KOMA-Script

a versatile LaTeX 2ε bundle

Note: This document is a translation of the German KOMA-Script manual. Several authors has been involved to this translation. Some of them are native English speakers, others like me are not. Improvement of the translation by native speakers or experts would be welcome always!

The Guide

KOMA-Script

Markus Kohm Jens-Uwe-Morawski

2012-05-15

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

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

Free screen version without any optimization of paragraph and page breaks

To All Friends of Typography!

Preface to new English KOMA-Script Guide

This is not a translation of the preface of the German KOMA-Script guide, because this translation of the German KOMA-Script guide is still a work in progress. Currently the chapters of part I and chapter 10, chapter 11, chapter 14, chapter 15, chapter 16, chapter 17 are up-to-date. There may still be dead-links and broken references at these chapters, because the referenced information in chapters of other parts could be missing.

The descriptions in chapter 12, chapter 13 aren't translations from the German manual but are primary descriptions from the implementation of these packages. They should be up-to-date — sometimes even more than the chapters from the German manual.

So this English guide is complete but nevertheless not as good as the German one, because my English is not as good as my German. Currently there's only one editor for the English guide, Krickette Murabayashi, who improves my translation for free. Many thanks to her for her very good job! Nevertheless, additional editors or translators would be welcome!

Contents

List of Figures

List of Tables

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 LaTeX should look at *The Not So Short Introduction to LaTeX 2ε* [OPHS99] or *LaTeX 2ε for Authors* [Tea05b] or a LaTeX reference book. You will also find useful information in the many LaTeX FAQs, including the *TeX Frequently Asked Questions on the Web* [FAQ11]. Although the length of the *TeX Frequently Asked Questions on the Web* is considerably long, it is nevertheless quite useful not only to those having problems using LaTeX or KOMA-Script.

Part II is recommended for advanced KOMA-Script users. These are all of you who already know LaTeX, 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 LaTeX 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, scrrprt, 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, it is called out in the margin, as shown here with scrartcl.

scrartcl

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

1.3. History of KOMA-Script

In the early 1990s, Frank Neukam needed a method to publish an instructor's lecture notes. At that time LaTeX was LaTeX2.09 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, LaTeX 2$_\varepsilon$ became available and brought with it many changes. Users of Script-2 were faced with either limiting their usage to LaTeX 2$_\varepsilon$'s compatibility mode or giving up Script altogether. This situation led Markus to put together a new LaTeX 2$_\varepsilon$ 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 scrlttr2 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 LaTeX Project Public License. You will find it in the file `lppl.txt`. An unofficial German-language translation is also available in `lppl-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 TeX distributions, MacTeX, MiKTeX, and TeX 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 TeX 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 LaTeX document that demonstrates the problem. You should only use the packages and definitions needed to demonstrate the problem; do not use any unusual packages.

By preparing such an example it often becomes clear whether the problem is truly a KOMA-Script bug or something else. 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 231. 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 LaTeX 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 [Ume00]), 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 10 from page 231 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.

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

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 [Tob00]), 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

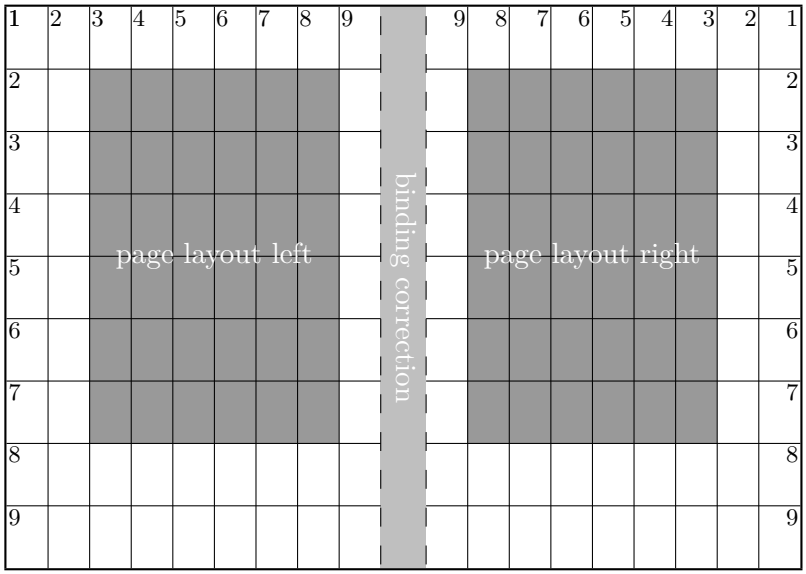


Figure 2.1.: Double-sided layout with the box construction of the classical division factor of 9, after subtraction of a binding correction

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.

- 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 32). If there is no binding correction ($BCOR = 0\,\mathrm{pt}$), the results roughly match the values of table 2.1, page 31.

In addition to the predefined values, one can specify *BCOR* and *DIV* as options when loading the package (see section 2.4, from page 29 onwards). There is also a command to explicitly calculate the type area by providing these values as parameters (also see section 2.4, page 35).

The typearea package can automatically determine the optimal value of *DIV* for the font and leading used. Again, see section 2.4, page 32.

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 doesn't choose the text area to match the font, but the author or typesetter has to choose the font according to the text area. This can be considered a "must".

In the typearea package this construction is changed slightly. By using a special (normally meaningless) *DIV* value or a special package option, a *DIV* value is chosen to match the perfect values of the late Middle Age method as closely as possible. See also section 2.4, page 32.

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 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 [?].

v3.00

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 LaTeX refuses multiple loadings of a package with different option settings.

```
\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*.

If in the *option list* one sets an option to a disallowed value, or the *value list* contains an invalid value, then an error is produced. If LaTeX is run in an interactive mode, then it stops at this point. Entering "`h`" displays a help screen, in which also the valid values for the corresponding option are given.

If a *value* includes an equal sign or a comma, then the *value* must be enclosed in curly brackets.

To implement this possibility KOMA-Script uses the commands `\FamilyOptions` and `\FamilyOption` with the family "`KOMA`". More information on these commands is found in section 10.2, page 234.

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
```

v3.01b

Since version 2.96a of KOMA-Script, for scrlttr2 since version 2.9t, and for typearea since version 3.01b, 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.

With `version=last` the most recent version will be selected at every LaTeX run. Be warned, though, that using `version=last` poses possibilities of compatibility issues for future LaTeX

v3.01a runs. Option `version` without any *value* means the same. This is the default behaviour as long as you don't 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` doesn't 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 can't 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 LaTeX literature, e. g. [OPHS99] 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 TeX 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.

Example: Assume one wants to write a meeting protocol, using the protocol class. The document should be double-sided. In the company 12 pt Bookman font is used.

Table 2.1.: Type-area dimensions dependent on *DIV* for A4

	Type-area		Margins	
DIV	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

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}
\usepackage{bookman}
```

The a4paper option can be left out when using the scrartcl class, as it is predefined

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

base font size:	10 pt	11 pt	12 pt
DIV:	8	10	12

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 explicity 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 15.2). 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=`*`Faktor`* 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 LaTeX 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*`}`

`areaset`	Recalculate page layout.
`calc`	Recalculate type-area including choice of appropriate *DIV* value.
`classic`	Recalculate type-area using Middle Age book design canon (circle-based calculation).
`current`	Recalculate type-area using current *DIV* value.
`default`	Recalculate type-area using the standard value for the current page format and current font size. If no standard value exists, `calc` is used.
`last`	Recalculate type-area using the same *DIV* 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 LaTeX 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, LaTeX 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 [Tob00]), 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 neesds 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
\KOMAoptions{DIV=last}
\begin{document}
\title{Title}
\author{Markus Kohm}
\begin{spacing}{1}
  \maketitle
  \tableofcontents
\end{spacing}
\chapter{Ok}
\end{document}
```

See further also the notes in section 2.8.

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 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

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

`current`	Recalculate type-area with the currently valid *BCOR* value.

`\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 LaTeX 2ε. When used within the document, a page break will result.

v3.00 Since `\typearea{current}{last}` or `\KOMAoptions{DIV=last}` are often needed for recalculation of the type-area, there exists specially the abbreviated command `\recalctypearea`.

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.

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.

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

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

Value	Description
`true`[1]	activates the option
`on`	activates the option
`yes`	activates the option
`false`	deactivates the option
`off`	deactivates the option
`no`	deactivates the option

[1] This value will be used also, if you use the option without assigning any value.

`\KOMAoption`. Use of the option after loading the typearea package results automatically in recalculation of the type-area using `\recalctypearea` (see page 35). 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 `\recalctypearea` (see page 35).

`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 [OPHS99] 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 number[1] 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., the header or footer is counted as part of the text.

v3.00

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

[1] Pagination refers to the indication of the page number.

[2] Running 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.

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 33) or the \recalctypearea command (see page 35).

mpinclude=*simple switch*

v2.8q

v3.00

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.

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 32) heeds 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 33) or the \recalctypearea command (see page 35).

`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 33) or the `\recalctypearea` command (see page 35).

`\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!

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 [Ume00]).

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 23 onwards), the entire page division and document layout depends on the paper format. Whereas the LaTeX 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 ° 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 also use option `pagesize`, which will be described next.

v3.01b Additionally, the *`format`* can also be specified in the form *`height`*`:`*`width`*. Note that until version 3.01a *`height`* and *`width`* has been interchanged. This is important if you use compatibility settings (see option `version`, section 2.5, page 28).

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=A8,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 33) or the `\recalctypearea` command (see page 35).

`pagesize=`*`output driver`*

The above-mentioned mechanisms for choice of paper format only affect the output insofar as internal LaTeX 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 TeX source file. When using the DVI driver `dvips` the information can be given in the form of a `\special` command. With pdfTeX or VTeX 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=false`. Usage of option `pagesize` without value is same like `pagesize=auto`.

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 pdfTeX 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 LaTeX or by pdfLaTeX — 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.

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

`auto`
: Uses output driver `pdftex` if pdfTEX-specific registers `\pdfpagewidth` and `\pdfpageheight` are defined. In addition, output driver `dvips` will be used.

`automedia`
: Almost the same as `auto` but if the VTEX-specific registers `\mediawidth` and `\mediaheight` are defined, they will be set additionally.

`false`, `no`, `off`
: Doesn't set any output driver and doesn't send page size information to the output driver.

`dvipdfmx` v3.05a
: Writes paper size into DVI files using `\special{pagesize=`*`width`*`,`*`height`*`}`. The name of the output driver is `dvipdfmx` because application `dvipdfmx` 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 `\special{pagesize=`*`width`*`,`*`height`*`}`. While application `dvips` 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 `\begin{document}` on your own risk, if you are using `dvips`!

`pdftex`
: Sets paper size using the pdfTEX-specific registers `\pdfpagewidth` and `\pdfpageheight`. You may do this at any time in your document.

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 TEX. In such a single-spaced document even correction signs 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 20^{th} 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.

LaTeX 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 LaTeX. 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 LaTeX 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 LaTeX 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 [Ume00]), which is not part of KOMA-Script, or change the type-area parameters of LaTeX. 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 LaTeX 2_ε in 1994. In the worst case one may use the instruction `\linespread`. The author recommends the package setspace (see [Tob00]), 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 LaTeX 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 LaTeX classes. This means that the KOMA-Script bundle contains replacements for the three standard classes: book, report, and article. There is also a replacement for the standard class letter. The document class for letters is described in a separate chapter, because it is fundamentally different from the three main classes (see chapter 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 LaTeX, just like before the substitution. The look, however, should be different. Additionally, the KOMA-Script classes provide new possibilities and options that are described in the following sections.

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.

3.2. Compatibility with Earlier Versions of KOMA-Script

It applies, mutatis mutandis, what is written in section 2.5.

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

standard class	KOMA-Script class
article	scrartcl
report	scrreprt
book	scrbook
letter	scrlttr2

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 don't 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 [Car99d]).

`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 37. 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 [Sch10].

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 23 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. TEX knows shrinkable and stretchable distances for this purpose. LATEX provides an automatism for this kind of vertical adjustment.

Using double-sided typesetting with option `twoside` (see section 2.4, page 36) switches on vertical adjustment also. 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 TEX 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 35). 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 depend on those packages. This may even result in error messages or typesetting errors, which cannot be considered a fault of KOMA-Script.

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 scrartcl 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

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 [OPHS99], [Tea05b], and [Tea05a].

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

The LaTeX-Kern already defines the command `\textsuperscript` to superscript text. Unfortunately, LaTeX 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 $C_6H_{12}O_6$ and O_2 to produce H_2O and CO_2. However, arsenic (As) has a quite detrimental effect on the metabolism.

Some time later you decide that the chemical formulas should be typeset in sans serif. Now you can see the advantages of using logical markup. You only have the 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 $\mathsf{C_6H_{12}O_6}$ and $\mathsf{O_2}$ to produce $\mathsf{H_2O}$ and $\mathsf{CO_2}$. However, arsenic (As) has a quite detrimental effect on the metabolism.

In the example above, the notation "`\molec C6`" is used. This makes use of the fact that arguments consisting of only one character do not have to be enclosed in parentheses. That is why

"\molec C6" is similar to "\molec{C}{6}". You might already know this from indices or powers in mathematical environments, such as "x^2" instead of "x^{2}" for "x^2".

```
\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 [OPHS99], [Tea05b], or [Tea05a]. Color switching commands like \normalcolor (see [Car99d] 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.

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

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

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

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

Table 3.2.: Elements whose type style can be changed with the KOMA-Script command \setkomafont or \addtokomafont

caption	text of a table or figure caption (see section 3.20, page 111)

...

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

`captionlabel`
: label of a table or figure caption; used according to the element `caption` (see section 3.20, page 111)

`chapter`
: title of the sectional unit `\chapter` (see section 3.16, page 85)

`chapterentry`
: table of contents entry of the sectional unit `\chapter` (see section 3.9, page 63)

`chapterentrypagenumber`
: page number of the table of contents entry of the sectional unit `\chapter`, variation on the element `chapterentry` (see section 3.9, page 63)

`chapterprefix`
: chapter number line at setting `chapterprefix=true` or `appendixprefix=true` (see section 3.16, page 80)

`descriptionlabel`
: labels, i. e., the optional argument of `\item` in the `description` environment (see section 3.18, page 101)

`dictum`
: dictum, wise saying, or smart slogan (see section 3.17, page 97)

`dictumauthor`
: author of a dictum, wise saying, or smart slogan; used according to the element `dictumtext` (see section 3.17, page 97)

`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 80)

`footnote`
: footnote text and marker (see section 3.14, page 76)

...

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

`footnotelabel`
: mark of a footnote; used according to the element `footnote` (see section 3.14, page 76)

`footnotereference`
: footnote reference in the text (see section 3.14, page 76)

`footnoterule`
v3.07
: horizontal rule above the footnotes at the end of the text area (see section 3.14, page 79)

`labelinglabel`
: labels, i. e., the optional argument of `\item` in the `labeling` environment (see section 3.18, page 102)

`labelingseparator`
: separator, i. e., the optional argument of the `labeling` environment; used according to the element `labelinglabel` (see section 3.18, page 102)

`minisec`
: title of `\minisec` (see section 3.16 ab page 90)

`pagefoot`
: only used if package scrpage2 has been loaded (see chapter 5, page 201)

`pagehead`
: another name for `pageheadfoot`

`pageheadfoot`
: the head of a page, but also the foot of a page (see section 3.12 ab page 68)

`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 85)

...

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

`part`
: title of the `\part` sectional unit, without the line containing the part number (see section 3.16, page 85)

`partentry`
: table of contents entry of the sectional unit `\part` (see section 3.9, page 63)

`partentrypagenumber`
: Page number of the table of contents entry of the sectional unit `\part` variation on the element `partentry` (see section 3.9, page 63)

`partnumber`
: line containing the part number in a title of the sectional unit `\part` (see section 3.16, page 85)

`section`
: title of the sectional unit `\section` (see section 3.16, page 85)

`sectionentry`
: table of contents entry of sectional unit `\section` (only available in scrartcl, see section 3.9, page 63)

`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 63)

`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 57)

`subparagraph`
: title of the sectional unit `\subparagraph` (see section 3.16, page 85)

`subsection`
: title of the sectional unit `\subsection` (see section 3.16, page 85)

`subsubsection`
: title of the sectional unit `\subsubsection` (see section 3.16, page 85)

...

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

`subtitle`	subtitle of the document, i. e., the argument of `\subtitle` on the main title page (see section 3.7, page 57)
`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 57)

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`*

v3.00

Using `\maketitle` (see page 56), this option switches between document title pages and in-page title. For *`simple switch`*, any value from table 2.5, page 37 may be used.

The option `titlepage=true` or `titlepage` makes LaTeX 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 LaTeX, 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.

The default of classes scrbook and scrreprt is usage of title pages. Class scrartcl, 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 70.

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 [Bra01]). Examples can be found at the end of this section.

`\extratitle{`*half-title*`}`

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

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

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

