layer exists *then-code* will be executed, otherwise *else-code*. Note, the command cannot really test whether a layer exists. It uses a heuristic, that will never be false negative, but may be false positive. Nevertheless, if it is false positive something went wrong, either an incompatible package has been used or the user made something he should not do.

`\DestroyLayer{layer name}`

This command sets all macros corresponding with the layer with given *layer name* to `\relax`, if a layer with that name exists. As a result the layer cannot be used any longer. It does not matter, if the layer is still part of the layer list of a page style, because such destroyed layers will be ignored. Nevertheless, destroyed layers may be defined again using `\DeclareNewLayer` or `\ProvideLayer`, but cannot be changed using `\RedeclareLayer` or `\ModifyLayer` any longer.

The command is intended to be used inside `\scrlayerOnAutoRemoveInterface` to remove layers, which have been defined using removable macros of an interface, whenever the interface would be removed.

`\layercontentsmeasure`

The command `\layercontentsmeasure` prints a ruler at each layer edge, of which the top and left one is labelled with centimeters, the right and bottom one with inches. This command is used internally, if option `draft` is enabled. The rulers will be drawn behind the content of each layer. It can also be used as exclusive content of a layer.

17.5. Declaration and Management of Page Styles

Until now we know layers, but we do not know how to use them. The perhaps astonishing answer is: with page styles. In \LaTeX , page styles usually define heads and foots of odd and even pages.

The head and foot of odd pages will be printed on pages with odd page number in two-side mode or on all pages in single-side mode. This is something like the layer attributes `oddpaper` and `evenpage`.

The page head will be printed before the main contents of a page. The page footer will be printed after the main contents of a page. So this is something like the layer attributes `background` and `foreground`.

So it suggests itself to declare page styles as a list of layers. But instead of having only four attributes `oddpaper`, `evenpage`, `background`, and `foreground` all the attributes of layers shown in [section 17.4](#) may be used.

The outcome of all such considerations is that layer page styles are one type of page styles `sclayer` provides. A layer page style consists of layers and several *hooks*. For description of layers see [section 17.4](#). The *hooks* are points in the expansion or execution of page styles you may add additional code to. Advanced users know this already from commands like `\AtBeginDocument` (see [\[Tea05b\]](#)) or `\BeforeClosingMainAux` (see [page 335](#)).

Alias page styles are another type of page styles provided by `sclayer`. An alias page style wraps another page style. In other words, the name of an alias page style is an alias name of another page style, the aliased or original page style. Because of this, the manipulation of an alias page style results in the manipulation of the original page style. If the original page style is an alias page style too, the manipulation will result in the manipulation of the aliased page style of that original page style and so on until a real page style will be manipulated. Aliasing is not restricted to `sclayer` page styles, but possible for all page styles.

`\currentpagestyle`

Package `sclayer` patches `\pagestyle` to set `\currentpagestyle` to the currently active page style. Note, `\thispagestyle` does not change `\currentpagestyle`. But if you use `\thispagestyle` the result of `\currentpagestyle` may be changed while executing the \LaTeX output routine. However, this change will only occur, if `\currentpagestyle` has been actively protected from expansion up to execution of the output routine.

Note, the layer page styles described later in this section, will not need the patch of `\pagestyle` to set `\currentpagestyle`. The patch has been made for usage with other page styles. Additionally, `\currentpagestyle` is empty before the first `\pagestyle` after loading `sclayer`. So if you define an end user page style interface, it may be useful to use an implicit `\pagestyle` to set the current page style to a default page style.

```
\BeforeSelectAnyPageStyle{code}
\AfterSelectAnyPageStyle{code}
```

The command `\BeforeSelectAnyPageStyle` adds `code` to the hook that will be executed inside of `\pagestyle` just before the page style will be selected. You may use `#1` as a placeholder for the argument of `\pagestyle`.

The command `\AfterSelectAnyPageStyle` is similar, but the `code` will be executed just after the page style has been selected and after `\currentpagestyle` has been set to the name of the real page style.

Note, `code` of both commands will be executed only if a page style will be selected using `\pagestyle`, but not, e. g., if a page style will be selected using `\thispagestyle`. Note also, you cannot remove `code` from the hook after adding it. But the `code` will be added locally, so you may use a group to limit the scope of `code`.

```
\DeclarePageStyleAlias{alias page style name}{original page style name}
\DeclareNewPageStyleAlias{alias page style name}{original page style name}
\ProvidePageStyleAlias{alias page style name}{original page style name}
\RedeclarePageStyleAlias{alias page style name}{original page style name}
```

These commands may be used to define a page style with name *alias page style name* that is simply an alias for an already existing page style with name *original page style name*. If there is already a page style *alias page style name*, then using `\DeclarePageStyleAlias` or `\RedeclarePageStyleAlias` will destroy the alias before recreating it.

`\DeclareNewPageStyleAlias` will throw an error message, if a page style *alias page style name* has already been defined before. It does not matter if the already defined page style is a layer page style, an alias page style or another page style.

`\ProvidePageStyleAlias` will define the alias only if a page style *alias page style name* has not been defined before. If a page style *alias page style name* already exists nothing will be done.

`\RedeclarePageStyleAlias` expects an already existing page style *alias page style name*. It will destroy that page style and afterwards define the alias. If the page style *alias page style name* does not exist, then you will get an error.

```
\DestroyPageStyleAlias{page style name}
```

This command makes the page style with given *page style name* L^AT_EX-undefined, if it is an alias for another page style. Afterwards, the page style may be defined newly with, e.g., `\DeclareNewAliasPageStyle` or `\ProvideAliasPageStyle`.

The command is intended to be used inside of `\scrlayerOnAutoRemoveInterface` to remove page styles that have been declared by an interface and use removable macros of that interface.

```
\GetRealPageStyle{page style name}
```

The command will result in the (recursive) real page name of the page style, if the page style with given name *page style name* is an alias of another page style. In all other cases, even if there’s no alias and no page style named *page style name*, the result would be simply *page style name*. The command is fully expandable and may be used, e.g., in the second argument of `\edef`.

```
\DeclarePageStyleByLayers[option list]{page style name}{layer list}
\DeclareNewPageStyleByLayers[option list]{page style name}{layer list}
\ProvidePageStyleByLayers[option list]{page style name}{layer list}
\RedeclarePageStyleByLayers[option list]{page style name}{layer list}
```

These commands declare a page style with *page style name*. The page style will consist of the layers given in *layer list*, a comma separated list of layer names. Note, the *page style name* and the layer names at the *layer list* must be fully expandable and should expand to letters. Several other characters are tolerated, but, nevertheless, not recommended.

The *option list* is a comma separated list of *key=value* options. These options may be used to set additional features. Currently they are used to set the code that should be expanded or executed at several *hooks*. See the introduction to this section for more general information about *hooks*. See [table 17.2](#) for detailed information on specific hooks.

Table 17.2.: The *hook* options for page styles (in order of execution)

<code>onselect=code</code>
Execute <i>code</i> whenever the page style is selected using, e.g., <code>\pagestyle</code> . Note, <code>\thispagestyle</code> does not select the page style immediately, but asynchronously inside L ^A T _E X’s output routine.

...

Table 17.2.: The *hook* options for page styles (*Continuation*)

`oninit=code`

Execute `code` whenever the output of page style’s layers is initialised. Note, this is done twice for every page: once for background layers and once for foreground layers.

`ononeside=code`

Execute `code` whenever the output of page style’s layers in single-side mode is initialised. Note, this is done twice for every page: once for background layers and once for foreground layers.

`ontwoside=code`

Execute `code` whenever the output of page style’s layers in two-side mode is initialised. Note, this is done twice for every page: once for background layers and once for foreground layers.

`onoddpage=code`

Execute `code` whenever the output of page style’s layers on an odd page is initialised. Note, this is done twice for every page: once for background layers and once for foreground layers. Note also that in single-side mode all pages are odd pages, not only pages with odd page numbers.

`onevenpage=code`

Execute `code` whenever the output of page style’s layers on an even page is initialised. Note, this is done twice for every page: once for background layers and once for foreground layers. Note also that there are not even pages in single-side mode, but all pages are odd pages, not only pages with odd page numbers.

`onfloatpage=code`

Execute `code` whenever the output of page style’s layers on a float page are initialised. Note, this is be done twice for every page: once for background layers and once for foreground layers. Note also that float pages are only those pages with p-placed floating objects.

`onnonfloatpage=code`

Execute `code` whenever the output of page style’s layers on a non-float page is initialised. Note, this is done twice for every page: once for background layers and once for foreground layers. Note also that non-float pages are all pages that are not float-pages. Those pages may have t-placed, h-placed, b-placed, or no floating objects.

Table 17.2.: The *hook* options for page styles (*Continuation*)

<code>onbackground=code</code>	Execute <i>code</i> whenever the output of page style's layers in the background of a page is initialised. Note, this is done once for every page.
<code>onforeground=code</code>	Execute <i>code</i> whenever the output page style's layers in the foreground of a page is initialised. Note, this is done once for every page.

The difference of `\DeclarePageStyleByLayers` and `\DeclareNewPageStyleByLayers` is that `\DeclareNewPageStyleByLayers` will result in an error, if a page style with name *page style name* already exists. Note, declaring a page style, which is an alias of another page style (see `\DeclareAliasPageStyle` prior in this section), will not re-declare the page style itself, but it's real page style (see `\GetRealPageStyle` prior in this section).

The difference of `\DeclarePageStyleByLayers` and `\ProvidePageStyleByLayers` is that `\ProvidePageStyleByLayers` will simply do nothing, if there's already a page style with name *page style name*. In difference to `\DeclareNewPageStyleByLayers` it will not raise an error.

The difference of `\DeclarePageStyleByLayers` and `\RedeclarePageStyleByLayers` is, that `\RedeclarePageStyleByLayers` may be used only if the real page style of *page style name* already exists. Otherwise an error occurs.

Please have also a look at the notes to following pseudo page style `@everystyle@`.

```
\pagestyle{@everystyle@}
\pagestyle{empty}
```

There are two default layer page styles that are somehow special. The first one is `@everystyle@`. This page style should not be used like any other page style, but the layers of this page style will be used by all the other layer page styles. So adding a layer to this page style is similar to adding this layer to all other layer page styles even the empty one. There's one difference: Layer referencing commands of the page style interface like `\ForEachLayerOfPageStyle`, `\AddLayersToPageStyleBeforeLayer`, or `\AddLayersToPageStyleAfterLayer` process only the layers of the page style that has been referenced but not the layers implicated by `@everystyle@`.

The other somehow special page style is `empty`. Normally page style `empty` is defined by the `LATEX` kernel, to be a page style without page head or page foot. Package `scrlayer` re-defines it to be a layer page style without any layer. Nevertheless, you may use it like every other layer page style too. The main advantage above the `LATEX` kernel's empty page style is that it also executes the layers of special layer page style `@everyself@`.


```
onpsselect=code
onpsinit=code
onpsoneside=code
onpstwo-side=code
onpsoddpage=code
onpsevenpage=code
onpsfloatpage=code
onpsnonfloatpage=code
onpsbackground=code
onpsforeground=code
```

There’s also a KOMA-Script option for each of those `hooks`. The names of the KOMA-Script options are similar to the names of the page style options, but with “ps” inserted behind “on”. The value of the KOMA-Script options are the initial defaults of the corresponding `hooks`. These defaults will be extended at every declaration of page style hook options via *option list*. You may remove the default, using `\ModifyLayerPageStyleOptions` described later in this section.

```
deactivatepagestylelayers=simple switch
\ForEachLayerOfPageStyle{page style name}{code}
\ForEachLayerOfPageStyle*{page style name}{code}
```

As long as KOMA-Script option `deactivatepagestylelayers` has not been activated command `\ForEachLayerOfPageStyle` can be used to process *code* for every member layer of *page style name*’s layers list. Inside of *code* the place holder #1 may be used to reference the name of the current layer.

Example: If you want to print the names of all layers of page style `scrheadings`, you may use:

```
\let\commaatlist\empty
\ForEachLayerOfPageStyle{scrheadings}{%
  \commaatlist#1\gdef\commaatlist{, }}
}
```

Usage of `\gdef` instead of `\def` is necessary in the example above, because `\ForEachLayerOfPageStyle` executes the *code* inside of a group to minimise side effects. Here `\gdef` redefines `\commaatlist` globally, so it would be still valid at the execution of *code* for the next layer.

v3.18

Alternatively, you can use `\def` but the star variant `\ForEachLayerOfPageStyle*`. This type does not use an additional group for executing *code*. On the other hand the user has to take care on side effects of *code*, i.e., deactivation of all layers using `deactivatepagestylelayers=true` in *code* would persist after `\ForEachLayerOfPageStyle*`.

Several other commands of `scrlayer` also use `\ForEachLayerOfPageStyle` internally. So these also do not process any layer if KOMA-Script option `deactivatepagestylelayers` would be activated. So you may use this options, e.g., to hide all layers of all layer page styles.

```
\AddLayersToPageStyle{page style name}{layer list}
\AddLayersAtBeginOfPageStyle{page style name}{layer list}
\AddLayersAtEndOfPageStyle{page style name}{layer list}
\RemoveLayersFromPageStyle{page style name}{layer list}
```

You can use these commands to add layers to a layer page style or to remove layers from a layer page style. The page style will be referenced by *page style name*. The layers are given by a comma separated *layer list*.

The commands `\AddLayersToPageStyle` and `\AddLayersAtEndOfPageStyle` add all layers of the comma separated list of layers *layer list* at the end of the layer list of layer page style *page style name*. Logically the added layers would be above or in front of the old layers of the page style. Nevertheless, new background layers would be behind the text layer and therefore behind all foreground layers.

Command `\AddLayersAtBeginOfPageStyle` adds the new layers at the begin of the layer list of the page style. Note, the layers will be added in the order of the *layer list*. The first layer at *layer list* will be added first, the second layer will be added second and so on. So with `\AddLayersAtBeginOfPageStyle` the last layer at *layer list* will become the new first layer of the layer list of layer page style *page style name*.

Command `\RemoveLayersFromPageStyle` may be used to remove layers from the layer list of layer page style *page style name* instead of adding them. Note, layers, which are part of *layer list*, but not part of the page style's layer list, will be ignored. But adding or removing layers from a page style, which is not a layer page style or an alias of a layer page style, would be a mistake and result in an error message.

```
\AddLayersToPageStyleBeforeLayer{page style name}{layer list}{reference layer name}
\AddLayersToPageStyleAfterLayer{page style name}{layer list}{reference layer name}
```

These commands are similar to the commands described before, but they do not add the layers at the begin or end of the layer list of a layer page style, but just before or after a reference layer at the layer list of a layer page style. Note, in this case the order of the *layer list* will be same in the layer list of *page style name* after adding. If the reference layer named *reference layer name* is not part of the layer list of the layer page style, nothing happens.

```
\UnifyLayersAtPageStyle{page style name}
```

With the commands described before in this section you may not only add different layers to a page style, but even add the same layer several times to a page style. In most cases it does not make sense to have one layer several times at the layer list of a layer page style. So you may use `\UnifyLayersAtPageStyle` to remove all dupes of layers from the layer list of a layer page style.

Note, the order of layers may change! So if you want a special order, you should remove all layers and add the layers in the order you want instead of using `\UnifyLayersAtPageStyle`.

```
\ModifyLayerPageStyleOptions{page style name}{option list}
```

```
\AddToLayerPageStyleOptions{page style name}{option list}
```

Command `\ModifyLayerPageStyleOptions` may be used to modify the page style options of a layer page style. Only options at the comma separated *option list* will be set to the new values given in *option list* if the new value is not empty. Options, which are not at *option list*, will stay unchanged. If you want to set an option to *do nothing* you may use value `\relax`. Note, setting an option to a new value using `\ModifyLayerPageStyleOptions` will remove the previous value including the global default value.

`\AddToLayerPageStyleOptions` differs from `\ModifyLayerPageStyleOptions` in that point. It will not overwrite the previous values, but adds — or more precisely: concatenates — the new values to the previous values of the options at *option list*.

```
\IfLayerPageStyleExists{page style name}{then code}{else code}
```

```
\IfRealLayerPageStyleExists{page style name}{then code}{else code}
```

Command `\IfLayerPageStyleExists` tests, whether or not the real page style of *page style name* is a layer page style. If the test is true, *then code* will be executed. If *page style name* is neither a layer page style, nor an alias of a layer page style, nor an alias of an alias of ... a layer page style, *else code* will be executed. Internally this command is often used to throw an error message if you use one of the layer page style commands with an *page style name* that does not correspond with a layer page style.

Command `\IfRealLayerPageStyleExists` is similar, but *then code* will only be executed, if *page style name* itself is the name of a layer page style. So *else code* will even be executed, if *page style name* is an alias name of a layer page style or the alias name of an alias name of ... a layer page style.

```
\IfLayerAtPageStyle{page style name}{layer name}{then code}{else code}
\IfSomeLayerAtPageStyle{page style name}{layer list}{then code}{else code}
\IfLayersAtPageStyle{page style name}{layer list}{then code}{else code}
```

Command `\IfLayerAtPageStyle` may be used to test, whether or not a layer named *layer name* is a member of the layer list of a given page style. If the test is true, the *then code* will be executed. If the layer is not a member of the layer list of *page style name*, the *else code* will be executed.

Commands `\IfSomeLayerAtPageStyle` and `\IfLayersAtPageStyle` do not only test one layer but several layers at a given, comma separated *layer list*. `\IfSomeLayerAtPageStyle` will execute the *then code* if *at least one* of the layers at *layer list* is a member of the layer list of *page style name*. In difference `\IfLayersAtPageStyle` executes the *then code* only if *all* of the layers at *layer list* are members of the layer list of *page style name*.

```
\DestroyRealLayerPageStyle{page style name}
```

Command `\DestroyRealLayerPageStyle` makes the page style named *page style name* undefined, if and only if it is a layer page style. Nothing will be happen if it is an alias name of a layer page style, if it is another page style, or if it is not a page style.

If *page style name* is the name of the current page style the current page style will become a kind of empty page style. If the special page style — this may be set using `\thispagestyle` — is *page style name*, this will be simply reset. So the previous `\thispagestyle` will become invalid.

Note, the layers of the page style will not be destroyed automatically. If you want to destroy the layers too, you may use

```
\ForEachLayerOfPageStyle{...}{\DestroyLayer{#1}}
```

before destroying the layer page style.

The command is intended to be used inside the auto-remove code of an interface See [section 17.8](#) below for more information about auto-remove code.

17.6. Head and Foot Height

Page head and footer are the main elements not only of a page style. Layer can also be restricted to these using corresponding options (see [table 17.1](#), [page 387](#)). Therefore the heights of these elements are very important and have to be well defined.

All of what is described in [section 5.2](#) is generally applicable. So if you have already read and understood [section 5.2](#) you can switch to [section 17.7](#), [page 405](#).

```
\footheight
\headheight
```

The package `scrlayer` introduces `\footheight` as a new length similar to `\headheight` of the \LaTeX kernel. Additionally `scrlayer-scrpage` interprets `\footskip` to be the distance from the last possible base line of the text area to the first normal base line of the footer. Package `typearea` interprets `footheight` in the same way. So `typearea`’s foot height options may also be used to setup the values for packages `scrlayer` and `scrlayer-scrpage`. See option `footheight` and `footlines` in [section 2.6, page 43](#)) and option `footinclude` at [page 40](#) of the same section.

If you do not use package `typearea`, you should setup the head and foot height using the lengths directly where necessary. At least for the head package `geometry` provides similar settings. If you setup a head or foot height that is too small for the effective content, `scrlayer-scrpage` will try to adjust the corresponding lengths properly. Furthermore, it will warn you and give you additional information about the changes and proper settings you may use yourself. The automatic changes will become valid immediately after the need for them has been detected. They will never be removed automatically, however, even if content with a lower height requirement should be detected at a later point in time.

17.7. Manipulation of Defined Page Styles

Even though `scrlayer` itself does not define a concrete page style with content — the already mentioned page styles `@everystyle` and `empty` are initially defined without any level and so empty and without content —, it provides some options and commands for the manipulation of the contents.

```
\automark[section level of the right mark]{section level of the left mark}
\automark*[section level of the right mark]{section level of the left mark}
\manualmark
automark
autooneside=simple switch
manualmark
```

In common classes the decision for a page style — generally `headings` and `myheading` — is also a decision to use either automatic or manual running heads. This difference between those two page styles has been abolished by `scrpage2` and also by `scrlayer`. Instead of distinguishing between automatic and manual running head by the selection of a page style, two new commands, `\automark` and `\manualmark`, are provided.

The command `\manualmark` switches to manual marks and deactivates the automatic filling of the marks. In contrast to this `\automark` and `\automark*` can be used to define, which section levels should be used for the automatic mark filling. The optional argument is the *section level of the right mark*, the mandatory argument the *section level of the*

left mark. The arguments always should be the name of a section level like `part`, `chapter`, `section`, `subsection`, `subsubsection`, `paragraph`, or `subparagraph`.

Generally the higher level should be used for the left mark and the lower level for the right mark. This is only a convention and not a need, but it makes sense.

Please note that not every class provides running heads for every section level. For example the standard classes never setup the running head of `\part`. The KOMA-Script classes provide running heads for every section level.

The difference in `\automark` and `\automark*` is, that `\automark` deletes all prior usages of `\automark` or `\automark*`, while `\automark*` changes only the behaviour of the section levels of its arguments. So you can even build more complex cases.

Instead of the commands you may also use the options `manualmark` and `automark` to switch between manual and automatic running heads. Thereby `automark` always uses the default

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

for classes with `\chapter` and

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

for classes without `\chapter`.

But normally in single side mode you do not want that the lower level influences the right mark, you want the higher level, that will fill only the left mark in double side layout, to be the running head of all pages. The default option `autooneside` corresponds to this behaviour. The option understands the values for simple switches, that can be found in [table 2.5 on page 39](#). If you'd deactivate the option, in single side layout the optional and the obligatory arguments of `\automark` and `\automark*` will influence the running head again.

Please note, only loading the package does not have any effect on the fact whether automatic or manual running heads are used or what kind of section headings do fill up the marks. Only using an explicit option `automark` or `manualmark` or one of the commands `\automark` or `\manualmark` can reach a well defined state.

```
draft=simple switch
```

This KOMA-Script option understands the values for simple switches, that are shown in [table 2.5 on page 39](#). If the option is active, all elements of the page styles will also show dimension lines. This might be useful while draft editing.

```
\MakeMarkcase{string}
```

The automatic running head and only this uses `\MakeMarkcase` for its output. If the command has not been defined, e.g., by the class, while loading `scrlayer`, it would be defined with the default of outputting the argument *string* without changes. But the default can be change either redefining `\MakeMarkcase` or using option `markcase`, that will be described next. Depending on the setting the argument could, e.g., be converted into upper or lower cases.

```
markcase=Wert
```

As already mentioned with `sclayer` you may switch between manual and automatic running heads. Using automatic running heads the corresponding marks will be filled by the section heading commands. Some culture areas do convert the running heads into upper case letters in opposite to the German typographic habit. The \LaTeX standard classes do so by default. Package `sclayer` optionally provides this too. Therefor you'd use option `markcase=upper` which results in a redefinition of `\MakeMarkcase`, a command, that is used by `sclayer` for automatic running heads.

Unfortunately \LaTeX 's command for upper case letter typesetting, `\MakeUppercase` results in an very inadequate typesetting, because it neither uses letter spacing nor does it space balancing. One reason for this behaviour might be, that a glyph analyzing would be needed, to incorporate the letter shapes and their combination while space balancing. Because of this KOMA-Script author suggests to abstain from upper case letter typesetting for running heads. This could be achieved using `markcase=used`. Nevertheless, some classes would add `\MarkUppercase` or even \TeX command `\uppercase` into the running heads. For such cases option `markcase=noupper` can be used. This will also deactivate `\MakeUppercase` and `\uppercase` inside the running heads.

You can find all valid values for `markcase` in [table 5.2, page 245](#).

```
\leftmark
\rightmark
\headmark
\pagemark
```

If you want to differ from the predefined page styles, usually you need to decide, where to place the marks' contents. With `\leftmark` you can state the contents of the left mark.

Similar you can use `\rightmark` to state the contents of the right mark. For more information about some intricacies of this see the further description of [21.1](#) in [section 21.1, page 441](#).

Somehow easier would be usage of `\headmark`. This extension of `sclayer` aliases either `\leftmark` or `\rightmark` depending on whether the current page is even or odd.

Strictly thought command `\pagemark` is not involved by \TeX 's mark mechanism. It is only used to output a formatted page number. The font of element `pagenumber` will be used for the output. This can be changed using command `\setkomafont` or `\addtokomafont` (see also [section 3.6, page 57](#)).

If you are interested in an example about usage of commands `\headmark` and `\pagemark`, have a look into [section 5.5, page 245](#). The package `sclayer-scrpage` internally uses a lot of such features of `sclayer`.

If all the mark features described above are not sufficient, experienced users can find more of them on [page 409](#). For example, you can find there `\leftfirstmark` and `\rightbotmark`, which seem to be useful for lexicon like documents.

```

\partmarkformat
\chaptermarkformat
\sectionmarkformat
\subsectionmarkformat
\subsubsectionmarkformat
\paragraphmarkformat
\subparagraphmarkformat

```

Usually the KOMA-Script classes and package `sclayer` use these commands internally to bring the section numbers into wanted form and additionally they support the `\autodot` mechanism of the KOMA-Script classes. If wanted these commands may be redefined to get another form of section numbers. See the example in [section 5.5, page 246](#) for more information.

```

\partmark{Text}
\chaptermark{Text}
\sectionmark{Text}
\subsectionmark{Text}
\subsubsectionmark{Text}
\paragraphmark{Text}
\subparagraphmark{Text}

```

Most classes use these commands to setup marks corresponding to the section commands. Thereby the only argument is the text without the number of the section heading, that should be used for the running head. For the number simply the number of the current section level will be used, if the current level uses numbers.

Unfortunately, not all classes use such a command for every section level. The standard classes for example do not call `\partmark`. The KOMA-Script classes support such commands for all section levels and therefore also use `\partmark`.

If users redefine these commands, they should take care to also use the counter `secnumdepth` for the test whether or not the section level should output a number, even in the case the user does not change counter `secnumdepth` himself, because packages and classes may do so locally and rely on correct handling of `secnumdepth`.

However, package `sclayer` redefines these commands whenever you use `\automark` or `\manualmark` or the corresponding options, to activate or deactivate the wanted running heads.

```

\markleft{left mark}
\markright{right mark}
\markboth{left mark}{right mark}

```

Independent of whether currently manual or automatic running heads are active, you may change the contents of the *left mark* or the *right mark* at any time using these commands. You should note, that the resulting contents of `\leftmark` is the *left mark* of the last `\markleft` or `\markboth` command of the current page. In opposite to this the resulting

contents of `\rightmark` is the *right mark* of the first `\rightmark` or `\markboth` command of the current page.

If you are using manual running heads, the marks will stay valid until one of the corresponding commands will be used again. If you are using automatic running heads the marks can loose their validity with the next section heading depending on the configuration of the automatism.

You may also use these commands together with the star versions of the section commands. You can find a detailed example about usage of `\markboth` with `sclayer-scrpage` in [section 5.5, page 247](#). Package `sclayer-scrpage` is a kind of extension of `sclayer`.

```
\GenericMarkFormat{name of the section level}
```

At the running head this command will be used by default for the formatting of all section numbers below the subsection and with classes without `\chapter` also for the section level and the subsection level, if the mark commands for those numbers have not been defined before loading `sclayer`. Thereby the package uses `\@secCNTmarkformat` if such a command has been defined, like the KOMA-Script classes do. Otherwise `\@secCNTformat` will be used, which is provided by the L^AT_EX kernel. The expected, mandatory argument of the command is the *name of a section level*, i.e., `chapter` or `section without` the backslash prefix.

You can change the default formatting of the numbers, that use this command, by redefining only this single command. Classes can change the default formatting also by defining this command.

A detailed example about cooperation of command `\GenericMarkFormat` and the commands `\sectionmarkformat` and `\subsectionmarkformat` that are described at [page 408](#) is shown in [section 18.1, page 413](#).

```
\righttopmark
\rightbotmark
\rightfirstmark
\lefttopmark
\leftbotmark
\leftfirstmark
```

v3.16

For page styles L^AT_EX uses a bipartite T_EX mark. Running heads can use the left part of that mark by `\leftmark` and the right part by `\rightmark`. Indeed, it seems that it was suggested to use `\leftmark` for the running head of left, even pages and `\rightmark` for the running head of right, odd pages of double-sided documents. Within single-sided layouts the standard classes even do not set the left part of the mark.

T_EX itself knows three alternatives to make usage of a mark. The `\botmark` is the last valid mark of the page that has been build last. If there has not been set any mark on the page, it is the last mark of the already shipped out pages. L^AT_EX command `\leftmark` uses

exactly this mark. So it shows the left part of the last mark of the page. This is the same like `\leftbotmark`. In opposite to this `\rightbotmark` shows the right part of that mark.

`\firstmark` is the first mark of the page that has been build last. This means the first mark, that has been set on the page. If there has not been set any mark on the page, this is the last mark of the already shipped out pages. L^AT_EX command `\rightmark` uses exactly this mark. So it shows the right part of the first mark of the page. This is the same like `\rightfirstmark`. In opposite to this `\leftfirstmark` shows the left part of that mark.

`\topmark` is the content of `\botmark` before building the current page. L^AT_EX itself does not use it. Nevertheless, `sclayer` provides `\lefttopmark` to show the left part of it and `\righttopmark` to show the right part of it.

Note that the left part and the right part of the mark can be made all together only. Even if you use `\markright` to change only the right part, the left part will be made again (unchanged). Accordingly in double-side mode using page style `headings` the higher section levels always make both parts. For example `\chaptermark` uses `\markboth` with an empty right argument in this case. This is the reason why `\rightmark` or `\rightfirstmark` always shows an empty value on pages with chapter beginnings, even if there was a `\sectionmark` or `\section` on the same page to make the right part of the mark.

Please note that using any of these commands to show the left or right part of the mark as part of the page content could have unexpected results. They are recommended to be used in the head or foot of a page style only. So with `sclayer` they should be part of the contents of a layer. But it does not matter whether or not a layer is restricted to the background or the foreground. Both kind of layers are shipped out after building the current page.

If you need more information about the mark mechanism of T_EX, please have a look at [Knu90, chapter 23]. You will find, that marks are an issue for real experts. So if the explanation above was at least more confusing than illuminative: Please don't worry.

```
\@mkleft{left mark}
\@mkright{right mark}
\@mkdouble{mark}
\@mkboth{left mark}{right mark}
```

Sometimes inside classes and packages the marks for running heads should be filled up only if automatic running heads are active (see option `automark` and command `\automark` on page 241). In the L^AT_EX standard classes only `\@mkboth` has been used for this. This command is either `\@gobbletwo`, that simply destroys both mandatory arguments, or `\markboth`, a command that fills up the left and `left mark` and the `right mark`. Packages like `babel` also change `\mkboth`, i. e., to add language switching to the running head.

But if a class or package author only wants to change either the `left mark` or the `right mark` without the other one, a proper command is missing. Package `sclayer` itself needs such commands for the implementation of all cases of automatic running heads. So if command `\@mkleft` to fill up only the left mark or `\@mkright` to fill up only the right mark or `\@mkdouble`

to fill up both marks with the same content is undefined while loading `scrlayer`, the package will define them. Thereby the actual definition of `\@mkboth` will be used as an indicator whether or not automatic running heads should be used. The new commands will fill up the marks only in the case of automatic running heads.

Class and package authors can use these commands too, if they want to fill up only one of the marks or both marks with the same content but only if automatic running heads are activated. This condition is also the difference to the commands `\markleft`, `\markright`, and `\markboth`.

For more information about manipulation of the contents of page styles see also [section 5.5](#) from [page 240](#).

17.8. End User Interfaces

Package `scrlayer` provides an interface to define and manage (concurrent) end user interfaces. Maybe future releases of KOMA-Script will provide parts of this by package `scrbase` and remove those commands from `scrlayer`. But currently this part of `scrlayer` is very experimental, so package `scrlayer` provides its own interface definition commands. Currently you should not depend on correct working of auto-removing a concurrent end user interface. Instead you should avoid using concurrent end user interfaces.

This section only describes the interface commands for defining end user interfaces. This is not interesting for end users, but only for authors of end user interfaces. End users will find information about the end user interfaces in the sections about the particular end user interface, e.g., [chapter 5](#), [chapter 18](#), and [chapter 19](#).

```
\scrlayerInitInterface[interface name]
```

Command `\scrlayerInitInterface` registers a new interface. The *interface name* must be unique. If you try to initialise an already initialised interface an error will occur. This command is obligatory and mandatory for interfaces. It should be the first interface command and therefore has been described first. If the optional argument is omitted, `\@currname.\@currentx` will be used instead. For classes and packages this will be the file name of the class or package while loading the class or package. But you may use any sequence of characters with category letter or other.

```
forceoverwrite=simple switch
autoremoveinterfaces=simple switch
\scrlayerAddToInterface[interface name]{command}{code}
\scrlayerAddCsToInterface[interface name]{command sequence}{code}
```

One of the special features of end user interfaces is that they should register all interface dependent commands (also known as *macros*). You may do this using `\scrlayerAddToInterface`. If your interface generates macros not only at load time but also at run time or if the interface name should not be the class's or package's name, you have to use the optional argument to add the command to a dedicated interface. An error will occur, if the interface has not been initialised before.

The first mandatory argument is the *command*¹ that should be added to the interface. If the command can be added to the interface, it will be added to the interface, will be set to `\relax` and *code* will be executed. You can use, e.g., `\newcommandcommand` inside of *code* to define *command*.

But when can a command be defined? If a command is undefined or `\relax` it can be defined. If a command has already been defined and registered for another interface *and* if KOMA-Script option `autoremoveinterfaces` has been switched on, the other interface will be removed automatically and the new command will be set to `\relax` and will be registered for the given interface. If a command has already been defined but is not part of another interface *and* if KOMA-Script option `forceoverwrite` has been switched on, the command will be set to `\relax` and will be registered for the given interface.

Command `\scrlayerAddCsToInterface` is similar to `\scrlayerAddToInterface` but does not expect a command as first, mandatory argument, but a command sequence².

```
\scrlayerOnAutoRemoveInterface[interface name]{code}
```

Command `\scrlayerOnAutoRemoveInterface` registers *code* to be executed, if the interface will be automatically removed (see `autoremoveinterfaces` prior in this section). This may be used, e.g., to automatically destroy layers or page styles (see `\DestroyLayer`, `\DestroyPageStyleAlias`, and `\DestroyRealLayerPageStyle`).

¹The *command* consists of the backslash followed by a *command sequence* consisting of characters with category code letter or one other character, or *command* consists of one active character (without backslash).

²A command sequence may consist of any characters with category code letter or other.

v3.12

Additional Features with package sctlayer-scrpage

Package sctlayer-scrpage provides features that have not been described in [chapter 5](#) of this user guide's [part I](#). In general, the average user will not need those extensions and some of them are only provided for compatibility with scrpage2.

18.1. Manipulation of Defined Page Styles

This section is an add-on to [section 17.7](#). It describes features that may be too complicated for beginners.

```
\GenericMarkFormat{name of the section level}
```

At the running head this command will be used by default for the formatting of all section numbers below the subsection and with classes without `\chapter` also for the section level and the subsection level, if the mark commands for those numbers have not been defined before loading sctlayer. Thereby the package uses `\@secntmarkformat` if such a command has been defined, like the KOMA-Script classes do. Otherwise `\@secntformat` will be used, which is provided by the L^AT_EX kernel. The expected, mandatory argument of the command is the *name of a section level*, i.e., `chapter` or `section without` the backslash prefix.

You can change the default formatting of the numbers, that use this command, by redefining only this single command. Classes can change the default formatting also by defining this command.

Example: Assume you want the section numbers of all levels at the running head of an article to be printed in white colour inside a black coloured box. Using standard L^AT_EX article class, package sctlayer defines the number mark commands `\sectionmarkformat` and `\subsectionmarkformat` using `\GenericMarkFormat`. So you need to redefine only this single command:

```
\documentclass{article}
\usepackage{blindtext}
\usepackage[automark]{sctlayer-scrpage}
\pagestyle{scrheadings}
\usepackage{xcolor}
\newcommand*{\numberbox}[1]{%
  \colorbox{black}{\strut~\textcolor{white}{#1}~}}
\renewcommand*{\GenericMarkFormat}[1]{%
  \protect\numberbox{\csname the#1\endcsname}\enskip}
\begin{document}
\blinddocument
\end{document}
```

The colour has been done using package `xcolor`. See the package manual for more information about this.

Additionally a not visible strut has been used. Every complete L^AT_EX introduction should explain the related command `\strut`.

A helper macro `\numberbox` has been defined for the formatting of the number within a box. In the redefinition of `\GenericMarkFormat` this has been prefixed by `\protect` to protect it from expansion. This is necessary because otherwise the upper case letter conversion of `\MakeUppercase` that will be used for the running head, would result in an ask for colours “BLACK” and “WHITE” instead of “black” and “white” and those colours are undefined. Alternatively you may define `\numberbox` using `\DeclareRobustCommand*` instead of `\newcommand*` and then omit `\protect` (see [Tea06]).

If you’d do the same with a KOMA-Script class or with one of the L^AT_EX standard classes `book` or `report`, you’d additionally have to redefine `\sectionmarkformat` and `\subsectionmarkformat` respectively `\chaptermarkformat` and `\sectionmarkformat` and too, because then these would not have been defined using `\GenericMarkFormat`:

```
\documentclass{scrbook}
\usepackage{blindtext}
\usepackage[automark]{scrlayer-scrpage}
\pagestyle{scrheadings}
\usepackage{xcolor}
\newcommand*\numberbox[1]{%
  \colorbox{black}{\strut~\textcolor{white}{#1}~}}
\renewcommand*\GenericMarkFormat[1]{%
  \protect\numberbox{\csname the#1\endcsname}\enskip}
\renewcommand*\chaptermarkformat{\GenericMarkFormat{chapter}}
\renewcommand*\sectionmarkformat{\GenericMarkFormat{section}}
\begin{document}
\blinddocument
\end{document}
```

```
\righttopmark
\rightbotmark
\rightfirstmark
\lefttopmark
\leftbotmark
\leftfirstmark
```

v3.16

For page styles L^AT_EX uses a bipartite T_EX mark. Running heads can use the left part of that mark by `\leftmark` and the right part by `\rightmark`. Indeed, it seems that it was suggested to use `\leftmark` for the running head of left, even pages and `\rightmark` for the running head of right, odd pages of double-sided documents. Within single-sided layouts the standard classes even do not set the left part of the mark.

T_EX itself knows three alternatives to make usage of a mark. The `\botmark` is the last valid mark of the page that has been build last. If there has not been set any mark on the page, it is the last mark of the already shipped out pages. L^AT_EX command `\leftmark` uses exactly this mark. So it shows the left part of the last mark of the page. This is the same like `\leftbotmark`. In opposite to this `\rightbotmark` shows the right part of that mark.

`\firstmark` is the first mark of the page that has been build last. This means the first mark, that has been set on the page. If there has not been set any mark on the page, this is the last mark of the already shipped out pages. L^AT_EX command `\rightmark` uses exactly this mark. So it shows the right part of the first mark of the page. This is the same like `\rightfirstmark`. In opposite to this `\leftfirstmark` shows the left part of that mark.

`\topmark` is the content of `\botmark` before building the current page. L^AT_EX itself does not use it. Nevertheless, scrlayer provides `\lefttopmark` to show the left part of it and `\righttopmark` to show the right part of it.

Note that the left part and the right part of the mark can be made all together only. Even if you use `\markright` to change only the right part, the left part will be made again (unchanged). Accordingly in double-side mode using page style `headings` the higher section levels always make both parts. For example `\chaptermark` uses `\markboth` with an empty right argument in this case. This is the reason why `\rightmark` or `\rightfirstmark` always shows an empty value on pages with chapter beginnings, even if there was a `\sectionmark` or `\section` on the same page to make the right part of the mark.

Please note that using any of these commands to show the left or right part of the mark as part of the page content could have unexpected results. They are recommended to be used in the head or foot of a page style only. So with scrlayer they should be part of the contents of a layer. But it does not matter whether or not a layer is restricted to the background or the foreground. Both kind of layers are shipped out after building the current page.

If you need more information about the mark mechanism of T_EX, please have a look at [Knu90, chapter 23]. You will find, that marks are an issue for real experts. So if the explanation above was at least more confusing than illuminative: Please don't worry.

```
\@mkleft{left mark}
\@mkright{right mark}
\@mkdouble{mark}
\@mkboth{left mark}{right mark}
```

Sometimes inside classes and packages the marks for running heads should be filled up only if automatic running heads are active (see option **automark** and command **\automark** on [page 241](#)). In the L^AT_EX standard classes only **\@mkboth** has been used for this. This command is either **\@gobbletwo**, that simply destroys both mandatory arguments, or **\markboth**, a command that fills up the left and **left mark** and the **right mark**. Packages like **babel** also change **\mkboth**, i. e., to add language switching to the running head.

But if a class or package author only wants to change either the *left mark* or the *right mark* without the other one, a proper command is missing. Package **scrlayer** itself needs such commands for the implementation of all cases of automatic running heads. So if command **\@mkleft** to fill up only the left mark or **\@mkright** to fill up only the right mark or **\@mkdouble** to fill up both marks with the same content is undefined while loading **scrlayer**, the package will define them. Thereby the actual definition of **\@mkboth** will be used as an indicator whether or not automatic running heads should be used. The new commands will fill up the marks only in the case of automatic running heads.

Class and package authors can use these commands too, if they want to fill up only one of the marks or both marks with the same content but only if automatic running heads are activated. This condition is also the difference to the commands **\markleft**, **\markright**, and **\markboth**.

18.2. Definition of new Pairs of Page Styles

In [section 5.4](#) the page styles **scrheadings** and **plain.scrheadings** have been described. These can be seen as a pair of page styles, with **scrheadings** being the main page style with a running head and **plain.scrheadings** being the corresponding **plain** page style without running head but generally with pagination. Additionally to the configuration of these page styles, **scrlayer-scrpage** provides also the feature of defining new such pairs of page styles. Thereby the name of the main page style, e. g., **scrheadings**, can be seen as the name of the pair of page styles.

Most users will not need more than the predefined pair **scrheadings**. So the commands of this section may be understood as an addition for special cases. And as each such case is very special, this section also lacks of handsome examples. Nevertheless, if the support will show me a real nice example in the future, I perhaps could use it in the future. However, I'm almost sure that all such cases could also be solved using the predefined pair only.


```

\defpairofpagestyles[parent pair]{name}{definition}
\newpairofpagestyles[parent pair]{name}{definition}
\renewpairofpagestyles[parent pair]{name}{definition}
\providepairofpagestyles[parent pair]{name}{definition}

```

With these commands you may define pairs of page styles similar to `scrheadings` and `plain.scrheadings`. Thereby *name* is the name of the main page style that is similar to `scrheadings`. The name of the corresponding `plain` page style will be prefixed by `plain.` automatically. So *name* is not only the name of the pair of page styles, but also the name of the main page style of that pair, while `plain.name` is the name of the `plain` page style of this pair.

If the optional argument *parent pair* is given, the settings of the pair of page styles with that name are the initial settings of the new pair of page styles. So the new pair inherits the configuration of the *parent pair*.

Reading [section 5.4](#) might have created the impression that the described commands are related to `scrheadings` and `plain.scrheadings` only. However, if there exist more pairs of page styles, these commands are related to the pair that has been activated last. In fact, this is the case for `\lehead`, `\cehead`, `\rehead`, `\lohead`, `\cohead`, `\rohead`, `\lefoot`, `\cefoot`, `\refoot`, `\lofoot`, `\cofoot`, `\rofoot`, `\ihead`, `\chead`, `\ohead`, `\ifoot`, `\cfoot`, and `\ofoot` of [section 5.4](#).

All those commands and the below described commands `\clearmainofpairofpagestyles`, `\clearplainofpairofpagestyles`, and `\clearpairofpagestyles` are also designed to be used inside the argument *definition*. In that case, they are a kind of default configuration of the pair of page styles that is executed each time the pair will be activated. A pair of page styles is activated, if one of the page styles of the pair will be activated, which is mostly done by using `\pagestyle`.

Please note that the commands of [section 5.5](#) are for general purpose. Therefore, they are related to every page style that has been defined using `scrlayer-scrpage`.

Whereas `\defpairofpagestyles` will define a pair no matter if the corresponding page styles are already defined, `\newpairofpagestyles` and `\providepairofpagestyles` will define the pair only, if the page styles are currently undefined. If at least one of the page styles is already defined, the new definition of `\providepairofpagestyles` will be ignored, but usage of `\newpairofpagestyles` would result in an error message. To redefine already existing pairs of page styles you may use `\renewpairofpagestyles`. With this an error would be thrown, if at least one of the page styles of the pair does not already exist.

```
\clearmainofpairstyles  
\clearplainofpairstyles  
\clearpairstyles
```

Command `\clearmainofpairstyles` will configure the main page style of the lastly activated pair of page styles to be empty. In contrast to this `\clearplainofpairstyles` will configure the `plain` page style of the lastly activated pair of page styles to be empty. Last but not least `\clearpairstyles` will configure both page styles of the lastly activated pair of page styles to be empty.

But please note that none of these commands will remove the definitions of argument *definition* that has been used when defining the pair of page styles (see above). So if you activate the pair of page styles again, those definitions will be used again!

These commands may be used inside of *definition* themselves. But you may use them outside the definition of a pair of page styles too. In this case they are related to the lastly activated pair of page styles.

The commands `\clearscrheadings`, `\clearscrplain`, and `\clearscrheadfoot` are aliases for these commands provided only for compatibility with `scrpage2`. Nevertheless, they are deprecated and should not be used.

18.3. Definition of Simple Page Styles with Three Parts in Head and Foot

Beside predefined page styles and the definition of new pairs of page styles, package `scrlayer-scrpage` also provides the definition of page styles with three parts in head and foot. But this user interface exists for compatibility with `scrpage2` only. However, it has been extended beyond the functionality of `scrpage2` to provide a more consistent set of commands. Nevertheless, for new documents it is recommended to instead use the advanced features described in the previous section.

```

\deftriplepagestyle{name of the page style}[thickness of the outer line][thickness
    of the inner line]{inner head element}{centre head element}
    {outer head element}{inner foot element}{centre foot element}
    {outer foot element}
\newtriplepagestyle{name of the page style}[thickness of the outer line][thickness
    of the inner line]{inner head element}{centre head element}
    {outer head element}{inner foot element}{centre foot element}
    {outer foot element}
\renewtriplepagestyle{name of the page style}[thickness of the outer line][thickness
    of the inner line]{inner head element}{centre head element}
    {outer head element}{inner foot element}{centre foot element}
    {outer foot element}
\providetriplespagestyle{name of the page style}[thickness of the outer line]
    [thickness of the inner line]{inner head element}{centre head
    element}{outer head element}{inner foot element}{centre foot
    element}{outer foot element}

```

With these commands you can define a single page style, whose head and foot are parted into three elements: an inner element, a centre element, and an outer element. The contents of the elements are stated via three mandatory arguments for the head and three mandatory arguments for the foot.

Each of the two inner elements will be printed right aligned on left pages and left aligned on right pages. The centre elements will be printed centred in the head respectively foot of left and right pages. Each of the two outer elements will be printed left aligned on left pages and right aligned on right pages.

Please note once again: There are only right pages in single side layout!

The two optional arguments can be used to activate horizontal lines in the head and foot of the page style. To do so, you put the wanted thickness of the line as an optional argument. The first optional argument is the *thickness of the outer line* in both, the page head and footer. In the page head it will be printed above the head contents. In the page footer it will be printed below the foot contents. The second optional argument is the *thickness of the inner line*. This will be printed below the page head and additionally above the page footer. If you have only one optional argument, this is the *thickness of the inner line*. In this case no outer line will be printed. And if you omit both optional arguments, both lines will be omitted.

Please note, an empty optional argument is not the same like omitting the optional argument! Empty optional arguments are not allowed in this case. But you may use simple dimension expressions as explained for KOMA-Script option `headwidth` (see [section 5.5, page 249](#)) for the optional arguments.

[Figure 18.1](#) shows a sketch for the interpretation of the arguments on a schematic double page. Thereby “outer”, “inner”, and “centre” are example contents. The corresponding arrows illustrate the expansion of those contents. The coloured texts inside the pages are only

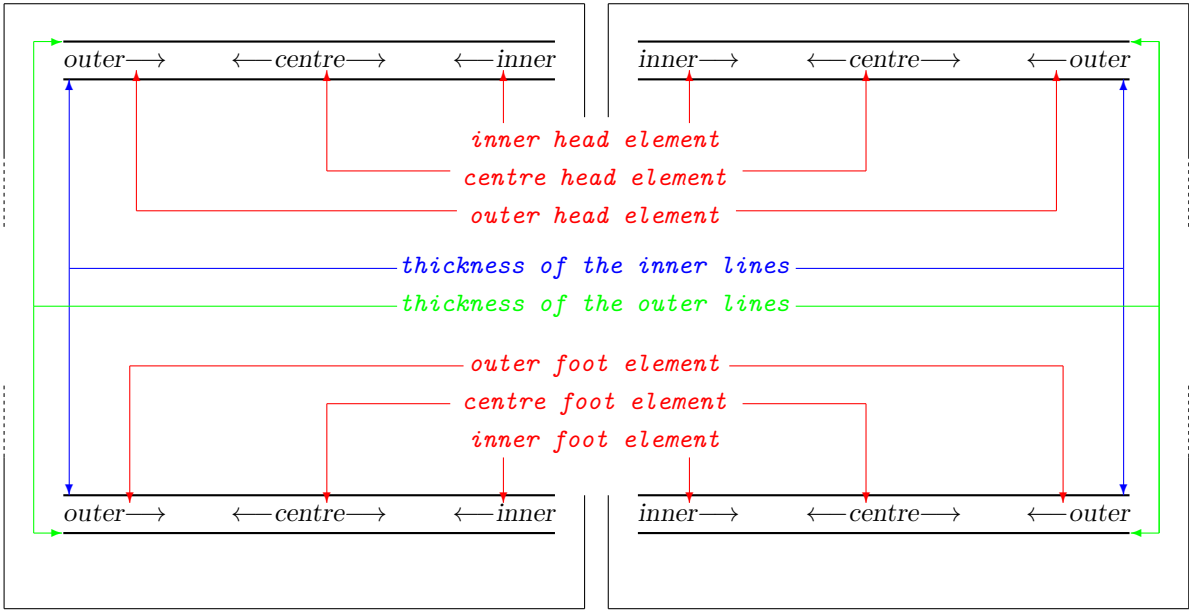


Figure 18.1.: Schematic double page with the elements of a tripartite page style of commands `\deftriplepagestyle`, `\newtriplepagestyle`, `\providetripelapagestyle`, and `\renewtriplepagestyle`

the names of the arguments of these commands. The related arrows in the same colour point to the corresponding elements of the page.

Using the described commands it is not possible to draw only a single line, e. g., only below the page head but not above the page footer. If you need to do so, please consult the previous or following section.

The length of a single element is not limited by the existence of a neighbour element. So in unfortunate circumstances it can happen that one elements overwrites a neighbour element or even both of them. The user himself is responsible to avoid such inappropriate usage. With automatic running heads this could, e. g., be done using the optional argument of the section commands. See the manual of the document class for more information about those commands.

Command `\deftriplepagestyle` defines a page style independent from whether or not a page style with the same *name of the page style* already exists. In difference to this `\newtriplepagestyle` and `\providetripelapagestyle` define the page style only, if the *name of the page style* is not the name of an already defined page style. Otherwise `\providetripelapagestyle` simply does nothing, but `\newtriplepagestyle` throws an error. Command `\renewtriplepagestyle` is something like the opposite of `\newtriplepagestyle`. It throws an error, if there has not been a page style with the *name of the page style* before,

and defines the page style only, if there has been already a page style with the given name.

The command `\deftriplestyle` of `scrpage2` corresponds to `\deftriplepagestyle`. Because of unification reasons it has been renamed and should not be used with its old name any longer. Using the old name would result in a warning message. Nevertheless, the result would be the expected one.

18.4. Definition of Complex Page Styles

Beside predefined page styles, the definition of new pairs of page styles and the deprecated definition of page styles with tripartite head and foot, the package `scrlayer-scrpage` additionally provides basic features to define new page styles. All already described page style definitions of `scrlayer-scrpage` use internally these basic features. Because of the complexity of this user interface it is recommended for advanced users only. As all most all imaginable use cases of page styles can be handled using the features described previous section, less advanced users may skip this section.

```
\defpagestyle{name}{head specification}{foot specification}
\newpagestyle{name}{head specification}{foot specification}
\providepagestyle{name}{head specification}{foot specification}
\renewpagestyle{name}{head specification}{foot specification}
```

These commands can be used to define a single page style with maximum flexibility. Thereby *name* is the name of the page style that should be defined.

The arguments *head specification* and *foot specification* have identical structure:

```
(length of the line above,thickness of the line above)%
{specification for the left page in two-side layout}%
{specification for the right page in two-side layout}%
{specification for all pages in one-side layout}%
(length of the line below,thickness of the line below)
```

The arguments in the round brackets are options. This means, you may omit them together with the brackets. In that case, the length and thickness of the corresponding horizontal line would be given by the KOMA-Script options `headtopline`, `headsepline`, `footsepline` or `footbotline` (see [section 5.5](#), [page 249](#)).

All three arguments in the curly brackets are mandatory. They hold arbitrary content specifications depending on the page and the layout settings. But for page styles with running heads usage of `\headmark`, `\leftmark`, or `\rightmark` is recommended inside the specification. Section numbers or section headings must not be used in the specifications. Due to the asynchronous page building of L^AT_EX, using them could result in wrong numbers or texts in the page header or footer.

Table 18.1.: The layers scrlayer-scrpage defines to a page style *name*

Name of the layer	Meaning of the layer
<i>Name</i> .head.above.line	horizontal line above the page head
<i>Name</i> .head.odd	page head of odd pages in two-side layout
<i>Name</i> .head.even	page head of even pages in two-side layout
<i>Name</i> .head.onside	page head in single-side layout
<i>Name</i> .head.below.line	horizontal line below the page head
<i>Name</i> .foot.above.line	horizontal line above the page foot
<i>Name</i> .foot.odd	page foot of odd pages in two-side layout
<i>Name</i> .foot.even	page foot of even pages in two-side layout
<i>Name</i> .foot.onside	page foot in single-side layout
<i>Name</i> .foot.below.line	horizontal line below the page foot

Command `\defpagestyle` defines the page style no matter if a page style with the same *name* already exists. Command `\newpagestyle` would throw an error, if such a page style already exists. In contrast to this, `\providepagestyle` simple does nothing in such a case. In contrast to `\newpagestyle` command `\renewpagestyle` throws an error, if a page style with the *name* does not already exist, and therefore may be used to redefine an existing page style.

All these commands are based on the command `\DeclarePageStyleByLayers` of package `scrlayer`. [Table 18.1](#) shows the layers that are defined for a page style *name*. More information about layers and layer page styles can be found starting on [page 382](#) in [chapter 17](#) of [part II](#).

Example: Assume you want to colour the whole background of page style `scrheadings`’ head. Because you read the introduction to this chapter, you know, that `scrheadings` last but not least is a layer page style with, e.g., layers `scrheadings.head.even`, `scrheadings.head.odd`, and `scrheadings.head.onside`. So you define three more layers for the backgrounds and add them at the very beginning of the page style:

```
\documentclass{scrartcl}
\usepackage[automark]{scrlayer-scrpage}
\usepackage{xcolor}
\usepackage{blindtext}
\DeclareLayer[clone=scrheadings.head.onside,
  contents={%
    \color{yellow}\rule[-\dp\strutbox]{\layerwidth}{\layerheight}}%
]{scrheadings.head.onside.background}
\DeclareLayer[clone=scrheadings.head.odd,
  contents={%
```

```

\color{yellow}\rule[-\dp\strutbox]{\layerwidth}{\layerheight}%
}%
]{scrheadings.head.odd.background}
\DeclareLayer[clone=scrheadings.head.even,
  contents={%
    \color{yellow}\rule[-\dp\strutbox]{\layerwidth}{\layerheight}%
  }%
]{scrheadings.head.even.background}
\AddLayersAtBeginOfPageStyle{scrheadings}{%
  scrheadings.head.onside.background,%
  scrheadings.head.odd.background,%
  scrheadings.head.even.background%
}
\pagestyle{scrheadings}
\begin{document}
\blinddocument
\end{document}

```

As you can see, the example uses three layers, so the position and size of that background layers may simply be copied from the corresponding head layer using option `clone`. This is much easier than using only one background layer and dynamically calculate the position time by time.

In this example the coloured background is printed using a `\rule` command. The size arguments of this `\rule` are given by `\layerwidth` and `\layerheight` which contain the current width and height of the layer itself. The optional argument of `\rule` is used to move the rule down by the height of a descender.

Instead of using new layers to colour the background in the example above, `\colorbox` and `\thead` could have been used. It is recommended to build such a solution as an exercise. Another approach could be to add the background layers one by one and just before the contents layers. You may do this as an exercise too.

hmode=simple switch

Package `scrpage2` outputs page heads and page foots in horizontal mode always. In opposite to this `scrlayer-scrpage` does not switch into horizontal mode itself, but the output of horizontal material will do so. Nevertheless, you can activate option `hmode` gain compatibility to `scrpage2` in this aspect. But this would have side effects, i. e., white spaces at the start of the output or the vertical alignment of the output.

The options allows for the standard values for simple switches that are given at [table 2.5](#) on [page 39](#). The option is deactivated by default.

Note Columns with `scrlayer-notecolumn`

Up to version 3.11b KOMA-Script supported note columns only by marginal notes that get their contents from `\marginpar` and `\marginline` (see [section 3.21](#), [page 136](#)). This kind of note columns has several disadvantages:

- Marginal notes can be output only completely on one page. Page breaks inside marginal notes are not possible. Sometimes this results in margin notes located close to the lower page margin.
- Marginal notes near page breaks sometimes float to the next page and then in case of two-sided layout with alternating marginal notes can be output at the wrong margin.. This problem can be solved by additional packages like `mparhack` or usage of `\marginnote` provided by package `marginnote`.
- Marginal notes inside floating environments or footnotes are not possible. This problem can be solved using `\marginnote` of package `marginnote`, too.
- There is only one marginal note column or at most two, if you work with `\reversemarginpar` and `\normalmarginpar`. You should know, that `\reversemarginpar` is of less usability on two-sided documents.

Usage of `marginnote` results in one more problem. Because the package does not have any collision detection, marginal notes that are set near to each other can partially or totally overlap. Moreover, usage of `\marginnote` sometimes and depending on the used settings can result in changes of the baseline distance of the normal text.

Package `scrlayer-notecolumn` should solve all these problems. To do so, it uses the basic functionality of `scrlayer`. Nevertheless, there is a disadvantage of using this package: Notes can be output only on pages that use a `scrlayer` based page style. This disadvantage may be easily resolved and maybe changed into an advantage using `scrlayer-scrpage`.

19.1. Note about the State of Development

Originally the package has been made as a so called *proof of concept* to demonstrate the potential of `scrlayer`. Despite the fact that it currently is in a very early state of development, the stability of most parts is less a question of `scrlayer-notecolumn` but of `scrlayer`. Nevertheless it is assumed that there are still bugs in `scrlayer-notecolumn`. You are welcome to report such bugs whenever you find one. Some of the disabilities are caused by minimisation of complexity. As an example note columns can break to several pages, but there is not a new line break of the paragraphs. `TEX` itself does not provide this.

Because the package is more experimental, the user manual belongs to the second part of the KOMA-Script manual, recommended for experienced users. If you are a beginner or a user

on the way to become an expert, some explanations could be ambiguous or incomprehensible. Please understand that I try to minimise the effort in time and work for the manual of an experimental package.

19.2. Early or late Selection of Options

All of what is described in [section 2.4](#) is generally applicable. So if you have already read and understood [section 2.4](#) you can switch to [section 19.3](#), [page 426](#).

In this section a peculiarity of KOMA-Script is presented, which, apart from the `sclayer-notecolumn` package, is also relevant to other KOMA-Script packages and classes. Such that the user can find all information corresponding to a single package or a single class in the relevant chapter, this section is found almost identically in several chapters. Users who are not only interested in a particular package or class, but wish to gain an overview of KOMA-Script as a whole, may read the section in one chapter and may thereafter skip it wherever coming across it in the document.

```
\documentclass[option list]{KOMA-Script class}
\usepackage[option list]{package list}
```

In \LaTeX , provision is made for the user to pass class options as a comma-separated list of keywords as optional arguments to `\documentclass`. Apart from being passed to the class, these options are also passed on to all packages which can understand the options. Provision is also made for the user to pass optional arguments as a comma-separated list of keywords as optional arguments to `\usepackage`. KOMA-Script expands the option mechanism for the KOMA-Script classes and various packages to use further possibilities. Thus, most KOMA-Script options can also take a value. An option may have not only the form *Option*, but may also have the form *option=value*. Apart from this difference `\documentclass` and `\usepackage` function the same in KOMA-Script as described in [\[Tea05b\]](#) or any introduction to \LaTeX , for example [\[OPHS11\]](#).

You should note, that in opposite to the interface described below the options interface of `\documentclass` and `\usepackage` is not robust. So commands, lengths, counters and such constructs may break inside the optional argument of these commands. Because of this, the usage of a \LaTeX length inside the value of an option would cause an error before KOMA-Script can get the control over the option execution. So, if you want to use a \LaTeX length, counter or command a part of the value of an option, you have to use `\KOMAOPTIONS` or `\KOMAOPTION`. These commands will be described next.

```
\KOMAOptions{option list}
\KMAOption{option}{value list}
```

v3.00

KOMA-Script offers most class and package options the opportunity to change the value of options even after loading of the class or package. One may then change the values of a list of options at will with the `\KOMAOptions` command. Each option in the *option list* has the form *option=value*.

Some options also have a default value. If one does not give a value, i. e., gives the option simply in the form *option*, then the default value will be used.

Some options can assume several values simultaneously. For such options there exists the possibility, with the help of `\KMAOption`, to pass a single *option* a list of values. The individual values are given as a comma-separated *value list*.

To implement this possibility KOMA-Script uses the commands `\FamilyOptions` and `\FamilyOption` with the family “KOMA”. For more information in these commands see [part II, section 12.2, page 313](#).

19.3. Text Markup

What is described in [section 3.6](#) applies, mutatis mutandis. So if you have already read and understood [section 3.7](#) you can switch to [page 427](#).

L^AT_EX offers different possibilities for logical and direct markup of text. Selection of the font family commands, as well as choosing the font size and width is supported. More information about the standard font facilities may be found at [\[OPHS11\]](#), [\[Tea05b\]](#), and [\[Tea05a\]](#).

```
\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`, and `\scshape`, as well as the font size commands `\Huge`, `\huge`, `\LARGE`, `\Large`, `\large`, `\normalsize`, `\small`, `\footnotesize`, `\scriptsize`, and `\tiny`. The description of these commands can be found in [\[OPHS11\]](#), [\[Tea05b\]](#), or [\[Tea05a\]](#). Color switching commands like `\normalcolor` (see [\[Car05\]](#) and [\[Ker07\]](#)) are also acceptable. The behavior when using other commands, especially those that make redefinitions 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 inside the document body, but only in the document preamble. Usage examples can be found in the paragraphs on the corresponding element.

With command `\usekomafont` the current font style may be changed into the font style of the selected *element*.