```
\documentclass{scrbook}
\begin{document}
  \extratitle{\textbf{\Huge Me}}
  \title{It's me}
  \maketitle
\end{document}
```

You can center the half-title 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

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

Element name	Default
subject	\normalfont\normalcolor\bfseries\Large
title	\usekomafont{disposition}
subtitle	\usekomafont{title}\large

define the corresponding elements.

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

v2.95 The *subject* is output immediately above the *title*. Thereby the font switching of element subject will be used. The default, that may be found in table 3.3, may be changed using the commands \setkomafont and \addtokomafont (see section 3.6, page 51).

v2.8p The *title* is output with a very large font size. Besides the change of size, the settings for the element title also take effect. By default these settings are identical to the settings for the element disposition (see table 3.2, page 51). The default settings may be changed using the commands \setkomafont and \addtokomafont (see section 3.6, page 51). The font size is, however, not affected (see table 3.2, page 59).

v2.97c The *subtitle* is set just below the title, in a font determined by the element subtitle. The default, seen in table 3.3, can be changed with the help of the commands \setkomafont and \addtokomafont (see section 3.6, page 51).

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

Below the author or authors appears the date. The default value is the present date, as produced by \today. The \date command accepts arbitrary information — 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.

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.

With the exception of *titlehead* and possible footnotes, all the items are centered horizontally. The information is summarised in table 3.4.

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

Table 3.4.: Font size and horizontal positioning of the elements in the main title page in the order of their vertical position from top to bottom when typeset with `\maketitle`

Element	Command	Font	Orientation
Title head	`\titlehead`	`\normalsize`	justified
Subject	`\subject`	`\usekomafont{subject}`	centered
Title	`\title`	`\huge\usekomafont{title}`	centered
Subtitle	`\subtitle`	`\usekomafont{subtitle}`	centered
Authors	`\author`	`\Large`	centered
Date	`\date`	`\Large`	centered
Publishers	`\publishers`	`\Large`	centered

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\\
  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.

`\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. 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

scrartcl, scrreprt

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*

scrreprt, scrartcl

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 *simplex switch*, any value from table 2.5, page 37 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.

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

scrartcl, scrreprt

Some LaTeX 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 89 onwards.

When using an in-page title (see option `titlepage`, section 3.7, page 55), the abstract is set using the environment `quotation` (see section 3.18, page 104) internally. Thereby paragraphs will be set with intention of the first line. If that first paragraph of the abstract should not be intended, 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 similiar 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 [Lap06]). 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`.

v3.00

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. KOMA-Script does not have support to ignore this general agreement.

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 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.

...

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

`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 LaTeX run.
`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.
`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.

`\tableofcontents`

The production of the table of contents is done by the `\tableofcontents` command. To get a correct table of contents, at least two LaTeX runs are necessary after every change. The 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 LaTeX runs are needed again.

The entry for the highest sectional unit below `\part`, i. e., `\chapter` with scrbook and scrreprt or `\section` with scrartcl is not indented. There are no dots between the text of the

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

Element	Default font style
partentry	\usekomafont{disposition}\large
partentrypagenumber	
chapterentry	\usekomafont{disposition}
chapterentrypagenumber	
sectionentry	\usekomafont{disposition}
sectionentrypagenumber	

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 scrartcl 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 51, and table 3.2, page 51). The default settings of the elements may be found at table 3.6.

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 scrartcl. The inclusion of a sectional unit in the table of contents is controlled by the counter `tocdepth`. This has the value −1 for `\part`, 0 for `\chapter`, and so on. By incrementing or decrementing the counter, one can choose the lowest sectional unit level to be included in the table of contents. The same happens with the standard classes.

The user of the scrpage2 package (see chapter 5) does not need to remember the numerical values of each sectional unit. They are given by the values of the macros `\chapterlevel`, `\sectionlevel`, and so on, down to `\subparagraphlevel`.

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

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

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

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

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

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

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 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`*

v3.00 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 1 em. 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 doesn't take care about the white space at the end of the last line of a paragraph.

v3.08 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 37 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 1 em white space at the end of the last line of each paragraph. Switching off the options would reactivate the default of 1 em 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

`false`, `off`, `no`	paragraph indentation instead of vertical space; the last line of a paragraph may be arbitrarily filled
`full`, `true`, `on`, `yes`	one line vertical space between paragraphs; there must be at least 1 em free space in the last line of a paragraph
`full-`	one line vertical space between paragraphs; the last line of a paragraph may be arbitrarily filled
`full+`	one line vertical space between paragraphs; there must be at least a third of a line free space at the end of a paragraph
`full*`	one line vertical space between paragraphs; there must be at least a quarter of a line free space at the end of a paragraph
`half`	half a line vertical space between paragraphs; there must be at least 1 em free space in the last line of a paragraph
`half-`	one line vertical space between paragraphs
`half+`	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
`half*`	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

...

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

`never`	there will be no inter-paragraph spacing even if additional vertical spacing is needed for the vertical adjustment with `\flushbottom`

v3.08

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 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 16.1, page 282 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 37. 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 doesn't mean that the header must now be moved farther from the text body. Instead, the header should be considered as belonging to the text body for the purpose of page layout calculations. KOMA-Script takes this into account by automatically passing the option `headinclude` to the typearea package whenever the `headsepline` option is used. KOMA-Script behaves similar to `footinclude` using `footsepline`. Package scrpage2 (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 scrreprt the titles of chapters and sections are repeated in the header for double-sided layout — with KOMA-Script on the outer side, with the standard classes on the inner side. The page number is set on the outer side of the footer with KOMA-Script; with the standard classes it is set on the inner side of the header. In one-sided layouts only the titles of the chapters are used and are, with KOMA-Script, centered in the header. The page numbers are set centered in the footer with KOMA-Script. scrartcl behaves similarly, but starting a level deeper in the section hierarchy with sections and subsections, because the chapter level does not exist in this case.

scrbook, scrreprt

scrartcl

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

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

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

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.

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 scrpage2 package, but instead via special instructions. The page styles `headings` and `myheadings` should not be used together with this package (see chapter 5, page 200).

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 51. 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 scrpage2 (see chapter 5, page 201).

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 51).

It is not possible to use these methods to force capitals to be used automatically for the running headings. For that, please use the scrpage2 package (see chapter 5, page 208).

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 scrpage2 (see chapter 5) for that, but, for example, the package fancyhdr (see [vO00]), 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 scrpage2 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 scrpage2, 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`

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

`\titlepagestyle` Page style for a title page when using *in-page* titles.
`\partpagestyle` Page style for the pages with `\part` titles.
`\chapterpagestyle` Page style for the first page of a chapter.
`\indexpagestyle` Page style for the first page of the index.

scrbook, scrreprt

and `\chapterpagestyle` are found only with classes scrbook and scrreprt, but not in scrartcl. The default value for all four cases is `plain`. The meaning of these macros can be taken from table 3.9. The page styles can be redefined with the `\renewcommand` macro.

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

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

As mentioned previously on page 68, the page style `empty` is exactly what is required in this example. Naturally you can also use a user-defined page style.

Assume you have defined your own page style for initial chapter pages with the package scrpage2 (see chapter 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

Table 3.10.: Available numbering styles of page numbers

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

will be that you don't need to know the current page style before the change to switch back at the end.

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

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

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

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

should read more about the background of `\rightmark` at section 16.1, page 282.

`\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 LaTeX kernel. You can specify with this command the *`numbering style`* of page numbers.

The changes take effect immediately, hence starting with the page that contains the command. It is recommended to use `\cleardoubleoddpage` 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 TeX 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. 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 aren't 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` to break until the wanted page. Every already defined *`page style`* (see section 3.12 from page 68 and chapter 5 from page 196) may be used. Besides this, `cleardoublepage=current` is valid. This case is the default until KOMA-Script 2.98c and v3.00 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 28).

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:

```
\KOMAoption{cleardoublepage=plain}
```

More information about page style `plain` may be found at section 3.12, page 69.

```
\clearpage
\cleardoublepage
\cleardoublepageusingstyle{page style}
\cleardoubleemptypage
\cleardoubleplainpage
\cleardoublestandardpage
\cleardoubleoddusingstyle{page style}
\cleardoubleoddemptypage
\cleardoubleoddplainpage
\cleardoubleoddstandardpage
\cleardoubleevenusingstyle{page style}
\cleardoubleevenemptypage
\cleardoubleevenplainpage
\cleardoubleevenstandardpage
```

The LaTeX 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 36) starts a new right-hand page. An empty left page in the current page style is output if necessary.

v3.00

With \cleardoubleoddstandardpage, KOMA-Script works as described above. The \cleardoubleoddplainpage 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 \cleardoubleoddemptypage, 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 is 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 contradition 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 therefor provides the commands \cleardoubleevenstandardpage, \cleardoubleevenplainpage, \cleardoubleevenemptypage, and \cleardoubleevenusingstyle, 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 \cleardoubleoddpage 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

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

`multiple`	At sequences of immediately following footnote marks, consecutive marks will be separated by `\multfootsep`.
`nomultiple`	Immediately following footnotes will be handled like single footnotes and not separated from each other.

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.

3.14. Footnotes

KOMA-Script, unlike the standard classes, provides features for configuration of 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 [Fai05]). 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 75.

```
\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 LaTeX 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 shouldn't 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 couldn't 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.,

```
Company SplishSplash\footnote{This is a registered trade name.
  All rights are reserved.\label{refnote}}
produces not only SplishPlump\footref{refnote}
but also SplishPlash\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 illustrates the layout parameters. The default configuration of the KOMA-Script classes is:

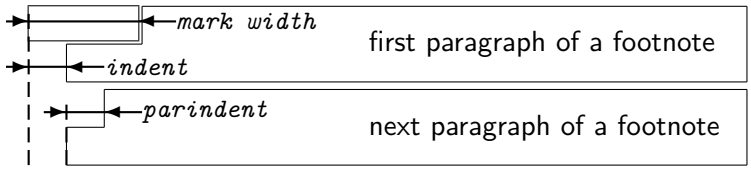


Figure 3.1.: Parameters that control the footnote layout

```
\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.

v2.8q The font element `footnote` determines the font of the footnote including the footnote mark. Using the element `footnotelabel` the font of the footnote mark can be changed separately with the commands `\setkomafont` and `\addtokomafont` (see section 3.6, page 51). Please refer also to table 3.2, page 51. 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 51). 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 51).

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 1 em. The following lines of the footnote text are also indented by 1 em 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 51:

```
\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* ha 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` (see section 3.6, page 51). 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 28). It is explicity noted that this creates a confusion with chapter numbers! The use of `\addsec` and `\section*` (see section 3.16, page 89 and page 90) 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 28).

\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 72.

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

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.

v3.00

Besides the implicit usage of \cleardoublepage at chapter starts, the option influences also the explicit usage of the commands \cleardoublepage, \cleardoublepageusingstyle, \cleardoublestandardpage, \cleardoubleplainpage, and \cleardoubleemptypage. See section 3.12, page 73 for more information about these. Since LaTeX doesn't differentiate between left-hand and right-hand pages in single-sided printing, the option doesn't have any influence in that case.

In class scrartcl the section is the first structural element below the part. Because of this, scrartcl doesn't support this option.

`chapterprefix=`*`simple switch`*
`appendixprefix=`*`simple switch`*

scrbook, scrreprt

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 37 may be used as *`simple switch`*. The default, however, is `chapterprefix=false`, which is opposite of the behaviour of the standard classes, which

[1] When using another language the word "Chapter" is naturally translated to the appropriate language.

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

`any`	Commands `\cleardoublepageusingstyle`, `\cleardoublestandardpage`, `\cleardoubleplainpage`, `\cleardoubleemptypage`, and `\cleardoublepage` result in a single page break and therefor are same like `\clearpage`.
`left`	Commands `\cleardoublepageusingstyle`, `\cleardoublestandardpage`, `\cleardoubleplainpage`, `\cleardoubleemptypage`, and `\cleardoublepage` result in a page break and add an interleaf page if needed to reach the next left-hand page.
`right`	Commands `\cleardoublepageusingstyle`, `\cleardoublestandardpage`, `\cleardoubleplainpage`, `\cleardoubleemptypage`, and `\cleardoublepage` result in a page break and add an interleaf page if needed to reach the next right-hand page.

would correspond to `chapterprefix=true`. These options also affect the automatic running titles in the headers (see section 3.12, page 68).

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 37). Since this results in an inconsistent document layout, I advise against using this option.

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 51). Default is the usage of element `chapter` (see page 85, as well as table 3.15, page 88).

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

v3.00

scrbook, scrreprt

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

scrbook, scrreprt

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` 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, R 3]). On the other hand, if roman numerals or letters are appear in the numbering, then a period should appear at the end of the numbering (see [DUD96, R 4]). 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 aux file and takes effect on the next LaTeX run.

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

`autoendperiod`, `autoenddot`, `auto`	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.
`endperiod`, `withendperiod`, `periodatend`, `enddot`, `withenddot`, `dotatend`	All numbers of structural commands and all dependent numbers will be set with ending period. Only references will be set without the ending period.
`noendperiod`, `noperiodatend`, `noenddot`, `nodotatend`	All the numbers are without ending period.

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`*

scrbook, scrreprt v2.96a

As mentioned in section 3.20, page 121, 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 28).

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 10 pt. 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 doesn't 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 doesn't 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 64) 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 isn't 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 LaTeX 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 scrartcl) 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 95) is set on a separate page.

scrbook, scrreprt

scrbook, scrreprt

`\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 scrartcl. 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 80).

scrbook, scrreprt

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 70).

v2.8p

The font of all headings can be changed with the commands `\setkomafont` and `\addtokomafont` (see section 3.6, page 51). In doing this, generally the element `disposition` is used, followed by a specific element for every section level (see table 3.2, page 51). 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 81). 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}
```

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:

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

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

```
\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 46 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 TEX 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.

scrbook, scrreprt

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

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 70).

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 scrartcl.

book, scrreprt

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 70).

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 51) 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`
`\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.

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 don't 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
\othersectionlevelsformat{sectioning name}{}{counter output}
\autodot
```

KOMA-Script has added a further logical level on top of `\the`*`sectioning 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`, and the command `\othersectionlevelsformat` that expect three arguments. Of course the command `\chapterformat` like `\thechapter` does not exist in the class scrartcl but only in the classes scrbook and scrreprt.

scrbook, scrreprt

As described for option `numbers` at the beginning of this section (see page 83), 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 being 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.

The command \othersectionlevelsformat takes as first parameter the name of the section level, such as section, subsection, subsection, subsubsection, paragraph, and subparagraph. The third parameter should be the output of the corresponding counter, usually \thesection, \thesubsection, \thesubsubsection, \theparagraph, or \thesubparagraph.

Per default, therefore, only the levels \part and \chapter have formatting commands of their own, while all other section levels are covered by one general formatting command. This has historical reasons. At the time that Werner Lemberg suggested a suitable extension of KOMA-Script for his CJK package, only this differentiation was needed. Users should ignore the third argument of \othersectionlevelsformat completely.