```
\usefontofkomafont{element}
\useencodingofkomafont{element}
\usesizeofkomafont{element}
\usefamilyofkomafont{element}
\useseriesofkomafont{element}
\useshapeofkomafont{element}
```

v3.12

Sometimes and despite the recommendation users use the font setting feature of elements not only for font settings but for other settings too. In this case it may be useful to switch only to the font setting of an element but not to those other settings. You may use `\usefontofkomafont` in such cases. This will activate the font size and baseline skip, the font encoding, the font family, the font series, and the font shape of an element, but no further settings as long as those further settings are local.

You may also switch to one of those attributes only using one of the other commands. Note, that `\usesizeofkomafont` will activate both, the font size and the baseline skip.

You should not misunderstand these commands as a legitimization of using all kind of commands at the font setting of an element. Hence this would result in errors sooner or later (see [section 21.3, page 444](#)).

19.4. Declaration of new Note Columns

Loading the package already declares a note column named `marginpar` automatically. The name already adumbrates that this note column is placed in the area of the normal marginal note column used by `\marginpar` and `\marginline`. It even recognises the setting of `\reversemarginpar` and `\normalmarginpar` page by page instead of note by note. So these switches are irrelevant when the user defines a note but relevant when the package outputs the note during L^AT_EX's page building. If you want to use notes at the left and at the right margin within one page, you should declare at least one more note column.

The default settings for every newly declared note column is the same like that of the predefined `marginpar`. But you can easily change them during declaration.

You should note that note columns can be output only on pages that use a page style made with package `scrlayer`. Package `scrlayer-notecolumn` automatically loads `scrlayer`, which by default provides only page style `empty`. If you need additional page styles, the use of `scrlayer-scrpage` is recommended.

```
\DeclareNoteColumn[option list]{note column name}
\DeclareNewNoteColumn[option list]{note column name}
\ProvideNoteColumn[option list]{note column name}
\RedeclareNoteColumn[option list]{note column name}
```

These commands can be used to declare note columns. `\DeclareNoteColumn` declares the note column independent of whether or not they already exist. `\DeclareNewNoteColumn` throws an error message if *note column name* has already been used for another note column. `\ProvideNoteColumn` simply does nothing in the same case. `\RedeclareNoteColumn` can be used only to declare an already existing note column again.

By the way: Already existing notes of a already existing note column are not lost if you re-declare it using `\DeclareNoteColumn` or `\RedeclareNoteColumn`.

Declaring a new note column does also define a new element of which you can change the font attributes using `\setkomafont` and `\addtokomafont` if such a element does not already exist. The name of the new element is `notecolumn.note column name`. Because of this the predefined note column `marginnote` has a element `notecolumn.marginpar`. The initial setting of the element's font can be done using option `font` as part of the *option list*.

The *option list* is a comma separated list of keys with or without values, also known as options. The available options are shown in [table 19.1](#). Option `marginpar` is set by default. But you can overwrite this default with your individual settings.

Because note columns are implemented using `scrlayer`, each note column also defines a layer. The name of the layer is the same as the name of the element, `notecolumn.note column name`. For more information about layers see [section 17.4](#), from [page 385](#).

Example: Assuming, you are a German professor of weird rights and like to write a paper about the new “Gesetz über die ausgelassene Verbreitung allgemeiner Späße e”, GüdaVaS. The main focus of the work should consist of comments to each section. So you use a two-columned layout. The comments would be the main text in the main column. The sections should be placed in a smaller note column on the right side of the main column using a smaller font and a different colour.

```
\documentclass{scrartcl}
\usepackage[ngerman]{babel}
\usepackage{selinput}
\SelectInputMappings{
  adieresis={ä},
  germandbls={ß},
}
\usepackage[T1]{fontenc}
\usepackage{lmodern}
\usepackage{xcolor}

\usepackage{scrjura}
\setkomafont{contract.Clause}{\bfseries}
```

Table 19.1.: Options for the declaration of note columns

font=font declaration

The initial font attributes to be used for the note column. Everything, that is allowed to be set by `\setkomafont` or `\addtokomafont` can be used as *font declaration* (see [section 3.6, page 57](#)). Note that `\normalfont\normalsize` will be used before. So you don't need one of them at your own initialisation.

Default: *empty*

marginpar

Sets up `position` and `width` to use the marginal note column of `\marginpar`. Note that this option does not expect or allow any value.

Default: *yes*

normalmarginpar

Sets up `position` and `width` to use the normal marginal note column and ignore `\reversemarginpar` and `\normalmarginpar`. Note that this option does not expect or allow any value.

Default: *no*

position=offset

Sets up the horizontal offset of the note column from the left edge of the paper. The *horizontal offset* can be either a L^AT_EX length, a T_EX dimension, a T_EX skip, a length value, or a dimensional expression using `+`, `-`, `*`, `/` and parenthesis (see [\[Tea98, section 3.5\]](#) for more information about dimensional expressions). The value will be calculated at usage time not at definition time.

Default: *by option marginpar*

reversemarginpar

Sets up `position` and `width` to use the reverse marginal note column and ignore `\reversemarginpar` and `\normalmarginpar`. Note that this option does not expect or allow any value.

Default: *no*

width=length

Sets up the horizontal size of the note column. You can use the same values for *size* like for *offset* of option `position`.

Default: *by option marginpar*

```

\setkeys{contract}{preskip=-\dp\strutbox}

\usepackage{scrlayer-scrpage}
\usepackage{scrlayer-notecolumn}

\newlength{\paragraphscolwidth}
\AfterCalculatingTypearea{%
  \setlength{\paragraphscolwidth}{%
    .333\textwidth}%
  \addtolength{\paragraphscolwidth}{%
    -\marginparsep}%
}
\recalctypearea
\DeclareNewNoteColumn[%
  position=\oddsidemargin+1in
    +.667\textwidth
    +\marginparsep,
  width=\paragraphscolwidth,
  font=\raggedright\footnotesize
    \color{blue}
]{paragraphs}

```

The paper should be a single-sided article. The German language (new spelling) is selected with package `babel`. The input encoding is selected and detected with package `selinput` automatically. The font is Latin Modern in 8-bit coding. The colour selection uses package `xcolor`.

For usage of package `scrjura` to typeset laws and contracts see the German manual of that package.

Next package `scrlayer-notecolumn` is loaded. The required width of the note column is calculated within `\AfterCalculatingTypearea` after package `typearea`'s each new calculation of the typing area. It should be one third of the typing area minus the distance between the main text and the note column.

With all this information we define the new note column. For the positions we simply use a dimension expression. Doing so, we have to note that `\oddsidemargin` is not the total left margin but for historical reasons the left margin minus 1 inch. So we have to add this value.

That is all of the declaration. But we have to note that currently the note column would be output inside the typing area. This means that the note column would overwrite the text.

```

\begin{document}

\title{Kommentar zum GüdVaS}

```

```

\author{Professor R. O. Tenase}
\date{11.\,11.-2011}
\maketitle
\tableofcontents

\section{Vormerkung}
Das GûdaVaS ist ohne jeden Zweifel das wichtigste
Gesetz, das in Spaßmanien in den letzten eintausend
Jahren verabschiedet wurde. Die erste Lesung fand
bereits am 11.\,11.-1111 im obersten spaßmanischen
Kongress statt, wurde aber vom damaligen Spaßvesier
abgelehnt. Erst nach Umwandlung der spaßmanischen,
aberwitzigen Monarchie in eine repräsentative,
witzige Monarchie durch W. Itzbold,
den Urkomischen, am 9.\,9.-1999 war der Weg für
dieses Gesetz endlich frei.

```

Because the text area has not been reduced, the preamble is output with the whole width of the typing area. To test this, you can temporarily add:

```
\end{document}
```

Until now, the example does not give any answer to the question, how the text of the comments will be printed with a smaller width. You will find this in the continuation of the example below.

19.5. Making a Note

After having declared a note column, we can make notes for this column. But these notes will not be output immediately. Instead of this, they are written into a helper file with extension “`.slnc`”. If you want to know the exactly workout: First they are written to the `aux`-file and while reading the `aux`-file inside of `\end{document}` they will be copied to the `slnc`-file. Thereby setting of `\nofiles` will be respected. At the next `LATEX` run, the helper file will be read step by step depending on the progress of the document and at the end of the page the note will be output.

But you should note that note columns are output only in pages with a page style of package `sclayer`. This package will be loaded by `sclayer-notecolumn` automatically and by default provides only one page style, `empty`. Usage of additional package `sclayer-scrpage` is recommended, if you need more page styles.

```
\makenote[note column name]{note}
```

This command may be used to make a new *note*. The current vertical position is used as the vertical position of the start of the *note*. The horizontal position depends only on the defined position of the note column. To work correct, the package needs `\pdfsavepos`, `\pdflastypos`, and `\pdfpageheight`. Without these primitives `sclayer-notecolumn` cannot be used. The primitives should act exactly as they would using PDF_T_EX.

However, if the package recognises a collision with the output of a former *note*, then the new *note* will be delayed until the end of the former *note*. If the *note* does not fit into the page, it will be moved at whole or only at part to the next page.

The optional argument *note column name* determines which note column should be used for the *note*. The predefined note column `marginpar` is used, if the optional argument is omitted.

Example: Let's add a commented section into the example of the previous section. The section itself is put into the note column:

```
\section{Analyse}
\begin{addmargin}[0pt]{.333\textwidth}
  \makenote[paragraphs]{%
    \protect\begin{contract}
      \protect\Clause{%
        title={Kein Witz ohne Publikum}%
      }
      Ein Witz kann nur dort witzig sein, wo er
      auf ein Publikum trifft.
    \protect\end{contract}%
  }
  Dies ist eine der zentralsten Aussagen des
  Gesetzes. Sie ist derart elementar, dass es
  durchaus angebracht ist, sich vor der Weisheit
  der Verfasser zu verbeugen.
```

Environment `addmargin`, which is described in [section 3.18, page 118](#), is used to reduce the width of the main text by the width of the section column.

Here you can see one of the few problems of using `\makenote`. Because the mandatory argument will be written into a helper file, commands inside this argument unfortunately can *break*. To avoid this, it is recommended to use `\protect` in front of all commands that should not expand while writing the helper file. Otherwise using a command inside this argument could result in an error message.

In principle you could finish this example already with:

```
\end{addmargin}
\end{document}
```


to see a first result.

If you'll test this example, you will see that the *paragraphs* column is longer than the comment. If you would add one more paragraph with one more section and comment, you may have the problem, that the new command will be put immediately after the comment above and not after the section. So the new section would move away from the corresponding comment. Next we will get a solution of this problem.

```
\syncwithnotecolumn[note column name]
```

This command adds a synchronisation point into the note column and into the main text of the document. Whenever a synchronisation point is found while output of a note column or the main text, a mark will be generated that consists of the current page and the current vertical position.

Together with the generation of synchronisation points it is recognised whether a mark has been set into the note column or the main text while the previous \LaTeX run. If so, the values will be compared. If the mark of the note column is lower at the current page or on a later page, the main text will be moved down to the position of the mark.

It is recommended to use synchronisation points not inside paragraphs of the main text but only between paragraphs. If you use `\syncwithnotecolumn` inside a paragraph, the synchronisation point will be delayed until the current line has been output. This behaviour is similar to the usage of, e. g., `\vspace`.

Because recognition of synchronisation points can be done first at the next \LaTeX run,, the mechanism needs at least three \LaTeX runs. Because of the new synchronisation later synchronisation points may be moved. This would result in the need of additional \LaTeX runs. You should have a look at the message: “ \LaTeX Warning: Label(s) may have changed. Rerun to get cross-references right.” to find out whether or not additional \LaTeX runs are needed.

If you do not use the optional argument, the predefined note column `marginpar` will be used. Please note, an empty optional argument is not the same as omitting the optional argument!

You must not use `\syncwithnotecolumn` inside a note itself, this means inside the mandatory Argument of `\makenote`! Currently the package cannot recognise such a mistake and would result in synchronisation point movement at each \LaTeX run. So the process will never terminate. So synchronise two or more note columns, you have to synchronise each of them with the main text. The recommended command for this will be described next.

Example: Let's extend the example above, now by adding a synchronisation point and one more section with one more comment:

```
\syncwithnotecolumn[paragraphs]\bigskip
\makenote[paragraphs]{%
  \protect\begin{contract}
    \protect\Clause{title={Komik der Kultur}}
    \setcounter{par}{0}%
  }
```

```

Die Komik eines Witzes kann durch das
kulturelle Umfeld, in dem er erzählt wird,
bestimmt sein.

Die Komik eines Witzes kann durch das
kulturelle Umfeld, in dem er spielt,
bestimmt sein.
\protect\end{contract}
}
Die kulturelle Komponente eines Witzes ist
tatsächlich nicht zu vernachlässigen. Über die
politische Korrektheit der Nutzung des
kulturellen Umfeldes kann zwar trefflich
gestritten werden, nichtsdestotrotz ist die
Treffsicherheit einer solchen Komik im
entsprechenden Umfeld frappierend. Auf der
anderen Seite kann ein vermeintlicher Witz im
falschen kulturellen Umfeld auch zu einer
echten Gefahr für den Witzeerzähler werden.

```

Beside the synchronisation point a vertical distance is added by `\bigskip` to better separate the section and the corresponding comments.

Again, one more potential problem is shown. Because the note columns uses boxes, that are build and split, counters inside note columns can sometimes jitter. In the example the first paragraph would be numbered 2 instead of 1. This can easily be fixed by a courageous reset of the counter.

Now, the example is almost finished. You just have to finish the environments:

```

\end{addmargin}
\end{document}

```

In fact, all other section of the law should also be commented. But let us focus on the main purpose.

But stop! What, if the section in the *paragraphs* note column wouldn't fit to the last page? Would it be output on the next pages? We will answer this question in the next paragraph.

`\syncwithnotecolumns[list of note column names]`

This command will synchronise the main text with all note columns of the comma-separated *list of note column names*. To do so, the main text will be synchronised with the note column, that have the mark closest to the end of the document. As a side effect the note columns will be synchronised with each other.

If the optional argument is omitted or empty (or begins with `\relax`), synchronisation will be done with all currently declared note columns.

19.6. Force Output of Note Columns

Sometimes it is necessary to have not only the normal note column output but to be able to output all notes that haven't been output already. An example for this effort could be that large notes result in moving lots of notes to new pages. A good occasion to force the output could be the end of a chapter or the end of the document.

```
\clearnotecolumn[note column name]
```

This command can be used to force the output of all notes of a selected note column that haven't been output until the end of the current page, but has been made in this page or previous pages. To force the output empty pages will be generated. During the output of the delayed notes of the selected note column, notes of other note columns are also output, but only as long as there are still delayed notes of the selected note column.

During the output of the delayed notes, notes may be output by mistake that have been placed to one of the inserted empty pages in the previous `LATEX` run. This will be solved automatically by one of the next `LATEX` runs. You can realise such movement of notes by the message: “`LATEX` Warning: Label(s) may have changed. Rerun to get cross-references right.”

The note column related to the output, is given by the optional argument, *note column name*. If this argument is omitted, the notes of the predefined note column `marginpar` will be output.

Attentive readers will have noticed that the forced output is something similar to the synchronisation. But if the forced output really results in an output you will be at the start of a new page and not just at the end of the output afterwards. Nevertheless, a forced output terminates with less `LATEX` runs.

```
\clearnotecolumns[list of note column names]
```

This command is similar to `\clearnotecolumn` with the difference that the optional argument is not only the name of one note column but a comma-separated *list of note column names*. It forces the output of the notes of all these note columns.

If you omit the optional argument, all delayed notes of all note columns will be output.

```
autoclearnotecolumns=simple switch
```

Usually you would like to force the output of notes whenever a document implicitly—e.g. because of a `\chapter` command—or explicitly executes `\clearpage`. Note that this is also the case at the end of the document inside `\end{document}`. Option `autoclearnotecolumns` manages whether or not `\clearpage` should also execute `\clearnotecolumns` without any optional argument.

According to the author's preference the option is active by default. But you can change this with proper values for simple switches (see [table 2.5, page 39](#)) at any time.

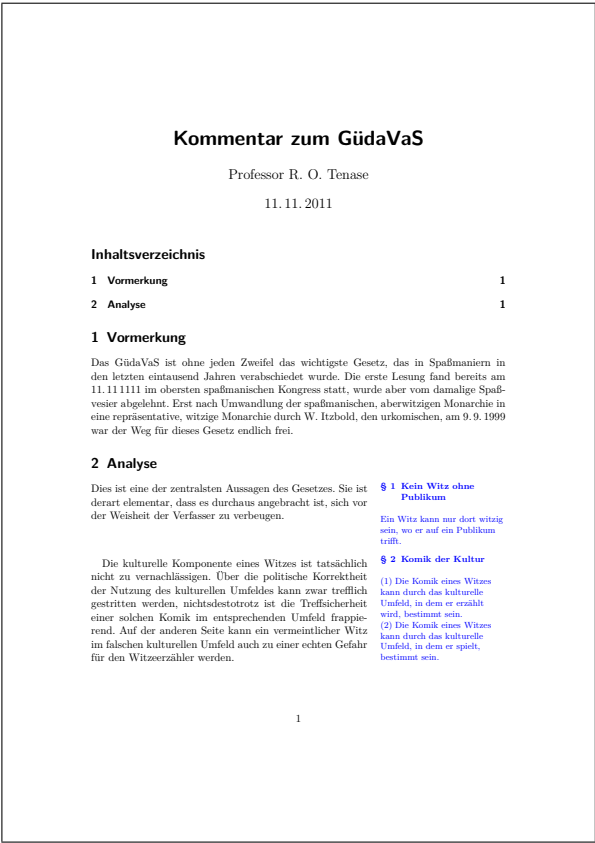


Figure 19.1.: An example page to the example of this chapter

You should note, that deactivation of the option can result in lost notes at the end of the document. So you should insert `\clearnotecolumns` before `\end{document}` if you deactivate the option.

This should answer the question from the end of the last section. By default the section would be output even if it would need the output of one more page. But because this would be done after the end of the `addmargin` environment, the forced output could overlap with main text after that environment. So it would be useful to either add one more synchronisation point or to force the output explicitly immediately at the end of the `addmargin` environment.

The result of the example is shown in figure 19.1.

Additional Information about package typearea

This chapter gives additional information about package `typearea`. Some parts of the chapter are subject to the KOMA-Script book [Koh14a] only. This should not be a problem, because the average user, who only want to use the package, does not need the information that is addressed to users with non-standard requirements or who want to write their own packages using `typearea`. Another part of the information describes features of `typearea` that exist only because of compatibility to former releases of KOMA-Script or the standard classes. The features, that exist only because of compatibility to former KOMA-Script releases, are printed with a sans serif font. You should not use them any longer.

20.1. Experimental Features

This section describes experimental features. Experimental in this context means, the function of a feature is not guaranteed. There are two reasons for this. First of all the final function is not yet defined or implementation is not yet established. Second reason is, that a feature may depend on internal functions of other packages and therefore the feature can not be guaranteed, if the other package changes.

```
usegeometry=simple switch
```

Usually `typearea` simply ignores whether or not it is used in any constellation with package `geometry` (see [Ume10]). This means, `geometry` does not recognize any page parameter change done with `typearea`, especially if you change the page size or page orientation—a feature not provided by `geometry` itself.

Once you set option `usegeometry`, `typearea` tries to translate all of its options into options of `geometry`. If you use `typearea` inside the document, it even calls `\newgeometry`. But since `geometry` does not provide changes of page size or page orientation with `\newgeometry`, `typearea` uses internal macros and lengths of `geometry` to, nevertheless, provide such changes. This has been tested with `geometry` 5.3 up to 5.6.

Note that using `geometry` and changing page size or orientation with `typearea` and active `usegeometry` does not mean, that `geometry` will automatically use the new paper size in an expected manner. `geometry` has an over estimated amount of options. So you often need to set up the margins and size of typing area newly using `\newgeometry`, because otherwise the typing area may be to large or to small. Nevertheless the combination of `typearea` with option `usegeometry` and package `geometry` can offer additional features or at least option `usegeometry` can improve the co-existence of both packages.

```
areasetadvanced=simple switch
\areaset[BCOR]{width}{height}
```

Usually `\areaset` does not care for options to define height of page header or page footer or whether or not margin elements should be part of the typing area same like `\typearea`. With option `areasetadvance` you can change this at most. Nevertheless, settings that result in same size of typing area still need not result in same top margin and bottom margin. This is, because `\typearea` only changes the bottom margin to have a integer number of lines in the typing area and `\areaset` always sets the ratio 1:2 for top margin to bottom margin. So the identical sizes of typing areas may be shifted vertically a little bit depending on which command has been used.

20.2. Expert Commands

This section describes commands that are not of interest for average users. Nevertheless these commands provide additional features for experts. Because the information is addressed to experts it's condensed.

```
\activateareas
```

Package `typearea` uses this command to assign settings of typing area and margins to internal L^AT_EX lengths, whenever the type area is newly calculated inside of the document, i.e., after `\begin{document}`. If option `pagesize` has been used, it will be executed again afterward. With this, e.g., the page size may vary inside of a PDF document.

Experts may use this command, if they change lengths like `\textwidth` or `\textheight` inside a document manually for any reason. Nevertheless the expert himself is responsible for eventually needed page breaks before or after usage of `\activateareas`. Moreover all changes of `\activateareas` are local!

```
\storeareas{\command}
\BeforeRestoreareas{code}
\BeforeRestoreareas*{code}
\AfterRestoreareas{code}
\AfterRestoreareas*{code}
```

With `\storeareas` a `\command` will be defined that may be used to restore all current settings of typing area and margins. This provides to store the current settings, change them, do anything with valid changed settings and restore the previous settings afterwards.

Example: You want a landscape page inside a document with portrait format. No problem using `\storeareas`:

```

\documentclass[pagesize]{scrartcl}

\begin{document}
\noindent\rule{\textwidth}{\textheight}

\storeareas\MySavedValues
\KOMAOptions{paper=landscape,DIV=current}
\noindent\rule{\textwidth}{\textheight}

\clearpage
\MySavedValues
\noindent\rule{\textwidth}{\textheight}
\end{document}

```

Command `\clearpage` before calling `\MySavedValues` is important to restore the saved values at the next page.

In the example `\noindent` has been used to avoid the paragraph indent of the black boxes. Without these commands the boxes would not show the typing area correctly.

Please note that neither `\storeareas` nor the defined `\command` should be used inside a group. Internally `\newcommand` is used to define the `\command`. So re-usage of a `\command` to store settings again would result in a corresponding error message.

v3.18

Often it is useful to automatically execute commands like `\cleardoubleoddpages` before restoring the settings of a `\command` generated by `\storeareas`. You can do so using `\BeforeRestoreareas` or `\BeforeRestoreareas*`. Analogously you can use `\AfterRestoreareas` or `\AfterRestoreareas*` to automatically execute *code* after restoring the settings. The variants with and without star differs so that the star variant changes only the auto-execution *code* of future *commands* and the variant without star also changes the auto-execution *code* of already defined *commands*.

```

\AfterCalculatingTypearea{instructions}
\AfterCalculatingTypearea*{instructions}
\AfterSettingArea{instructions}
\AfterSettingArea*{instructions}

```

These commands manage *hooks*. `\AfterCalculatingTypearea` and its star version provide experts to execute *instructions* every time `typearea` has recalculated the typing area and margins either implicitly or because of an explicit usage of `\typearea`. Similar `\AfterSettingArea` and its star version provide execution of *instructions* every time `\areaset` has been used. While normal versions work globally the influence of the star versions is only local. The *instructions* will be executed instantly before execution of `\activateareas`.

v3.11

20.3. Local Settings with File `typearea.cfg`

Sorry, currently additional information to this may be found at the same point of the German KOMA-Script book [\[Koh14a\]](#) only.

20.4. More or Less Obsolete Options and Commands

Sorry, currently additional information to this may be found at the same point of the German KOMA-Script book [\[Koh14a\]](#) only.

Additional Information about the Main Classes `scrbook`, `scrreprt`, and `scrartcl` as well as the Package `scrextend`

This chapter gives additional information about the KOMA-Script classes `scrbook`, `scrreprt`, and `scrartcl`. Some of the features are also available for package `scrextend`. Some parts of the chapter are subject to the KOMA-Script book [Koh14a] only. This should not be a problem, because the average user, who only want to use the package, will not need the information, that is addressed to users with non-standard requirements or who want to write their own classes using a KOMA-Script class. Another part of the information describes features of the classes that exist only because of compatibility to former releases of KOMA-Script or the standard classes. The features, that exist only because of compatibility to former KOMA-Script releases, are printed with a sans serif font. You should not use them any longer.

Sorry, currently additional information to this may be found at the same point of the German KOMA-Script book [Koh14a] only.

21.1. Additional Information to User Commands

Sorry, currently additional information to this may be found at the same point of the German KOMA-Script book [Koh14a] only.

21.2. Cooperation and Coexistence of KOMA-Script and Other Packages

Sorry, currently additional information to this may be found at the same point of the German KOMA-Script book [Koh14a] only.

21.3. Expert Commands

This sections described commands, that are more or less out of average user's interest. Nevertheless these commands provide additional features for experts. Because the information is addressed to experts it's condensed.

`\KOMAClassName`
`\ClassName`

`\KOMAClassName` stores the name of the currently used KOMA-Script class. If someone wants to know, whether or not or a KOMA-Script class is used or which KOMA-Script is used this may be tested with this command. In difference to this, `\ClassName` tells which would be the standard class, that has been replaced by a KOMA-Script class.

Please note, that the existence of `\KOMAScript` is not a indication for the usage of a KOMA-Script class. First of all: Every KOMA-Script package and not only KOMA-Script classes

define `\KOMAScript`. Furthermore other packages may also define the KOMA-Script word mark with this name.

`\addtocentrydefault{level}{number}{heading}`

v3.08

The KOMA-Script classes do not use `\addcontentsline` directly. Instead they call `\addtocentrydefault` with similar arguments. The command may be used for both, entries with and without number. Thereby *level* is the textual sectioning level, i.e., `part`, `chapter`, `section`, `subsection`, `subsubsection`, `paragraph`, or `subparagraph`. The already formatted sectioning number is given by the second argument, *number*. This argument may be empty. The text of the entry is given by argument *heading*. It is recommended to protect fragile commands inside this argument with prefix `\protect`.

There's one speciality for argument *number*. An empty argument signalizes, that an entry without number should be generated. KOMA-Script uses

`\addcontentsline{toc}{level}{heading}`

for this. Nevertheless, if the argument is not empty an entry with number will be made and *number* is the already formatted heading number. KOMA-Script uses

```
\addcontentsline{toc}{level}{%
  \protect\numberline{number}heading%
}
```

to make this.

Package authors and authors of wrapper classes may redefine this command to manipulate the entries. For example one could suggest

```
\renewcommand{\addtocentrydefault}[3]{%
  \ifstr{#3}{-}{%
    \ifstr{#2}{-}{%
      \addcontentsline{toc}{#1}{#3}%
    }{%
      \addcontentsline{toc}{#1}{\protect\numberline{#2}#3}%
    }%
  }%
}%
```

to omit entries with empty *heading*. In real life this would not be needed, because the KOMA-Script classes already use another method to suppress empty entries. See the description of the structuring commands in [section 3.16](#) from [page 95](#) onward for this.

```

\addparttocentry{number}{heading}
\addchaptertocentry{number}{heading}
\addsectiontocentry{number}{heading}
\addsubsectiontocentry{number}{heading}
\addsubsubsectiontocentry{number}{heading}
\addparagraphtocentry{number}{heading}
\addsubparagraphtocentry{number}{heading}

```

v3.08

The KOMA-Script classes call the previously described command `\addtocentrydefault` directly only if no individual command for the *level* has been defined or if that command is `\relax`. Nevertheless, the default definition of all these individual commands simply call `\addtocentrydefault` with their own *level* passing their arguments through.

```

\raggedchapterentry

```

v3.21

Previous versions of KOMA-Script provide a feature for printing chapter entries at the table of contents left-aligned instead of justified by defining command `\raggedchapterentry` to be `\raggedright`. Officially this feature has been removed from KOMA-Script version 3.21 on.

Indeed attribute `raggedentrytext` of toc-entry style `tocline` of package `tocbasic` has been implemented to use such a macro `\raggedentry levelentry` as an indicator for left-aligned text. If such a macro is `\raggedright`, the text is printed left-aligned. With any other definition the text is printed justified.

With this implementation of `raggedentrytext` full compatibility to the previous documentation of `\raggedchapterentry` is reached. As stated formerly, other definitions of `\raggedchapterentry` — and now also of `\raggedsectionentry` and similar macros for other entry levels — could result in the potentially unexpected effect of justified text.

Nevertheless, it is recommended to use the attribute of style `tocline` to select justified or left-aligned text.

```

\@fontsizefilebase
\changefontsizes{font size}

```

The prefix `scrsz` for file names of font size files, that has been mentioned in [section 21.1](#) at [page 441](#) is only the default of the internal macro `\@fontsizefilebase`. This default is used only, if the macro has not already be defined when loading a KOMA-Script class or package `scrextend`. Authors of wrapper classes may define this macro with another file name prefix to use completely different font size files. Also authors of wrapper classes could change or deactivate the *fallback* solution for unknown font sizes by redefinition of macro `\changefontsizes`. This macro has exactly one argument: the wanted *font size*.

```
\newkomafont[warning message]{element}{default}
\aliaskomafont{alias name}{element}
```

Experts may use `\newkomafont` to define a *default* for the font style of an *element*. After this that default may be changed using commands `\setkomafont` and `\addtokomafont` (see [section 3.6, page 57](#)). Of course this is not enough to use the defined font style. The expert himself has to prepare his code to use command `\usekomafont` (see [page 57](#)) with that *element* at his code definitions.

The optional argument *warning message* defines a warning message, that KOMA-Script will show whenever the default font style of that *element* will be changed. The sender of the warning in such cases will be the used KOMA-Script class or package `scrextend`.

Command `\aliaskomafont` defines an *alias name* to an already defined *element*. KOMA-Script will inform the user automatically about the real name of the element, whenever an *alias name* will be used. An *alias name* may be used, e.g., if a developer finds a better name for an element, that has been defined formerly with another name, if the old name should still be usable because of compatibility. Also an *alias names* may increase usability, because different users may find different names more or less intuitive. KOMA-Script itself defines a lot of *alias names* for several *elements*.

```
\addtokomafontrelaxlist{macro}
\addtokomafontgobblelist{macro}
```

As already mentioned in [part I](#) of this manual, font settings of elements should consist only in commands to select the size, family, coding, series, shape, or colour. At least changing the colour is not always transparent in L^AT_EX and therefore could result in unwanted effects if someone uses `\usekomafont` at an inappropriate state.

Users tend to use different, somehow critical things in the font setting of an element, e.g., `\MakeUppercase` at the very end of the setting. As long as possible, the internal usage of font settings has been implemented so that such forbidden settings do not matter. Even using a command that expects an argument, e.g., `\textbf` instead of `\bfseries`, would work mostly, if it the last one in the font setting of an element — but without warranty.

Inside KOMA-Script, sometimes, it was necessary to restrict the font change to real font settings. This has been done, e.g., using `\usefontofkomafont` instead of `\usekomafont` (see [section 3.6, page 62](#)).

Nevertheless, command `\usefontofkomafont` and it's siblings have some limitations. Therefore you must not use a command that always needs a fully expandable argument inside the font setting of an element. But this is exactly what `\MakeUppercase` needs. Therefore KOMA-Script holds a list of macros, which should become `\relax` inside `\usefontofkomafont` and it's siblings. Amongst others, `\normalcolor`, `\MakeUppercase`, and `\MakeLowercase` are part of that list. More macros can be added one by one using `\addtokomafontrelaxlist`.

Note that *macro* will be set simply to `\relax`. So if *macro* really has an argument, the argument will be execute locally. Therefore you must not add commands like

`\setlength` to the list. The user himself is responsible for all errors resulting in the usage of `\addtokomafontrelaxlist`. Additionally this command should not be misunderstood as a legitimation for adding all kind of commands to the font setting of an element!

v3.19

If a *macro* and it's first argument should be ignored locally inside `\usefontofkomafont` and it's siblings, you can use `\addtokomafontgobblelist` instead of `\addtokomafontrelaxlist`. An example for this is the command `\color`, that has to be ignored with the colour name and therefore is a default member of the gobble-list.

```
\IfExistskomafont{element}{then code}{else code}
```

v3.15

Which elements are defined depends on the version of KOMA-Script. So sometimes it may be useful to be able to test, whether or not an *element* exists. The command executes the *then code* only if an *element* has been defined using `\newkomafont` or `\aliaskomafont` and therefore can be changed using `\setkomafont` or `\addtokomafont` and can be used by one of the `\use...komafont` commands. Otherwise it executes the *else code*.

```
\setparsizes{indent}{distance}{last line end space}
```

With this command KOMA-Script provides to set the indent of the first line of a new paragraph, the distance between paragraphs and the white space that has to be at the end of the last line of each paragraph. This command should be used whenever changes should also be recognized by option `parskip=relative`. KOMA-Script itself uses this command, e. g., with

```
\setparsizes{0pt}{0pt}{0pt plus 1fil}
```

to switch of paragraph indent and distance between paragraphs and to allow any white space at the end of the last line of paragraphs. This make sense, if a paragraph consists of a box only, that should be printed without vertical distance but with the whole column width. If, in opposite to that, the box should only span the whole width but should be printed with the current settings of paragraph indent and distance between paragraphs, usage of

```
\setlength{\parfillskip}{0pt plus 1fil}
```

would be recommended.

v3.17

Since KOMA-Script 3.17 changing or reactivation of the typing area or the margins (see [chapter 2](#)) will also reactivate the values of `\setparsizes` if they have not been changed since the last usage of this command. This is one more reason not to change these values without using KOMA-Script. With compatibility to a KOMA-Script version prior to 3.17 (see [section 3.2](#), [page 53](#), option `version`) this reactivation of the `\setparsizes` values is deactivated.

Sorry, currently additional information to this may be found at the same point of the German KOMA-Script book [[Koh14a](#)] only.

```
\DeclareSectionCommand[attributes]{name}
\DeclareNewSectionCommand[attributes]{name}
\RedeclareSectionCommand[attributes]{name}
\ProvideSectionCommand[attributes]{name}
```

v3.15 With these commands you can either define a new section-like command `\name` or change an already defined sectioning command `\name`. To do so you use the optional argument to setup several *attributes*. The *attributes* are a comma-separated list of *key=value* assignments. Beside the style-independent attributes shown in table 21.1, there are style dependent attributes, too. Currently the following styles are provided:

v3.18 **chapter:** Style of chapter headings. This style is currently used for `\chapter` and indirectly for `\addchap`. You can define new section-like commands using this style. To define new or to reconfigure existing commands you can also use the additional attributes of table 21.3, page 449. Note that command `\addchap` and the star variants are configured automatically together with `\chapter` and should not be changed independently. Note that `scrartcl` does not provide this style.

scrbook,
scrreprt
v3.18 **part:** Style part headings. This style is currently used for `\part` and indirectly for `\addpart`. You can define new section-like commands using this style. To define new or to reconfigure existing commands you can also use the additional attributes of table 21.4, page 450. Note that command `\addpart` and the star variants are configured automatically together with `\part` and should not be changed independently.

section: Style of section headings. This style is currently used for `\section`, `\subsection`, `\subsubsection`, `\paragraph`, and `\subparagraph`. You can define new section-like commands using this style. To define new or to reconfigure existing commands you can also use the additional attributes of table 21.2, page 448. Definitions of new commands need at least the *keys* style, *afterskip*, *beforeskip*, *font*, *indent*, *level*, *tocindent*, and *tocnumwidth*. This is also true if a command that was not a section-like command before will be redefined as a section-like command using `\RedeclareSectionCommand`. Note that command `\addsec` and the star variants are configured automatically together with `\section` and should not be changed independently. Defining a new section-like command with this style will also define an element with the same *name* and the possibility to change its font using `\setkomafont` or `\addtokomafont` (see section 3.6, page 57), if such an element does not already exist.

`\DeclareNewSectionCommand` defines a new section-like command. If the same *name* is already used by T_EX for something else, the command will result in an error message and will not define anything.

`\ProvideSectionSommand` is similar but does not show any error message.

Table 21.1.: Available *keys* and *values* for the style-independent *attributes* declaring a section-like command

	<i>key</i>	<i>value</i>	Description
	counterwithin	<i>counter name</i>	The value of the counter of the heading should depend on <i>counter name</i> . Whenever <code>\stepcounter</code> or <code>\refstepcounter</code> increases the value of <i>counter name</i> , the value of the counter of this heading should be set to 0. Also the output of the value of this heading should be prefixed by <code>\thecounter name</code> followed by a dot.
v3.19	counterwithout	<i>counter name</i>	Cancels a prior <code>counterwithin</code> setting. Therefore it makes sense only if you re-define an existing section-like command.
	expandtopt	<i>switch</i>	If the switch is on, all following values for lengths will be completely expanded, evaluated and stored as <code>pt</code> values. If the switch is off, all following values for lengths will be completely expanded, tentatively evaluated but only expanded stored. Any values from table 2.5, page 39 may be used. Default is <code>false</code> .
	level	<i>integer</i>	A number, denoting depth of section (see <code>counter secnumdepth</code> , section 3.16, page 106); the value should be unique.
	style	<i>name</i>	Defines the style of the heading.
Beta-Feature	tocstyle	<i>name</i>	Defines the style of the entries into the table of contents. You can use every already defined toc-entry style (see section 15.3). An empty <i>name</i> prevents a new definition of the toc-entry command.
Beta-Feature	tocoption	<i>value</i>	Additional options depending on the TOC-entry style given by <code>tocstyle</code> . See section 15.3, page 356 for additional information about TOC-entry styles. See table 15.1, page 360 for information about the attributes of the predefined TOC-entry styles of package <code>tocbasic</code> , that can be used as <i>option</i> .

Table 21.2.: Additional *keys* and *values* of attributes declaring a section-like command with style section

key	value	Description
afterskip	<i>length</i>	A negative value activates a run-in heading. The absolute value is the skip to leave to right of the run-in heading. A positive value is the vertical skip below the heading.
beforeskip	<i>length</i>	The absolute value of the vertical skip before the heading. If the value is negative, then the paragraph indent of the text following the heading is suppressed.
font	<i>font commands</i>	The initial font setting that should be used beside disposition for the heading. You can use all settings, that are allowed for \setkomafont and \addtokomafont for the element of the heading.
indent	<i>length</i>	Indentation of heading from the left margin.

`\RedeclareSectionCommand` can only change an existing command to a section-like command with given *attributes*. There is no pre-validation to make sure that `\name` was a section-like command before. However, it has to be an already used command sequence *name*.

`\DeclareSectionCommand` does not check whether or not *name* is an already used `TEX` command sequence. It just defines a section-like command `\name` with the given *attributes*.

Every section-like command has also a corresponding counter with the same *name*, that will be allocated using `\newcounter` if necessary. The same rule applies to the corresponding output of the counter: `\thename`, the output format: `\nameformat`, the command to generate a running head: `\namemark`, the counter format of the running head: `\namemarkformat`, the font element: *name*, and the section depth number: `\namenumdepth`. The command for the running head `\namemark` will be defined to not generate any running head. The output of the counter, `\thename`, will show an Arabic number. If the *key* `counterwithin` defines a counter the heading number depends on, the output of the counter will be prefixed by the output of that counter followed by a dot.

Beta-Feature
Beside the section-like command, a command for corresponding entries to the table of contents is defined also. This is done using package `tocbasic`. Attribute *tocstyle* defines the style of those entries. If you set an empty *name*, e.g., using `tocstyle=` or `tocstyle={}`, the entry command will not be changed. This is important, if you use another package to modify the table of contents. If you do not set attribute *tocstyle* the already configured style will be used again.

Beta-Feature
The different toc-entry styles also have different, additional attributes. You can set them here too, if you prefix them with `toc`. For example, you can setup the level of the toc-entries, `level`, using `toclevel`, the indention, `indent`, using `tocindent`, or the number width,

Table 21.3.: Additional *keys* and *values* of attributes declaring a section-like command with style chapter

key	value	Description
afterskip	<i>length</i>	The absolute value is the vertical skip below the heading.
beforeskip	<i>length</i>	The absolute value of the vertical skip before the heading. If the value is negative, then the paragraph indent of the text following the heading is suppressed. This will be done also for positive values if class option version is set to a value less than 3.22.
font	<i>font commands</i>	The initial font setting that should be used beside disposition for the heading. You can use all settings, that are allowed for \setkomafont and \addtokomafont for the element of the heading.
innerskip	<i>length</i>	The vertical skip between the prefix line and the heading’s text, if a prefix line is used.
pagestyle	<i>page style name</i>	The name of the page style that should be used automatically on pages with the heading. There is no validation whether or not <i>page style name</i> is really the name of a page style. Therefore, wrong names will result in error messages at usage of the section-like command.
prefixfont	<i>font commands</i>	The initial font setting that should be used beside disposition and the element of the heading for the prefix line of the heading. You can use all settings that are allowed for \setkomafont and \addtokomafont for the element of the prefix line.

numwidth, using tocnumwidth. For more toc-entry style attributes see [section 15.3, page 356](#).

Example: Because of incomprehensible reasons, the headings of **\paragraph** shouldn’t be run-in headings any longer but headings similar to **\subsubsection**. The vertical skip above the heading should be 10 pt and the following text should be set without any vertical distance. To do so, you can simply use:

```
\RedeclareSectionCommand[%  
  beforeskip=-10pt,%  
  afterskip=1sp%  
]{paragraph}
```

The negative value of **beforeskip** does not only result in a positive vertical skip

Table 21.4.: Additional *keys* and *values* of attributes declaring a section-like command with style part

<i>key</i>	<i>value</i>	Description
<code>afterskip</code>	<code>length</code>	The absolute value is the vertical skip below the heading.
<code>beforeskip</code>	<code>length</code>	The absolute value of the vertical skip before the heading.
<code>font</code>	<code>font commands</code>	The initial font setting that should be used beside <code>disposition</code> for the heading. You can use all settings, that are allowed for <code>\setkomafont</code> and <code>\addtokomafont</code> for the element of the heading.
<code>innerskip</code>	<code>length</code>	The vertical skip between the prefix line and the heading’s text of <code>scrbook</code> and <code>scrreprt</code> .
<code>pagestyle</code>	<code>page style name</code>	The name of the page style that should be used automatically on pages with the heading. There is no validation whether or not <code>page style name</code> is really the name of a page style. Therefore, wrong names will result in error messages at usage of the section-like command. This feature is only available with <code>scrbook</code> or <code>scrreprt</code> .
<code>prefixfont</code>	<code>font commands</code>	The initial font setting that should be used beside <code>disposition</code> and the element of the heading for the prefix line of the heading. You can use all settings that are allowed for <code>\setkomafont</code> and <code>\addtokomafont</code> for the element of the prefix line.

before the heading, but also deactivates the paragraph indentation of the following text. Despite the specification of no vertical skip after the heading a value of 1 sp has been given to `afterskip`. This is because L^AT_EX here doesn’t accept 0 pt as positive value. So 1 sp is the lowest possible positive value.

Generally, for the vertical adjustment it is better to have some tolerance in the declaration of gaps—the so called *glue*:

```
\RedeclareSectionCommand[%
  beforeskip=-10pt plus -2pt minus -1pt,%
  afterskip=1sp plus -1sp minus 1sp%
]{paragraph}
```

Doing so you have to know, that the glue also switches the algebraic sign before becoming a skip, if the value is negative. That is the reason for the negative glue values in the example. Additionally we used the occasion to minimize the vertical

skip after the heading using glue too.

In the example above only `before skip` and `after skip` were needed, because since v3.15 KOMA-Script defines `\paragraph` itself using `\DeclareSectionCommand`. All other values are just reused with their initial definition. Furthermore, the original definition of `\paragraph` in `scrartcl` reads:

```
\DeclareSectionCommand[%  
  level=4,  
  indent=0pt,  
  before skip=3.25ex plus 1ex minus .2ex,  
  after skip=-1em,  
  font={},  
  tocindent=7em,  
  tocnumwidth=4.1em,  
  counter within=subsubsection  
{\paragraph}
```

`scrreprt` and `scrbook` use slightly different values.

Some settings of `\chapter` depend on option `headings` (see [section 3.16, page 91](#)). [Table 21.5](#) shows the default values of these settings. An overview of all settings is shown in [table 21.6](#). For more information about the default of the toc-entry styles see [section 15.3, page 356](#). Please note, that `1ex` and `\baselineskip` depend on the default font size of the heading or the table of contents entry.

Beta-Feature

Table 21.6.: Default of the headings of `scrbook` and `scrreprt`

`\part:`

Attribute	Default Value
<code>after skip</code>	<code>0pt plus 1fil</code>
<code>before skip</code>	<code>0pt plus 1fil + \baselineskip</code>
<code>font</code>	see element <code>part</code> , table 3.15, page 98
<code>innerskip</code>	<code>20pt</code>
<code>level</code>	<code>-1</code>
<code>prefixfont</code>	see element <code>partnumber</code> , table 3.15, page 98
<code>tocindent</code>	<code>0pt</code>
<code>toclevel</code>	<code>-1</code>
<code>tocnumwidth</code>	<code>2em</code>
<code>tocstyle</code>	<code>part</code>

Table 21.6.: Default of the headings of scrbook and scrreprt (*Continuation*)

`\chapter:`

Attribute	Default Value
<code>afterskip</code>	see table 21.5
<code>beforeskip</code>	see table 21.5
<code>font</code>	see element chapter , table 3.15 , page 98
<code>innerskip</code>	0pt
<code>level</code>	0
<code>prefixfont</code>	see element chapterprefix , table 3.15 , page 98
<code>tocindent</code>	0pt
<code>toclevel</code>	0
<code>tocnumwidth</code>	1.5em
<code>tocstyle</code>	chapter

`\section:`

Attribute	Default Value
<code>afterskip</code>	2.3ex plus .2ex
<code>beforeskip</code>	-3.5ex plus -1ex minus -.2ex
<code>font</code>	see element section , table 3.15 , page 98
<code>indent</code>	0pt
<code>level</code>	1
<code>tocindent</code>	1.5em
<code>toclevel</code>	1
<code>tocnumwidth</code>	2.3em
<code>tocstyle</code>	section

...

Table 21.6.: Default of the headings of scrbook and scrreprt (*Continuation*)

`\subsection:`

Attribute	Default Value
afterskip	1.5ex plus .2ex
beforeskip	-3.25ex plus -1ex minus -.2ex
font	see element <code>subsection</code> , table 3.15, page 98
indent	0pt
level	2
tocindent	3.8em
toclevel	2
tocnumwidth	3.2em
tocstyle	section

`\subsubsection:`

Attribute	Default Value
afterskip	1.5ex plus .2ex
beforeskip	-3.25ex plus -1ex minus -.2ex
font	see element <code>subsubsection</code> , table 3.15, page 98
indent	0pt
level	3
tocindent	7.0em
tocnumwidth	4.1em
toclevel	3
tocstyle	section

`\paragraph:`

Attribute	Default Value
afterskip	-1em
beforeskip	3.25ex plus 1ex minus .2ex
font	see element <code>paragraph</code> , table 3.15, page 98
indent	0pt
level	4
tocindent	10em
toclevel	4
tocnumwidth	5em
tocstyle	section

Table 21.6.: Default of the headings of scrbook and scrreprt (*Continuation*)

`\subparagraph`:

Attribute	Default Value
<code>afterskip</code>	<code>-1em</code>
<code>beforeskip</code>	<code>3.25ex plus 1ex minus .2ex</code>
<code>font</code>	see element <code>subparagraph</code> , table 3.15, page 98
<code>indent</code>	<code>\scr@parindent</code>
<code>level</code>	<code>5</code>
<code>tocindent</code>	<code>12em</code>
<code>toclevel</code>	<code>5</code>
<code>tocnumwidth</code>	<code>6em</code>
<code>tocstyle</code>	<code>section</code>

```
\DeclareSectionCommands[attributes]{list of names}  
\DeclareNewSectionCommands[attributes]{list of names}  
\RedeclareSectionCommands[attributes]{list of names}  
\ProvideSectionCommands[attributes]{list of names}
```

v3.15

With these commands you can either define or change several section-like commands at once. The names of the section-like commands are given by the comma-separated *list of names*.

There are two more differences to the previously described commands that only define or change a single section-like command. Firstly, in case of error—if `\DeclareNewSectionCommands` is used for an already defined T_EX command sequence or if `\RedeclareSectionCommands` is used for an undefined T_EX command sequence—the definition will be done in spite of raising an error message.

Secondly, there is one more attribute `increaselevel` with an optional integer argument. This attribute changes the meaning of the attributes `level` and `toclevel` (see table 21.1) so their values become start values for the first section-like command of the *list of names*. From section-like command to section-like command of the *list of names* the value of `level` and `toclevel` will be increased by the value of `increaselevel`. If attribute `increaselevel` is used without an assignment the increase value is 1.

Table 21.5.: Defaults of the chapter headings of scrbook and scrreprt subject to option `headings`

With `headings=big`:

Attribute	Default Value
<code>afterskip</code>	<code>1.725\baselineskip plus .115\baselineskip minus .192\baselineskip</code>
<code>beforeskip</code>	<code>-3.3\baselineskip-\parskip</code>
<code>font</code>	<code>\huge</code>

With `headings=normal`:

Attribute	Default Value
<code>afterskip</code>	<code>1.5\baselineskip plus .1\baselineskip minus .167\baselineskip</code>
<code>beforeskip</code>	<code>-3\baselineskip-\parskip</code>
<code>font</code>	<code>\LARGE</code>

With `headings=small`:

Attribute	Default Value
<code>afterskip</code>	<code>1.35\baselineskip plus .09\baselineskip minus .15\baselineskip</code>
<code>beforeskip</code>	<code>-2.8\baselineskip-\parskip</code>
<code>font</code>	<code>\Large</code>

```
\chapterheadstartvskip
\chapterheadmidvskip
\chapterheadendvskip
\partheadstartvskip
\partheadmidvskip
\partheadendvskip
\partheademptypage
```

These commands are used inside of the definition of the headings `\chapter`, `\part`, `\addchap`, `\addpart` and their star-variations. Thereby `\chapterheadstartvskip` is designed to be a command, that inserts a vertical distance before the chapter heading. Analogues `\chapterheadendvskip` is designed to be a command, that inserts a vertical distance after the chapter heading. If the chapter heading has a prefix number line (see option `chapterprefix` in [section 3.16, page 91](#)), `\chapterheadmidvskip` will be used between the number line and the heading text.

Vertical distance above and below part headings will be inserted using the commands `\partheadstartvskip` and `\partheadendvskip`. A page break would be interpreted to be part of the vertical distance and therefore is already part of the default of `\partheadendvskip`

scrbook,
scrreprt

in scrbook and scrreprt. Command `\partheademptypage` is used to produce the empty page after the part heading page of scrbook and scrreprt.

v3.15

Since KOMA-Script 3.15 the defaults of these seven commands are independent from option `headings` (see [section 3.16, page 91](#)). The original definitions of the chapter heading reads since KOMA-Script 3.157:

v3.17

```
\newcommand*{\chapterheadstartvskip}{\vspace{\@tempskipa}}
\newcommand*{\chapterheadmidvskip}{\par\nobreak\vskip\@tempskipa}
\newcommand*{\chapterheadendvskip}{\vskip\@tempskipa}
```

Every usage of `headings=big`, `headings=normal`, or `headings=small` reactivates these default definitions.

Command `\chapter` automatically sets the internal length `\@tempskipa` to the value of the `\DeclareSectionCommand` attribute `before skip` before calling `\chapterheadstartvskip`. Correspondingly it sets the length to the value of attribute `after skip` before calling `\chapterheadendvskip` and `inner skip` before calling `\chapterheadmidvskip`. If you redefine `\chapterheadstartvskip`, `\chapterheadmidvskip`, or `\chapterheadendvskip` you should also correspond the new definition to `\@tempskipa` to respect the values of `\DeclareSectionCommand`.

v3.17

The default values of the distances of `\part` do not depend on option `headings`. So the corresponding commands will not be redefined using the options. The original definitions of these commands of scrbook and scrreprt read:

```
\newcommand*{\partheadstartvskip}{%
  \null\vskip-\baselineskip\vskip\@tempskipa
}
\newcommand*{\partheadmidvskip}{%
  \par\nobreak
  \vskip\@tempskipa
}
\newcommand*{\partheadendvskip}{%
  \vskip\@tempskipa\newpage
}
```

and of scrartcl:

```
\newcommand*{\partheadstartvskip}{%
  \addvspace{\@tempskipa}%
}
\newcommand*{\partheadmidvskip}{%
  \par\nobreak
}
\newcommand*{\partheadendvskip}{%
  \vskip\@tempskipa
}
```


Again `\part` sets `\@tempskipa` to the values of `\DeclareSectionCommand` before the internal usage of the command.

v3.17

It is not recommended to redefine the command for the distances above or below the headings only for changing these distances, because you can reconfigure these distances much easier using `\RedeclareSectionCommand`. Redefinition of the commands should be reserved to more complex changes. An example that redefines `\chapterheadstartvskip` and `\chapterheadendvskip` to print extra rules above and below the chapter heading may be found at [KDP] (in German).

```
\chapterlineswithprefixformat{level}{number}{text}
\chapterlinesformat{level}{number}{text}
```

v3.19

These commands are used by headings with style `chapter` to output the heading number and heading text depending on option `chapterprefix` (see section 3.16, page 91). If the option is `true` `\chapterlineswithprefixformat` is used, otherwise `\chapterlinesformat`.

The arguments *number* and *text* are already formatted, i.e., they contain font selections. At least the order and layout of both has to be done with these commands. If a heading has no number argument *number* will be completely empty also without any format or font commands.

The default definitions are very simple:

```
\newcommand{\chapterlinesformat}[3]{%
  \@hangfrom{#2}{#3}%
}
\newcommand{\chapterlineswithprefixformat}[3]{%
  #2#3%
}
```

Example: You want to have chapter headings with yellow background. For the headings without prefix line you use the follow definition in the document preamble:

```
\makeatletter
\renewcommand{\chapterlinesformat}[3]{%
  \colorbox{yellow}{%
    \parbox{\dimexpr\linewidth-2\fboxrule-2\fboxsep}{%
      \@hangfrom{#2}{#3}%
    }%
  }%
}
\makeatother
```

For chapter headings without prefix line you use:

```
\renewcommand{\chapterlineswithprefixformat}[3]{%
  \colorbox{yellow}{%
    \parbox{\dimexpr\linewidth-2\fboxrule-2\fboxsep}{%
      \@hangfrom{#2}{#3}%
    }%
  }%
}
```

```

        \parbox{\dimexpr\linewidth-2\fbboxrule-2\fbboxsep}{%
            #2#3%
        }%
    }%
}

```

Unfortunately you will find, that these re-definitions result in justified text for the headings. The reason is the `\parbox` command. To change this, you can use `\raggedchapter` (see [section 3.16](#), [page 102](#)) inside the argument of `\parbox`. Otherwise `\raggedchapter` would be used only before `\chapterlineswithprefixformat` and `\chapterlinesformat`:

```

\makeatletter
\renewcommand{\chapterlinesformat}[3]{%
    \colorbox{yellow}{%
        \parbox{\dimexpr\linewidth-2\fbboxrule-2\fbboxsep}{%
            \raggedchapter
            \@hangfrom{#2}#3%
        }%
    }%
}
\makeatother
\renewcommand{\chapterlineswithprefixformat}[3]{%
    \colorbox{yellow}{%
        \parbox{\dimexpr\linewidth-2\fbboxrule-2\fbboxsep}{%
            \raggedchapter
            #2#3%
        }%
    }%
}

```

Remember to use `\makeatletter` and `\makeatother` only at the document preamble. Do not use it inside your own wrapper-class or package. It is only needed because of `\@hangfrom` in the definition of `\chapterlinesformat`.

As the example shows, users, who change the definition of one of the commands, should take care of several side effects. The alignment of the text is only one thing. They also have to prevent page breaks in the headings, e.g., if they add extra space into the output. The example above does not have page break problems. Not only the boxes prevent page breaks. KOMA-Script itself also changes `\interlinepenalty` as part of *text* to prevent page breaks. It also finishes *text* with an internal paragraph break using `\@par`.

Command `\raggedchapter` is not part of *text* but executed before `\chapterlineswithprefixformat` or `\chapterlinesformat`. This makes it much easier to use, e.g., `\MakeUppercase` at a re-definition. Please note, typographers use individual letter spacing at majuscule text, but L^AT_EX command `\MakeUppercase` does not.

The first argument, *level*, is not used by the default definition and also not needed in the example above. It is of interest only, if a user would define several commands with `chapter` style and need to distinguish the different levels. The predefined commands `\chapter`, `\chapter*`, `\addchap`, and `\addchap*` all share the same *level chapter*.

```
\sectionlinesformat{level}{indent}{number}{text}
\sectioncatchphraseformat{level}{indent}{number}{text}
```

v3.19

These commands are used by headings with style `section` to output the heading number and heading text. If the heading is printed as catch phrase at the very beginning of the following paragraph text `\sectioncatchphraseformat` is used, otherwise `\sectionlinesformat`.

In both cases *indent* is the value of the horizontal indentation of the heading relative to the text area. If the value is negative the heading should even move into the left margin.

The arguments *number* and *text* are already formatted, i.e., they contain font selections. At least the order and layout of both has to be done with these commands. If a heading has no number argument *number* will be completely empty also without any format or font commands.

The default definitions are:

```
\newcommand{\sectionlinesformat}[4]{%
  \@hangfrom{\hskip #2#3}{#4}%
}
\newcommand{\sectioncatchphraseformat}[4]{%
  \hskip #2#3#4%
}
```

If the user re-defines one of these commands, he has to take care to prevent page breaks inside the printing of the heading. KOMA-Script itself only changes `\interlinepenalty` to impede page breaks.

Example: Like chapter headings in the previous example, headings of *level section* should be printed with a background colour. The lower levels should be affected:

```
\makeatletter
\renewcommand{\sectionlinesformat}[4]{%
  \@tempswafalse
  \ifstr{#1}{section}{%
    \hspace*{#2}%
    \colorbox{yellow}{%
      \parbox{\dimexpr\linewidth-2\fboxrule-2\fboxsep-#2}{%
        \raggedsection
        \@hangfrom{#3}{#4}%
      }%
    }%
  }%
}
```

```

        \@hangfrom{\hskip #2#3}{#4}%
      }%
    }
    \makeatother

```

With this code the indentation part of indented headings would not be coloured, but with negative indentation the background of the margin part of the headings will also become yellow. You can move the `\hspace*` command into the `\colorbox` to change this.

Again remember to use `\makeatletter` and `\makeatother` only at the document preamble. Do not use it inside your own wrapper-class or package. It is only needed because of `\@hangfrom` in the definition of `\sectionlinesformat`.

The first argument, *level*, is not used by the default definition. The example shows how to use it to distinguish different heading levels of the same style `section`.

```

\SecDef{star command}{command}
\scr@startsection{name}{level}{indent}{beforeskip}{afterskip}{style commands}
    {short version}{heading}
\scr@startsection{name}{level}{indent}{beforeskip}{afterskip}{style commands}*
    {heading}
\At@startsection{code}
\Before@ssect{code}
\Before@ssect{code}

```

v3.15

As already explained in [section 3.16](#) in the description of the sectioning commands beginning with [page 95](#), KOMA-Script provides additional features for the optional argument of those commands. Therefore, it was necessary to replace some L^AT_EX kernel commands, especially `\secdef` and `\@startsection`. The meaning of the parameters of these commands can be found in the L^AT_EX kernel manual [BCJ⁺05].

Unfortunately these commands are often redefined by other packages in a way that collides with the functionality of KOMA-Script. So KOMA-Script not only changes the definition of these commands but also defines the additional, alternative commands `\SecDef` and `\scr@startsection`. Package authors are permitted to use these commands like they would use the corresponding L^AT_EX kernel commands and therefore participate in the additional features of KOMA-Script. Nevertheless these commands should not be redefined, because they could be changed in future and so the redefinition would impair KOMA-Script again.

KOMA-Script internally uses `\SecDef` and `\scr@startsection` instead of `\secdef` and `\@startsection`, e.g., defining section-like commands by `\DeclareSectionCommand`. So later redefinition of `\secdef` or `\@startsection` will not influence the sectioning commands of KOMA-Script.

As an alternative to the redefinition of such commands, KOMA-Script provides the execution of additional *code* at several states of the KOMA-Script replacements. The *code* of each usage of `\At@startsection`, `\Before@sect`, or `\Before@ssect` will be cumulated independently. Later removing of added *code* is not provided.

The *code* of `\At@startsection` is executed immediately after the evaluation of *beforeskip* and calculation of the resulting `\@tempskipa`, before adding the vertical gap itself. So you still may change the value of `\@tempskipa`.

The *code* of `\Before@sect` and `\Before@ssect` is executed just before `\@sect` and `\@ssect` inside `\scr@startsection`, respectively. At this state the vertical gap of `\@tempskipa` has already been added using `\addvspace`.

The commands `\At@startsection`, `\Before@sect`, and `\Before@ssect` are suggested for package authors.

`\appendixmore`

scrbook,
scrreprt

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` (see [section 3.16, page 91](#)). You should take note of this in case you decide to define or redefine the macro `\appendixmore`. In case one of this option has been used, 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` in [section 3.16, page 91](#)). 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 layout option `appendixprefix`, 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 can write in the preamble of your document:

```
\newcommand*{\appendixmore}{%
  \renewcommand*{\sectionformat}{%
    \appendixname~\thesection\autodot\enskip}%
  \renewcommand*{\sectionmarkformat}{%
    \appendixname~\thesection\autodot\enskip}}
```

Redefined commands are explained in more detail in [section 3.16, page 103](#) and [page 105](#).