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

```
\newcommand*{\partformat}{\partname~\thepart\autodot}
\newcommand*{\chapterformat}{%
  \chapappifchapterprefix{\ }\thechapter\autodot\enskip}
\newcommand*{\othersectionlevelsformat}[3]{%
  #3\autodot\enskip}
```

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

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

Strictly speaking, you could do without \autodot at this point and insert a fixed dot instead. As \part is numbered with roman numerals, according to [DUD96] a 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 83.

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

```
\renewcommand*{\othersectionlevelsformat}[3]{%
    \llap{\#3\autodot\enskip}}
```

The little known TEX command \llap in the definition above puts its argument left of the current position without changing the position thereby. A much better LATEX solution would be:

```
\renewcommand*{\othersectionlevelsformat}[3]{%
```

```
\makebox[0pt][r]{%
  #3\autodot\enskip}}
```

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

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

scrbook, scrreprt (≥2.96)

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 80) `\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 `chapterprefix=false` 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}
\sectionmark{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`.

scrbook, scrreprt

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 scrpage2 package (see chapter 5).

scrartcl

Similar to `\chapterformat` and `\othersectionlevelsformat`, the commands `\chaptermarkformat` (not in scrartcl), `\sectionmarkformat`, and `\subsectionmarkformat` (only in scrartcl) define the formatting of the sectioning numbers in the automatic running 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`

Per default, in the classes scrbook and scrreprt, the section levels from `\part` down to `\subsection`, and in the class scrartcl, the levels from `\part` down to `\subsubsection` are numbered. This is controlled by the LaTeX counter `secnumdepth`. The value −1 represents `\part`, 0 the level `\chapter`, and so on. By defining, incrementing, or decrementing this counter, you can determine down to which level the headings are numbered. The same applies in the standard classes. Please refer also to the explanation concerning the counter `tocdepth` in section 3.9, page 64.

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

scrbook, scrreprt

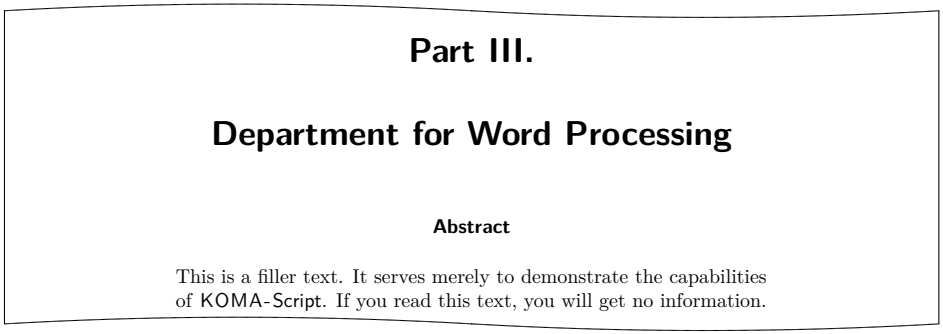
Parts and chapters in KOMA-Script can be started with a *preamble*. This is particularly useful when you are using a two column layout with the class option `twocolumn`, since the heading together with the *preamble* is always set in a one column layout. The *preamble* can comprise more than one paragraph. 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 81) and the options for the `abstract` environment (see section 3.8, page 60), 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 61).

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.

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

Element	Default
dictumtext	\normalfont\normalcolor\sffamily\small
dictumauthor	\itshape

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 16.3, page 285 for this.

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 dictumtext, which can be changed with the commands \setkomafont and \addtokomafont (see section 3.6, page 51). 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 `dictumtext` 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 51).

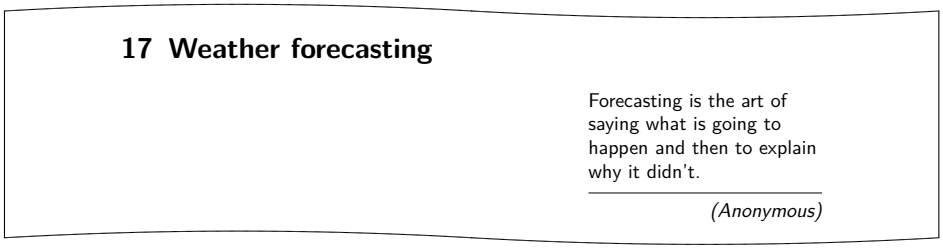
If `\dictum` is used within the macro `\setchapterpreamble` or `\setpartpreamble` you have to take care of the following: the horizontal positioning is always done with `\raggeddictum`. Therefore, the optional argument for horizontal positioning which is implemented for these two commands has no effect. `\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 LaTeX 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}
      \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 LaTeX kernel. Depending on the level, the numbering uses the following characters: Arabic numbers, small letters, small roman numerals, and capital letters. The kind of numbering is defined with the macros `\theenumi` down to `\theenumiv`. The output format is determined

by the macros `\labelenumi` to `\labelenumiv`. While the small letter of the second level is followed by a round parenthesis, the values of all other levels are followed by a dot. Every item is introduced with `\item`.

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

Vehicles

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

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

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

v2.8p

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 fontwhich is responsible for emphasizing the item can be changed with the commands `\setkomafont` and `\addtokomafont` (see section 3.6, page 51) for the element `descriptionlabel` (see table 3.2, page 51). 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 fontwhich is responsible for emphasizing the item and the separator can be changed with the commands `\setkomafont` and `\addtokomafont` (see section 3.6, page 51) for the element `labelinglabel` and `labelingseparator` (see table 3.2, page 51).

v3.01

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:

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

As can be seen in this example, a font changing command may be set in the usual way. But if you don't 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 1 em. In turn the outer margin is decreased by 1 em. The result is a shift by 1 em to the outside. Instead of `1em` you can of course use a length, for example, `2\parindent`.

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 3.11, page 67) and `\ifthispagewasodd` (section 16.1, page 282).

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 16.1, page 282. Also at the expert part, in section 16.1, page 282, 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 LaTeX kernel have been supported. With this also, the options `leqno` and `fleqn` are available.

You won't 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 [OPHS99]. 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 wouldn't 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 wouldn't 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, LaTeX offers a very capable and comfortable mechanism for automatic placement of figures and tables. But often these floating environments are slightly misunderstood by beginners. They often ask for a fixed position of a table or figure within the text. However, since these floating environments are being referenced in the text this is not necessary in most cases. It is also not sensible because such an object can only be set on the page if there is enough space left for it. If this is not the case the object would have to be shifted onto the next page, thereby possibly leaving a huge blank space on the page before.

Often one finds in a document for every floating object the same optional argument for positioning the object. This also makes no sense. In such cases one should rather change the standard parameter globally. For more details refer to [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 wouldn't be useful to change the caption format from signatures to headings.

Nevertheless sometimes all floating environments shall use headings. For this KOMA-Script
v3.09 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 118).

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 shouldn't 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 `captionsbeside`, that will be described from page 115. 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 `captionsbeside` (see section 3.20, page 115) will be vertically align depending on the bottommost base lines.
`centeredbeside`, `besidecentered`, `middlebeside`, `besidemiddle` Captions and contents of environment `captionsbeside` (see section 3.20, page 115) will be vertically centered

...

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

`figureheading`, `figureabove`, `abovefigure`, `topatfigure`

v3.09

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

`figuresignature`, `belowfigure`, `bottomatfiggure`

v3.09

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

`heading`, `above`, `top`

v3.09

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`.

`innerbeside`, `besideinner`

Captions of environment `captionsbeside` (siehe section 3.20, page 115) 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 `captionsbeside` (siehe section 3.20, page 115) 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 `captionsbeside` (siehe section 3.20, page 115) 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 `captionsbeside` (siehe section 3.20, page 115) will be placed right beside the contents of the environment.

...

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

`signature`, `below`, `bot`, `bottom` v3.09 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`.
`topbeside`, `besidetop` Captions and contents of environment `captionsbeside` (see section 3.20, page 115) 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 `captions=tableheading`. 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:

```
\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 `captions=tableheading` instead (see page 108). 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 51). The respective elements for this are `caption` and `captionlabel` (see table 3.2, page 51). 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:

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

element	default
caption	\normalfont
captionlabel	\normalfont

```
\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}]%
```

Figure 3.2.: A rectangle

Table 3.19.: Rectangle size

Width:	4 cm
Height:	3 cm

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

```
    {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 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. Therefor 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 an tabular. An additional `flushleft` environment around the `minipage` environment may be used, to have a pleasing distance above and below is 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:

`l` – 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 my be changed using option `captions` (see page 108) with values like `innerbeside`, `leftbeside`, `outerbeside`, and `rightbeside`. If either `o` or `i` are used you may need to run LaTeX 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 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 LaTeX runs for this to work correctly.

KOMA-Script

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

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

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

With `\dimexp` a ε-TEX command has been used. This shouldn't be a problem at all, because KOMA-Script itself needs ε-TEX and every almost up-to-date LATEX distribution uses ε-TEXalready.

Figure 3.5 shows a centered caption with:

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

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

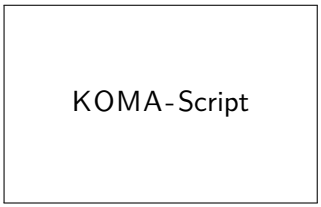
Even if you are not a typographer you may see, that this is not a recommended suggestion.

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}
  \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:maincls.captionbesidetop}
\end{figure}
\end{document}
```

You may use such a movement not only at graphics replacements like show, but also using `\includegraphics` (see [Car99d]).

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




```
\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*.

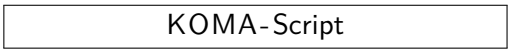


Figure 3.7.: Equivalent to the standard setting, similar to the usage of `\setcaphanging`

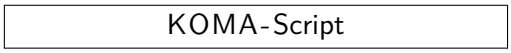


Figure 3.8.: With slightly hanging indentation starting at the second line using `\setcapindent{1em}`

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

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

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 109.

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.

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

v2.8q

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

With the command `\setcapwidth` you can decrease this *width*. The obligatory argument determines the maximum *width* of the caption. As an optional argument you can supply exactly one letter which specifies the horizontal justification. The possible justifications are given in the following list.

`l` – left-aligned

`c` – centered

`r` – right-aligned

`i` – alignment at the inner margin in double-sided output

`o` – alignment at the outer margin in double-sided output

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

With the command `\setcapmargin` you can specify a *margin* which is to be left free next to the description in addition to the normal text margin. If you want margins with different widths at the left and right side you can specify these using the optional argument *margin left*. The starred version `\setcapmargin*` defines instead of a *margin left* a *margin inside* in a double-sided layout. In case of longtable tables you have to deal with the same problem with justification inside or outside as mentioned with the macro `\setcapwidth`. Furthermore, the usage of `\setcapmargin` or `\setcapmargin*` switches on the option `captions=nooneline` (see page 109) 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 [KM08].

`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 `toc=nolistof`, `toc=listof`, and `toc=listofnumbered` described in section 3.9, page 61, 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 scrartcl. 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 90.

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 `toc=flat` for the table of contents. Thereby the needed with for printing the number will be determined at each LaTeX run. See option `toc=flat`, section 3.9, page 62 for information about how it works. Please note again, that you need more than one LaTeX 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 wouldn't 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 wouldn't give any additional information and for this wouldn't 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 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

v3.00 `listof=chaptergapsmall`. In this case a gap of width 10 pt will be used. With option `listof=chaptergapline` a gap of the height of one standard text line will be used. The gap

may be switched of with `listof=nochaptergap`. 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 doesn't have any floating environment. Additional influence of chapters to the lists of floating environments is available with option `chapteratlists`. See section 3.16, page 84 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`
v3.06
: 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 with of the first column depends on the previous LaTeX 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.

...

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

`nochaptergap`, `ignorechapter` Chapter starts won't be marked at the lists of floating environments.
`notoc`, `plainheading` The lists of floating environments, e. g., list of figures and list of tables, won't generate an entry at the table of contents.
`numbered`, `totocnumbered`, `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 won't be numbered.

```
\listoftables
\listoffigures
```

These commands generate a list of tables or figures. Changes in the document that modify these lists will require two LaTeX runs in order to take effect. The layout of the lists can be influenced by the option `listof` with values `graduated` or `flat` (see page 121). 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 LaTeX are inserted with the command `\marginpar`. They are placed in the outer margin. In documents with one-sided layout the right border is used. Though `\marginpar` can take an optional different margin note argument in case the output is in the left margin, margin notes are always set in justified layout. However, experience has

shown that many users prefer left- or right-aligned margin notes instead. To facilitate this, KOMA-Script offers the command `\marginline`.

Example: In this chapter, sometimes, the class name scrartcl 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 16.1, page 282. 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 83.

scrbook, scrreprt

Die output of the chapter headings in the appendix are influenced by the options `chapterprefix` and `appendixprefix`. See section 3.16, page 80 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 [OPHS99].

`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 BibTeX which you may select using `\bibstyle`. While BibTeX 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 16.3 at page 288.

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 62.

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 needn't 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=totocnumbered`, this command can only be successful if you have not loaded a package which prevents this by

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

`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.

redefining the `thebibliography` environment. Even though the natbib package makes unauthorized use of internal macros of KOMA-Script it could be achieved that `\setbibpreamble` works with the current version of natbib (see [Dal99]).

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

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

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

`\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 is currently only if they make the bibliography on their own. Because of this usage is very limited.

Table 3.22.: Available values of option `index`

`totoc`, `toc`, `notnumbered`	The index will have an entry at the table of contents, but will not have a heading number.
`default`, `nottotoc`, `plainheading`	The index will not have an entry at the table of contents.

`\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 this 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 16.2, page 282).

3.24. Index

For general information about making an index see [OPHS99], [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 [Koh06]).

`index=`*selection*

v3.00 The index is chapter (scrbook or scrreprt) or section (scrartcl) without heading number or entry at the table of contents by default or option `index=default`. The index doesn't 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`. See also option `toc` with value `index` in section 3.9 from page 62 onward.

A summary of all available values for option `index` may be found in table 3.22.

`\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 70).

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 scrlttr2

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 scrlttr2. 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 51). 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 scrlttr2

`addresseeimage`
: instuctions, that will be used to print the Port-Payé head of option `addrfield=backgroundimage` or the Port-Payé addressee of option `addrfield=image` (section 4.10, page 164)

`backaddress`
: return address for window envelopes (section 4.10, page 164)

`backaddressseparator`
: separator within the return address (section 4.10, page 164)

...

Table 4.1.: Alphabetical list of all supported variables in scrlttr2 (*continuation*)

`ccseparator`
: separator between title of additional addressees, and additional addressees (section 4.7, page 142)

`customer`
: customer number (section 4.10, page 171)

`date`
: date (section 4.10, page 170)

`emailseparator`
: separator between e-mail name and e-mail address (section 4.10, page 159)

`enclseparator`
: separator between title of enclosure, and enclosures (section 4.7, page 144)

`faxseparator`
: separator between title of fax, and fax number (section 4.10, page 159)

`firstfoot`
v3.08
: page foot of the note paper (section 4.10, page 176)

`firsthead`
v3.08
: page head of the note paper (section 4.10, page 164)

`fromaddress`
: sender's address without sender name (section 4.10, page 154)

`frombank`
: sender's bank account (section 4.10, page 177)

`fromemail`
: sender's e-mail (section 4.10, page 159)

`fromfax`
: sender's fax number (section 4.10, page 159)

`fromlogo`
: commands for inserting the sender's logo (section 4.10, page 162)

`fromname`
: complete name of sender (section 4.10, page 154)

Table 4.1.: Alphabetical list of all supported variables in scrlttr2 (*continuation*)

`fromphone`
: sender's telephone number (section 4.10, page 159)

`fromurl`
: a URL of the sender, e. g., the URL of his homepage (section 4.10, page 159)

`fromzipcode`
: zip code or postcode of the sender used at the Port-Payé head of option `addrfield=PP` (section 4.10, page 164)

`invoice`
: invoice number (section 4.10, page 171)

`location`
: more details of the sender (section 4.10, page 169)

`myref`
: sender's reference (section 4.10, page 171)

`nextfoot`
v3.08
: page foot using page style `headings` or `myheadings` (section 4.13, page 182)

`nexthead`
v3.08
: page head using page style `headings` or `myheadings` (section 4.13, page 182)

`phoneseparator`
: separator between title of telephone and telephone number (section 4.10, page 159)

`place`
: sender's place used near date (section 4.10, page 164)

`placeseparator`
: separator between place and date (section 4.10, page 172)

`PPdatamatrix`
: instruction, that print the data array of option `addrfield=PP` (section 4.10, page 164)

`PPcode`
: instructions for the sender's identification code of option `addrfield=PP` (section 4.10, page 164)

...