```
\newbibstyle[parent style]{name}{instructions}
\newblock
\@openbib@code
\bib@beginhook
\bib@endhook
```

The standard classes already provide command `\newblock` for structuring of bibliography entries. The exact purpose of this command depends on the class options. Using option `openbib` redefine commands `\@openbib@code` and `\newblock` at the end of the used standard class. These classes execute command `\@openbib@code` at the begin — or more precise: at the specification of the parameters of the — list environment, that will be used for the bibliography. It should be assumed, that many packages will execute this command in the same kind, if they redefine the bibliography.

The KOMA-Script classes do something similar. Nevertheless, they do not redefine `\@openbib@code` at the end of the class. Instead of, the bibliography style `openstyle` is defined using `\newbibstyle`. The *instructions*, that has been defined as part of the implementation, contain the appropriate redefinition of `\@openbib@code` and `\newblock`. Now, if this bibliography style will be selected using option `bibliography=openstyle`, then the *instructions* will be executed immediately. This will redefine `\@openbib@code` and `\newblock`.

Beside `\@openbib@code` and `\newblock` also `\bib@beginhook` and `\bib@endhook` may be redefined by the *instructions* of the style. Command `\bib@beginhook` will be executed immediately after heading and preamble of the bibliography, but before the begin of the list with the bibliography entries. Command `\bib@endhook` will be executed immediately after this list at the end of the bibliography. If `\BreakBibliography` (see [section 3.23, page 139](#)) intercepts the bibliography, these commands will also executed at the begin and end of each part of the bibliography, this would be immediately before and after `\BreakBibliography`.

The commands `\newblock`, `\@openbib@code`, `\bib@beginhook`, and `\bib@endhook` will be reset to an empty definition at the start of each new bibliography style. After this the *instructions* of the *parent style* of the bibliography style will be executed. After this the

instructions of the bibliography style itself will be executed. Because of this these commands has to be defined using `\renewcommand` not `\newcommand` inside of argument *instructions*.

If the user declares additional *instructions* using `\AtEndBibliography` and `\AfterBibliographyPreamble` to be executed after the preamble or at the end of the bibliography, the *instructions* of `\AfterBibliographyPreamble` will be executed only once at the begin of the bibliography but after the `\bib@beginhook` *instructions*, and the *instructions* of `\AtEndBibliography` will be executed only once at the end of the bibliography but before `\bib@endhook`.

Package `multicol` (see [Mit11]) could be used to define a bibliography style for printing the bibliography with two columns:

```
\newbibstyle{twocolumstyle}{%
  \renewcommand*{\bib@beginhook}{\begin{multicols}{2}}%
  \renewcommand*{\bib@endhook}{\end{multicols}}}%
```

If also an *open*-variation of this style should be defined, one may use the provided heredity feature and specify a *parent style*:

```
\newbibstyle{twocolumopenstyle}[openstyle]{%
  \renewcommand*{\bib@beginhook}{\begin{multicols}{2}}%
  \renewcommand*{\bib@endhook}{\end{multicols}}}%
```

These new defined styles may be selected using option `bibliography` as usual.

21.4. More or Less Obsolete Options and Commands

Sorry, currently additional information to this may be found at the same point of the German KOMA-Script book [Koh14a] only.

Additional Information about the Letter Class `scrlettr2` and the Letter Package `scrletter`

This chapter gives additional information about the KOMA-Script class `scrlettr2`. Some parts of the chapter are subject to the KOMA-Script book [Koh14a] only. This should not be a problem, because the average user, who only want to use the package, does not need the information. Other information is addressed to users, who want additional information about details. For example the first section will describe pseudo-lengths in detail. These may be used to modify the note paper. Another part of the information describes features of the class that exist only because of compatibility to deprecated `scrlettr` class. Last but not least it will be described in detail how to change a document from the old `scrlettr` class to be used with the current `scrlettr2` class.

v3.15

Since KOMA-Script 3.15 the additional package `scrletter` exists. It can be used with one of the KOMA-Script classes `scartcl`, `scrprrt`, or `scrbook` and provides all the features of `scrlettr2` for those classes. There are, however, small differences described later in this section.

22.1. Pseudo-Lengths for Experienced Users

\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 `scrlettr2` 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 `scrlettr2` are internally defined with macros instead of lengths. The drawback of this approach is that computations with macros is somewhat more complicated than with real lengths.

Please note: Even though these pseudo-lengths are internally implemented as macros, the commands for pseudo-length management expect only the names of the pseudo-lengths not the macros representing the pseudo-lengths. The names of pseudo-lengths are without backslash at the very beginning similar to the names of \LaTeX counters and in opposite to macros or \LaTeX 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 `scrlettr2`.

The pseudo-lengths defined and uses by `scrlettr2` are listed in [table 22.1](#). Cross references to the detailed descriptions of each pseudo-lengths in the following sub-sections are also given in the table.

A schematic display of the most important distances of the note paper is shown in [figure 22.1](#) at [page 469](#). Beside the pseudo-lengths for the modifiable distances, also some lengths, which may not be modified, are shown in light gray. Some rarely needed pseudo-length of the note paper have been omitted to get a more clear arrangement.

Table 22.1.: Pseudo-lengths provided by class `scrLtr2`

<code>backaddrheight</code>	height of the return address at the upper edge of the address field (section 22.1.3 , page 475)
<code>bfoldmarklength</code>	length of the bottommost folding mark (section 22.1.1 , page 470)
<code>bfoldmarkvpos</code>	vertical distance of bottommost folding mark from top paper edge (section 22.1.1 , page 470)
<code>firstfoothpos</code>	horizontal distance of the letter footer from the left paper edge; values greater than the width of the paper or smaller than the negative value of the width of the paper will activate special handling (section 22.1.8 , page 479)
<code>firstfootvpos</code>	vertical distance of letter footer from top paper edge (section 22.1.8 , page 479)
<code>firstfootwidth</code>	width of letter footer (section 22.1.8 , page 479)
<code>firstheadhpos</code>	horizontal distance of the letterhead from the left paper edge; values greater than the width of the paper or smaller than the negative value of the width of the paper will activate special handling (section 22.1.2 , page 472)
<code>firstheadvpos</code>	vertical distance of letterhead from top paper edge (section 22.1.2 , page 472)
<code>firstheadwidth</code>	width of the letterhead (section 22.1.2 , page 473)

Table 22.1.: Pseudo-lengths provided by class scrlltr2 (*continued*)

<code>foldmarkhpos</code>	horizontal distance of the horizontal folding marks from left paper edge (section 22.1.1, page 471)
<code>foldmarkvpos</code>	vertical distance of the vertical folding marks from the top paper edge (section 22.1.1, page 472)
<code>fromrulethickness</code>	thickness of an optional horizontal rule in the letterhead (section 22.1.2, page 473)
<code>fromrulewidth</code>	length of an optional horizontal rule in letterhead (section 22.1.2, page 473)
<code>lfoldmarkhpos</code>	horizontal distance of the vertical folding mark from the left paper edge (section 22.1.1, page 471)
<code>lfoldmarklength</code>	length of the vertical folding mark (section 22.1.1, page 471)
<code>locheight</code>	height of the field with the extended sender’s information, of the value is not zero; <code>toaddrheight</code> will be used instead of zero value (section 22.1.4, page 476)
<code>lochpos</code>	horizontal distance of the field with the extended sender’s information from the right paper edge, if the value is positive; negative horizontal distance from the left paper edge, if the value is negative; negative value of <code>toaddrhpos</code> will be used instead of zero value (section 22.1.4, page 476)
<code>locvpos</code>	vertical distance of the field with the extended sender’s information from the top paper edge, if the value is not zero; <code>toaddrvpos</code> will be used instead of zero value (section 22.1.4, page 476)
<code>locwidth</code>	width of the field with the extended sender’s information; for zero value width is calculated automatically with respect to option <code>locfield</code> that is described in section 4.10, page 187 (section 22.1.4, page 476)

Table 22.1.: Pseudo-lengths provided by class `scrLtr2` (*continued*)

<code>mfoldmarklength</code>	length of the middle horizontal folding mark (section 22.1.1, page 471)
<code>mfoldmarkvpos</code>	vertical distance of the middle horizontal folding mark from the top paper edge (section 22.1.1, page 470)
<code>pfoldmarklength</code>	length of the puncher hole mark (section 22.1.1, page 471)
<code>refaftervskip</code>	vertical skip below reference fields or business line (section 22.1.5, page 477)
<code>refhpos</code>	horizontal distance of reference fields or business line from left paper edge; for zero value reference fields line is centered horizontally on letter paper (section 22.1.5, page 477)
<code>refvpos</code>	vertical distance of reference fields or business line from top paper edge (section 22.1.5, page 476)
<code>refwidth</code>	width of reference fields line (section 22.1.5, page 477)
<code>sigbeforevskip</code>	vertical skip between closing and signature (section 22.1.7, page 478)
<code>sigindent</code>	indentation of signature with respect to text body (section 22.1.7, page 478)
<code>specialmailindent</code>	left indentation of mode of dispatch within address field (section 22.1.3, page 475)
<code>specialmailrightindent</code>	right indentation of mode of dispatch within address field (section 22.1.3, page 475)
<code>subjectaftervskip</code>	vertical distance after the subject (section 22.1.6, page 478)
<code>subjectbeforevskip</code>	additional vertical distance before the subject (section 22.1.6, page 478)

Table 22.1.: Pseudo-lengths provided by class scrlltr2 (*continued*)

subjectvpos	vertical distance of the subject from the top paper edge; zero value will set the subject depending on option subject (section 22.1.6, page 478)
tfoldmarklength	length of the topmost horizontal folding mark (section 22.1.1, page 471)
tfoldmarkvpos	vertical distance of the topmost horizontal folding mark from the top paper edge (section 22.1.1, page 470)
toaddrheight	height of address field (section 22.1.3, page 474)
toaddrhpos	horizontal distance of the address field from left paper edge, for positive values; negative horizontal distance of the address field from right paper edge, for negative values (section 22.1.3, page 473)
toaddrindent	left and right indentation of address within address field (section 22.1.3, page 474)
toaddrvpos	vertical distance of the address field from the top paper edge (section 22.1.3, page 473)
toaddrwidth	width of address field (section 22.1.3, page 474)

`\@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.

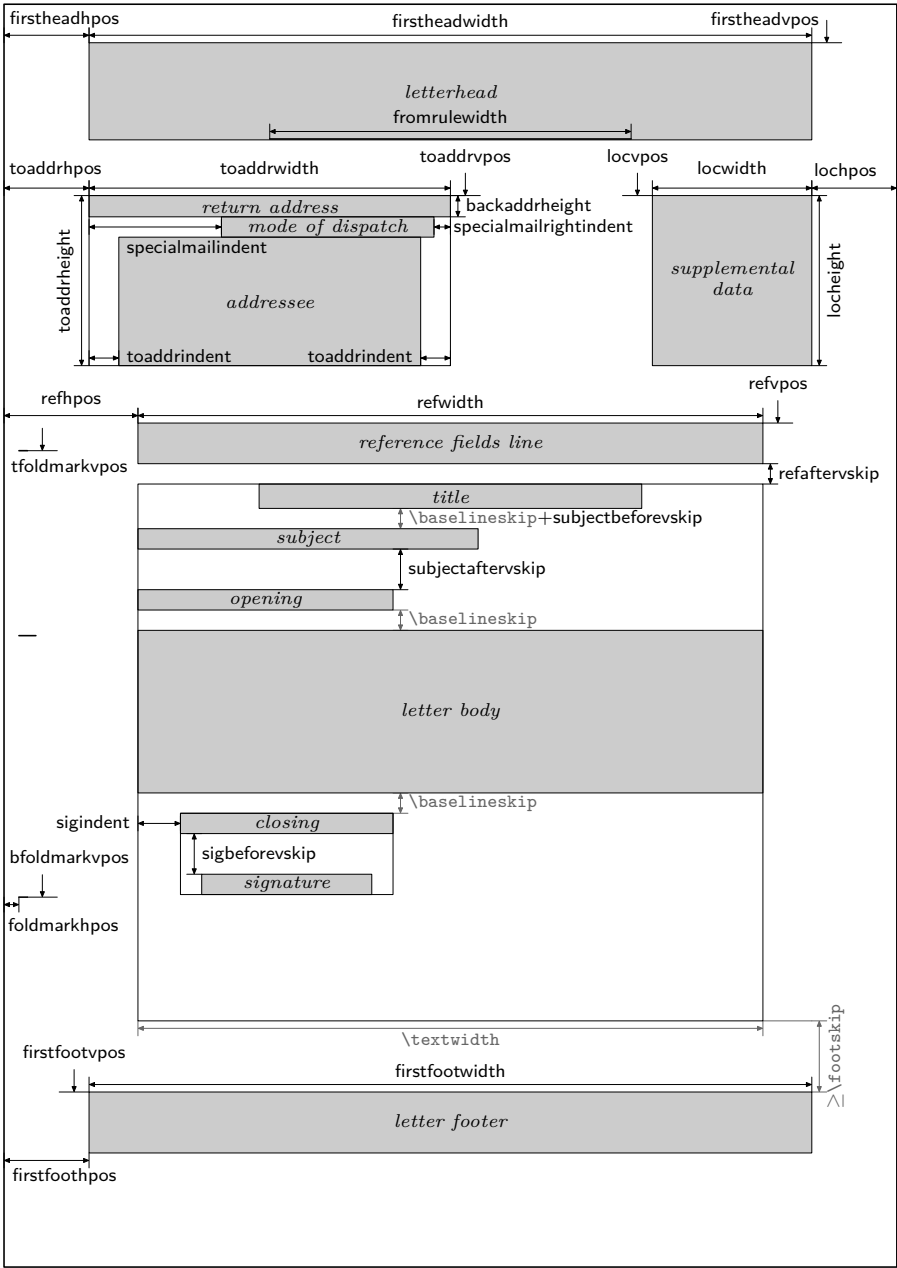


Figure 22.1.: Schematic of the pseudo-lengths for a letter

```
\@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`, section 4.2, page 147).

The command `\@addtoplength` adds the *value* to a *pseudo-length*. Again a *factor* may be given as an optional argument.

To assign, or to add the multiple of, one *pseudo-length* to another pseudo-length, the command `\useplength` (siehe section 4.2, page 147) 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.

22.1.1. Folding Marks

Folding mark are short horizontal lines at the left edge, and short vertical lines at the upper edge of the paper. KOMA-Script at present supports three configurable horizontal and one configurable vertical foldmarks. In addition, there is support for a puncher hole mark or center mark which cannot be shifted in the vertical direction.

```
\@setlength{tfoldmarkvpos}{length}
\@setlength{mfoldmarkvpos}{length}
\@setlength{bfoldmarkvpos}{length}
```

The letter class `scrlltr2` knows a total of three vertically placed configurable foldmarks. The position of the topmost foldmark, taken from the upper edge of the paper, is governed by the pseudo-length `tfoldmarkvpos`; the position of the middle foldmark by pseudo-length `mfoldmarkvpos`; the position of the bottommost foldmark by pseudo-length `bfoldmarkvpos`. With the addition of the puncher hole or center mark, there is still a fourth horizontal mark. This one is however always placed at the vertical center of the paper.

The topmost and bottommost foldmarks do not serve to divide the paper into exactly equal thirds. Instead, with their help, the paper should be folded such that the address field appears correctly in the space available in the chosen window envelope format, which is determined by choice of `lco`-file. Several such files are available offering predefined formats. An anomaly is present with `DINmtext`: for this format, an envelope format of C6/5 (also known as “C6 long”) is assumed. Letters written with this option are not suited to envelopes of formats C5 or C4.

The middle foldmark is not normally required for Western letters. In Japan, however, a larger number of envelope formats exists, requiring one more foldmark (see the Japanese `lco`-

files). At this point attention is drawn to the fact that reference to “topmost”, “middle”, and “bottommost” foldmarks is simply a convenience. In fact, it is not defined that `tfoldmarkvpos` must be smaller than `mfoldmarkvpos`, which in turn must be smaller than `bfoldmarkvpos`. If on the other hand one of the pseudo-lengths is set to null, then the corresponding foldmark will not be set even if the option `foldmarks` (see [section 4.10, page 168](#)) is explicitly activated.

```
\@setlength{tfoldmarklength}{length}
\@setlength{mfoldmarklength}{length}
\@setlength{bfoldmarklength}{length}
\@setlength{pfoldmarklength}{length}
```

v2.97e

These four pseudo-lengths determine the lengths of the four horizontal foldmarks. One exceptional behaviour exists. If the length is given as null, then the three vertically-configurable pseudo-lengths `tfoldmarklength`, `mfoldmarklength` and `bfoldmarklength` are set to 2 mm in length. The length of the punchmark, `pfoldmarklength`, is instead set to 4 mm.

```
\@setlength{foldmarkhpos}{length}
```

This pseudo-length gives the distance of all horizontal foldmarks from the left edge of the paper. Normally, this is 3.5 mm. This value can be changed in the user’s own `lco`-file, in case the user’s printer has a wider unprintable left margin. Whether the foldmarks are typeset at all depends on the option `foldmarks` (see [section 4.10, page 168](#)).

```
\@setlength{lfoldmarkhpos}{length}
```

v2.97e

Apart from the horizontal foldmarks there is also a vertical foldmark, whose position from the left margin is set via the pseudo-length `lfoldmarkhpos`. This foldmark is used, for example, in Japanese Chou- or You-format envelopes, when one wishes to use A4 size sheets with them. This can also be useful for envelopes in C6 format.

```
\@setlength{lfoldmarklength}{length}
```

v2.97e

The length of the vertical foldmark is set via the pseudo-length `lfoldmarklength`. Here too there is an exceptional behaviour. When the length is set to null, a length of 4 mm is actually used.

```
\@setlength{foldmarkvpos}{length}
```

v2.97e This pseudo-length gives the distance of all vertical foldmarks from the upper edge of the paper. Normally this is 3.5 mm, but the value can be changed in the user's personal `lco`-file in case the user's printer has a wider unprintable top margin. Whether the foldmarks are typeset at all depends on the option `foldmarks` (see [section 4.10](#), [page 168](#)). At present there is only one vertical foldmark, which is designated the left vertical foldmark.

```
\@setlength{foldmarkthickness}{length}
```

v2.97c This pseudo-length determines the thickness of all foldmarks. Default value is 0.2 pt, in other words a very thin hairline. In particular, if the color of the foldmarks is changed, this can be too thin!

22.1.2. 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`.

```
\@setlength{firstheadvpos}{length}
```

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.

```
\@setlength{firstheadhpos}{length}
```

v3.05 A positive value of pseudo-length `firstheadhpos` gives the distance between the left edge of the paper and the start of the letterhead. If is actually greater than or equal to the paper width, `\paperwidth`, then the letterhead will be centered to the note paper width. A negative value gives the distance between the distance between the right paper edge and the end of the letterhead. If the value is even less or equal to the negative value of the width of the paper, then the letterhead will be left aligned to the left edge of the typing area.

Typical default is a value of `\maxdimen`, though the greatest allowed value of a length. This will result in horizontal centering.


```
\@setlength{firstheadwidth}{length}
```

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` and has a definite value of 170 mm in `NF`.

```
\@setlength{fromrulethickness}{length}
```

```
\@setlength{fromrulewidth}{length}
```

Depending on the class option `fromrule` (see [section 4.10, page 172](#)), a horizontal rule can be drawn the predefined letterheads under or within the sender address. If the pseudo-length `fromrulewidth` has a value of 0 pt, 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 `\setlength` (see [page 470](#)). The default thickness of the line, `fromrulethickness`, is 0.4 pt.

v2.97c

22.1.3. Addressee

The term addressee 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.

```
\@setlength{toaddrvpos}{length}
```

```
\@setlength{toaddrhpos}{length}
```

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` or `NF`. 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 4.10, page 182](#)).

```
\@setlength{toaddrheight}{length}
```

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.

```
\@setlength{toaddrwidth}{length}
```

This pseudo-length 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`.

```
\@setlength{toaddrindent}{length}
```

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.

With each of the settings `addrfield=PP`, `addrfield=image`, and `addrfield=backgroundimage` (see [section 4.10, page 182](#)) a value of 0 pt will be replaced by a value of 8 mm. If really no indent should be used in this case, then 1 sp may be used to set a negligible small indent. Additionally `toaddrindent` will be used also for the distance to the right edge of the address window with the mentioned `addrfield` settings.

```
\@setlength{backaddrheight}{length}
```

For window envelopes the sender is often printed with small font at one line above the addressee. This kind of sender's information is known as return address, because it is visible at the address window and will be used by the post officers to return the letter (back) to the sender. In this return address only that information should be that is needed for this purpose.

The height, that is reserved for the return address at the top of the address field, is given by pseudo-length `backaddrheight`. A typical value for this is 5mm in the predefined `lco-fileslco-file=lco-file`. Whether or not to print the return address depend on option `addrfield` (see [section 4.10, page 182](#)) and `backaddress` (see [section 4.10, page 182](#)).

```
\@setlength{specialmailindent}{length}
```

```
\@setlength{specialmailrightindent}{length}
```

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 1em. Thus the dispatch type is set 1em from the address field's right margin.

```
\@setlength{PPheadheight}{length}
```

```
\@setlength{PPheadwidth}{length}
```

v3.03

The pseudo-length `PPheadheight` is the height, that will be reserved for the Port-Payé head using the options `addrfield=PP` and `addrfield=backgroundimage`. Pseudo-length `PPheadwidth` will be used only with `addrfield=PP` (see [section 4.10, page 182](#)) and gives the width of the left field in the Port-Payé head that contains P.P. logo, zip-code and place. The width of the right field with the sender's code and the priority is the remaining width.

Class `scrLtr2` automatically changes pseudo-length `PPheadheight`'s usual default value from 0mm into 20,74pt and `PPheadwidth`'s default from 0mm into 42mm.

```
\@setlength{PPdatamatrixvskip}{length}
```

v3.03

This pseudo-length gives the vertical distance between the Port-Payé head and the data array or data matrix of option `addrfield=PP` (see [section 4.10, page 182](#)). Class `scrLtr2` automatically changes the default value 0mm into 9mm. The data matrix will be set right aligned with the Port-Payé head.

22.1.4. 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*. In former manuals it has been called *location field*.

```
\@setlength{locheight}{length}
\@setlength{lochpos}{length}
\@setlength{locvpos}{length}
\@setlength{locwidth}{length}
```

v2.97d

The pseudo-lengths `locwidth` and `locheight` set the width and height of the sender's extension field. The pseudo-lengths `lochpos` and `locvpos` determine the distances from the right and upper paper edges. These value is typically set to 0pt in the predefined `lco` files. This does not mean that the sender's extension has no width; instead, it means that the actual width is set within `\opening` when the paper width, address window width, and the distance between the left and upper edges of the paper and the address window are known. The option `locfield` (see [section 4.10](#), [page 187](#)) is also taken into account. As is the case for `toaddrhpos`, negative values of `lochpos` take on a special meaning. In that case, instead of referring to a distance from the right edge of the paper, `lochpos` now means a distance from the left edge of the paper. The meaning is thus the opposite to that of `toaddrhpos` (see [section 22.1.3](#), [page 473](#)).

22.1.5. Business 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. This line is sometimes called *reference fields line* or *reference line*, sometimes *business line*. The business 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 forced white space or `\null`. If you want to have your letter without a business line, then instead of it the label and contents of the variable `date` will be set. Information about adding variables to the business line or clean up the business line may be found in [section 22.2](#), [page 480](#).

```
\@setlength{refvpos}{length}
```

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.

```
\@setlength{refwidth}{length}
\@setlength{refhpos}{length}
```

This pseudo-length gives the width that is available for the reference fields line. The value is set typically to 0pt 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 within the command `\opening`. Thus the calculated width depends on the determination of the options `refline` (see [section 4.10, page 189](#)). 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 inside the typing area.

If `refwidth` is not 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.

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.

```
\@setlength{refaftervskip}{length}
```

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.

22.1.6. Subject

In different countries the letter's subject will be set different. Some like to have the subject before the opening phrase, some prefer the subject below the opening phrase. Some professional guilds at least want the subject before the business line.

```
\@setlength{subjectvpos}{length}
```

v3.01

If the value of this pseudo-length is 0pt, the position of the subject is given by option `subject` (see [section 4.10, page 193](#)). Every other value defines the distance between the top edge of the paper and the subject. It is recommended to take care of the available space to surely avoid interferences with other elements.

Example: Some professional guilds prefer to have the subject at least before the business line. To achieve this, the position may be defined like this:

```

\ProvidesFile{lawsubj.lco}
      [2008/11/03 lawyers lco file]
\@setlength{subjectvpos}{\uselength{refvpos}}
\@addtoplength{refvpos}{3\baselineskip}
\endinput

```

which also changes the position of the business line itself. If at least one empty line between subject and business line should stay empty, this provides a maximum of two subject lines.

```

\@setlength{subjectbeforevskip}{length}
\@setlength{subjectaftervskip}{length}

```

v3.01

If the subject is not positioned absolutely, but before or after the opening phrase, additional vertical spaces may be inserted before and after the subject. Thereby, the space before the subject may interfere with other distances like the automatic distance of one line after the title. Because of this the default is to use no additional space here. In contrast, the class's default for the space after the subject is two lines.

22.1.7. 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.

```

\@setlength{sigindent}{length}
\@setlength{sigbeforevskip}{length}

```

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.

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 two lines. In this space you can then write your inscription.

22.1.8. Letter Footer

As the first page of a letter, the note paper, 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`.

`\@setlength{firstfootvpos}{length}`

This pseudo-length gives the distance between the letter footer and the upper edge of the paper. It also takes care of preventing text from jutting into the footer area. For this the text height on the first page will be decreased using `\enlargethispage` if needed. Likewise, and if it is wanted, the text height can conversely be extended with the help of the option `enlargefirstpage` (see [section 4.10, page 170](#)). This way, the distance between text area and the first letter footer can be reduced to the value `\footskip`.

v2.9t

With the compatibility option set up to version 2.9t (see `version` in [section 4.4, page 149](#)) the footer is set independently of the type area in all predefined `lco`-files (see [section 4.21](#)) except for KOMAold and NF. 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`.

v2.9t

If the letter footer be switched off using option `firstfoot=false` (see [section 4.10, page 195](#)), then the setting of `firstfootvpos` is ignored, and instead `\paperheight` is applied. Thus, there remains a minimum bottom margin of length `\footskip`.

`\@setlength{firstfootpos}{length}`

v3.05

A positive value of pseudo-length `firstfootpos` gives the distance between the letter footer and the left edge of the paper. If the value is even greater than or equal to the paper width, `\paperwidth`, then the footer will be centered horizontally to the note paper. But if the value is less than or equal to the negative width of the paper, then the footer will be aligned left to the typing area.

Typical default for this value is `\maxdimen`, that is the maximum allowed value of a length. This results in horizontal centering.

`\@setlength{firstfootwidth}{length}`

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.

22.2. Variables for Experienced Users

KOMA-Script provides beside the feature of using predefined variable also commands to define new variable or to manipulate the automatic usage of variables in the business line.

```
\newkomavar[description]{name}
\newkomavar*[description]{name}
\removeeffields
\defaultreffields
\addtoreffields{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*. In opposite to *name* the *description* is not used to reference a variable. In fact the *description* defines an addition to the content of a variable, that may be output similar to the variable content.

Using the command `\addtoreffields` you can add the variable *name* to the reference fields line (see [section 4.10, page 189](#)). 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 `\addtoreffields`. 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 field line can be removed. This also includes the predefined variables of the class. The reference fields line is then empty. 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.

The date should not be added to the reference fields line using `\addtoreffields`. Instead option `date` should be used to select the date left, right or not at the business line. This option also will change the position of the date if no reference fields will be output.

```
\usekomavar[command]{name}
\usekomavar*[command]{name}
```

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 `scrlayer-scrpage`. However, these kinds of commands can not be used within commands like

`\MakeUppercase` which directly influence their argument. 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}
```

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. However, they are robust enough to be used as the argument of, e. g., `\markboth` or `\footnote`.

22.3. Differences in the Page Styles of scrletter

As described in [section 4.13](#), the page style `headings` shows the content of variables `nexthead` and `nextfoot`. Page style `headings` has already another meaning in the KOMA-Script classes `scrbook`, `scrreprt`, and `scrartcl`. Therefore `scrletter` defines a new pair of page styles (see [section 18.2, page 416](#)). The definition is done using package `scrlayer-scrpage`.

Hence, you cannot use `scrletter` with the outdated packages `scrpage2` and `scrpage`. Same applies to `fancyhdr` that is in general not recommended for use with KOMA-Script. Additionally, internally loading package `scrlayer-scrpage` automatically activates page style `scrheadings`. This activation changes page styles `headings` and `plain` into aliases of `scrheadings` and `plain.scrheadings`. So these page styles are also controlled by `scrlayer-scrpage`.

```
\pagestyle{letter}
\pagestyle{plain.letter}
\setkomavar{nexthead}[description]{contents}
\setkomavar{nextfoot}[description]{contents}
```

Package `scrletter` defines the pair of page style `letter` using `scrlayer-scrpage` to become independent from the page styles of the document class. Variables `nexthead` and `nextfoot` are used by `scrletter`'s `letter` in the same way as for `scrletter2`'s `headings`. The corresponding `plain` page style is also similar to `scrletter2`'s page style `plain`. Notably the position of the page number depends on option `pagenumber` (see [page 200](#)).

```
headsepline
footsepline
```

Because of the different internal approach of `scrletter`, the options `headsepline` and `footsepline` differ slightly from class `scrletter`, because they are controlled by package `scrlayer-scrpage`. Especially the horizontal lines in page head and foot of the `plain` page style depend on the options of `scrlayer-scrpage`. See [section 5.5, page 249](#) and [page 251](#) for information about those options.

```
\pagemark
\letterpagemark
```

The KOMA-Script classes already define and use `\pagemark`. So when `scrletter` is loaded, it cannot simply redefine `\pagemark` to add the prefix “Page” using `\pagename`, because this would not only influence the pagination of the letter pages but the pagination of the whole document. Therefore `scrletter` defines a new command `\letterpagemark` with the prefix and lets `\pagemark` become `\letterpagemark` in `\begin{letter}` until the end of the letter only. If you want to redefine `\pagemark` for the whole document including the letter pages, you have to add

```
\let\letterpagemark\pagemark
```

just after the normal redefinition of `\pagemark`. This add-on also will not violate anything if you later decide to switch from package `scrletter` to class `scrletter`. Note that the class `scrletter` does not provide or use `\letterpagemark`.

22.4. Differences in the Handling of `lco`-Files in `scrletter`

As shown in [section 4.21](#), `scrletter` can load `lco`-files via the optional argument of `\documentclass`. Package `scrletter` does not support this.

```
\LoadLetterOption{name}
\LoadLetterOptions{list of names}
```

For `scrletter` it is only recommended to load `lco`-files using `\LoadLetterOption` or `\LoadLetterOptions`. With `scrletter` you must use these commands to load `lco`-files. Of course, you can use these commands at the earliest after loading `scrletter`.

22.5. `lco`-Files for Experienced Users

Even though each paper size, that may be set up using package `typearea`, may be also used with `scrletter`, the result of the first page may be unwanted with some of those page sizes. The conception of the class is not the reason for this, but the fact, that there are mainly parameter sets for ISO A4 paper. Unfortunately there are not any universal rules, to calculate, e. g., the

v3.17

v3.15

position of the address field or similar for every available paper sizes. But it is possible to make parameter sets for any paper size that is needed.

22.5.1. Survey of Paper Size

At present there exist only parameter sets and lco-files for A4-sized or letter-sized paper. Nevertheless, class scrlltr2 supports many more paper sizes. Because of this it's necessary to survey setting up the correct paper size.

```
\LetterOptionNeedsPapersize{option name}{paper size}
```

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.

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 topmost and bottommost folding marks. If there would not be options for this, the easiest way to achieve this would be to place the marks outside of the paper area.

```
\ProvidesFile{a5.lco}
      [2002/05/02 letter class option]
\LetterOptionNeedsPapersize{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 loading “a5.lco”.

22.5.2. Visualization of Positions

If someone develops a new lco-file, e.g., to adapt the positions of the several fields of the note paper because of own wishes or because it's simply necessary, then it often will be useful to

v3.04

make at least some elements directly visual. This is the sense of lco-file `visualize.lco`. This file may be loaded like each other lco-file. But in difference to other lco-files it has to be done in the document preamble and it's not possible to switch off the effects of that lco-file. This lco-file uses packages `eso-pic` and `graphicx`, that are not part of KOMA-Script.

`\showfields{field list}`

This command activates the visualization of note paper fields. Argument *field list* is a comma separated list of fields that should be visualized. Following are the supported fields:

- test** – is a test field of size 10 cm by 15 cmd with position 1 cm down and right from the topmost and leftmost edges of the paper. This field exists for debugging purpose. It may be used as an measure comparison in the case, that the measures will be adulterated while printing.
- head** – is the header area of the note paper. This area has an open bottom.
- foot** – is the footer area of the note paper. This area has an open top.
- address** – is the address window area used by window envelopes.
- location** – is the field of the sender's extension.
- refline** – is the business line. This are has an open bottom.

The color of the visualization may be changed using commands `\setkomafont` und `\addtokomafont` (see [section 4.9, page 164](#)) with the element `field`. Default is `\normalcolor`.

`\setshowstyle{visualization style}`

`\edgesize`

The default for the visualization of single areas are frames around the areas. Areas with open top or bottom are not framed completely but have an open edge with arrows at the end of the ending lines. Alternatively the *visualization style* rule may be used. In this case a background color will be used to visualize the areas. This does not differ open and closed areas. Instead a minimal height will be used for open areas. The third available *visualization style* is *edges*. This will mark the corners of the areas. The corner marks at the open edge of open areas will be omitted. The size of two edges of the corner marks are given by the macro `\edgesize` with default 1 ex.

```
\showenvelope(width,height)(h-offset,v-offset)[instructions]
\showISOenvelope{format}[instructions]
\showUScommercial{format}[instructions]
\showUScheck[instructions]
\unitfactor
```

These commands may be used to output a graphics of an envelope. The envelope of the graphic will be rotated by 90° and printed in measure 1:1 to one document page. The addressee window will be generated automatically using the current data of the address position of the note paper: `toaddrvpos`, `toaddrheight`, `toaddrwidth`, and `toaddrhpos`. For this the differences *h-offset* and *v-offset* of size of the folded letter sheet to the size of the envelope, *width* and *height*, will be needed. If both values, *h-offset* and *v-offset*, will be omitted using `\showenvelope`, then these will be calculated from the folding marks and the paper size.

Commands `\showISOenvelope`, `\showUScommercial`, and `\showUScheck` base on `\showenvelope`. With `\showISOenvelope` ISO-envelopes with *format* C4, C5, C5/6, DL (also known as C5/6) or C6 may be generated. With `\showUScommercial` an US-commercial envelope of *format* 9 or 10 may be generated. `\showUScheck` may be used for envelopes in format US-check.

The *instructions* may be used to print additional elements inside the envelope.

The position of the letter sheet will be signed with dash lines inside the envelope. The color of this dash lines may be changed using commands `\setkomafont` und `\addtokomafont` (see [section 4.9, page 164](#)) with element `letter`. Default is `\normalcolor`.

The envelope will be dimensioned automatically. The color of the dimensioning may be changed using commands `\setkomafont` und `\addtokomafont` (see [section 4.9, page 164](#)) with element `measure`. Default is `\normalcolor` The dimensioning will be done in multiply of `\unitlength` with accuracy of `1/\unitfactor`. Nevertheless accuracy of the T_EX arithmetic also limits the accuracy of dimensioning. Default is 1. The `\unitfactor` may be changed using `\renewcommand`.

Example: An example letter in format ISO-A4 will be produced. The supported fields should be visualized with yellow frame lines. Additionally the position of the window of an envelope with size DL should be checked with a graphics. The measure lines of the dimensioning should be red and the measure numbers should use a small font. The accuracy of the dimensioning should be 1 mm. The dashed note paper sheet in the envelope should be colored green.

```
\documentclass[visualize]{scr1ttr2}
\usepackage{xcolor}
\setkomafont{field}{\color{yellow}}
\setkomafont{measure}{\color{red}\small}
\setkomafont{letter}{\color{green}}
\showfields{head,address,location,refline,foot}
\usepackage[ngerman]{babel}
```

```

\usepackage{lipsum}
\begin{document}
\setkomavar{fromname}{Peter Musterfrau}
\setkomavar{fromaddress}{Hinter dem Tal 2\\
                        54321 Musterheim}

\begin{letter}{%
    Joana Public\\
    Hillside 1\\
    12345 Public City%
}
\opening{Hello,}
\lipsum[1]
\closing{Good bye}
\end{letter}
\setlength{\unitlength}{1cm}
\renewcommand*{\unitfactor}{10}
\showISOenvelope{DL}
\end{document}

```

This will show the note paper on the first and the envelope graphic on the second document page.

Please note, that inauspicious combinations of `\unitlength` and `\unitfactor` may provoke \TeX errors like *arithmetic overflow* very soon. Also shown measure numbers may differ a little bit from the real value. Both are not mistakes of `visualize` but simple implementation limitations of \TeX .

22.6. Language Support

The document class `scrlettr2` supports many languages. These include German (`german` for old German orthography, `ngerman` for the new orthography, `austrian` for Austrian with old German orthography, and `naustrian` for Austrian with new orthography), 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, Finnish, Norsk, and Swedish.

If the package `babel` (see [BB13]) 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.

There is one thing more to mention about language packages. The package `french` (see [Gau07]) redefines not only the terms of table 22.3, but also other, for instance some versions of that package even redefine the command `\opening`, since the package assumes that the definition of the standard letter is used. With `scrlettr2` this is not the case, therefore the package

v3.09

v3.08

If one utilizes the `babel` package in order to switch to language `french` while the package `french` is simultaneously installed, then the same problems may likely occur, since `babel` employs definitions from the `french` package.

`\usepackage[...frenchb,...]{babel}`

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.

If one switches the language of a letter then using these commands the language-dependent terms from [table 22.3, page 490](#) are redefined. If the used language selection scheme does not support this then the commands above can be used directly.

Table 22.2.: Language-dependent forms of the date

Command	Date example
<code>\dateenglish</code>	24/12/1993
<code>\dateUSenglish</code>	12/24/1993
<code>\dateamerican</code>	12/24/1993
<code>\datebritish</code>	24/12/1993
<code>\dateUKenglish</code>	24/12/1993
<code>\dategerman</code>	24. 12. 1993
<code>\datengerman</code>	24. 12. 1993
<code>\dateaustrian</code>	24. 12. 1993
<code>\datefrench</code>	24. 12. 1993
<code>\dateitalian</code>	24. 12. 1993
<code>\datespanish</code>	24. 12. 1993
<code>\datedutch</code>	24. 12. 1993
<code>\datecroatian</code>	24. 12. 1993.
<code>\datefinnish</code>	24.12.1993.
<code>\datenorsk</code>	24.12.1993
<code>\dateswedish</code>	24/12 1993

`\dateenglish`
`\dateUSenglish`
`\dateamerican`
`\datebritish`
`\dateUKenglish`
`\dategerman`
`\datengerman`
`\dateaustrian`
`\datenaustrian`
`\datefrench`
`\dateitalian`
`\datespanish`
`\datedutch`
`\datecroatian`
`\datefinnish`
`\datenorsk`
`\dateswedish`

The numerical representation of the date (see option `numericaldate` in [section 4.10](#), [page 189](#)) will be written depending on the selected language. Some examples can be found in [table 22.2](#).


```

\yourrefname
\yourmailname
\myrefname
\customername
\invoicename
\subjectname
\ccname
\enclname
\headtoname
\headfromname
\datename
\pagename
\mobilephonenumber
\phonename
\faxname
\emailname
\wwwname
\bankname

```

The commands 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 12.4](#). The definitions become active only at `\begin{document}`. Therefore they are not available in the L^AT_EX preamble and cannot be redefined there. In [table 22.3](#) the default settings for `english` and `ngerman` can be found.

22.7. From Obsolete `scrlettr` to Current `scrlettr2`

With the June 2002 release of `scrlettr2` (see [chapter 4](#)) 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 L^AT_EX several times, like this:

```

latex scrlettr.dtx
mkindex scrlettr
latex scrlettr.dtx
mkindex scrlettr
latex scrlettr.dtx

```

Then you obtain the file `scrlettr.dvi` containing the old German manual. If you want `scrlettr.pdf` instead of `scrlettr.dvi` you should use `pdflatex` instead of `latex`.

To facilitate the transition to the new class, there is the compatibility option `KOMAold`. In general, the complete older functionality still remains in the new class. Without `KOMAold`, the

Table 22.3.: Default settings for language-dependent terms using languages `english` and `ngerman`, as long as language selection packages have not been used

Command	english	ngerman
<code>\bankname</code>	Bank account	Bankverbindung
<code>\ccname¹</code>	cc	Kopien an
<code>\customername</code>	Customer no.	Kundennummer
<code>\datename</code>	Date	Datum
<code>\emailname</code>	Email	E-Mail
<code>\enclname¹</code>	encl	Anlagen
<code>\faxname</code>	Fax	Fax
<code>\headfromname</code>	From	Von
<code>\headtoname¹</code>	To	An
<code>\invoicename</code>	Invoice no.	Rechnungsnummer
<code>\myrefname</code>	Our ref.	Unser Zeichen
<code>\pagename¹</code>	Page	Seite
<code>\mobilephonenumber</code>	Mobile phone	Mobiltelefon
<code>\phonenumber</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

¹ Normally these terms are defined by language packages like `babel`. In this case they are not redefined by `scrلتtr2` and may differ from the table above.

user interface and the defaults will be different. More details on this option are provided in [section 4.21](#), [table 4.18](#).

Sorry, currently additional information to this may be found at the same point of the German KOMA-Script book [\[Koh14a\]](#) only.

Japanese Letter Support for scrlltr2¹

Since version 2.97e scrlltr2 provides support not only for European ISO envelope sizes and window envelopes, but also for Japanese envelopes, in the form of lco files which set the layout of the paper. This chapter documents the support, and provides a few examples of using the provided lco files for printing letters intended for Japanese envelopes.

A.1. Japanese standard paper and envelope sizes

The Japan Industrial Standard (JIS) defines paper sizes and envelope sizes for national use, which both overlap with the ISO and US sizes and include some metricated traditional Japanese sizes. Envelope window size and position have not been defined internationally as yet; hence, there exists a plethora of envelopes with differing window sizes and positions. The below subsections give some background on Japanese paper sizes and envelopes.

A.1.1. Japanese paper sizes

The JIS defines two main series of paper sizes:

1. the JIS A-series, which is identical to the ISO A-series, but with slightly different tolerances; and
2. the JIS B-series, which is not identical to the ISO/DIN B-series. Instead, the JIS B-series paper has an area 1.5 times that of the corresponding A-series paper, so that the length ratio is approximately 1.22 times the length of the corresponding A-series paper. The aspect ratio of the paper is the same as for A-series paper.

Both JIS A-series and B-series paper is widely available in Japan and most photocopiers and printers are loaded with at least A4 and B4 paper. The ISO/JIS A-series, and the different ISO and JIS B-series sizes are listed in [table A.1](#).

There are also a number of traditional paper sizes, which are now used mostly only by printers. The most common of these old series are the Shiroku-ban and the Kiku paper sizes. The difference of these types compared to the JIS B-series are shown in [table A.2](#). Finally, there are some common stationary sizes, listed in [table A.3](#). You may come across these when buying stationary.

The ISO C-series is not a paper size as such, but is a standard developed for envelopes, intended for the corresponding A-series paper, and is discussed in the next subsection.

¹This chapter has been written originally by Gernot Hassenpflug.

Table A.1.: ISO and JIS standard paper sizes

ISO/JIS A	W×H in mm	ISO B	W×H in mm	JIS B	W×H in mm
A0	841×1189	B0	1000×1414	B0	1030×1456
A1	594×841	B1	707×1000	B1	728×1030
A2	420×594	B2	500×707	B2	515×728
A3	297×420	B3	353×500	B3	364×515
A4	210×297	B4	250×353	B4	257×364
A5	148×210	B5	176×250	B5	182×257
A6	105×148 ¹	B6	125×176	B6	128×182
A7	74×105	B7	88×125	B7	91×128
A8	52×74	B8	62×88	B8	64×91
A9	37×52	B9	44×62	B9	45×64
A10	26×37	B10	31×44	B10	32×45
A11	18×26			B11	22×32
A12	13×18			B12	16×22

¹ Although Japan's official postcard size appears to be A6, it is actually 100×148 mm, 5 millimeters narrower than A6.

A.1.2. Japanese envelope sizes

ISO (International Organization for Standardization) envelope sizes are the official international metric envelope sizes; however, Japan uses also JIS and metricated traditional envelope sizes. Sizes identified as nonstandard do not conform to Universal Postal Union requirements for correspondence envelopes.

Table A.2.: Japanese B-series variants

Format Size	JIS B-series W×H in mm	Shiroku-ban W×H in mm	Kiku W×H in mm
4	257×364	264×379	227×306
5	182×257	189×262	151×227
6	128×182	189×262	
7	91×128	127×188	

Table A.3.: Main Japanese contemporary stationary

Name	W×H in mm	Usage and Comments
Kokusai-ban	216×280	“international size” i. e., US letter size
Semi B5 or Hyoujun-gata	177×250	“standard size” (formerly called “Hyoujun-gata”), semi B5 is almost identical to ISO B5
Oo-gata	177×230	“large size”
Chuu-gata	162×210	“medium size”
Ko-gata	148×210	“small size”
Ippitsu sen	82×185	“note paper”

ISO envelope sizes

The ISO C-series envelope sizes, and possibly B-series envelope sizes, are available in Japan. C-series envelopes can hold the corresponding A-series paper, while B-series envelopes can hold either the corresponding A-series paper or the corresponding C-series envelope. The ISO envelope sizes commonly for Japan are listed in [table A.4](#), with the corresponding paper they are intended for, and the folding required.

JIS and traditional envelope sizes

The JIS classifies envelopes into three categories based on the general shape of the envelope, and where the flap is located:

You: these envelopes are of the ‘commercial’ type, rectangular, and correspond largely to Western envelope sizes, and also have the flap on the long dimension (‘Open Side’) in ‘commercial’ or ‘square’ style. ‘You-kei’ means Western-style.

Chou: these are also ‘commercial’ type envelopes, with the same shape as the corresponding ‘You’ type, but with the flap on the short dimension (‘Open End’) in ‘wallet’ style. ‘Chou-kei’ means long-style.

Kaku: these envelopes are more square in appearance and are made for special use, and correspond to ‘announcement’ envelopes. The flap is on the long side, in the ‘square’ style. They generally do not fall under the ordinary envelope postage rates. ‘Kaku-kei’ means square-style.

The main JIS and traditional envelope sizes and the corresponding paper and its required folding are listed in [table A.5](#).

Table A.4.: Japanese ISO envelope sizes

Name	W×H in mm	Usage and Comments
C0	917×1297	for flat A0 sheet; nonstandard
C1	648×917	for flat A1 sheet; nonstandard
C2	458×648	for flat A2 sheet, A1 sheet folded in half; nonstandard
C3	324×458	for flat A3 sheet, A2 sheet folded in half; nonstandard
B4	250×353	C4 envelope
C4	229×324	for flat A4 sheet, A3 sheet folded in half; very common; nonstandard
B5	176×250	C5 envelope
C5	162×229	for flat A5 sheet, A4 sheet folded in half; very common; nonstandard
B6	125×176	C6 envelope; A4 folded in quarters; very common
C6	114×162	for A5 sheet folded in half, A4 sheet folded in quarters; very common
C6/C5	114×229	A4 sheet folded in thirds; very common
C7/6	81×162	for A5 sheet folded in thirds; uncommon; nonstandard
C7	81×114	for A5 sheet folded in quarters; uncommon; nonstandard
C8	57×81	
C9	40×57	
C10	28×40	
DL ¹	110×220	for A4 sheet folded in thirds, A5 sheet folded in half lengthwise; very common

¹ Although DL is not part of the ISO C-series, it is a very widely used standard size. DL, probably at one time the abbreviation of DIN Lang (Deutsche Industrie Norm, long), is now identified as “Dimension Lengthwise” by ISO 269.

Table A.5.: Japanese JIS and other envelope sizes

JIS	Name	W× in mm	Usage and Comments
	Chou 1	142×332	for A4 folded in half lengthwise; nonstandard
Yes	Chou 2	119×277	for B5 folded in half lengthwise; nonstandard
Yes	Chou 3	120×235	for A4 folded in thirds; very common
	Chou 31	105×235	for A4 folded in thirds
	Chou 30	92×235	for A4 folded in fourths ³
	Chou 40	90×225	for A4 folded in fourths ³
Yes	Chou 4	90×205	for JIS B5 folded in fourths ³ ; very common
	Kaku A3	320×440	for A3 flat, A2 folded in half ; nonstandard
	Kaku 0	287×382	for B4 flat, B3 folded in half; nonstandard
	Kaku 1	270×382	for B4 flat, B3 folded in half; nonstandard
Yes	Kaku 2	240×332	for A4 flat, A3 folded in half; nonstandard
	Kaku Kokusai A4	229×324	for A4 flat, A3 folded in half; same size as ISO C4; nonstandard
Yes	Kaku 3	216×277	for B5 flat, B4 folded in half; nonstandard
Yes	Kaku 4	197×267	for B5 flat, B4 folded in half; nonstandard
Yes	Kaku 5	190×240	for A5 flat, A4 folded in half ; nonstandard
Yes	Kaku 6	162×229	for A5 flat, A4 folded in half; same size as ISO C5; nonstandard
Yes	Kaku 7	142×205	for B6 flat, B5 folded in half; nonstandard
Yes	Kaku 8	119×197	pay envelope (for salaries, wages) ; common for direct mail

Table A.5.: Japanese JIS and other envelope sizes (*continued*)

JIS	Name	W× in mm	Usage and Comments
Yes	You 0 ¹ or Furusu 10	235×120	for A4 folded in thirds; same size as Chou 3 but with ‘Open Side’ style flap
	You 0 ¹	197×136	for kyabine ¹ (cabinet) size photos (165 mm×120 mm); nonstandard
	You 1 ²	176×120	for B5 folded in quarters
	You 1 ²	173×118	for B5 folded in quarters
Yes	You 2	162×114	for A5 folded in half, A4 folded in quarters; same size as ISO C6
Yes	You 3	148×98	for B6 folded in half
Yes	You 4	235×105	for A4 folded in thirds
Yes	You 5	217×95	for A4 folded in fourths ³
Yes	You 6	190×98	for B5 folded in thirds
Yes	You 7	165×92	for A4 folded in quarters, B4 folded in quarters

¹Because two different sizes are called You 0, the JIS You 0 is normally called Furusu 10; Furusu (‘fools’) derives from ‘foolscap’; Kyabine is a metricated traditional Japanese size.
²Two slightly different sizes are sold as You 1; the smaller size (173 mm×118 mm) is the paper-industry standard size.
³Twice in the same direction.

Window variants

There are a large number of window subtypes existing within the framework explained in the previous subsection. The most common window sizes and locations are listed in [table A.6](#).

A.2. Provided lco files

In `scrlltr2` support is provided for Japanese envelope and window sizes through a number of `lco` files which customize the foldmarks required for different envelope sizes and subvariants with different window positions and sizes.

The provided `lco` files together with the envelope types for which they provide support are listed at [table A.7](#). See [table A.4](#) for the full list of Japanese envelopes and the paper they take, and [table A.6](#) for the common window sizes and locations. The rightmost column indicates which `lco` file provides the support.

Table A.6.: Supported Japanese envelope types and the window sizes and locations.

Envelope type	Window name ¹	- size ²	- location ³	lco file ⁴
Chou 3	A	90×45	l 23, t 13	NipponEL
Chou 3	F	90×55	l 23, t 13	NipponEH
Chou 3	Hisago	90×45	l 23, t 12	NipponEL
Chou 3	Mutoh 1	90×45	l 20, t 11	NipponEL
Chou 3	Mutoh 101	90×55	l 20, t 11	NipponEH
Chou 3	Mutoh 2	80×45	l 20, t 11	NipponEL
Chou 3	Mutoh 3	90×45	l 25, t 11	NipponLL
Chou 3	Mutoh 301	90×55	l 25, t 11	NipponLH
Chou 3	Mutoh 6	100×45	l 20, t 11	NipponEL
Chou 3	v.2 ⁵	90×45	l 24, t 12	NipponLL
Chou 40	A	90×45	l 23, t 13	NipponEL
Chou 4	A	90×45	l 23, t 13	NipponEL
Chou 4	B	80×45	l 98, t 28	NipponRL
Chou 4	C	80×45	l 21, t 13	NipponEL
Chou 4	K	80×45	l 22, t 13	NipponEL
Chou 4	Mutoh 1	80×45	l 40, b 11	—
Chou 4	Mutoh 2	80×45	l 20, t 11	NipponEL
Chou 4	Mutoh 3	90×45	l 20, t 11	NipponEL
Chou 4	Mutoh 6	100×45	l 20, t 11	NipponEL
Chou 4	v.2 ⁵	80×45	l 20, t 12	NipponEL
Chou 4	v.3 ⁵	90×45	l 20, t 12	NipponEL
Kaku A4	v.1 ⁶	95×45	l 20, t 24	KakuLL
You 0	Cruise 6	90×45	l 20, t 12	NipponEL
You 0	Cruise 601	90×55	l 20, t 12	NipponEH
You 0	Cruise 7	90×45	l 20, b 12	NipponEL
You 0	Cruise 8	90×45	l 24, t 12	NipponLL
You 0	v.2 ⁵	90×45	l 24, t 12	NipponEL
You 0	v.3 ⁵	90×45	l 23, t 13	NipponEL
You 4	A	90×45	l 23, t 13	NipponEL

¹Names (acting as subtype information) are taken from the manufacturer catalog.²Given as width by height in millimeters.³Given as offset from left (l) or right (r), followed by offset from bottom (b) or top (t).⁴The lco file, which provides support (see [table A.7](#)).⁵In the absence of any other information, a numerical variation number for the subtype name is provided.⁶Dimensions apply when envelope is held in portrait mode.

Table A.7.: lco files provided by scrlltr2 for Japanese window envelopes

lco file	Supported	Window size ¹	Window location ¹
NipponEL	Chou/You 3 and 4	90×45	l 22, t 12
NipponEH	Chou/You 3 and 4	90×55	l 22, t 12
NipponLL	Chou/You 3 and 4	90×45	l 25, t 12
NipponLH	Chou/You 3 and 4	90×55	l 25, t 12
NipponRL	Chou/You 3 and 4	90×45	l 98, t 28
KakuLL	Kaku A4	90×45	l 25, t 24

¹Window size is given in width by height, location as offset from left (l) or right (r), followed by offset from bottom (b) or top (t). All Values in millimeters.

The tolerances for location is about 2 mm, so it is possible to accommodate all the envelope and window variants of [table A.6](#) with just a small number of lco files. The difference between Chou/You 3 and Chou/You 4 is determined by paper size.

A.3. Examples of Japanese letter usage

Assume you want to write a letter on A4 size paper and will post it in a Japanese envelope. If the envelope has no window, then it is enough to determine whether the envelope dimensions match a European one — the standard DIN.lco style may suffice for many such cases.

If you wish to use a windowed envelope, please note that owing to the large variety, not all existing subvariants are currently supported. If you should note that you particular windowed envelope has its window dimensions and positions significantly (more than approximately 2 mm) different from any of the supported subvariants, please contact the author of KOMA-Script to obtain support as soon as possible, and in the meanwhile create a customized lco file for your own use, using one of the existing ones as a template and reading the KOMA-Script documentation attentively.

If your window envelope subvariant is supported, this is how you would go about using it: simply select the required lco file and activate the horizontal and vertical foldmarks as required. Another, independent, mark is the punching mark which divides a sheet in two horizontally for easy punching and filing.

A.3.1. Example 1:

Your favourite envelope happens to be a You 3 with window subvariant Mutoh 3, left over from when the company had its previous name, and you do not wish them to go to waste. Thus, you write your letter with the following starting code placed before the letter environment:

```
\LoadLetterOption{NipponLL}\setkomavar{myref}{NipponLL}
```

```
\begin{letter}{Martina Muster\\Address}
...
\end{letter}
```

A.3.2. Example 2:

You originally designed your letter for a You 3 envelope, but suddenly you get handed a used electrical company envelope with cute manga characters on it which you simply cannot pass up. Surprisingly, you find it conforms fairly closely to the Chou 4 size and C window subvariant, such that you realize you can alter the following in your document preamble:

```
\LoadLetterOption{NipponEL}\setkomavar{myref}{NipponEL}
\begin{letter}{Martina Muster\\Address}
...
\end{letter}
```

Now, scrlltr2 automatically reformats the letter for you to fit the required envelope.

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.

scrartcl

v2.8p	57, 65, 78, 97, 99, 100, 113, 124, 164, 209, 228, 269, 287, 426
v2.8q	41, 70, 87, 88, 126, 132, 134
v2.96a	30, 53, 100, 149, 266
v2.97c	65, 72
v3.00	29, 30, 52, 54, 62, 68, 69, 70, 74, 77, 82, 83, 85, 86, 91, 127, 133, 134, 138, 139, 140, 148, 149, 150, 226, 265, 267, 286, 287, 379, 383, 425, 426
v3.01a	31, 53, 149, 266
v3.02	114
v3.05	125
v3.06	88, 133, 134, 135
v3.07	59, 89
v3.08	74, 75, 442, 443
v3.09	120, 121, 122, 125
v3.09a	125
v3.10	92, 93, 95, 109, 129
v3.12	58, 60, 61, 62, 63, 66, 106, 107, 138, 139, 167, 230, 270, 288, 427
v3.15	71, 445, 446, 454, 456, 460
v3.17	55, 103, 444, 445, 456, 457
v3.18	69, 71, 140, 141, 446
v3.19	445, 459
v3.20	132, 447, 448, 451

scrartl

v3.19	447
-------	-----

scrbase

v3.05a	329
v3.08	319
v3.12	310, 315, 318, 319, 320, 323, 324, 326
v3.15	311, 318
v3.18	312
v3.20	313

scrbook

v2.8o	104
-------	-----

v2.8p	57, 65, 78, 97, 99, 100, 108, 113, 124, 164, 209, 228, 269, 287, 426
v2.8q	41, 70, 87, 88, 126, 132, 134
v2.96a	30, 53, 91, 94, 100, 149, 266
v2.97c	65, 72
v2.97e	89
v3.00	29, 30, 52, 54, 62, 69, 70, 74, 77, 82, 83, 85, 86, 90, 91, 127, 133, 134, 138, 139, 140, 148, 149, 150, 226, 265, 267, 286, 287, 379, 383, 425, 426
v3.01a	31, 53, 149, 266
v3.02	114, 456
v3.05	125
v3.06	88, 133, 134, 135
v3.07	59, 89
v3.08	74, 75, 442, 443
v3.09	120, 121, 122, 125
v3.09a	125
v3.10	92, 93, 95, 109, 129
v3.12	58, 60, 61, 62, 63, 66, 106, 107, 138, 139, 167, 230, 270, 288, 427
v3.15	70, 72, 102, 445, 446, 454, 455, 456, 460
v3.17	55, 103, 444, 445, 456, 457
v3.18	69, 71, 91, 140, 141, 446
v3.19	445, 447, 457, 459
v3.20	132, 447, 448, 451
v3.21	443
scrdate	
v3.05a	252, 253, 254
v3.08b	255
v3.13	255
scrxtextd	
v3.00	29, 30, 52, 148, 149, 226, 265, 286, 287, 379, 383, 425, 426
v3.01a	31, 53, 149, 266
v3.02	284
v3.06	282
v3.07	282
v3.10	283
v3.12	58, 60, 61, 62, 167, 230, 270, 271, 273, 288, 427
scrjura	
v0.7	293
v0.9	298
scrlayer	
v3.16	387, 392, 395, 409, 415

v3.18	388, 389, 391, 393, 401
v3.19	391, 393, 394
scrlayer-scrpage	
v3.12	225, 413
v3.14	233, 237, 239, 249
scrletter	
v3.15	464, 482
v3.17	167, 182, 184, 185, 482
v3.19	154, 199
scrfile	
v2.96	337
v3.03	336
v3.08	339, 340
v3.09	333
v3.12	339, 340, 341
scrltr2	
v2.9i	146, 147
v2.9t	30, 53, 149, 266, 479
v2.95	154
v2.96	184
v2.97	215
v2.97c	170, 184, 185, 189, 193, 472, 473
v2.97d	476
v2.97e	168, 171, 195, 470, 471, 472, 479
v3.00	54, 150, 202, 204, 205, 206, 267
v3.01	477, 478
v3.01a	31, 53, 149, 266
v3.02	209, 486
v3.03	146, 182, 185, 186, 474, 475
v3.04	214, 484
v3.05	472, 479
v3.06	207
v3.07	166, 208
v3.08	143, 144, 196, 198, 203, 486
v3.09	189, 190, 486
v3.12	143, 166, 176, 177, 190
v3.14	215
v3.15	445
v3.17	167, 182, 184, 185
v3.19	154, 199

scrrcpt

v2.8o

104

v2.8p

57, 65, 78, 97, 99, 100, 108, 113, 124, 164, 209, 228, 269, 287, 426

v2.8q

41, 70, 87, 88, 126, 132, 134

v2.96a

30, 53, 91, 94, 100, 149, 266

v2.97c

65, 72

v3.00

29, 30, 52, 54, 62, 68, 69, 70, 74, 77, 82, 83, 85, 86, 90, 91, 127, 133, 134, 138, 139, 140, 148, 149, 150, 226, 265, 267, 286, 287, 379, 383, 425, 426

v3.01a

31, 53, 149, 266

v3.02

114, 456

v3.05

125

v3.06

88, 133, 134, 135

v3.07

59, 89

v3.08

74, 75, 442, 443

v3.09

120, 121, 122, 125

v3.09a

125

v3.10

92, 93, 95, 109, 129

v3.12

58, 60, 61, 62, 63, 66, 106, 107, 138, 139, 167, 230, 270, 288, 427

v3.15

70, 72, 102, 445, 446, 454, 455, 456, 460

v3.17

55, 103, 444, 445, 456, 457

v3.18

69, 71, 91, 140, 141, 446

v3.19

445, 447, 457, 459

v3.20

132, 447, 448, 451

v3.21

443

scrttime

v3.05a

258

tocbasic

v3.01

355

v3.06

373

v3.09

373

v3.10

355

v3.12

355, 370

v3.17

354

v3.20

355, 356, 357, 360, 365, 366, 367, 370, 373, 374, 375

v3.21

360, 364, 373, 374, 375

typearea

v3.00

29, 30, 31, 32, 34, 35, 38, 39, 40, 41, 42, 45, 52, 148, 149, 226, 265, 286, 287, 379, 383, 425, 426

v3.01b

30, 45, 53, 149, 266

v3.02c	45
v3.05a	48
v3.11	439
v3.12	39, 43
v3.17	47
v3.18	439
v3.22	45
v0.7	
scrjura	293
v0.9	
scrjura	298
v2.8o	
scrbook	104
scrreprt	104
v2.8p	
scrtctl	57, 65, 78, 97, 99, 100, 113, 124, 164, 209, 228, 269, 287, 426
scrbook	57, 65, 78, 97, 99, 100, 108, 113, 124, 164, 209, 228, 269, 287, 426
scrreprt	57, 65, 78, 97, 99, 100, 108, 113, 124, 164, 209, 228, 269, 287, 426
v2.8q	
scrtctl	41, 70, 87, 88, 126, 132, 134
scrbook	41, 70, 87, 88, 126, 132, 134
scrreprt	41, 70, 87, 88, 126, 132, 134
v2.9i	
scrlltr2	146, 147
v2.9t	
scrlltr2	30, 53, 149, 266, 479
v2.95	
scrlltr2	154
v2.96	
scrfile	337
scrlltr2	184
v2.96a	
scrtctl	30, 53, 100, 149, 266
scrbook	30, 53, 91, 94, 100, 149, 266
scrreprt	30, 53, 91, 94, 100, 149, 266
v2.97	
scrlltr2	215
v2.97c	
scrtctl	65, 72

scrbook	65, 72
scrlltr2	170, 184, 185, 189, 193, 472, 473
scrreprt	65, 72
v2.97d	
scrlltr2	476
v2.97e	
scrbook	89
scrlltr2	168, 171, 195, 470, 471, 472, 479
v3.00	
scartcl	29, 30, 52, 54, 62, 68, 69, 70, 74, 77, 82, 83, 85, 86, 91, 127, 133, 134, 138, 139, 140, 148, 149, 150, 226, 265, 267, 286, 287, 379, 383, 425, 426
scrbook	29, 30, 52, 54, 62, 69, 70, 74, 77, 82, 83, 85, 86, 90, 91, 127, 133, 134, 138, 139, 140, 148, 149, 150, 226, 265, 267, 286, 287, 379, 383, 425, 426
scrextend	29, 30, 52, 148, 149, 226, 265, 286, 287, 379, 383, 425, 426
scrlltr2	54, 150, 202, 204, 205, 206, 267
scrreprt	29, 30, 52, 54, 62, 68, 69, 70, 74, 77, 82, 83, 85, 86, 90, 91, 127, 133, 134, 138, 139, 140, 148, 149, 150, 226, 265, 267, 286, 287, 379, 383, 425, 426
typearea	29, 30, 31, 32, 34, 35, 38, 39, 40, 41, 42, 45, 52, 148, 149, 226, 265, 286, 287, 379, 383, 425, 426
v3.01	
scrlltr2	477, 478
tocbasic	355
v3.01a	
scartcl	31, 53, 149, 266
scrbook	31, 53, 149, 266
scrextend	31, 53, 149, 266
scrlltr2	31, 53, 149, 266
scrreprt	31, 53, 149, 266
v3.01b	
typearea	30, 45, 53, 149, 266
v3.02	
scartcl	114
scrbook	114, 456
scrextend	284
scrlltr2	209, 486
scrreprt	114, 456
v3.02c	
typearea	45
v3.03	
scrfile	336

scrlltr2	146, 182, 185, 186, 474, 475
v3.04	
scrlltr2	214, 484
v3.05	
scrartcl	125
scrbook	125
scrlltr2	472, 479
scrreprt	125
v3.05a	
scrbase	329
scrdate	252, 253, 254
scrtime	258
typearea	48
v3.06	
scrartcl	88, 133, 134, 135
scrbook	88, 133, 134, 135
scrextend	282
scrlltr2	207
scrreprt	88, 133, 134, 135
tocbasic	373
v3.07	
scrartcl	59, 89
scrbook	59, 89
scrextend	282
scrlltr2	166, 208
scrreprt	59, 89
v3.08	
scrartcl	74, 75, 442, 443
scrbase	319
scrbook	74, 75, 442, 443
scrfile	339, 340
scrlltr2	143, 144, 196, 198, 203, 486
scrreprt	74, 75, 442, 443
v3.08b	
scrdate	255
v3.09	
scrartcl	120, 121, 122, 125
scrbook	120, 121, 122, 125
scrfile	333
scrlltr2	189, 190, 486

scrreprt	120, 121, 122, 125
tocbasic	373
v3.09a	
scrartcl	125
scrbook	125
scrreprt	125
v3.10	
scrartcl	92, 93, 95, 109, 129
scrbook	92, 93, 95, 109, 129
scrextend	283
scrreprt	92, 93, 95, 109, 129
tocbasic	355
v3.11	
typearea	439
v3.12	
scrartcl	58, 60, 61, 62, 63, 66, 106, 107, 138, 139, 167, 230, 270, 288, 427
scrbase	310, 315, 318, 319, 320, 323, 324, 326
scrbook	58, 60, 61, 62, 63, 66, 106, 107, 138, 139, 167, 230, 270, 288, 427
scrextend	58, 60, 61, 62, 167, 230, 270, 271, 273, 288, 427
scrlayer-scrpage	225, 413
scrfile	339, 340, 341
scrlltr2	143, 166, 176, 177, 190
scrreprt	58, 60, 61, 62, 63, 66, 106, 107, 138, 139, 167, 230, 270, 288, 427
tocbasic	355, 370
typearea	39, 43
v3.13	
scrdate	255
v3.14	
scrlayer-scrpage	233, 237, 239, 249
scrlltr2	215
v3.15	
scrartcl	71, 445, 446, 454, 456, 460
scrbase	311, 318
scrbook	70, 72, 102, 445, 446, 454, 455, 456, 460
scrletter	464, 482
scrlltr2	445
scrreprt	70, 72, 102, 445, 446, 454, 455, 456, 460
v3.16	
scrlayer	387, 392, 395, 409, 415
v3.17	

scrartcl	55, 103, 444, 445, 456, 457
scrbook	55, 103, 444, 445, 456, 457
scrletter	167, 182, 184, 185, 482
scrlltr2	167, 182, 184, 185
scrreprt	55, 103, 444, 445, 456, 457
tocbasic	354
typearea	47
v3.18	
scrartcl	69, 71, 140, 141, 446
scrbase	312
scrbook	69, 71, 91, 140, 141, 446
scrlayer	388, 389, 391, 393, 401
scrreprt	69, 71, 91, 140, 141, 446
typearea	439
v3.19	
scrartcl	445, 459
scrartl	447
scrbook	445, 447, 457, 459
scrlayer	391, 393, 394
scrletter	154, 199
scrlltr2	154, 199
scrreprt	445, 447, 457, 459
v3.20	
scrartcl	132, 447, 448, 451
scrbase	313
scrbook	132, 447, 448, 451
scrreprt	132, 447, 448, 451
tocbasic	355, 356, 357, 360, 365, 366, 367, 370, 373, 374, 375
v3.21	
scrbook	443
scrreprt	443
tocbasic	360, 364, 373, 374, 375
v3.22	
typearea	45

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 <http://mirror.ctan.org/>. 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.

- [Ame02] American Mathematical Society:
*User's guide for the **amsmath** package*, February 2002.
[CTAN://macros/latex/required/amslatex/math/](http://ctan.org/macros/latex/required/amslatex/math/).
- [BB13] Johannes Braams and Javier Bezos:
Babel, December 2013.
[CTAN://macros/latex/required/babel/](http://ctan.org/macros/latex/required/babel/).
- [BCJ⁺05] Johannes Braams, David Carlisle, Alan Jeffrey, Leslie Lamport, Frank Mittelbach, Chris Rowley, and Rainer Schöpf:
The $\text{\LaTeX}2\epsilon$ Source, December 2005.
- [Car99a] David Carlisle:
*The **keyval** package*, March 1999.
[CTAN://macros/latex/required/graphics/](http://ctan.org/macros/latex/required/graphics/).
- [Car99b] David Carlisle:
*The **tabularx** package*, January 1999.
[CTAN://macros/latex/required/tools/](http://ctan.org/macros/latex/required/tools/).
- [Car04] David Carlisle:
*The **longtable** package*, February 2004.
[CTAN://macros/latex/required/tools/](http://ctan.org/macros/latex/required/tools/).
- [Car05] David P. Carlisle:
Packages in the ‘graphics’ bundle, November 2005.
[CTAN://macros/latex/required/graphics/](http://ctan.org/macros/latex/required/graphics/).
- [Che11] Florent Chervet:
tabu and longtabu, February 2011.
[CTAN://macros/latex/contrib/tabu/](http://ctan.org/macros/latex/contrib/tabu/).

- [Dal10] Patrick W. Daly:
Natural sciences citations and references, September 2010.
[CTAN://macros/latex/contrib/natbib/](http://ctan.org/macros/latex/contrib/natbib/).
- [DUD96] DUDEN:
Die deutsche Rechtschreibung. DUDENVERLAG, Mannheim, 21st edition, 1996.
- [Fai11] Robin Fairbairns:
footmisc — a portmanteau package for customising footnotes in L^AT_EX, June 2011.
[CTAN://macros/latex/contrib/footmisc/](http://ctan.org/macros/latex/contrib/footmisc/).
- [FAQ13] *Tex frequently asked questions on the web*, June 2013.
<http://www.tex.ac.uk/faq>.
- [Gau07] Bernard Gaulle:
Les distributions de fichiers de francisation pour latex, May 2007.
[CTAN://language/french/](http://ctan.org/language/french/).
- [Gre12] Enrico Gregorio:
The xpatch package, extending etoolbox patching commands, October 2012.
[CTAN://macros/latex/contrib/xpatch/](http://ctan.org/macros/latex/contrib/xpatch/).
- [KDP] *KOMA-Script Homepage*.
<http://www.komascript.de>.
- [Keh97] Roger Kehr:
XINDY, A Flexible Indexing System, 1997.
- [Ker07] Dr. Uwe Kern:
Extending L^AT_EX's color facilities: the xcolor package, January 2007.
[CTAN://macros/latex/contrib/xcolor/](http://ctan.org/macros/latex/contrib/xcolor/).
- [Kie10] Axel Kielhorn:
adrconv, April 2010.
[CTAN://macros/latex/contrib/adrconv/](http://ctan.org/macros/latex/contrib/adrconv/).
- [Knu90] Donald E. Knuth:
The T_EXbook, volume A of *Computers and Typesetting*. Addison-Wesley Publishing Company, Reading, Mass., 19th edition, 1990.
- [Koh02] Markus Kohm:
Satzspiegelkonstruktionen im Vergleich. Die T_EXnische Komödie, 4:28–48, 2002.
DANTE e. V.
- [Koh03] Markus Kohm:
Moderne Briefe mit KOMA-Script. Die T_EXnische Komödie, 2:32–51, 2003.
DANTE e. V.

- [Koh14a] Markus Kohm:
KOMA-Script. Edition DANTE. Lehmanns Media, Berlin, 5th edition, 2014,
ISBN 978-3-86541-613-1.
- [Koh14b] Markus Kohm:
Creating more than one index using `splitidx` and `splitindex`, April 2014.
[CTAN://macros/latex/contrib/splitindex/](http://macros/latex/contrib/splitindex/).
- [Lam87] Leslie Lamport:
MakeIndex: An index processor for \LaTeX , February 1987.
[CTAN://indexing/makeindex/doc/makeindex.pdf](http://indexing/makeindex/doc/makeindex.pdf).
- [Lap08] Olga Lapko:
The floatrow package, August 2008.
[CTAN://macros/latex/contrib/floatrow/](http://macros/latex/contrib/floatrow/).
- [Leh11] Philipp Lehman:
The etoolbox package, January 2011.
[CTAN://macros/latex/contrib/etoolbox/](http://macros/latex/contrib/etoolbox/).
- [Lin01] Anselm Lingnau:
An improved environment for floats, November 2001.
[CTAN://macros/latex/contrib/float/](http://macros/latex/contrib/float/).
- [Mit11] Frank Mittelbach:
An environment for multicolumn output, June 2011.
[CTAN://macros/latex/required/tools/](http://macros/latex/required/tools/).
- [Nie15] Rolf Niepraschk:
The eso-pic package, July 2015.
[CTAN://macros/latex/contrib/eso-pic/](http://macros/latex/contrib/eso-pic/).
- [Obe09] Heiko Oberdiek:
The picture package, October 2009.
[CTAN://macros/latex/contrib/oberdiek/picture.dtx](http://macros/latex/contrib/oberdiek/picture.dtx).
- [Obe10] Heiko Oberdiek:
The engord package, March 2010.
[CTAN://macros/latex/contrib/oberdiek/](http://macros/latex/contrib/oberdiek/).
- [OPHS11] Tobias Oetker, Hubert Partl, Irene Hyna, and Elisabeth Schlegl:
The Not So Short Introduction to \LaTeX 2 ϵ , April 2011.
[CTAN://info/lshort/english/](http://info/lshort/english/).
- [Pac] Jean Marie Pacquet:
KomaLetter2; Example by Jean-Marie Pacquet (French style). Wiki.
<http://wiki.lyx.org/Examples/KomaLetter2#toc6>.

- [Rai98a] Bernd Raichle:
german package, July 1998.
[CTAN://language/german/](http://ctan.org/language/german/).
- [Rai98b] Bernd Raichle:
ngerman package, July 1998.
[CTAN://language/german/](http://ctan.org/language/german/).
- [Sch09] Martin Schröder:
The ragged2e package, June 2009.
[CTAN://macros/latex/contrib/ms/](http://ctan.org/macros/latex/contrib/ms/).
- [Sch13] R Schlicht:
The microtype package: An interface to the micro-typographic extensions of pdfTEX, May 2013.
[CTAN://macros/latex/contrib/microtype/](http://ctan.org/macros/latex/contrib/microtype/).
- [Tea98] The $\mathcal{N}\mathcal{T}\mathcal{S}$ Team:
The ε -TEX manual, February 1998.
[CTAN://systems/e-tex/v2/doc/etex_man.pdf](http://ctan.org/systems/e-tex/v2/doc/etex_man.pdf).
- [Tea05a] L^AT_EX3 Project Team:
L^AT_EX 2_ε font selection, November 2005.
[CTAN://macros/latex/doc/fntguide.pdf](http://ctan.org/macros/latex/doc/fntguide.pdf).
- [Tea05b] L^AT_EX3 Project Team:
L^AT_EX 2_ε for authors, November 2005.
[CTAN://macros/latex/doc/usrguide.pdf](http://ctan.org/macros/latex/doc/usrguide.pdf).
- [Tea06] L^AT_EX3 Project Team:
L^AT_EX 2_ε for class and package writers, February 2006.
[CTAN://macros/latex/doc/clsguide.pdf](http://ctan.org/macros/latex/doc/clsguide.pdf).
- [TF11] Geoffrey Tobin and Robin Fairbairns:
setspace L^AT_EX package, December 2011.
[CTAN://macros/latex/contrib/setspace/](http://ctan.org/macros/latex/contrib/setspace/).
- [Tsc87] Jan Tschichold:
Ausgewählte Aufsätze über Fragen der Gestalt des Buches und der Typographie.
Birkhäuser Verlag, Basel, 2nd edition, 1987.
- [Ume10] Hideo Umeke:
The geometry package, September 2010.
[CTAN://macros/latex/contrib/geometry/](http://ctan.org/macros/latex/contrib/geometry/).
- [vO04] Piet van Oostrum:
Page layout in L^AT_EX, March 2004.
[CTAN://macros/latex/contrib/fancyhdr/](http://ctan.org/macros/latex/contrib/fancyhdr/).

- [WF00] Hans Peter Willberg and Friedrich Forssman:
Erste Hilfe in Typografie. Verlag Hermann Schmidt, Mainz, 2000.
- [Wik] Wiki:
Deutsche T_EX-FAQ.
<http://projekte.dante.de/DanteFAQ/WebHome>.

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

@everystyle@ (page style)	400	center mark	see punch hole mark
A		chapter	95
abstract	68–69	heading	104
address	see also addressee	number	103
database	263	page style	79
file	220–224, 259, 263	preamble	107
list	223	start	90
address field	155	title	91
addressee	153, 182–186, 473–475	circular letters	220–224
additional	159	citations	116, 210
adjustment		class	
vertical	55	→ Index of Files etc.	532
appendix	91, 104, 137	clause	288
author	65, 273	closing	156, 212–213, 478
auxiliary file	346	phrase	478
B		CM fonts	99
back matter	89	command	
bank account	196	→ Index of Commands etc.	518
bibliography	89, 138, 137–140	Compatibility	30–31, 53, 149–150, 266
BiBTeX	138	contents	
binding	25	table of	99
binding correction	25, 26, 27, 31	contract	286, 289–294
boxed (float style)	130	counter	
business line .. see reference line, 476–477, 479, 480		→ Index of Commands etc.	518
C		→ Index of Lengths etc.	529
caption		cover	63, 67, 271, 273
of figure	123	D	
of table	123	date	65, 189, 273, 488
carbon copy	159	day	
cell phone	176	of the week	252
cellphone	176	dedication	67, 274
		delimiter	159
		dictum	109–110, 282–283
		dispatch type	185

distribution list	159	font	56–62, 97–99, 163–167, 227–230, 268–270, 287–288, 426–427
document		size	55–56, 97–99, 161–163, 267–268
title	62–68, 270–274	style	78–79, 97–99, 113, 114, 124, 193, 202, 209, 284
document structure	91	foot	
double-sided	25, 78	width	249
draft mode	54, 150, 267	footer	
DVI	47	of letter	478
		of note paper	478
E		footnotes	65, 84–89, 205–208, 273, 278–282
e-mail	176	foreword	89
EC fonts	99	formulas	119, 211
element		front matter	89
→ Index of Elements	530		
empty (page style)	77–79, 83, 200, 204, 277, 400, 427, 431	G	
environment		gutter	25, 26
→ Index of Commands etc.	518		
equation		H	
alignment	119	half-title	63, 271
number	89, 119	head	
equations	119, 211	width	249
excerpt	109–110, 282–283	header	99
		heading	100, 105, 108
F		headings (page style)	77–79, 107, 144, 200, 203, 230, 241, 410, 415
fax	176	hook	154
figure	119		
number	89	I	
figures	119–136	identification	
list of	89, 133–136	code	189
file		indentation	74
extension	346–377	index	89, 140–141
file		page style	79
→ Index of Files etc.	532	instruction	
float styles		→ Index of Commands etc.	518
boxed	130	interleaf page	81–84, 203–205, 275–278
komaabove	129–130		
komabelow	129–130	K	
plain	130	komaabove (float style)	129–130
ruled	130	komabelow (float style)	129–130
floating environments	119		
floats	119–136	L	
folding mark	168	language	486–489
folding marks	155, 470–472	Croatian	486
foldmark	168	definition	325–328
following page	168		

dependent terms see language definition
Dutch	486
English	486
Finnish	486
French	486
German	486
Italian	486
Norsk	486
Spanish	486
Swedish	486
layer	382–412, 428
lco-file	.. 146, 147, 214–220, 468, 472, 473, 475, 476, 477, 478, 479, 482–486
lco-files	474
leading	26, 36
length	→ Index of Commands etc. 518 → Index of Lengths etc. 529
letter	
class option	214–220, 482
closing	212–213
first page	167–197
foot	195–197
footer	478–479
head	155, 171, 172–175, 176–182
Japanese	491
opening	155
signature	212–213
structure	151–161
letter (page style)	199, 481
letter class option	214–220, 482
letterfoot	195–197
letterhead	.. 171, 172–175, 176–182, 472–473
letters	142–224
line	
separator	200
line length	27
list	
of contents	346–377
list of	
figures	89, 133–136
tables	133–136
list of contents	emph see tble of contents356
lists	111–119, 208–211, 283–285

LM fonts	99
logical markup	56, 163, 227, 268, 287, 426
Logo	180
M	
macro	→ Index of Commands etc. 518
main matter	89
margin	26, 26, 118, 210, 284
notes	136–137, 211–212, 285
margins	40
markright	202
marks	
folding	see folding marks
markup	56, 163, 227, 268, 287, 426
mathematics	119, 211
matter	
back	89
front	89
main	89
mobile phone	176
mode of dispatch	185
myheadings (page style)	.. 79, 78–79, 144, 202, 202, 203, 241
N	
note	
columns	424–436
note paper	167–197
footer	see letter footer
numbering	99, 106–107, 112, 209
O	
option	215
option	→ Index of Options 533
options	29–30, 51–53, 148–149, 225–227, 264–265, 286–287, 378–379, 382–383, 425–426
P	
package	→ Index of Files etc. 532
page	26
break	432
counter	81

even 76, 198–199, 274–275

following 168

foot 199–203

 height 227, 404–405

footer 155

head 199–203

 height 227, 404–405

interleaf 90

layout 54–55, 151

number 81

odd 76, 198–199, 274–275

style 76–84, 199–205, 240–249, 275–278, 405–411, 413–416, 481–482

page footer 40

page header 40

page layout 25, 29

page styles

 @everystyle@ 400

 empty . 77–79, 83, 200, 204, 277, 400, 427, 431

 headings .. 77–79, 107, 144, 200, 203, 230, 241, 410, 415

 letter 199, 481

 myheadings . 79, 78–79, 144, 202, 202, 203, 241

 plain .. 78–79, 83, 202, 204, 230, 275, 276

 plain.letter 199, 481

 plain.scrheadings 230–231, 239

 scrheadings 202, 230–231, 239, 481

 scrplain 231

pagination 40

paper 26

 format 45–47

 orientation 45

 size

 limitation 482–483

paper format 25

paragraph 73

 markup 73–76, 197–198

 number 292–294

 omission 293

 spacing 74

part 95

 number 103

 page style 79

 preamble 107

PDF 47

phone 176

plain (float style) 130

plain (page style) ... 78–79, 83, 202, 204, 230, 275, 276

plain.letter (page style) 199, 481

plain.scrheadings (page style) . 230–231, 239

poems 115, 210

PostScript 47

pseudo-length

 → Index of Lengths etc. 529

pseudo-lengths 147–148, 464–479

publisher 65, 273

puncher hole mark 470

R

reference

 field line *see* business line

 line *see* business line

reference line 155, 189, 189–192

return address 475

rule

 separator 200

ruled (float style) 130

running head 105

running headings 40

running heads

 automatic 241

 manual 241

 static 241

S

scrheadings (page style) ... 202, 230–231, 239, 481

scrplain (page style) 231

section 95, *see also* clause

 number 89, 103

sender

 additional information 187–188

 extension *see* sender’s extension

sender’s extension 171, 475–476

sentence

 number 294

separator 159, 193

serial letters *see* circular letters220

serifs	26
signature	212–213, 478
smart slogan	109–110, 282–283
subject	65, 155, 193–195, 273, 477–478
subscript	56, 164, 268
summary	68, 68–69
superscript	56, 164, 268
synchronisation	433

T

table	119
caption	123
number	89
of contents	99
table of contents	69–73, 80, 89, 346–377
entry	356–367, 374, 375
tables	119–136
list of	89, 133–136
telephone	176
telephone list	223
terms	
language-dependent .	<i>see</i> language definition

text	
markup	56–62, 163–167, 227–230, 268–270, 287–288, 426–427
subscript	56, 164, 268
superscript	56, 164, 268
text area	28
textblock height	26
time	257
title	62–68, 91, 192–193, 270–274
back	67, 274
flipside	67, 274
head	65, 273
in-page	62, 69, 271
page style	79, 275
pages	62, 271
rear side	67, 274
TOC	346, 350
TOC-file	346, 371
type area	40, 44

V

variables	142–147, 479–481
-----------------	------------------

Index of Commands, Environments, and Variables

\@addtoplength	214, 470
\@currentt	348, 350, 351, 353, 368, 369
@everystyle@	
→ General Index	514
\@firstofone	353
\@fontsizefilebase	443, 443
\@mkboth	410–411, 416
\@mkdouble	410–411, 416
\@mkleft	410–411, 416
\@mkright	410–411, 416
\@newplength	214, 468
\@openbib@code	462–463
\@ptsize	380
\@setplength	214, 470
\@startsection	460
\@starttoc	342, 343, 351, 368
\@tempskipa	
→ Index of Lengths etc.	529
\@writefile	342

A

abstract (environment)	69, 108
\abstractname	327
\activateareas	438, 445
\addchap	69, 100, 105, 446
\addchap*	100
\addchapmark	105–106
\addchaptocentry	443
\addcontentsline	365, 442
\addcontentslinetoeachtocfile	351
\AddLayersAtBeginOfPageStyle	402
\AddLayersAtEndOfPageStyle	402
\AddLayersToPageStyle	402
\AddLayersToPageStyleAfterLayer	402
\AddLayersToPageStyleBeforeLayer	402
addmargin (environment) ..	118–119, 210–211, 284–285, 432
addmargin* (environment) ..	118–119, 210–211, 284–285
\addparagraphtocentry	443

\addpart	100, 446	\AfterSelectAnyPageStyle	397
\addpart*	100	\AfterSettingArea	439
\addparttocentry	443	\AfterSettingArea*	439
\addrchar	223–224, 259–260	\AfterStartingTOC	353
\addrentry	221, 259–260	\AfterTOCHead	37, 353
\Address	260	\aliaskomafont	444
addresseeimage (variable)	142, 182–186	\alsoname	327
\addsec	100, 105	\and	65–67, 272–274
\addsec*	100	\appendix	137
\addsecmark	105–106	\appendixmore	461–462
\addsectiontocentry	443	\appendixname	327
\addsubparagraphtocentry	443	\areaset	44–45, 438
\addsubsectiontocentry	443	\At@startsection	460–461
\addsubsubsectiontocentry	443	\AtAddToTocList	348
\addtocentrydefault	442, 443	\AtBeginDocument	154
\addtoeachtocfile	350	\AtBeginLetter	154
\addtokomafont	57–61, 97, 164–167, 228–229, 269, 287–288, 426–427	\AtEndBibliography	140, 463
\addtokomafontgobblelist	444–445	\AtEndLetter	154
\addtokomafontrelaxlist	444–445	\AtEndOfClass	154
\AddToLayerPageStyleOptions	403	\AtEndOfFamilyOptions	315
\addtolengthlength	147–148	\author	65–67, 272–274
\addtoeffields	480	\autodot	103–104
\addtotoclist	347–348	\automark	241–243, 405–406
\addxcontentsline	350–351, 365	\automark*	241–243, 405–406
\addxcontentslinetoeachtocfile	351	B	
\adrchar	223–224, 259–260	backaddress (variable)	142, 182–186
\adrentry	220–221, 259–260	backaddresseparator (variable)	142, 182–186
\AfterAtEndOfClass	333–335	\backmatter	89
\AfterAtEndOfPackage	333–335	\bankname	489
\AfterBibliographyPreamble	140, 463	\Before@sect	460–461
\AfterCalculatingTypearea	439	\Before@ssect	460–461
\AfterCalculatingTypearea*	439	\BeforeClass	332
\AfterClass	333–335	\BeforeClosingMainAux	335–336
\AfterClass!	333–335	\BeforeFamilyProcessOptions	312
\AfterClass*	333–335	\BeforeFile	332
\AfterClass+	333–335	\BeforePackage	332
\AfterFile	332	\BeforeRestoreareas	438–439
\AfterPackage	333–335	\BeforeRestoreareas*	438–439
\AfterPackage!	333–335	\BeforeSelectAnyPageStyle	397
\AfterPackage*	333–335	\BeforeStartingTOC	353
\AfterPackage+	333–335	\BeforeTOCHead	353
\AfterReadingMainAux	335–336	\bib@beginhook	462–463
\AfterRestoreareas	438–439	\bib@endhook	462–463
\AfterRestoreareas*	438–439	\bibname	327

\bigskip	110, 115, 139, 210	\chapter	95–99, 106, 446, 455
\blinddocument	245	\chapter*	69, 99
boxed		\chapterformat	93, 103–104
→ General Index	514	\chapterheadendvskip	455–457
\BreakBibliography	139–140	\chapterheadmidvskip	455–457
C			
\caption	120, 123–124, 373, 376	\chapterheadstartvskip	455–457
\captionabove	123–124	\chapterlinesformat	457–459
\captionaboveof	125–126	\chapterlineswithprefixformat	457–459
\captionbelow	123–124	\chaptermark	105–106, 107, 247, 408
\captionbelowof	125–126	\chaptermarkformat .	105–106, 246, 408, 414
captionbeside (environment) ...	121, 126–129	\chaptername	327
\captionformat	130	\chapternumdepth	106–107
\captionof	125–126	\chapterpagestyle	79–81
captionofbeside (environment)	129	\chead	238–239
\captionsofbeside (environment)	129	\chead*	239
\captionsofamerican	487	\ClassInfoNoLine	329
\captionsofaustrian	487	\ClassName	441–442
\captionsofbritish	487	\Clause	290–291
\captionsofcroatian	487	\Clauseformat	291
\captionsofdutch	487	\cleardoubleemptypage	83–84, 204–205, 276–278
\captionsofenglish	487	\cleardoubleevenemptypage	83–84, 204–205, 276–278
\captionsoffinnish	487	\cleardoubleevenpage	83–84, 204–205, 276–278
\captionsoffrench	487	\cleardoubleevenpageusingstyle	83–84, 204–205, 276–278
\captionsofgerman	487	\cleardoubleevenplainpage	83–84, 204–205, 276–278
\captionsofitalian	487	\cleardoubleevenstandardpage	83–84, 204–205, 276–278
\captionsofnaustrian	487	\cleardoubleoddemptypage .	83–84, 204–205, 276–278
\captionsofngerman	487	\cleardoubleoddpage	83–84, 204–205, 276–278
\captionsofnsorsk	487	\cleardoubleoddpageusingstyle	83–84, 204–205, 276–278
\captionsofspanish	487	\cleardoubleoddplainpage .	83–84, 204–205, 276–278
\captionsofswedish	487	\cleardoubleoddstandardpage	83–84, 204–205, 276–278
\captionsofUKenglish	487	\cleardoublepage .	83–84, 204–205, 276–278
\captionsofUSenglish	487	\cleardoublepageusingstyle	83–84, 204–205, 276–278
\cc	159	\cleardoubleplainpage	83–84, 204–205, 276–278
\ccname	327, 489		
ccseparator (variable)	143, 159		
\cefoot	234–236		
\cefoot*	237		
\cehead	231–233		
\cehead*	233–234		
\CenturyPart	252		
\cfoot	238–239, 246		
\cfoot*	239		
\changefontsize	443		
\chapapp	104–105		
\chapappifchapterprefix	104–105		

D	
\date	65–67, 255, 272–274
date (variable)	143, 189
\dateamerican	488
\dateaustrian	488
\datebritish	488
\datecroatian	488
\datedutch	488
\dateenglish	488
\datefinnish	488
\datefrench	488
\dategerman	488
\dateitalian	488
\datename	489
\datenaustrian	488
\datengerman	488

D

\datenorsk	488
\datespanish	488
\dateswedish	488
\dateUKenglish	488
\dateUSenglish	488
\day	254
\DayName	253–254, 254
\DayNameByNumber	253–254, 25
\DayNumber	252–253
\DecadePart	252
\DeclareLayer	386–393
\DeclareNewJuraEnvironment	298
\DeclareNewLayer	386–393
\DeclareNewNoteColumn	428–431
\DeclareNewPageStyleAlias	397
\DeclareNewPageStyleByLayers	395, 398–400
\DeclareNewSectionCommand	446–454
\DeclareNewSectionCommands	454
\DeclareNewTOC	366, 373–377
\DeclareNoteColumn	428–431
\DeclarePageStyleAlias	397
\DeclarePageStyleByLayers	398–400, 422
\DeclareSectionCommand ...	446–454, 456, 457
\DeclareSectionCommands	454
\DeclareSectionNumberDepth	384–385
\DeclareTOCEntryStyle	365–366
\DeclareTOCStyleEntry	357–364, 373
\dedication	67–68, 274
\defaulttreffields	480
\defcaptionname	325–328
\defcaptionname*	325–328
\deffootnote	87–88, 207, 280–282
\deffootnotemark	87–88, 207, 280–282
\DefineFamily	309–310
\DefineFamilyKey	310–311
\DefineFamilyMember	309–310
\DefineTOCEntryBooleanOption	365–366