Table 4.1.: Alphabetical list of all supported variables in scrlttr2 (*continuation*)

`signature`
: signature beneath the ending of the letter (section 4.20, page 183)

`specialmail`
: mode of dispatch (section 4.10, page 164)

`subject`
: letter's subject (section 4.10, page 174)

`subjectseparator`
: separator between title of subject and subject (section 4.10, page 174)

`title`
: letter title (section 4.10, page 173)

`toaddress`
: address of addressee without addressee name (section 4.10, page 164)

`toname`
: complete name of addressee (section 4.10, page 164)

`yourmail`
: date of addressee's referenced mail (section 4.10, page 171)

`yourref`
: addressee's reference (section 4.10, page 171)

`zipcodeseparator`
: separator between the zip code's or postcode's title and the code itself (section 4.10, page 164)

`\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 17.3, page 305 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 186 onward), but used in another `lco`-file if it exists only.

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

LaTeX processes length with commands `\newlength`, `\setlength`, `\addtolength`, and `\the`*length*. Many packages also use macros, that are commands, to storage 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-Scriptcalls this kind of lengths, that are stored at macros instead of real LaTeX lengths, pseudo-lengths.

A list of all pseudo-lengths in scrlttr2 is shown in table 17.1 starting at page 291. The meaning of the various pseudo-lengths is shown graphically in figure 17.1. The dimensions used in the figure correspond to the default settings of scrlttr2. 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 17.1, page 295. 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 295.

`\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 186).

`\setlengthtoplength[`*factor*`]{`*length*`}{`*pseudo-length*`}`
`\addtolengthplength[`*factor*`]{`*length*`}{`*pseudo-length*`}`

While you can simply prepend a factor to a length, this is not possible with pseudo-lengths. Suppose you have a length `\test` with the value 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 `\addtolengthplength` 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.

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 scrlttr2 since version 2.9t.

4.5. Draft Mode

What is written in section 3.3 applies, mutatis mutandis.

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 23 onwards.

Normally it makes no sense to distinguish letters with single-side layout and letters with double-side layout. Mostly letters won't be 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 48.

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

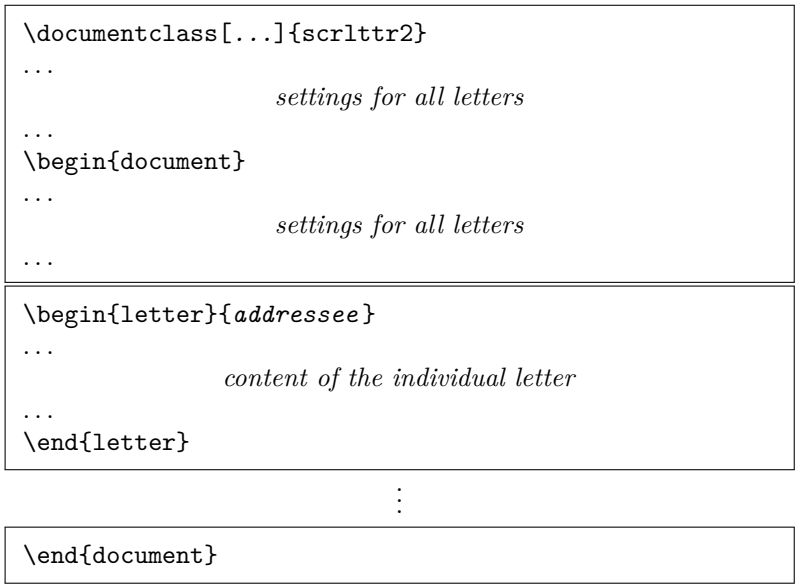


Figure 4.1.: General structure of a letter document with several individual letters (the structure of a single letter is shown in figure 4.2)

changed within a single letter is compatibility to prior versions of scrlttr2 (see option `version` in section 4.4, page 28).

It is recommended that only general settings such as the loading of packages and the setting of options be placed before `\begin{document}`. All settings that comprise the setting of variables or other text features should be done after `\begin{document}`. This is particularly recommended when the babel package (see [Bra01]) is used, or language-dependent variables of scrlttr2 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, and the closing. In the introduction, all settings pertaining only to the current letter are defined. It is important that this introduction always ends with `\opening`. Similarly, the closing always starts with `\closing`. The two arguments *`opening`* and *`closing`* can be left empty, but both commands must be used and must have an argument.

It should be noted that several settings can be changed between the individual letters. Such changes then have an effect on all subsequent letters. For reasons of maintainability of your letter documents, it is however not recommended to use further general settings with limited scope between the letters.

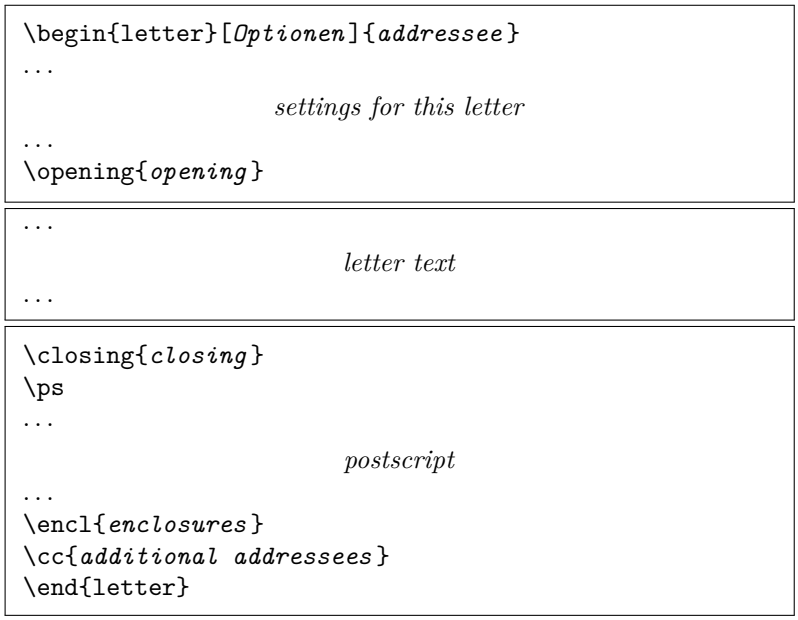


Figure 4.2.: General structure of a single letter within a letter document (see also figure 4.1)

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

The `letter` environment is one of the key environments of the letter class. A special scrlttr2 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]{scrlttr2}
\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 wouldn't be an addressee at the note paper sheet. The reason for this will be explained at the description of command \opening at page 139.

\AtBeginLetter{*instruction code*}
\AtEndLetter{*instruction code*}

LaTeX 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 LaTeX kernel. The class scrlttr2 provides two more hooks. The *instruction code* for these may be declared using \AtBeginLetter and
v2.95 \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.

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 scrlttr2, 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}{l@{}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 [Car99c]).

\opening{*opening*}

This is one of the most important commands in scrlttr2. 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 138 by an opening:

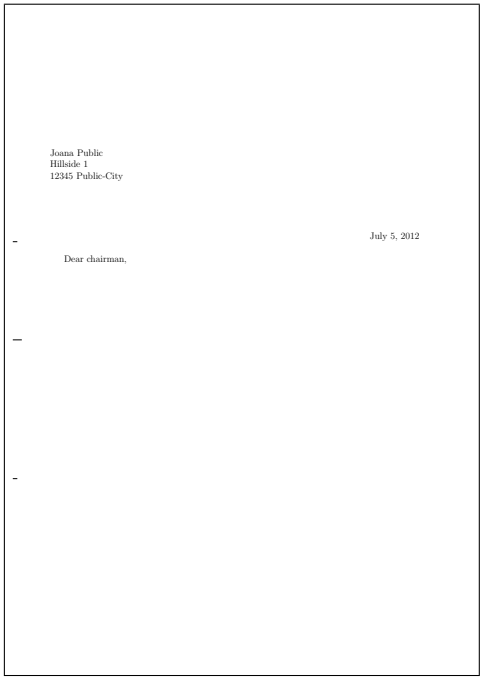


Figure 4.3.: result of a minimalistic letter with addressee and opening only (date and folding marks are defaults of DIN-letters)

```
\documentclass[version=last]{scrlttr2}
\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.

`\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 183.

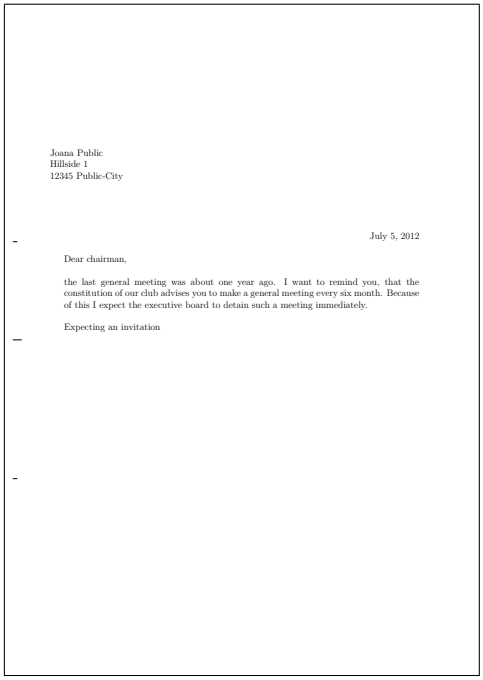


Figure 4.4.: result of a small letter with addressee, opening, text, and closing (date and folding marks are defaults of DIN-letters)

Example: Let's extend the our example by some lines of text and a closing phrase:

```
\documentclass[version=last]{scrlttr2}
\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}
\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 scrlttr2.

Example: The example letter extended by a postscript

```
\documentclass[version=last]{scrlttr2}
\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.
\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*]{*content*}

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.

Figure 4.5.: result of a small letter with addressee, opening, text, closing, and postscript (date and folding marks are defaults of DIN-letters)

Example: This time, the example letter should be send not only to the chairman, but also to all club members:

```
\documentclass[version=last]{scrlttr2}
\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}
```

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)

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]{content}
```

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 also, because there is only one enclosure and the default may be prepared for several enclosures:

```
\documentclass[version=last]{scrlttr2}
\usepackage[english]{babel}
\begin{document}
\begin{letter}{%
    Joana Public\\
    Hillside 1\\
    12345 Public-City%
  }
\opening{Dear chairman,}
```