\DefineTOCEntryCommandOption	365–366
\DefineTOCEntryIfOption	365–366
\DefineTOCEntryLengthOption	365–366
\DefineTOCEntryNumberOption	365–366
\DefineTOCEntryOption	365–366
\defpagestyle	421–423
\defpairofpagestyles	417
\deftocheading	354

<code>\deftriplepagestyle</code>	419–421	<code>\FamilyNumericalKey</code>	316–318
<code>\deftripstyle</code>	421	<code>\FamilyOption</code>	314–315
<code>description</code> (environment)	113–114, 209	<code>\FamilyOptions</code>	313–314
<code>\DestroyLayer</code>	395	<code>\FamilyProcessOptions</code>	311–312
<code>\DestroyPageStyleAlias</code>	398	<code>\FamilySetBool</code>	315–316
<code>\DestroyRealLayerPageStyle</code>	404	<code>\FamilySetCounter</code>	318
<code>\dictum</code>	108, 109–110, 283	<code>\FamilySetCounterMacro</code>	318
<code>\dictumauthorformat</code>	109–110, 283	<code>\FamilySetLength</code>	318–319
<code>\dictumrule</code>	109–110, 283	<code>\FamilySetLengthMacro</code>	318–319
<code>\dictumwidth</code>	109–110, 283	<code>\FamilySetNumerical</code>	316–318
<code>displaymath</code> (environment)	119, 211	<code>\FamilyStringKey</code>	319
<code>\documentclass</code>	29–30, 52, 148–149, 226, 265, 286–287, 379, 383, 425	<code>\FamilyUnknownKeyValue</code>	319–320
<code>\doforeachtocfile</code>	349	<code>\faxname</code>	489
E		<code>faxseparator</code> (variable)	143, 176–180
<code>\edgesize</code>	484	<code>figure</code> (environment)	131
<code>\ellipsispar</code>	293–294	<code>\figureformat</code>	130–131
<code>\emailname</code>	489	<code>\figurename</code>	327
<code>emailseparator</code> (variable)	143, 176–180	<code>\firstfoot</code>	196
<code>empty</code>		<code>firstfoot</code> (variable)	143, 196–197
→ General Index	514	<code>\firsthead</code>	182
<code>\encl</code>	159–161	<code>firsthead</code> (variable)	143, 182
<code>\enclname</code>	327, 489	<code>\FirstName</code>	260
<code>enclseparator</code> (variable)	143, 159–161	<code>\float@addtolists</code>	380
<code>\enlargethispage</code>	170, 479	<code>\float@listhead</code>	380
<code>enumerate</code> (environment)	112–113, 209	<code>\flushbottom</code>	27, 54–55, 75
<code>eqnarray</code> (environment)	119, 211	<code>foot</code> (variable)	250
<code>equation</code> (environment)	119, 211	<code>footbotline</code> (variable)	250
<code>\evensidemargin</code>		<code>\footheight</code>	
→ Index of Lengths etc.	529	→ Index of Lengths etc.	529
<code>\extratitle</code>	64–65, 272	<code>\footnote</code> .	85, 85–86, 205, 206, 278, 279–280
F		<code>\footnotemark</code>	85, 85–86, 205, 206, 278, 279–280
<code>\FamilyBoolKey</code>	315–316	<code>\footnotetext</code>	85–86, 206, 279–280
<code>\FamilyCounterKey</code>	318	<code>\footref</code>	86–87, 206, 280
<code>\FamilyCounterMacroKey</code>	318	<code>footsepline</code> (variable)	250
<code>\FamilyElseValues</code>	320–321	<code>\footskip</code>	
<code>\FamilyExecuteOptions</code>	313	→ Index of Lengths etc.	529
<code>\FamilyKeyState</code>	310–311	<code>\ForEachLayerOfPageStyle</code>	401–402
<code>\FamilyKeyStateNeedValue</code>	310–311	<code>\ForEachLayerOfPageStyle*</code>	401–402
<code>\FamilyKeyStateProcessed</code>	310–311	<code>\FreeI</code>	260
<code>\FamilyKeyStateUnknown</code>	310–311	<code>\FreeII</code>	260
<code>\FamilyKeyStateUnknownValue</code>	310–311	<code>\FreeIII</code>	260
<code>\FamilyLengthKey</code>	318–319	<code>\FreeIV</code>	260
<code>\FamilyLengthMacroKey</code>	318–319	<code>fromaddress</code> (variable)	143, 172–175
		<code>frombank</code> (variable)	143, 196–197

fromemail (variable)	143, 176–180	\ifisnumexpr	324
fromfax (variable)	143, 176–180	\ifisskip	323
fromlogo (variable)	143, 180–182	\ifkomavar	146
frommobilephone (variable)	143, 176–180	\ifkomavareempty	147, 481
fromname (variable)	144, 172–175, 177, 200	\ifkomavareempty*	147, 481
fromphone (variable)	144, 176–180	\IfLayerAtPageStyle	404
fromurl (variable)	144, 176–180	\IfLayerExists	395
fromzipcode (variable)	144, 182–186	\IfLayerPageStyleExists	403
\frontmatter	89	\IfLayersAtPageStyle	404
G			
\g@addto@macro	369	\ifnotundefined	322
\GenericMarkFormat	409, 413–414	\ifnumber	324
\GetLayerContents	395	\ifnumbered	107
\GetRealPageStyle	398	\ifoot	238–239
\glossaryname	327	\ifoot*	239
H			
head (variable)	250	\ifpdfoutput	322
\headfromname	489	\ifpdftex	321
\headheight		\ifpsoutput	322
→ Index of Lengths etc.	529	\IfRealLayerPageStyleExists	403
headings		\IfSomeLayerAtPageStyle	404
→ General Index	514	\ifstr	322
\headmark	244–246, 407	\ifstrstart	323
headsepline (variable)	250	\ifthispageodd	76, 198–199, 274–275
\headtoname	327, 489	\iftocfeature	356
headtopline (variable)	250	\ifundefinedorrelax	321
I			
\if@atdocument	325	\ifunnumbered	107
\if@chapter	384	\IfUsePrefixLine	103–104
\if@mainmatter	384	\ifVTeX	322
\ifattoclist	346–347	\ihead	238–239, 245
\IfChapterUsesPrefixLine	91	\ihead*	239
\ifdimen	325	\indexname	327
\ifdvioutput	322	\indexpagestyle	79–81
\IfExistskomafont	445	\InputAddressFile	259–260
\ifiscount	324	invoice (variable)	144, 189–190
\ifiscounter	324	\invoicename	489
\ifisdimen	323	\ISODayName	253–254, 254
\ifisdimension	323	\ISODayNumber	252–253
\ifisdimexpr	323	\ISOToday	254, 255
\ifisglue	323	\IsoToday	254, 255
\ifisglueexpr	324	\item	111–115, 208, 209–210, 284
\ifisinteger	324	itemize (environment)	111–112, 208
K			
komaabove			
→ General Index	514		
komabelow			
→ General Index	514		

\KOMAClassName	441–442	letter (environment)	153
\KOMAoption ..	30, 52–53, 149, 226–227, 265, 287, 379, 383, 426	\letterlastpage	154–155
\KOMAoptions ..	30, 52–53, 149, 226–227, 265, 287, 373, 379, 383, 426	\LetterOptionNeedsPapersize	483
\KOMAScript	328, 441	\letterpagemark	482
\KOMAScriptVersion	328	\letterpagestyle	199–200
L			
\l@addto@macro	329	\linespread	26, 36
\label	86, 155, 206, 280, 295	\lipsum	232, 242
\labelenumi	112–113, 209	\listfigurename	327
\labelenumii	112–113, 209	\listofextensionname	351–352
\labelenumiii	112–113, 209	\listofeachtoc	351–352
\labelenumiv	112–113, 209	\listoffigures	136
labeling (environment) ...	114–115, 209–210, 284	\listoftables	136
\labelitemi	111–112, 208	\listoftoc	351–352
\labelitemii	111–112, 208	\listoftoc*	351–352
\labelitemiii	111–112, 208	\listtablename	327
\labelitemiv	111–112, 208	\LoadLetterOption	215–220, 482
landscape (environment)	381	\LoadLetterOptions	215–220, 482
\LastName	260	location (variable)	144, 187–188
\LastTOCLevelWasHigher	367	\lofoot	234–236
\LastTOCLevelWasLower	367	\lofoot*	237
\LastTOCLevelWasSame	367	\lohead	231–233
\layercontentsmeasure	396	\lohead*	233–234
\layerhalign	393	\lowertitleback	67, 274
\layerheight	393	M	
\layervalign	393	\mainmatter	89
\layerwidth	393	\makeatletter	370
\layerxoffset	393	\makeatother	370
\layeryoffset	393	\MakeLowercase	254
\lefoot	234–236	\MakeMarkcase	244, 353–354, 406, 407
\lefoot*	237	\makenote	432–433, 433
\leftbotmark	409–410, 415	\maketitle	63, 63–68, 271, 271–274
\leftfirstmark	409–410, 415	\MakeUppercase ...	244, 353, 375, 377, 407, 481
\leftmark ...	244–246, 247, 407, 408, 409, 415	\manualmark	241–243, 405–406
\lefttopmark	409–410, 415	\marginline	136–137, 212, 285
\lehead	231–233	\marginpar	136–137, 212, 285
\lehead*	233–234	marginpar (variable)	250
\LenToUnit	394	\markboth	78, 79, 202, 202, 202–203, 247–249, 408–409, 480
letter		\markleft	79, 202, 247–249, 408–409, 480
→ Index of Lengths etc.	529	\markright ...	78, 79, 202, 202–203, 247–249, 408–409, 410, 415, 480
letter		\maxdimen	
→ General Index	514	→ Index of Lengths etc.	529
		\mediaheight	
		→ Index of Lengths etc.	529

`\mediawidth`
→ Index of Lengths etc. 529

`\medskip` 115, 210

`\microtypesetup` 355

`\minisec` 100–101

`\mobilephonenumber` 489

`mobilephoneseparator` (variable) 176–180

`\ModifyLayer` 386–393

`\ModifyLayerPageStyleOptions` 403

`\month` 254

`\multfootsep` 85, 85–86, 205, 206, 278, 279–280

`\multiplefootnoteseparator` 85–86, 206, 279–280

`myheadings`
→ General Index 514

`myref` (variable) 144, 189–190

`\myrefname` 489

N

`\Name` 260

`\nameday` 255

`\newbibstyle` 138, 462–463

`\newblock` 138, 139, 462–463

`\newcaptionname` 325–328

`\newcaptionname*` 325–328

`\newcommand*` 439

`\newdaylanguage` 255–256

`\newkomafont` 444

`\newkomavar` 480

`\newkomavar*` 480

`\newpagestyle` 421–423

`\newpairofpagestyles` 417

`\newtriplepagestyle` 419–421

`\nextfoot` 203

`nextfoot` (variable) 144, 166, 202, 203, 481

`\nextthead` 203

`nexthead` (variable) 144, 202, 203, 481

`\nobreakspace` 86, 280

`\nofiles` 431

`\noindent` 69, 116, 210

`\nonumberline` 370

`\nopagebreak` 116

`\normalfont` 228, 229, 358

`\numberline` 357, 361

`\numexpr` 252, 329

O

`\ofoot` 238–239

`\ofoot*` 239

`\ohead` 238–239, 245

`\ohead*` 239

`\onecolumn` 355

`\opening` 153, 155, 200, 202, 215, 472, 477, 478

`\othersectionlevelsformat` 103–104

P

`\PackageInfoNoLine` 329

`page` (variable) 250

`\pagemark` 244–246, 407, 482

`\pagename` 328, 489

`\pagenumbering` 81

`\pageref` 155

`\pagestyle` 77–79, 200–202, 417

`paper` (variable) 250

`\paperheight`
→ Index of Lengths etc. 529

`\paperwidth`
→ Index of Lengths etc. 529

`par`
→ Index of Lengths etc. 529

`\paragraph` 95–99, 446

`\paragraph*` 99

`\paragraphformat` 103–104

`\paragraphmark` 247, 408

`\paragraphmarkformat` 246, 408

`\paragraphnumdepth` 106–107

`\parbox` 109, 283

`\parellipsis` 293–294

`\parformat` 293

`\parformatseparation` 293

`\parindent`
→ Index of Lengths etc. 529

`\parname` 299

`\parshortname` 299

`\parskip`
→ Index of Lengths etc. 529

`\part` 95–99, 106, 446

`\part*` 99

`\partformat` 103–104

`\partheademptypage` 455–457

`\partheadendvskip` 455–457

`\partheadmidvskip` 455–457

[illegible]

\rehead	231–233	→ General Index	514
\rehead*	233–234	\SecDef	460–461
\relax	443	\secdef	460
\RelaxFamilyKey	311	secnumdepth	
\removefromtoclist	349	→ Index of Lengths etc.	529
\RemoveLayersFromPageStyle	402	\section	95–99, 106, 446
\removereffields	480	\section*	99
\renewcaptionname	325–328	\sectioncatchphraseformat	459–460
\renewcaptionname*	325–328	\sectionformat	103–104
\renewcommand	463	\sectionlinesformat	459–460
\renewpagestyle	421–423	\sectionmark	105–106, 247, 408
\renewpairofpagestyles	417	\sectionmarkformat .	105–106, 246, 408, 414
\renewtriplepagestyle	419–421	\sectionnumdepth	106–107
\ReplaceClass	337–339	\seenname	328
\ReplaceInput	337	\selectfont	74, 198
\ReplacePackage	337–339	\Sentence	294
\RequirePackage	308, 339, 340	sentence	
\RequirePackageWithOptions	339	→ Index of Lengths etc.	529
\ResetPreventPackageFromLoading .	340–341	\sentencename	299
\rightbotmark	409–410, 415	\sentenceshortname	299
\rightfirstmark	409–410, 415	\setbibpreamble	138–139
\rightmark	244–246, 247, 407, 409, 415	\setcapdynwidth	132–133
\righttopmark	409–410, 415	\setcaphanging	131
\rofoot	234–236	\setcapindent	131
\rofoot*	237	\setcapindent*	131
\rohead	231–233	\setcapmargin	132–133
\rohead*	233–234	\setcapmargin*	132–133
ruled		\setcapwidth	132–133
→ General Index	514	\setchapterpreamble	107–108
S			
\S	291	\setfootbotline	251
\scr@ifdviooutput	322	\setfootnoterule	88–89, 207–208, 282
\scr@ifpdfoutput	322	\setfootsepline	251
\scr@ifpdftex	321	\setheadsepline	251
\scr@ifpsoutput	322	\setheadtopline	251
\scr@ifundefinedorrelax	321	\setindexpreamble	141
\scr@ifVTeX	322	\setkomafont .	57–61, 97, 164–167, 228–229, 269, 287–288, 426–427
\scr@startsection	460–461	\setkomavar	145–146
scrheadings		\setkomavar*	145–146
→ General Index	514	\setlengthtoplength	147–148
\sclayerAddCsToInterface	412	\setparsizes	445
\sclayerAddToInterface	412	\setpartpreamble	107–108
\sclayerInitInterface	411	\setshowstyle	484
\sclayerOnAutoRemoveInterface	412	\settime	257
scrplain		\setuptoc	354–356

<code>\showenvelope</code>	485–486	<code>\textsubscript</code>	56–57, 164, 268
<code>\showfields</code>	484	<code>\textsuperscript</code> ...	56, 56–57, 164, 164, 268, 268
<code>\showISOenvelope</code>	485–486	<code>textwithmarginpar</code> (variable)	250
<code>\showUScheck</code>	485–486	<code>\thanks</code>	65–67, 272–274
<code>\showUScommercial</code>	485–486	<code>\the</code>	252
<code>signature</code> (variable)	145, 212	<code>\theenumi</code>	112–113, 209
<code>specialmail</code> (variable)	145, 182–186	<code>\theenumii</code>	112–113, 209
<code>\storeareas</code>	438–439	<code>\theenumiii</code>	112–113, 209
<code>\StorePreventPackageFromLoading</code> .	340–341	<code>\theenumiv</code>	112–113, 209
<code>\SubClause</code>	290–291	<code>\thefootnotemark</code>	87–88, 207, 280–282
<code>\subject</code>	65–67, 272–274	<code>\thepar</code>	293
<code>subject</code> (variable)	145, 193–195, 200	<code>\thesentence</code>	294
<code>\subjectname</code>	489	<code>\thisletter</code>	154–155
<code>subjectseparator</code> (variable)	145, 193–195	<code>\thispagestyle</code>	77–79, 200–202
<code>\subparagraph</code>	95–99, 446	<code>\thistime</code>	257
<code>\subparagraph*</code>	99	<code>\thistime*</code>	257
<code>\subparagraphformat</code>	103–104	<code>\title</code>	65–67, 272–274
<code>\subparagraphmark</code>	247, 408	<code>title</code> (variable)	145, 192–193
<code>\subparagraphmarkformat</code>	246, 408	<code>\titlehead</code>	65–67, 272–274
<code>\subparagraphnumdepth</code>	106–107	<code>titlepage</code> (environment)	63, 271
<code>\subsection</code>	95–99, 106, 446	<code>\titlepagestyle</code>	79–81, 275
<code>\subsection*</code>	99	<code>toaddress</code> (variable)	145, 182–186
<code>\subsectionformat</code>	103–104	<code>\tocbasic@@after@hook</code>	369
<code>\subsectionmark</code>	105–106, 247, 408	<code>\tocbasic@@before@hook</code>	369
<code>\subsectionmarkformat</code> ..	105–106, 246, 408, 414	<code>\tocbasic@extension@after@hook</code>	369
<code>\subsectionnumdepth</code>	106–107	<code>\tocbasic@extension@before@hook</code>	369
<code>\subsubsection</code>	95–99, 106, 446	<code>\tocbasic@addxcontentsline</code>	370
<code>\subsubsection*</code>	99	<code>\tocbasic@DependOnPenaltyAndTOCLevel</code> .	370
<code>\subsubsectionformat</code>	103–104	<code>\tocbasic@extend@babel</code>	368
<code>\subsubsectionmark</code>	247, 408	<code>\tocbasic@listhead</code>	369
<code>\subsubsectionmarkformat</code>	246, 408	<code>\tocbasic@listhead@extension</code>	369
<code>\subsubsectionnumdepth</code>	106–107	<code>\tocbasic@SetPenaltyByTOCLevel</code>	370
<code>\subtitle</code>	65–67, 272–274	<code>\tocbasic@starttoc</code>	368
<code>\syncwithnotecolumn</code>	433–434	<code>\tocbasicautomode</code>	349
<code>\syncwithnotecolumns</code>	434	<code>\TOCclone</code>	343–344
T		<code>tocdepth</code>	
<code>\tableformat</code>	130–131	→ Index of Lengths etc.	529
<code>\tablename</code>	328	<code>\TOCEntryStyleInitCode</code>	366–367
<code>\tableofcontents</code>	72	<code>\TOCEntryStyleStartInitCode</code>	366–367
<code>\Telephone</code>	260	<code>\TOCLineLeaderFill</code>	367
<code>text</code> (variable)	250	<code>\today</code>	65, 190, 273
<code>\textellipsis</code>	294	<code>\today'sname</code>	254
<code>\textheight</code>		<code>\today'snumber</code>	254
→ Index of Lengths etc.	529	<code>toname</code> (variable)	145, 182–186

\topmargin		
→ Index of Lengths etc.	529	
\typearea	37–38	
U		
\UnifyLayersAtPageStyle	403	
\unitfactor	485–486	
\UnPreventPackageFromLoading	341	
\UnPreventPackageFromLoading*	341	
\UnReplaceClass	339	
\UnReplaceInput	339	
\UnReplacePackage	339	
\unsettoc	354–356	
\uppertitleback	67, 274	
urlseparator (variable)	176–180	
\useencodingofkomafont ..	62, 167, 230, 270, 288, 427, 444	
\usefamilyofkomafont	62, 167, 230, 270, 288, 427, 444	
\usefontofkomafont ..	62, 167, 230, 270, 288, 427, 444	
\usekomafont	57–61, 164–167, 228–229, 269, 287–288, 426–427, 444	
\usekomavar	146, 480–481	
\usekomavar*	146, 480–481	
\usepackage ..	29–30, 52, 148–149, 226, 265, 286–287, 339, 379, 383, 425	
\useplength	147	
\useseriesofkomafont	62, 167, 230, 270, 288, 427, 444	
\useshapeofkomafont	62, 167, 230, 270, 288, 427, 444	
\usesizeofkomafont ..	62, 167, 230, 270, 288, 427, 444	
\usetocbasicnumberline	357	
V		
verse (environment)	115–116, 210	
\vspace	433	
W		
\wwwname	489	
X		
\XdivY	329–330	
\XmodY	329–330	
Y		
\year	254	
yourmail (variable)	145, 189–190	
\yourmailname	489	
yourref (variable)	145, 189–190	
\yourrefname	489	
Z		
zipcodeseparator (variable)	145, 182–186	

Index of Lengths and Counters

\@tempskipa (length)	456, 457	
B		
backaddrheight	465, 475	
bfoldmarklength	465, 471	
bfoldmarkvpos	465, 470–471	
E		
\evensidemargin (length)	63, 271	
F		
firstfoothpos	465, 479	
firstfootvpos	465, 479	
firstfootwidth	465, 479	
firsttheadhpos	465, 472	
firsttheadvpos	465, 472	
firsttheadwidth	465, 473, 479	
foldmarkhpos	466, 471	
foldmarkthickness	472	
foldmarkvpos	466, 472	
\footheight (length)	43–44, 227, 405	
\footskip (length)	227, 479	
fromrulethickness	466, 473	
fromrulewidth	172, 466, 473	
H		
\headheight (length)	227, 405	
L		
letter (counter)	154–155	

lfoldmarkhpos	466, 471	refhpos	467, 477
lfoldmarklength	466, 471	refvpos	467, 476
locheight	466, 476	refwidth	467, 477
lochpos	466, 476		
locvpos	466, 476		
locwidth	466, 476		
M		S	
\maxdimen (length)	472, 479	secnumdepth (counter)	73, 106–107
\mediaheight (length)	48	sentence (counter)	294, 298
\mediawidth (length)	48	sigbeforevskip	212, 467, 478
mfoldmarklength	467, 471	sigindent	212, 467, 478
mfoldmarkvpos	467, 470–471	specialmailindent	467, 475
		specialmailrightindent	467, 475
		subjectaftervskip	467, 478
		subjectbeforevskip	467, 478
		subjectvpos	468, 477–478
P		T	
\paperheight (length)	479	\textheight (length)	381
\paperwidth (length)	472, 479	tfoldmarklength	468, 471
par (counter)	293	tfoldmarkvpos	468, 470–471
\parindent (length)	368	toaddrheight	468, 474
\parskip (length)	368	toaddrhpos	217, 468, 473
\pdfpageheight (length)	48, 432	toaddrindent	468, 474
\pdfpagewidth (length)	48	toaddrvpos	468, 473
pfoldmarklength	467, 471	toaddrwidth	468, 474
PPdatamatrixvskip	475	tocdepth (counter) .	72–73, 288, 357, 358, 362, 365
PPheadheight	475	\topmargin (length)	63, 271
PPheadwidth	475		
R			
refaftervskip	467, 477		

Index of Elements with Capability of Font Adjustment

		A		Clause	291
		addressee	165, 182–184, 185	contract.Clause	291
		author	58, 65, 273		
				D	
		B		date	58, 65, 273
		backaddress	165, 185, 184–185	dedication	58, 67, 274
				descriptionlabel	58, 113–114, 165, 209
		C		dictum	58, 109–110, 283
		caption	58, 124	dictumauthor	59, 110, 283
		captionlabel	58, 124	dictumtext	59
		chapter	58, 91, 98, 97–99	disposition	59, 98, 99, 97–99, 100, 102
		chapterentry	58, 72		
		chapterentrydots	72	F	
		chapterentrypagenumber	58, 72	field	484
		chapterprefix	58, 91	foldmark	165, 170

footbotline 228, 249–251
footnote 59, 87–88, 165, 207, 281–282
footnotelabel 59, 87–88, 165, 207, 281–282
footnotereference 59, 88, 165, 207, 281–282
footnoterule 59, 88–89, 166, 208, 282
footsepline 228, 249–251
fromaddress 173, 173
fromname 173, 173
fromrule 173, 173

H

headsepline 229, 249–251
headtopline 229, 249–251

L

labelinglabel .. 59, 114–115, 166, 209–210, 284
labelingseparator 59, 114–115, 166, 209–210, 284
letter 485
lettersubject 167, 193
lettertitle 167, 192

M

measure 485
minisec 59, 100

N

notecolumn.note column name 428
notecolumn.marginpar 428

P

pagefoot .. 59, 78, 78–79, 166, 202, 228, 229, 234
pagefoothead 78
pagehead 60, 166, 229, 229, 231
pageheadfoot . 60, 78–79, 166, 202, 228, 229, 229, 231, 234

pagenumber 60, 78, 78–79, 166, 202, 229, 245, 407
pagination 60, 166
paragraph 60, 91, 98, 97–99
part 60, 98, 97–99
partentry 60, 72
partentrypagenumber 60, 72
partnumber 60, 98, 97–99
placeanddate 166, 190–191
PPdata 185, 186
PPlogo 185, 185
priority 185, 185
prioritykey 185, 185
publishers 60, 65, 273

R

refname 166, 189–190
refvalue 166, 189–190

S

section 60, 91, 98, 97–99
sectionentry 60, 72
sectionentrydots 72
sectionentrypagenumber 61, 72
sectioning 61
specialmail 166, 185, 185
subject 61, 65, 193, 273
subparagraph 61, 91, 98, 97–99
subsection 61, 91, 98, 97–99
subsubsection 61, 91, 98, 97–99
subtitle 61, 65, 273

T

title 61, 65, 66, 192, 273, 273
titlehead 61, 65, 272–273
toaddress 167, 182–184, 185
toneame 167, 182–184, 185

Index of Files, Classes, and Packages

A		L	
addrconv (package)	263	lco	482–486
article (class)	51	letter (class)	51
B		lipsum (package)	232, 242
babel (package)	64, 151, 189, 256, 272, 294, 299, 346, 354, 368, 380, 430, 486, 487	listings (package)	380
babelbib (package)	137	longtable (package)	120, 132, 133, 133
blindtext (package)	245	lscape (package)	381
book (class)	51	M	
bublatex (package)	137	marginnote (package)	424
C		marvosym (package)	177
capt-of (package)	125	microtype (package)	54, 150, 267, 355
caption (package)	125	mparhack (package)	424
E		multicol (package)	355, 463
engord (package)	252	N	
eso-pic (package)	394, 484	natbib (package)	137, 139
etoolbox (package)	329	ngerman (package)	189, 256, 325, 486
F		P	
fancyhdr (package)	79, 225, 481	pdflscape (package)	381
float (package)	69, 120, 121, 129, 379, 380	picture (package)	394
floatrow (package)	380	R	
fontenc (package)	36	report (class)	51
footmisc (package)	85, 205, 278	S	
french (package)	486, 487	scraddr (package)	259–262
G		scrartcl (class)	72, 77, 106, 51–141, 441–463
geometry (package)	25, 45, 437	scrbase (package)	308–330
german (package)	256, 325, 486	scrbook (class)	72, 77, 106, 51–141, 441–463
graphics (package)	54, 150, 180, 186, 267	scrdate (package)	252–256
graphicx (package)	54, 84, 150, 186, 267, 278, 484	scrextend (package)	264–285, 441–463
H		scrhack (package)	378–381
hyperref (package)	286	scrjura (package)	286–306, 430
I		scrlayer (package)	225, 227, 381, 382–405, 412, 424, 427, 431
index (package)	140	scrlayer-notecolumn (package)	424–436
isodate (package)	189	scrlayer-scrpage (package)	78, 79, 200, 202, 225–251, 413–423, 424, 427, 431, 481
K		scrletter (package)	464–490
keyval (package)	308, 310	scrfile (package)	331–341
		scrlltr2 (class)	142–224, 464–490
		scrpage (package)	481

scrpage2 (package)	481
scrrprt (class)	72, 77, 106, 51–141, 441–463
scrttime (package)	257–258
scrwfile (package)	342–345, 352, 354, 368
selinput (package)	430
setspace (package)	26, 49, 380
showframe (package)	381
splitidx (package)	140

T	
tabu (package)	120
tabularx (package)	154
titletoc (package)	345
tocbasic (package)	346–377, 380, 443, 448
typearea (package)	25–50, 227, 405, 437–440
typearea.cfg	440

X	
xcolor (package)	89, 208, 282, 414, 430

Index of Class and Package Options

12h= <i>simple-switch</i>	258
24h	258

A	
abstract= <i>simple switch</i>	68
addrfield= <i>mode</i>	182–186
addrfield=backgroundimage	474, 475
addrfield=image	474
addrfield=PP	474, 475
adrFreeIVempty	262
adrFreeIVshow	262
adrFreeIVstop	262
adrFreeIVwarn	262
appendixprefix= <i>simple switch</i>	91
areasetadvanced= <i>simple switch</i>	438
autoclearnotecolumns= <i>simple switch</i>	435–436
automark	241–243, 405–406
autooneside= <i>simple switch</i>	241–243, 405–406
autoremoveinterfaces= <i>simple switch</i>	412

B	
backaddress= <i>value</i>	182–186
BCOR	44
BCOR= <i>correction</i>	31–32
BCOR=current	38
bibliography= <i>selection</i>	138
bibliography=leveldown	138, 139
bibliography=nottotoc	138, 139
bibliography=oldstyle	138, 139
bibliography=openstyle	138, 139, 462

C	
captions	123, 127
captions= <i>selection</i>	120–123
captions=bottombeside	121, 127
captions=centeredbeside	121, 127
captions=figureheading	120, 121
captions=figuresignature	120, 121
captions=heading	120, 121
captions=innerbeside	122, 127
captions=leftbeside	122, 127
captions=nooneline	121, 122
captions=online	122
captions=outerbeside	122, 127
captions=rightbeside	122, 127
captions=signature	120, 122
captions=tableheading	120, 122
captions=tablesignature	120, 122
captions=topbeside	123, 127
chapteratlists	94
chapteratlists= <i>value</i>	94
chapteratlists=entry	94
chapterentrydots= <i>simple switch</i>	72
chapterprefix	457
chapterprefix= <i>simple switch</i>	91
clausemark= <i>value</i>	292
clausemark=both	292
clausemark=false	292
clausemark=forceboth	292
clausemark=forceright	292

clausemark=right	292	footlines=number of lines	43–44
cleardoublepage	82, 204, 276	footnotes=setting	85, 205, 278
cleardoublepage=page style ...	82, 204, 276	footnotes=multiple	85
cleardoublepage=current	82, 204, 276	footnotes=nomultiple	85
clines	249, 251	footsepline	202, 482
contract	289	footsepline=simple switch	77, 200
<div>D</div>			
deactivatepagestylelayers=simple switch	401–402	footsepline=thickness:length ...	249–251
DIN	218	footwidth=width:offset:offset	249
DINmtext	218	forceoverwrite=simple switch	412
DIV	34–37, 45	fromalign=method	171
DIV=factor	32–34	fromemail=simple switch	176–180
DIV=areaset	45	fromfax=simple switch	176–180
DIV=calc	34–35	fromlogo=simple switch	180–182
DIV=classic	34–35	frommobilephone=simple switch ...	176–180
DIV=current	35–37	fromphone=simple switch	176–180
DIV=last	35–37	fromrule=position	172–175
DIV=areaset	36	fromurl=simple switch	176–180
DIV=calc	36, 42	<div>H</div>	
DIV=classic	36	headheight	44
DIV=current	36	headheight=height	42–43
DIV=default	36	headinclude=simple switch	40–41
DIV=last	36	headings	456
draft	396	headings=selection	91–93
draft=simple switch 54, 150, 243–244, 267, 406		headings=big	91, 92
headings=normal			
headings=onelineappendix			
headings=onelinechapter			
headings=openany			
headings=openleft			
headings=openright			
headings=optiontoheadandtoc			
headings=optiontohead			
headings=optiontotoc			
headings=small			
headings=twolineappendix			
headings=twolinechapter			
headlines			
headlines=number of lines			
headsepline			
headsepline=simple switch			
headsepline=thickness:length ...			
headtopline=thickness:length ...			
headwidth=width:offset:offset			
hmode			
headwidth=width:offset:offset			

I

ilines	249, 251
index=selection	140
index=default	140, 141
index=numbered	140, 141
index=totoc	140, 141
internalonly=value	308–309

J

juratitlepagebreak=simple switch	292
juratocindent=indent	289
juratocnumberwidth=number width	289
juratotoc=level number	288–289
juratotoc=simple switch	288–289

K

KakuLL	218
KOMAOld	218
komastyle	240

L

legno	119
listings=false	380
listof	356
listof=setting	133–136, 380
listof=chapterentry	134, 135
listof=chaptergapline	134, 135
listof=chaptergapsmall	94, 134, 135
listof=entryprefix	135
listof=flat	134, 135
listof=graduated	134, 135
listof=leveldown	135, 354
listof=nochaptergap	134, 135
listof=nonumberline	355
listof=notoc	135
listof=numbered	136, 355
listof=numberline	355
listof=totoc	136, 355
locfield=selection	187–188
lscap=false	381

M

manualmark	241–243, 405–406
markcase=Wert	244, 407
markcase=lower	245
markcase=noupper	245

markcase=upper	240, 245
markcase=used	240, 245
mpinclude=simple switch	41–42

N

NF	218
NipponEH	218
NipponEL	219
NipponLH	219
NipponLL	219
NipponRL	219
numbers=selection	93–94
numbers=autoendperiod	94
numbers=endperiod	94
numbers=noendperiod	94
numericaldate=simple switch	189

O

olines	249, 251
onpsbackground=code	401
onpsevenpage=code	401
onpsfloatpage=code	401
onpsforeground=code	401
onpsinit=code	401
onpsnonfloatpage=code	401
onpsoddpag=code	401
onpsoneside=code	401
onpsselect=code	401
onpsttwo=side=code	401
open=method	90
open=any	90
open=left	90
open=right	90
origlongtable	133

P

pagenumber	203, 481
pagenumber=position	200
pagesize=output driver	47
pagesize=automedial	48
pagesize=auto	48
pagesize=dvipdfmx	48
pagesize=dvips	48
pagesize=false	48
pagesize=pdfTeX	48
pagestyle=setting	239–240

pagestyleset=standard	239–240
paper=format	45–46
parnumber=value	293
parnumber=auto	293
parnumber=false	293
parnumber=manual	293
parskip	292
parskip=manner	74–76, 197–198
parskip=false	74
parskip=full*	75
parskip=full+	75
parskip=full-	75
parskip=full	75
parskip=half+	75
parskip=half-	75
parskip=half	75
parskip=never	75
plainfootbotline=simple switch	251
plainfootsepline=simple switch	251
plainheadsepline=simple switch	251
plainheadtopline=simple switch	251
priority=priority	182–186

R

ref=value	297
ref=long	296
ref=numeric	296
ref=parlong	296
ref=parnumeric	296
ref=paroff	296
ref=parshort	296
ref=sentencenumeric	296
ref=sentenceoff	296
ref=sentenceshort	296
ref=value	296
refline=Einstellung	477
refline=selection	189–192
refline=dateleft	480
refline=dateright	480
refline=narrow	477
refline=nodate	480
refline=wide	477

S

sectionentrydots=simple switch	72
setspace=false	381

SN	217, 219
SNleft	219
standardstyle	240
subject=Einstellung	477
subject=selection	193–195
symbolicnames=simple switch	176–180

T

titlepage	62–63, 271
titlepage=simple switch	62–63, 271
titlepage=firstiscover	67, 274
titlepage=firstiscover	62–63, 271
toc=selection	69–71
toc=bibliographynumbered	69, 70
toc=bibliography	70
toc=chapterentrywithdots	70
toc=chapterentrywithoutdots	70
toc=flat	70, 134
toc=graduated	70, 71
toc=indexnumbered	69
toc=index	71
toc=listofnumbered	69, 71, 133
toc=listof	69, 71, 133
toc=nobibliography	71
toc=noindex	71
toc=nolistof	71, 133
toc=nonumberline	355
toc=numberline	355, 361
toc=sectionentrywithdots	71
toc=sectionentrywithoutdots	71
twocolumn	55, 107
twocolumn=simple switch	39
twoside	38–39, 55
twoside=simple switch	38–39
twoside=semi	38–39

U

UScommercial9	220
UScommercial9DW	220
usegeometry=simple switch	437

V

version .	30–31, 46, 53, 55, 149–150, 266, 445
version=value	30–31, 53, 149–150, 266
version=2.9t	479
version=2.9u	479

version=first	30–31, 53, 149–150, 266	visualize	483–486
version=last	30–31, 53, 149–150, 266		