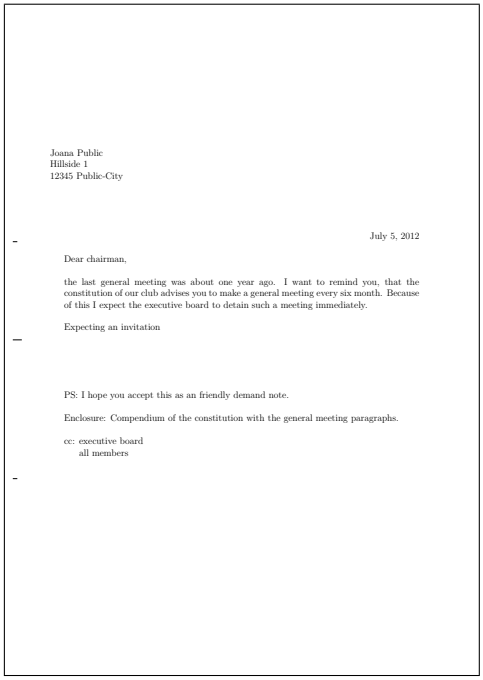


Figure 4.7.: result of a small letter with addressee, opening, text, closing, postscript, distribution list, and enclosure (date and folding marks are defaults of DIN-letters)

```
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 result in figure 4.7.

4.8. Selection of Document or Letter 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 TEX 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 35). 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 depend on those packages. This may even result in error messages or typesetting errors, which cannot be considered a fault of KOMA-Script.

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 scrlttr2 is `fontsize=12pt`.

Example: Assumed, the example is a letter to *"The friends of insane font sizes"* and therefor 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]{scrlttr2}
\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]{scrlttr2}
\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. 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 51.

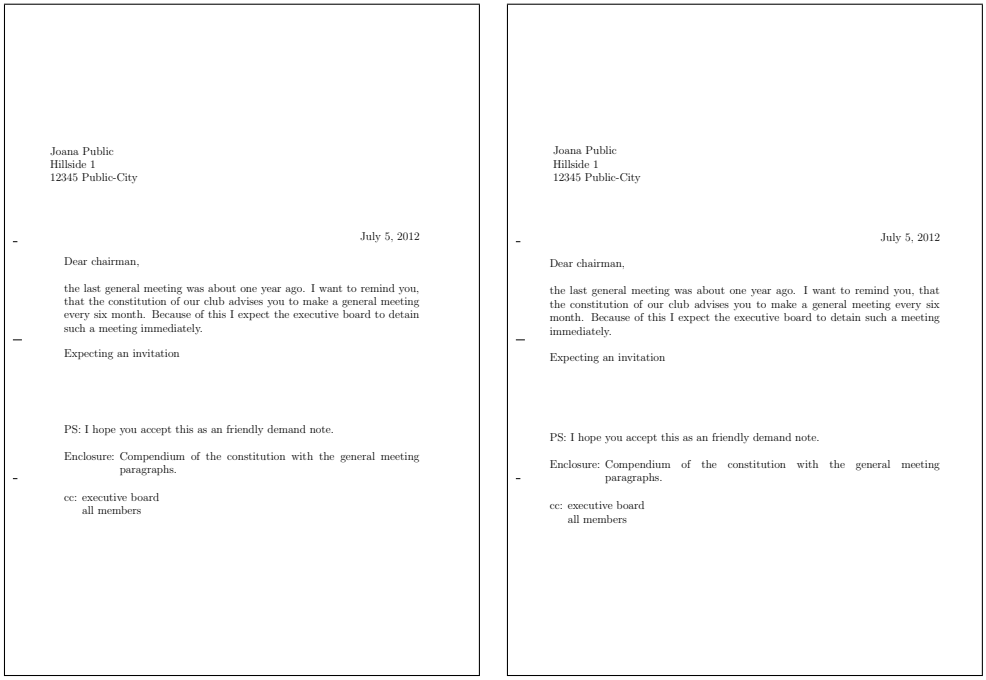


Figure 4.8.: result of a small letter with addressee, opening, text, closing, postscript, distribution list, enclosure, and insane large font size (date and folding marks are defaults of DIN-letters): at left one the font size has been defined by the optional argument of `letter`, at the right one the optional argument of `\documentclass` has been used

Table 4.2.: Alphabetical list of elements whose font can be changed in scrlttr2 using the commands `\setkomafont` and `\addtokomafont`

`addressee`	name und address in address field (section 4.10, page 164)
`backaddress`	return address for a window envelope (section 4.10, page 164)
`descriptionlabel`	label, i. e., the optional argument of `\item`, in a `description` environment (section 4.16, page 101)
`foldmark`	foldmark on the letter page; intended for color settings (section 4.10, page 150)

...

Table 4.2.: Elements whose font can be changed (*continuation*)

`footnote`
: footnote text and marker (see section 4.15, page 76)

`footnotelabel`
: mark of a footnote; used according to the element `footnote` (see section 4.15, page 76)

`footnotereference`
: footnote reference in the text (see section 4.15, page 76)

`footnoterule`
v3.07
: horizontal rule above the footnotes at the end of the text area (see section 3.14, page 79)

`labelinglabel`
: labels, i. e., the optional argument of `\item` in the `labeling` environment (see section 4.16, page 102)

`labelingseparator`
: separator, i. e., the optional argument of the `labeling` environment; used according to the element `labelinglabel` (see section 4.16, page 102)

`pagefoot`
: used after element `pageheadfoot` for the page foot, that has been defined with variable `nextfoot`, or for the page foot of package scrpage2 (chapter 5, page 201)

`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 180)

`pagenumber`
: page number in the header or footer (see section 4.13, page 180)

`pagination`
: another name for `pagenumber`

`refname`
: description or title of the fields in the reference line (section 4.10, page 171)

...

Table 4.2.: Elements whose font can be changed (*continuation*)

`refvalue`	content of the fields in the reference line (section 4.10, page 171)
`specialmail`	mode of dispatch in the address field (section 4.10, page 164)
`subject`	subject in the opening of the letter (section 4.10, page 174)
`title`	title in the opening of the letter (section 4.10, page 173)
`toaddress`	variation of the element `addressee` for setting the addressee address (less the name) in the address field (section 4.10, page 164)
`toname`	variation of the element `addressee` for the name (only) of the addressee in the address field (section 4.10, page 164)

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 the destined fields instantaneously. The independent position is provided by pseudo-lengths (see section 4.2 from page 135 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

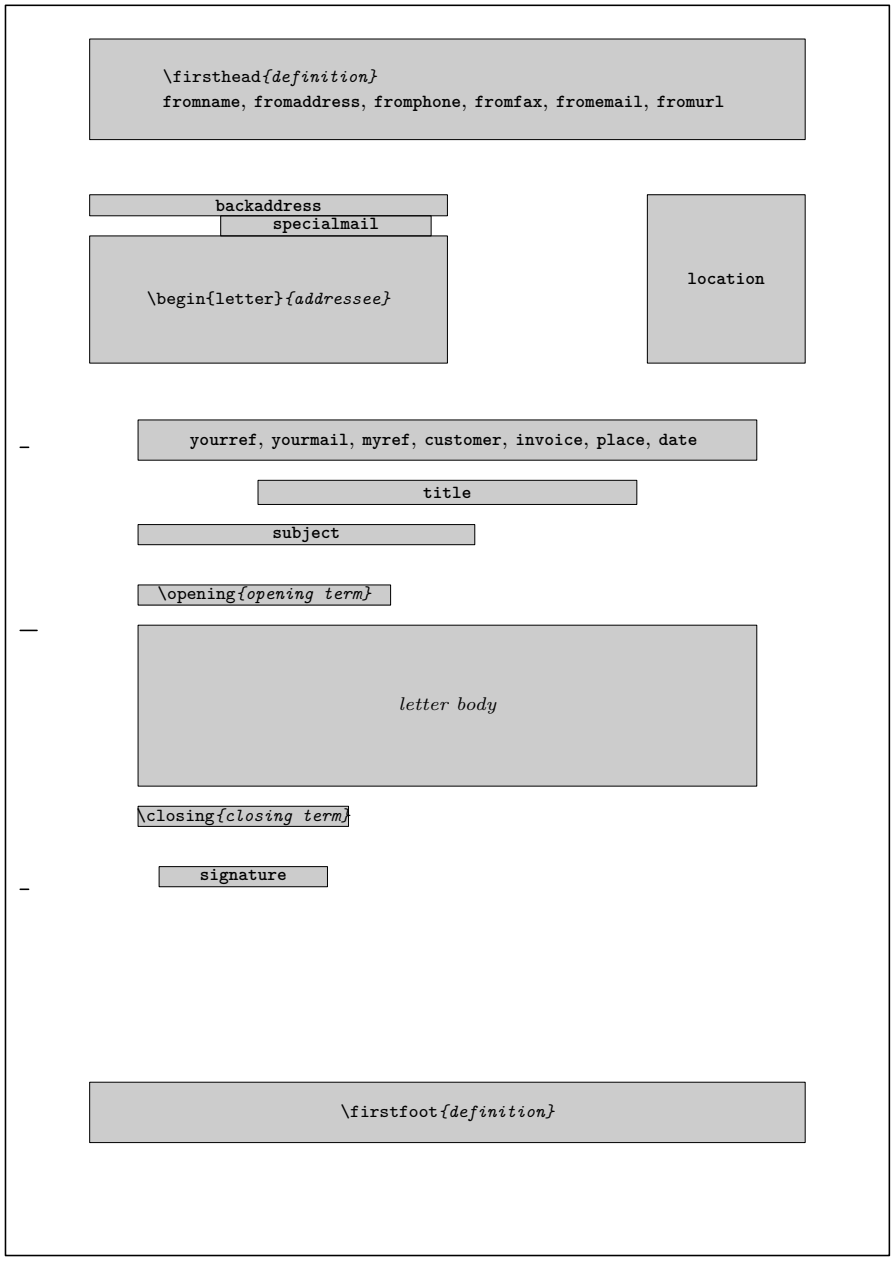


Figure 4.9.: schematic display of the note paper with the most important commands and variables for the drafted elements

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

in equal-sized parts. The position of the four horizontal and the single vertical marks are configurable via pseudo-lengths (see section 17.1.1 from page 298 onwards).

The user has a choice: Either one may use the standard values for simple switches, as described in table 2.5, page 37, 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 186 onward) chosen for a letter. Default values are `true` and `TBMPL`.

v2.97e

Example: Assume that you would like to deactivate all folding marks except the punching mark. This you can accomplish with, for example:

```
\KOMAoption{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]{scrlttr2}
\usepackage[english]{babel}
\begin{document}
\begin{letter}{%
    Joana Public\\
    Hillside 1\\
    12345 Public-City%
```

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)

```
  }
\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 51) with element `foldmark`. Default is not change.

`enlargefirstpage=`*simple switch*

The first page of a letter always uses a different page layout. The scrlttr2 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 on the first page. See also the description of the pseudo-length `firstfootvpos` on page 305. This option can take the standard values for simple switches, as listed in table 2.5, page 37. 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 37. Default is for the letterhead to be set.

`fromalign=`*method*

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

v2.97e

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`.

`fromrule=`*position*
`\setkomavar{fromname}[`*description*`]{`*content*`}`
`\setkomavar{fromaddress}[`*description*`]{`*contents*`}`

The sender's name will be determined by variable `fromname`. Thereby the *description* (see also table 4.6, page 160) 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 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`

Table 4.4.: Available values for option `fromalign` to define the position of the from address in the letterhead with scrlttr2

`center`, `centered`, `middle` return address centered; an optional logo will be above the extended return address; letterhead extensions will be activated
`false`, `no`, `off` standard design will be used for the return address; the letterhead extensions are deactivated
`left` left-justified return address; an optional logo will be right justified; letterhead extensions will be activated
`locationleft`, `leftlocation` 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 `firsthead`.
`locationright`, `rightlocation`, `location` 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 `firsthead`.
`right` right-justified return address; an optional logo will be left justified; letterhead extensions will be activated

Table 4.5.: Possible values of option `fromrule` for the position of the rule in the from address with scrlttr2

`afteraddress`, `below`, `on`, `true`, `yes` rule below the return address
`aftername` rule directly below the sender's name
`false`, `no`, `off` no rule

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]{scrlttr2}
\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
    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 183 onward.

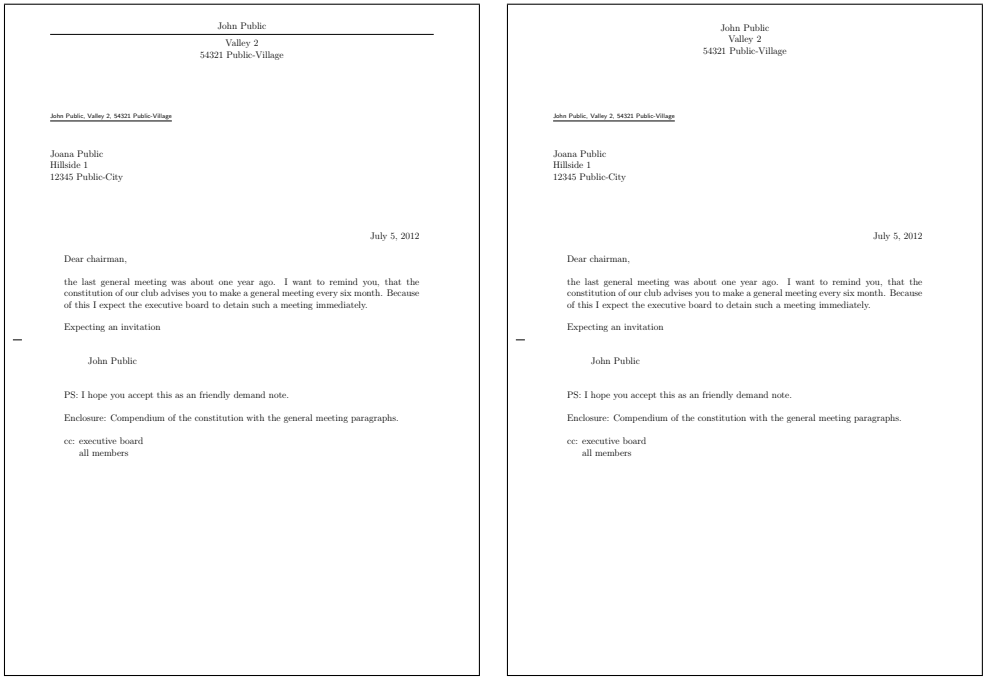


Figure 4.11.: result of a small letter with sender, addressee, opening, text, closing, postscript, distribution list, and enclosure (date and folding marks are defaults of DIN-letters): at left one the standard letterhead using `fromalign=false`, at right one the extended letterhead using `fromalign=center`

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]{scrlttr2}
\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.
```

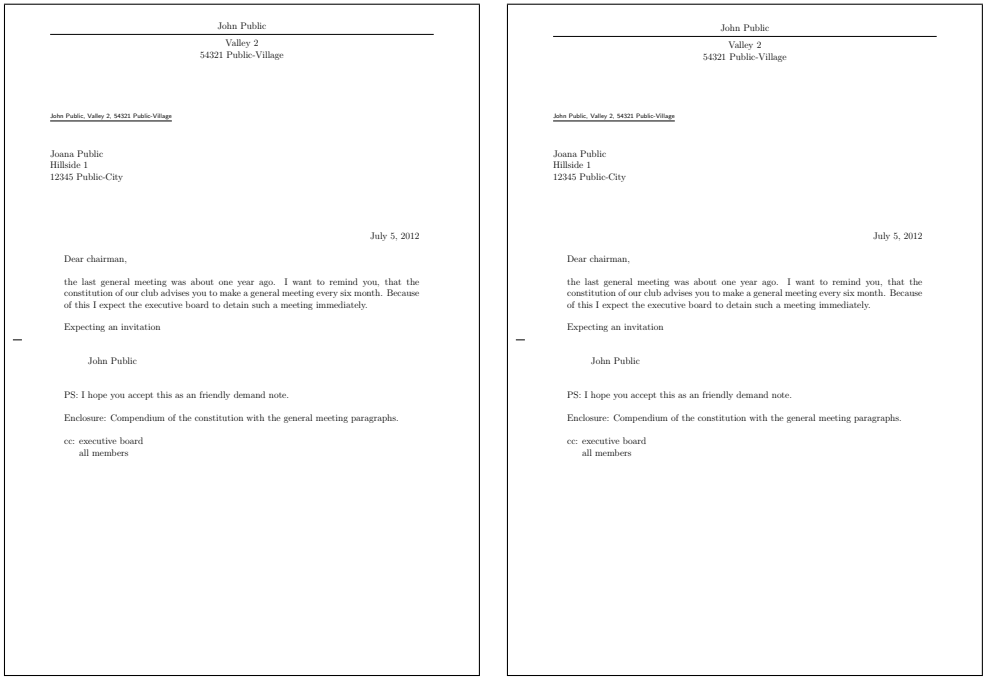


Figure 4.12.: result of a small letter with sender, separation rule, addressee, opening, text, closing, signature, postscript, distribution list, enclosure and puncher hole mark (the date is a default of DIN-letters): at left one the standard letterhead using `fromalign=false`, at right one the extended letterhead using `fromalign=center`

```
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 doesn't 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 scrlttr2 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 164) and signature (see variable `signature`, page 183) oneself.

```
fromphone=simple switch
fromfax=simple switch
fromemail=simple switch
fromurl=simple switch
\setkomavar{fromphone}[description]{content}
\setkomavar{fromfax}[description]{content}
\setkomavar{fromemail}[description]{content}
\setkomavar{fromurl}[description]{content}
\setkomavar{phoneseparator}[description]{content}
\setkomavar{faxseparator}[description]{content}
\setkomavar{emailseparator}[description]{content}
\setkomavar{urlseparator}[description]{content}
```

Whether or not a telephone number, a fax number, an e-mail address, or a sender's URL should be set as part of the letterhead may be configured by the options `fromphone`, `fromfax`, `fromemail`, and `fromurl`. Any standard value for simple switches from table 2.5, page 37 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.

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]{scrlttr2}
\usepackage[english]{babel}
\begin{document}
\setkomavar{fromname}{John Public}
\setkomavar{fromaddress}{Valley 2\\
                         54321 Public-Village}
```

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 \usekomavar*{emailseparator}\usekomavar{emailseparator}
fromfax \usekomavar*{faxseparator}\usekomavar{faxseparator}
fromname \headfromname
fromphone \usekomavar*{phoneseparator}\usekomavar{phoneseparator}
fromurl \usekomavar*{urlseparator}\usekomavar{urlseparator}

```
\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}
```

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.

Table 4.7.: Predefined description and content of the separators at the letterhead

variable name	description	content
emailseparator	\emailname	:~
faxseparator	\faxname	:~
phoneseparator	\phonename	:~
urlseparator	\wwwname	:~

```
\documentclass[foldmarks=true,foldmarks=blmtP,
  fromalign=center,fromrule=afteraddress,
  fromphone,fromemail,
  version=last]{scrlttr2}
\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.

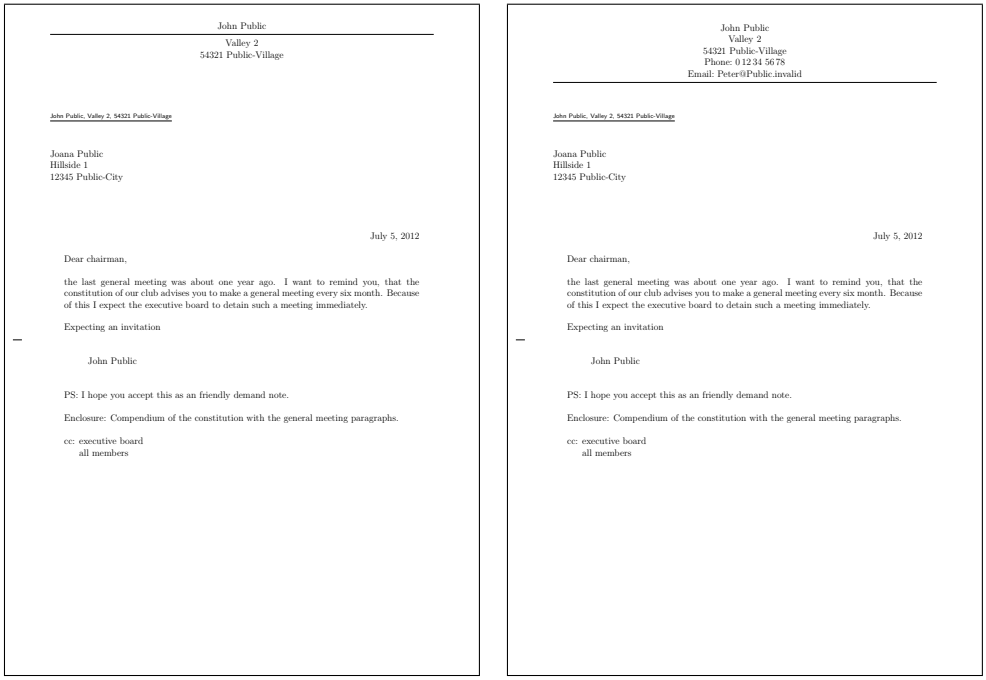


Figure 4.13.: result of a small letter with sender, separation rule, addressee, opening, text, closing, signature, postscript, distribution list, enclosure and puncher hole mark (the date is a default of DIN-letters): at left one the standard letterhead using `fromalign=false`, at right one the extended letterhead using `fromalign=center`

`fromlogo=`*`simple switch`*
`\setkomavar{fromlogo}[`*`description`*`]{`*`content`*`}`

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 37. 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 won't 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 [Car99d]).

```
\documentclass[foldmarks=true,foldmarks=blmtP,
  fromrule=afteraddress,
  fromphone,fromemail,fromlogo,
  version=last]{scrlttr2}
\usepackage[english]{babel}
\usepackage{graphics}
```

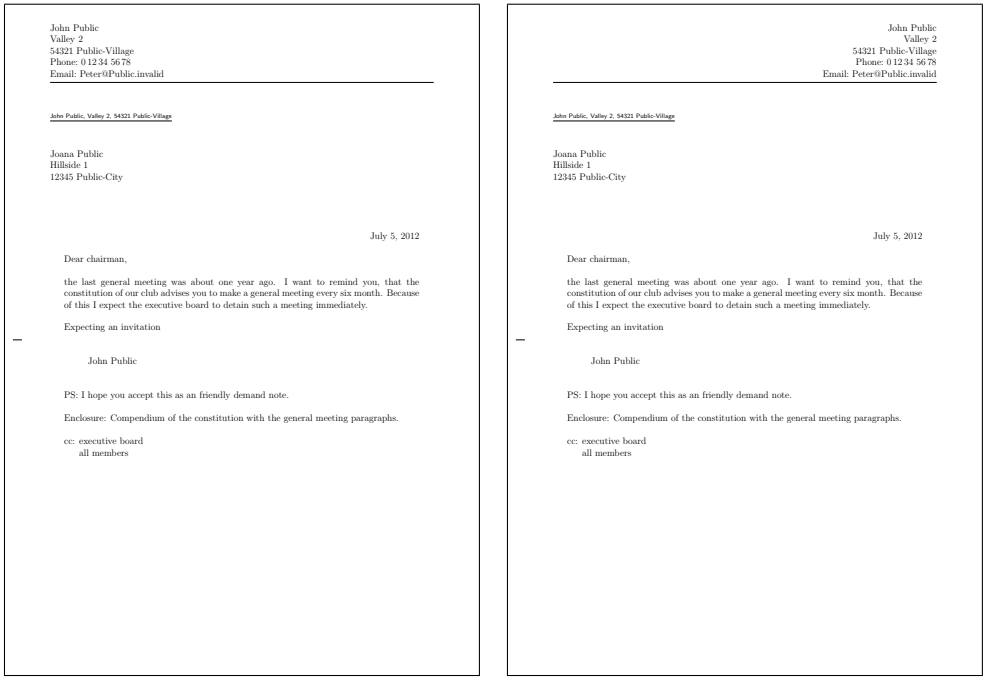


Figure 4.14.: result of a small letter with sender, separation rule, addressee, opening, text, closing, signature, postscript, distribution list, enclosure and puncher hole mark (the date is a default of DIN-letters): at left one left aligned letterhead using `fromalign=left`, at right one right aligned letterhead using `fromalign=right`

```
\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
    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*`]{`*content*`}`

For many cases, scrlttr2 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 doesn't use the *description* of variable `firsthead`.

The command `\firsthead` exists only for reason of compatibility to former scrlttr2 versions. However it is deprecated and you should not use it anymore.

```
addrfield=mode
backaddress=value
priority=priority
\setkomavar{toname}[description]{content}
\setkomavar{toaddress}[description]{content}
\setkomavar{backaddress}[description]{content}
\setkomavar{backaddressseparator}[description]{content}
\setkomavar{specialmail}[description]{content}
\setkomavar{fromzipcode}[description]{content}
\setkomavar{zipcodeseparator}[description]{content}
\setkomavar{place}[description]{content}
\setkomavar{PPcode}[description]{content}
\setkomavar{PPdatamatrix}[description]{content}
\setkomavar{addresseeimage}[description]{content}
```

v3.03

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

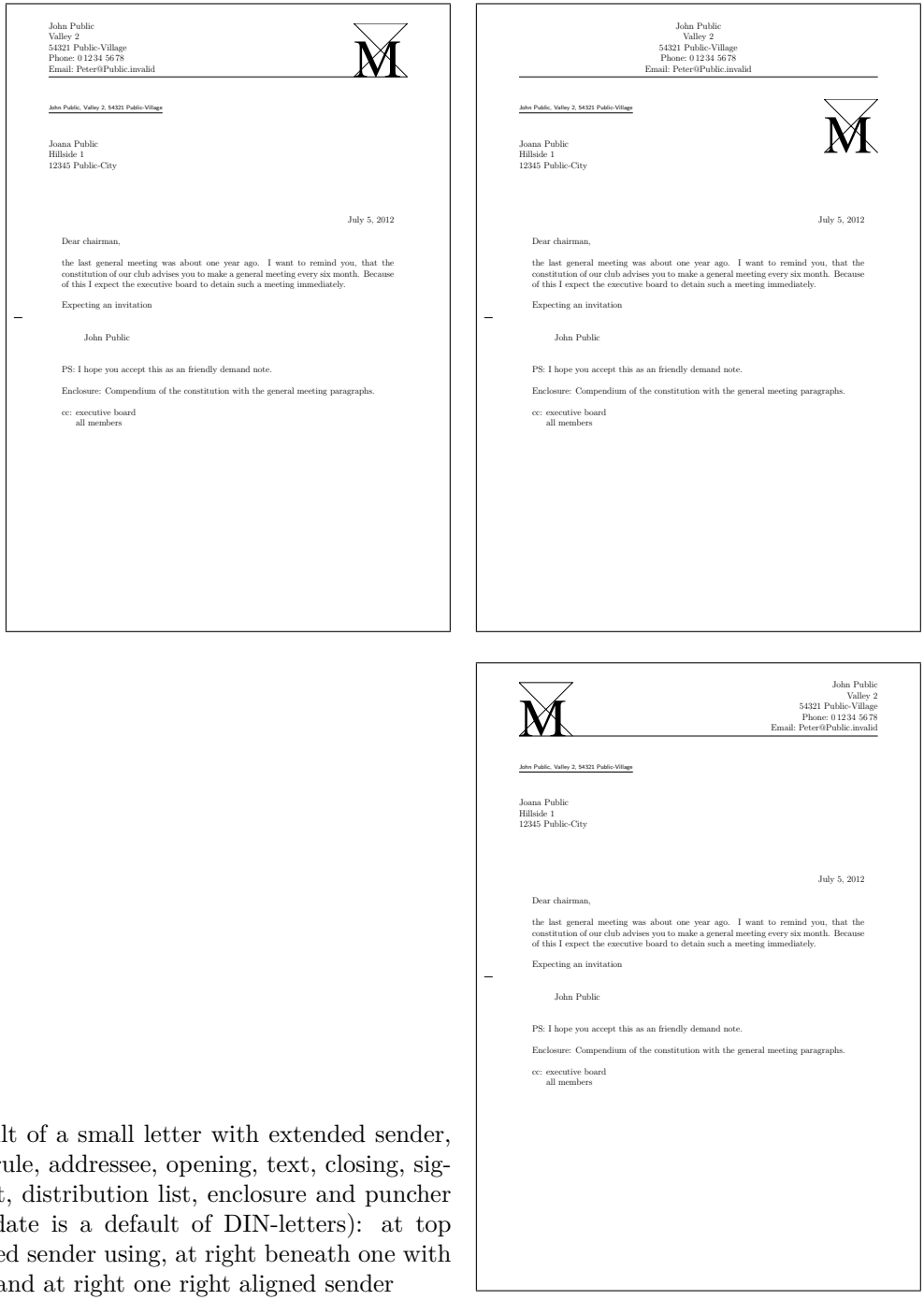


Figure 4.15.: result of a small letter with extended sender, logo, separation rule, addressee, opening, text, closing, signature, postscript, distribution list, enclosure and puncher hole mark (the date is a default of DIN-letters): at top left one left aligned sender using, at right beneath one with centered sender, and at right one right aligned sender

Table 4.8.: available values for option `addrfield` to change the addressee mode of scrlttr2

`true`, `on`, `yes`	Prints an address field including a return address and a mode of dispatch or priority.
`false`, `off`, `no`	Omits the address field.
`PP`, `pp`, `PPexplicite`, `PPExplicite`, `ppexplicite`, `ppExplicite`	Prints an address field with Port-Payé head, defined by the variables `fromzipcode`, `place`, and `PPcode` and when indicated with priority and data array defined by `PPdatamatrix` but without return address and mode of dispatch.
`backgroundimage`, `PPbackgroundimage`, `PPBackgroundImage`, `PPBackGroundImage`, `ppbackgroundimage`, `ppBackgroundImage`, `ppBackGroundImage`	Prints an address field with Port-Payé head, in variable `addresseimage` defined background graphics, but without return address or mode of dispatch.
`image`, `Image`, `PPimage`, `PPImage`, `ppimage`, `ppImage`	Prints variable `addresseeimage` as addressee with Port-Payé, but ignores addressee information and definitions for return address, mode of dispatch or priority.

`true`, `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 138). These 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 51). 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.

v2.97c

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 37. These values don't 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.

v2.96

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 `backaddressseparator`. The predefined separator *`content`* is a comma followed by a non-breakable white space. The *`description`* of variable `backaddress`

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	

isn't 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.

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 scrlttr2 itself doesn't use the *description* of variable specialmail. The alignment is defined by the pseudo-lengths PLengthspecialmailindent and specialmailrightindent (siehe page 301). 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 51).

v3.03 v2.97c

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 default font styles are listed in table 4.9 and may be changed using commands \setkomafont und \addtokomafont (siehe section 4.9, page 51).

v3.03

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 (see section 4.21 from page 186 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

Table 4.10.: available values for option `priority` to select the international priority at the address field ofscrlttr2

`false`, `off`, `no`, `manual`	Priority will not be printed.
`B`, `b`, `economy`, `Economy`, `ECONOMY`, `B-ECONOMY`, `B-Economy`, `b-economy`	Use international priority B-Economy. With `addrfield=true` this will be printed instead of the mode of dispatch.
`A`, `a`, `priority`, `Priority`, `PRIORITY`, `A-PRIORITY`, `A-Priority`, `a-priority`	Use international priority A-Priority. With `addrfield=true` this will be printed instead of the mode of dispatch.

for this.

v3.03 Zip-code, place and code will be printed with default font size 8 pt. 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 51).

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 doesn't 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 301) as well as `PPheadwidth` and `PPheadheight` (siehe page 301). The alignment of the data array is defined by pseudo-length `PPdatamatrixvskip` (see page 302).

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.

`locfield=`*`selection`*

scrlttr2 places a field with additional sender attributes next to the address field. This can be used, for example, for bank account or similar additional information. Depending on the `fromalign` option, it will also be used for the sender logo. The width of this field may be defined within an `lco` file (see section 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 17.1.4, page 302. Possible values for this option are shown in table 4.11. Default is `narrow`.

Table 4.11.: Possible values of option `locfield` for setting the width of the field with additional sender attributes with scrlttr2

`narrow`	narrow sender supplement field
`wide`	wide sender supplement field

`\setkomavar{location}[`*description*`]{`*content*`}`

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 doesn't use the *description* of the variable.

Example: Mr Public likes to show some additional information about his membership. Therefor he uses the `location` variable:

```
\documentclass[foldmarks=true,foldmarks=blmtP,
  fromphone,fromemail,fromlogo,
  version=last]{scrlttr2}
\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 nember no.~4711\\
  since 11.09.2001\\
  chairman 2003--2005}
\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.
```


John Public
Valley 2
54321 Public-Village
Phone: 0 12 34 56 78
Email: Peter@Public.invalid

John Public, Valley 2, 54321 Public-Village

Club member no. 4711
since 11.09.2001
chairman 2003–2005

Joana Public
Hillside 1
12345 Public-City

July 5, 2012

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.

Expecting an invitation

John Public

PS: I hope you accept this as an friendly demand note.

Enclosure: Compendium of the constitution with the general meeting paragraphs.

cc: executive board
all members

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)

```
\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 scrlttr2 itself. This option can take the standard values for simple switches, as listed in table 2.5, page 37. Default is `false`, which results in standard date presentation.

`\setkomavar{date}[`*description*`]{`*content*`}`

The date in the selected presentation will become the *content* of variable `date`. The selection of option `numericaldate` doesn't 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. This is the case

Table 4.12.: Possible value of option `refline` for setting the width of the reference fields line with scrlttr2

`dateleft`	
v3.09	The date will be placed leftmost at the reference line.
`dateright`	
v3.09	The date will be placed rightmost at the reference line.
`narrow`	The reference line will be restricted to type area.
`nodate`	
v3.09	The date won't be placed automatically into the reference line.
`wide`	The with of the reference line corresponds to address and sender's additional information.

even if all other elements of the reference line will be empty and therefore unused. For more information about the automatic printing of the date may be found in following description of option `refline`.

`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 scrlttr2 class, the header, footer, address, and sender attributes may extend beyond the normal type area to the left and to the right. Option `reflinewide` 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.
v3.09 Default is `narrow` and `dateright`.

```
\setkomavar{yourref}[description]{content}
\setkomavar{yourmail}[description]{content}
\setkomavar{myref}[description]{content}
\setkomavar{customer}[description]{content}
\setkomavar{invoice}[description]{content}
```

These five variables represent typical fields of the reference line. Their meanings are given in table 4.1 on page 130. 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 17.3 from page 305 onward.

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
yourref	\yourrefname	Your reference
yourmail	\yourmailname	Your letter from
myref	\myrefname	Our reference
customer	\customername	Customer No.:
invoice	\invoicename	Invoice No.:
date	\datename	date

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` and `\addtokomafont` (see section 4.9, page 51) should be used. The default configuration of both elements is listed in table 4.14.

```
\setkomavar{place}[description]{content}
\setkomavar{placeseparator}[description]{content}
```

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 170).

Example: Now Mr Public also sets the place:

```
\documentclass[foldmarks=true,foldmarks=blmtP,
  fromphone,fromemail,fromlogo,
  version=last]{scrlttr2}
\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 nember no.~4711\\
  since 11.09.2001\\
  chairman 2003--2005}
\setkomavar{date}{29th February 2011}
\setkomavar{place}{Public-Village}
```

Table 4.14.: default font style configuration of the elements of the reference line

element	default configuration
`refname`	`\sffamily\scriptsize`
`refvalue`	

```
\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}
```

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*`]{`*content*`}`

With scrlttr2 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 `title` can be changed with commands `\setkomafont` und `\addtokomafont` (see section 4.9, page 51). 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.

Example: Assume that you are to write a reminder. Thus you put as title:

```
\setkomavar{title}{Reminder}
```

This way the addressee will recognize a reminder as such.

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

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]{content}
\setkomavar{subjectseparator}[description]{content}
```

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

Table 4.15.: predefined descriptions of subject-related variables

variable name	description
`subject`	`\usekomavar*{subjectseparator}%` `\usekomavar{subjectseparator}`
`subjectseparator`	`\subjectname`

Table 4.16.: available values of option `subject` for the position and formatting of the subject with scrlttr2

`afteropening`	subject after opening
`beforeopening`	subject before opening
`centered`	subject centered
`left`	subject left-justified
`right`	subject right-justified
`titled`	add title/description to subject
`underlined`	set subject underlined (see note in text please)
`untitled`	do not add title/description to subject

v2.97c

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 51). or the element `subject` the predetermined font in scrlttr2 is `\normalcolor\bfseries`.

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]{scrlttr2}
\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 nember 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}
```

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 154) 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 37. Default is the setting of the letter foot.

`\setkomavar{firstfoot}[`*description*`]{`*content*`}`

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 doesn't use the *description* of the variable.

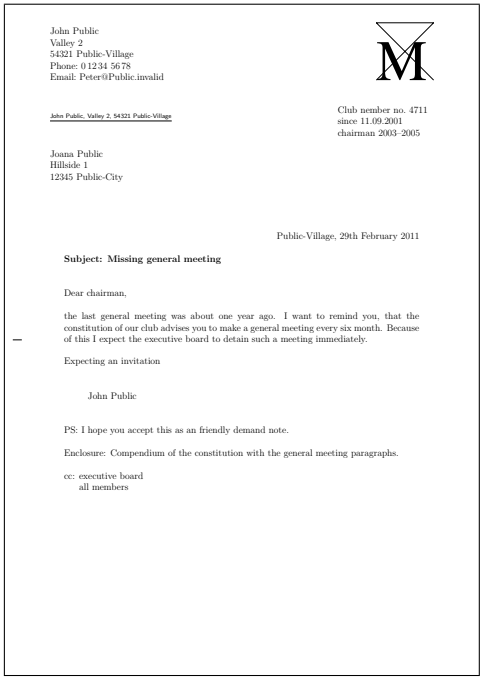


Figure 4.18.: result of a small letter with extended sender, logo, addressee, additional sender information, place, date, subject opening, text, closing, signature, postscript, distribution list, enclosure and puncher hole mark

For more information see the example following the next description. Only for compatibility reason the deprecated command `\firstfoot` of scrlttr2 before release 3.08 still exists. Nevertheless it shouldn't be used any longer.

`\setkomavar{frombank}[`*description*`]{`*content*`}`

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 169) 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}%
    \ifkomavarempty{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 28) the space will in general not suffice. In that case, you may need to reduce `firstfootvpos` (see page 305) appropriately.

4.11. Paragraph Markup

In the preliminaries of section 3.10, page 65 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* needn't 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.

4.12. Detection of Odd and Even Pages

What is described in section 3.11 applies, mutatis mutandis.

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.

`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 37. 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 of the corresponding rules.

Obviously option `headsepline` doesn't influence page styles `empty` and `plain` (see afterwards at this section). These page styles explicitly don't use any page head.

Typographically such a rule make a hard optical connection of head or foot and the text area. This wouldn't 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 scrpage2 (see chapter 5) provides additional features for rules at head and foot and may be combined with scrlttr2.

`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 scrpage2 package, if set before loading the package (see chapter 5). It can

take values only influencing horizontal, only vertical, or both positions. Available value are shown in table 4.17. Default is `botcenter`.

```
\pagestyle{page style}
\thispagestyle{local page style}
```

In letters written with scrlttr2 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 139).

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 154 and page 174). At which position these 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 scrpage2 also `\markleft` (see section 5.1.2, page 200) 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 scrpage2 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 v3.00 `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 scrpage2 (see chapter 5, page 201). The element for the page number within the header or footer is named `pagenumber`. Default settings are listed in table 3.8, page 69. Please have also a look at the example in section 3.12, page 69.

Table 4.17.: available values of option `pagenumber` for the position of the page number in page styles `headings`, `myheadings`, and `plain` with scrlttr2

`bot`, `foot` page number in footer, horizontal position not changed
`botcenter`, `botcentered`, `botmittle`, `footcenter`, `footcentered`, `footmiddle` page number in footer, centered
`botleft`, `footleft` page number in footer, left justified
`botright`, `footright` page number in footer, right justified
`center`, `centered`, `middle` page number centered horizontally, vertical position not changed
`false`, `no`, `off` no page number
`head`, `top` page number in header, horizontal position not changed
`headcenter`, `headcentered`, `headmiddle`, `topcenter`, `topcentered`, `topmiddle` page number in header, centered
`headleft`, `topleft` page number in header, left justified
`headright`, `topright` page number in header, right justified
`left` page number left, vertical position not changed
`right` page number right, vertical position not changed

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

The possibilities that are offered with variables and options in scrlttr2 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 scrlttr2 uses to create the letterhead. The commands `\markboth` and `\markright` can in particular be used together with pagestyle `myheadings`. If the package scrpage2 is used then this, of course, is valid also for pagestyle `scrheadings`. There the command `\markleft` is furthermore available.

```
\setkomavar{nexthead}[description]{content}
\setkomavar{nextfoot}[description]{content}
```

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 of footers of his own. Within *content* you can and should of course also make use of other variables by using `\usekomavar`. KOMA-Script doesn't use the *description* of both variables.

v3.08

Only for compatibility reason the deprecated commands `\nexthead` and `\nextfoot` of former scrlttr2 releases before 3.08 are also implemented. Nevertheless, you shouldn't use those.

4.14. Interleaf Pages

What is described in section 3.13 applies, mutatis mutandis. But at letters interleaf pages are unusual. This may be benefited by the case, that real two-sided letters are seldom, because binding of letters won't be done often. Nevertheless scrlttr2 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 72 upward.

4.15. Footnotes

What is described in section 3.14 applies, mutatis mutandis.

4.16. Lists

What is described in section 3.18 applies, mutatis mutandis.

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 scrlttr2 doesn't 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 scrlttr2.

You won't 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 [OPHS99]. 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 scrlttr2 doesn't 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 13). Nevertheless, tabulars and pictures or graphics without floating environment may still be done with the letter class scrlttr2.

4.19. Margin Notes

It applies, mutatis mutandis, what is described in section 3.21. Nevertheless, margin notes are unusual at letters and should be used seldomly.

4.20. Closing

That the letter closing will be set by `\closing` has been explained already in section 4.7, page 140. 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*`]{`*content*`}`

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 17.1.7, page 304). 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 91).

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]{scrlttr2}
\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 nember 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\\
```

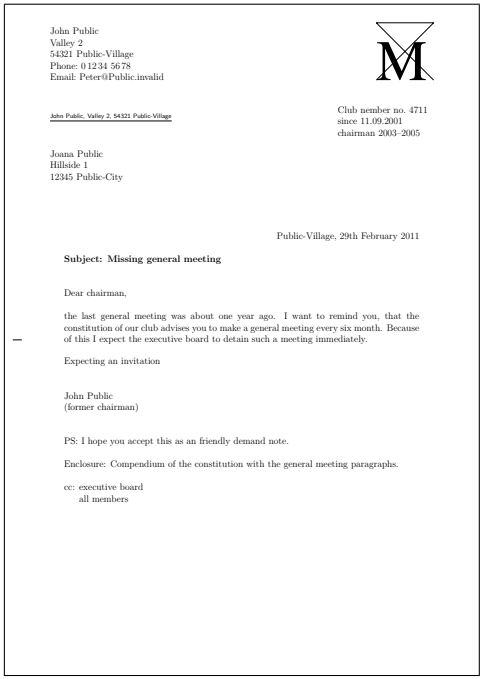


Figure 4.19.: result of a small letter with extended sender, logo, addressee, additional sender information, place, date, subject opening, text, closing, modified signature, postscript, distribution list, enclosure and puncher hole mark

```
    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.

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 scrlttr2 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 scrlttr2, these are in particular the commands `\@newplength`, `\@setplength`, and `\@addtoplength` (see section 17.1).

v3.04

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 189.

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 `lppl.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*`}`

Usually, the `lco`-files will be loaded by the `\documentclass` command. You enter the name of the `lco`-file without suffix as an option. The `lco`-file will be loaded right after the class file.

However, it is also possible to load an `lco`-file later, or even from within another `lco`-file. This may be done with the `\LoadLetterOption` command, which takes the *name* of the `lco`-file without suffix as a parameter.

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{scrlttr2}
```

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.

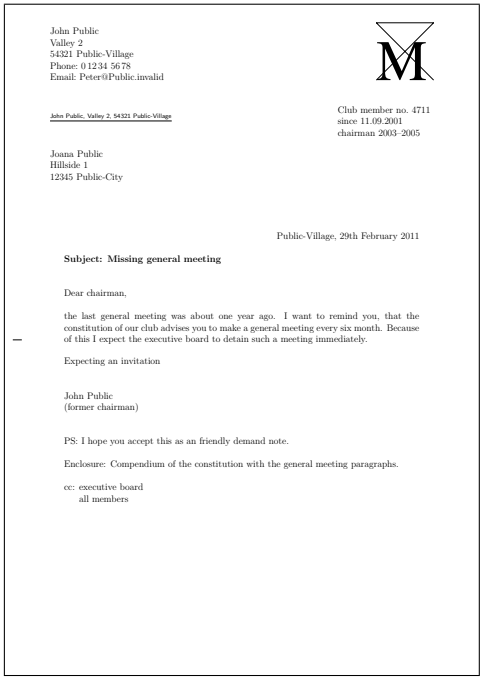
v2.97 If an `lco`-file is loaded via `\documentclass`, then it may no longer have the same name as an option.

Example: Since Mr Public often writes letters with the same options and parameters, he doesn't 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]{scrlttr2}
\usepackage[english]{babel}
\usepackage{graphics}
\begin{document}
\setkomavar{date}{29th February 2011}
```


John Public
Valley 2
54321 Public-Village
Phone: 0 1234 56 78
Email: Peter@Public.invalid

John Public, Valley 2, 54321 Public-Village

Club member no. 4711
since 11.09.2001
chairman 2003–2005

Joana Public
Hillside 1
12345 Public-City

Public-Village, 29th February 2011

Subject: Missing general meeting

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.

Expecting an invitation

John Public
(former chairman)

PS: I hope you accept this as an friendly demand note.

Enclosure: Compendium of the constitution with the general meeting paragraphs.

cc: executive board
all members

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

```
\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}
```

Nevertheless, as shown in figure 4.20, the result doesn't 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 TEX's 7-bit encoding for all characters, e. g., use "\ss" to produce a German "ß".

In table 4.18, page 189 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 171). If you have this kind of problem, create your own lco file that loads SN first and then changes toaddrhpos (see section 17.1.3, page 300) 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 packages from within an lco-file that have already been loaded by the class. Normally, this only applies to the typearea, scrlfile, 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).

...

Table 4.18.: predefined `lco`-files (*continuation*)

`KOMAold`
: parameter set for letters on A4-size paper using a layout close to the now obsolete scrlettr 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 scrlettr commands are defined; scrlttr2 may behave slightly different when used with this `lco`-file than with the other `lco`-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).

`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).

...

Table 4.18.: predefined `lco`-files (*continuation*)

`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).

`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 scrlttr2 provides basic support for this task.

`\adrentry{`*last-name*`}{`*first-name*`}{`*address*`}{`*phone*`}{`*F1*`}{`*F2*`}{`*comment*`}{`*key*`}`

The class scrlttr2 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 scrlttr2 can help. Assume that all your business partners have an entry in your `partners.adr` address file. If you now have to reply to Mr McEnvy again, then you can save typing as follows:

```
\input{partners.adr}
\begin{letter}{\FLANN}
  Your correspondence of today \dots
\end{letter}
```

Your TEX system must be configured to have access to your address file. Without access, the `\input` command results in an error. You can either put your address file in the same directory where you are running LATEX, or configure your system to find the file in a special directory.

`\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 TEX supports maximally nine parameters per command, the comment parameter has fallen away. Other than this difference, the use is the same as that of `\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 TEX.

Example: Suppose you are member of a society and want write an invitation for the next general meeting to all members.

```
\documentclass{scrlttr2}
\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 `\addrentry` commands too, than an additional definition for `\addrentry` is required before loading the address file:

```
\renewcommand*{\addrentry}[9]{%
  \adrentry{#1}{#2}{#3}{#4}{#5}{#6}{#7}{#9}%
}
```

In this simple example the extra freely-definable parameter is not used, and therefore `\addrentry` is defined with the help of `\adrentry`.

With some additional programming one can let the content of the letters depend on the address data. For this the free parameters of the `\adrentry` and and `\addrentry` commands can be used.

Example: Suppose the 5^{th} parameter of the `\adrentry` command contains the gender of a member (`m/f`), and the 6^{th} parameter contains what amount of subscription has not yet been paid by the member. If you would like to write a more personal reminder to each such member, then the next example can help you:

```
\renewcommand*{\adrentry}[8]{
  \ifdim #6pt>0pt\relax
  % #6 is an amount greater than 0.
  % Thus, this selects all members with due subscription.
    \begin{letter}{#2 #1\\#3}
      \if #5m \opening{Dear Mr.\,#2,} \fi
```

```
      \if #5f \opening{Dear Mrs.\,#2,} \fi

      Unfortunately we have to remind you that you have
      still not paid the member subscription for this
      year.

      Please remit EUR #6 to the account of the society.
     \closing{Regards,}
    \end{letter}
   \fi
}
```

As you can see, the letter text can be made more personal by depending on attributes of the letter's addressee. The number of attributes is only restricted by number of two free parameters of the `\adrentry` command, or four free parameters of the `\addrentry` command.

`\adrchar{`*initial letter*`}`
`\addrchar{`*initial letter*`}`

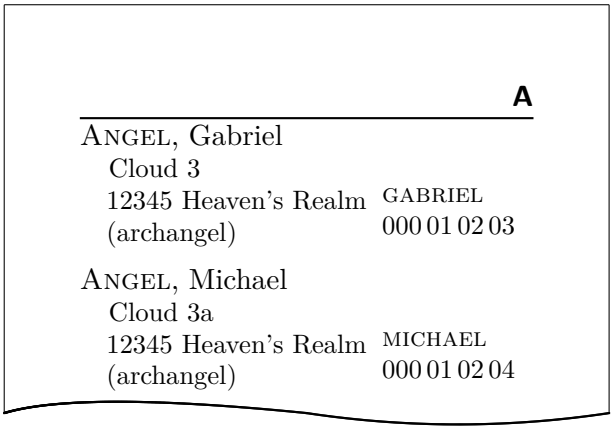
As already mentioned above, it is possible to create address and telephone lists using `adr` files. For that, the additional package adrconv by Axel Kielhorn (see [Kie99]) is needed. This package contains interactive LaTeX documents which help to create those lists.

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 `adrdir.tex` of the adrconv package [Kie99]. The result should look like this:



The letter in the page header is created by the `\adrchar` command. The definition can be found in `adrdir.tex`.

More about the adrconv package can be found in its documentation. There you should also find information about whether the current version of adrconv supports the `\addrentry` and `\addrchar` commands. Former versions only know the commands `\adrentry` and `\adrchar`.

Adapting Page Headers and Footers with scrpage2

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

In place of scrpage2 you can of course make use of fancyhdr (see [vO00]). However, scrpage2 integrated markedly better with the KOMA-Script bundle. For this reason, and because at the time the forerunner to fancyhdr was missing many features, scrpage2 was developed. Naturally, scrpage2 is not limited to use only with the KOMA-Script classes, but can just as easily be used with other document classes.

5.1. Basic Functionality

To understand the following description, an overview of LaTeX's fairly involved header and footer mechanism is needed. The LaTeX kernel defines the page styles `empty`, which produces a completely empty header and footer, and `plain`, which produces usually only a page number in the footer and an empty header. Apart from these, many document classes provide the style `headings`, which allows more complex style settings and running headings. The `headings` style often has a related variant, `myheadings`, which is similar except for switching off the running headings and reverting them to manual control by the user. A more detailed description is given in section 3.12 where it is also noted that some LaTeX commands automatically switch to another page style — usually page style `plain` — for the current page.

Package scrpage2 doesn't distinguish between page styles with automatic, running headings and page styles with manual headings. The way to deal with automatic and manual headings is independent from the page style and so the page style is independent from the choice of automatic or manual headings. More information about this in section 5.1.2.

5.1.1. Predefined Page Styles

One of the basic features of scrpage2 is a set of predefined, configurable page styles.

`scrheadings`
`scrplain`

Package scrpage2 delivers its own page style, named `scrheadings`, which can be activated with the `\pagestyle{scrheadings}`. When this page style is in use, an appropriate `scrplain` page style is used for the plain page style. In this case *appropriate* means that this new plain page style is also configureable by the commands introduced in section 5.1.3, which, for example, configure the header and footer width and complies within the basic layout. Neither

the activation of scrheadings nor the attendant change to the appropriate plain page style, scrplain, influences the mode of manual or automatic headings (see section 5.1.2). The scrplain page style can also be activated directly with \pagestyle.

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

The page style of scrpage2 are defined to have flexible configurable header and footer. To achieve this, the page styles include three boxes in both the header and the footer. The contents of these boxes may be modified easily. The commands modifying the content of these boxes can be seen in figure 5.1. Commands in the middle column modify the box contents on both odd and even pages. All of the commands have an optional and a mandatory argument. The option Argument influences the content of corresponding box of the plain page style, scrplain. The mandatory argument influences the content of the corresponding box of the page style scrheadings.

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

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

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

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

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

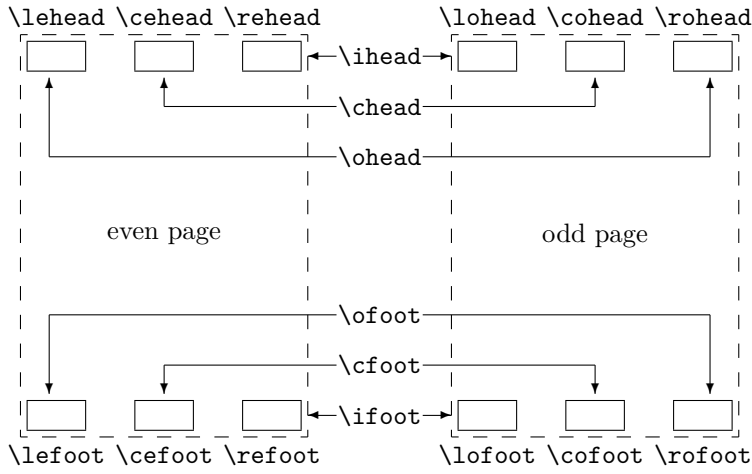


Figure 5.1.: Commands for modification of page styles `scrheadings` and `scrplain` and their relationship to header and footer elements

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

Example: Assuming one has the order to write an annual report of a company, one could use commands like this:

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

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

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

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

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

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

\clearscrheadings
\clearscrplain
\clearscrheadfoot

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

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

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

Without the commands `\clearscrheadfoot`, `\clearscrheadings` and `\clearscrplain`, six commands with additional nine empty arguments would be required:

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

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

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

\leftmark
\rightmark

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

\headmark

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

`\pagemark`

This command returns the formatted page number. The formatting can be controlled by `\pnumfont`, introduced in section 5.1.3, page 201, which `\pagemark` heeds automatically.

`useheadings`

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

5.1.2. Manual and Running Headings

Usually there is a *my*-version of the `headings` page style. If such a page style is active, then the running headings are no longer updated no longer automatically and become manual headings. With scrpage2 a different path is taken. Whether the headings are running or manual is determined by the instructions `\automark` and `\manualmark`, respectively. The default can be set already while loading of the package, with the options `automark` and `manualmark` (see section 5.1.4, page 207).

`\manualmark`

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

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

v2.2

The macro `\automark` activates the automatic updating, that is, running headings. For the two parameters the designations of the document sectioning level whose title is to appear in appropriate place are to be used. Valid values for the parameters are: `part`, `chapter`, `section`, `subsection`, `subsubsection`, `paragraph`, and `subparagraph`. For most of the classes use of `part` will not produce the expected result. So far only KOMA-Script classes from version 2.9s up are known to support this value. The optional argument *`right page`* is understandably meant only for double-sided documents. In the single-sided case one should normally not use it. With the help of the option `autooneside` one can also set that the optional argument in single-sided mode is ignored automatically (see section 5.1.4, page 208).

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

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

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

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

For the upper section level, the data of the headings is set by the command `\markboth` (see section 3.12, page 70), while that for the lower section level by `\markright` or `\markleft`. These commands are called indirectly by the sectioning commands. The macro `\markleft` is provided by the package scrpage2 and is defined similarly to `\markright` (see section 3.12, page 70) in the LaTeX kernel. Although `\markleft` is not defined as an internal command, the direct use is not recommended.

5.1.3. Formatting of Header and Footer

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

`\headfont`
`\footfont`
`\pnumfont`

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

Example: If, for example, one wants the header to be typeset in bold sans serif, the footer in non-bold sans serif, and the page number in a slanted serif style, then one can use the following definitions:

```
\renewcommand{\headfont}{\normalfont\sffamily\bfseries}
\renewcommand*{\footfont}{\normalfont\sffamily}
\renewcommand{\pnumfont}{\normalfont\rmfamily\slshape}
```

From version 2.8p of the KOMA-Script classes a new unified user interface scheme is implemented for font attributes. If scrpage2 is used together with one of these classes, then it is recommended to set up font attributes in the manner described in section 3.6 from page 50 onward.

Example: Instead of `\renewcommand` the command `\setkomafont` should be used to configure the font attributes. The previous definitions can then be written as:

```
\setkomafont{pagehead}\normalfont\sffamily\bfseries}
\setkomafont{pagefoot}{\normalfont\sffamily}
\setkomafont{pagenumber}{\normalfont\rmfamily\slshape}
```

\setheadwidth[*shift*]{*width*}
\setfootwidth[*shift*]{*width*}

Normally the widths of header and footer lines correspond to the width of the text body. The commands \setheadwidth and \setfootwidth enable the user to adapt in a simple manner the widths to his needs. The mandatory argument *width* takes the value of the desired width of the page header or footer, while *shift* is a length parameter by which amount the appropriate item is shifted toward the outside page edge.

For the most common situations the mandatory argument *width* accepts the following symbolic values:

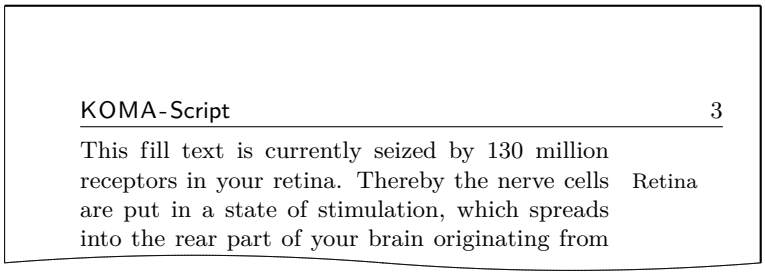
paper	– the width of the paper
page	– the width of the page
text	– the width of the text body
textwithmarginpar	– the width of the text body including margin
head	– the current header width
foot	– the current footer width

The difference between paper and page is that page means the width of the paper less the binding correction if the package typearea is used (see chapter 2). Without typearea both values are identical.

Example: Assume that one wants a layout like that of *The LaTeX Companion*, where the header projects into the margin. This can be obtained with:

```
\setheadwidth[0pt]{textwithmarginpar}
```

which appears like this on an odd page:



If the footer line should have the same width and alignment, then two ways to set this up are possible. The first way simply repeats the settings for the case of the footer line:

```
\setfootwidth[0pt]{textwithmarginpar}
```

In the second way the symbolic value `head` is used, since the header already has the desired settings.

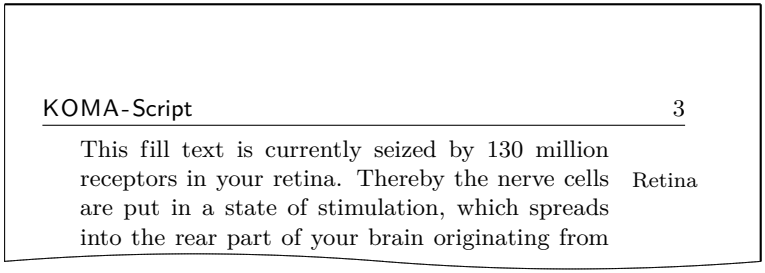
```
\setfootwidth[0pt]{head}
```

If no *shift* is indicated, i. e., without the optional argument, then the header or footer appears arranged symmetrically on the page. In other words, a value for the *shift* is determined automatically to correspond to the current page shape.

Example: Continuing with the previous example, we remove the optional argument:

```
\setheadwidth{textwithmarginpar}
```

which appears like this on an odd page:



As can be seen, the header is now shifted inward, while the header width has not changed. The shift is calculated in a way that the configuration of the typearea become visible also here.

```
\setheadtopline[length]{thickness}[commands]
\setheadsepline[length]{thickness}[commands]
\setfootsepline[length]{thickness}[commands]
\setfootbotline[length]{thickness}[commands]
```

Corresponding to the size configuration parameters of header and footer there are commands to modify the rules above and below the header and footer. But first of all the rules should be activated. See options `headtopline`, `headsepline`, `footsepline`, and `footbotline` in section 5.1.4, page 206 for this.

`\setheadtopline` – configures the line above the header

`\setheadsepline` – configures the line below the header

`\setfootsepline` – configures the line above the footer

`\setfootbotline` – configures the line below the footer

The mandatory argument *thickness* determines how strongly the line is drawn. The optional argument *length* accepts the same symbolic values as *width* for \setheadwidth, as well as also a normal length expression. As long as the optional argument *length* is not assigned a value, the appropriate line length adapts automatically the width of the header or the footer.

Use auto in the length argument to restore this automation for the length of a line.

v2.2 The optional argument *commands* may be used to specify additional commands to be executed before the respective line is drawn. For example, such commands could be used for changing the color of the line. When using a KOMA-Script class you could also use \setkomafont to specify commands for one of the elements headtopline, headsepline, footsepline, footbottomline, or footbotline. These can then be extended via \addtokomafont. See section 3.6, page 51 for details on the \setkomafont and \addkomafont commands.

```
\setheadtopline[auto]{current}
\setheadtopline[auto]{}
\setheadtopline[auto]{}[]
```

The arguments shown here for the command \setheadtopline are of course valid for the other three configuration commands too.

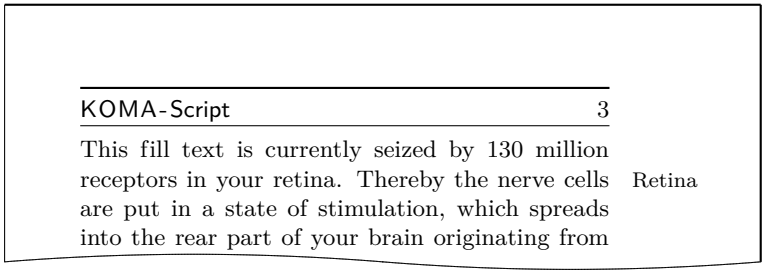
If the mandatory parameter has the value current or has been left empty, then the line thickness is not changed. This may be used to modify the length of the line without changing its thickness.

If the optional argument *commands* is omitted, then all command settings that might have been specified before will remain active, while an empty *commands* argument will revoke any previously valid commands.

Example: If the header, for example, is to be contrasted by a strong line of 2 pt above and a normal line of 0.4 pt between header and body, one can achieve this with:

```
\setheadtopline{2pt}
\setheadsepline{.4pt}
```

Additionally the options headtopline and headsepline have to be used preferably globally in the optional argument of \documentclass. In this case the result may be the following.



To specify that this line is to be drawn also, e. g., in red color, you would change the commands like this:

```
\setheadtopline{2pt}[\color{red}]
\setheadsepline{.4pt}[\color{red}]
```

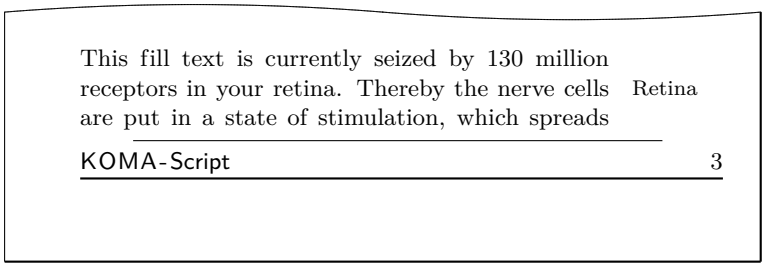
In this example, as well as in the following one, line color is activated by applying the syntax of the color package, so this package must of course be loaded. Since scrpage2 comes without built-in color handling, any package providing color support may be used.

KOMA-Script classes also support the following way of color specification:

```
\setheadtopline{2pt}
\setheadsepline{.4pt}
\setkomafont{headtopline}[\color{red}]
\setkomafont{headsepline}[\color{red}]
```

The automatic adjustment to the header and footer width is illustrated in the following example:

```
\setfootbotline{2pt}
\setfootsepline[text]{.4pt}
\setfootwidth[0pt]{textwithmarginpar}
```



Now not everyone will like the alignment of the line above the footer; instead, one would expect the line to be left-aligned. This can only be achieved with a global package option, which will be described together with other package options in the next section 5.1.4.

5.1.4. Package Options

In opposite to the KOMA-Script classes, where the most options may be changed using `\KOMAoptions` or `\KOMAoption` also after loading the class, package scrpage2 doesn't provide this feature currently. All options to scrpage2 have to be global options, i. e. be part of the optional argument of `\documentclass`, or package option, i. e. be part of the optional argument of `\usepackage`.

`headinclude`
`headexclude`
`footinclude`
`footexclude`

v2.3 Since KOMA-Script 3 this options shouldn't be passed to scrpage2 any longer using `\PassOptionsToPackage` or the optional argument of `\usepackage`. Only for compatibility reason scrpage2 still declares them and pass them as `headinclude`, `headinclude=false`, `footinclude` und `footinclude=false` to package typearea (see section 2.6, page 37) weitergereicht.

`headtopline`
`plainheadtopline`
`headsepline`
`plainheadsepline`
`footsepline`
`plainfootsepline`
`footbotline`
`plainfootbotline`

Basic adjustment of the lines under and over header and footer can be made with these options. These adjustments are then considered the default for all page styles defined with scrpage2. If one of these options is used, then a line thickness 0.4 pt is set. Since there is a corresponding plain page style to the page style `scrheadings`, the corresponding line in the plain style can also be configured with the `plain...` options. These `plain` options do however work only if the corresponding options without `plain` are activated. Thus, `plainheadtopline` shows no effect without the `headtopline` option set.

With these options, it is to be noted that the appropriate page part, header or footer, is considered as a part of the text area for the calculation of the type area in case a line has been activated. This means that, if the separation line between header and text is activated with `headsepline`, then the package typearea calculates the type area in such a way that the page header is part of the text block automatically.

The conditions for the options of the preceding paragraph apply also to this automation. That means that the package typearea must be loaded after scrpage2, or that on use of a KOMA-Script class, the options `headinclude` and `footinclude` must be set explicitly with `\documentclass` in order to transfer header or footer line in the text area.

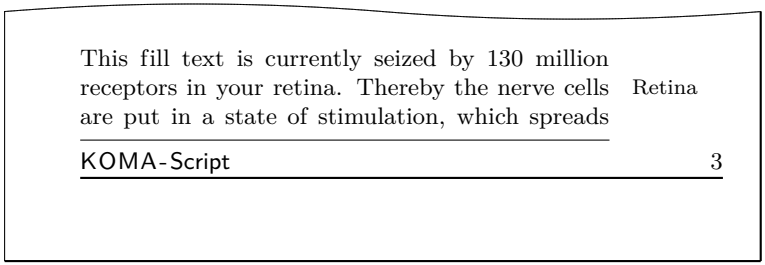
`ilines`
`clines`
`olines`

With the definition of the line lengths the case can arise where the lengths are set correctly, but the justification is not as desired because the line will be centered in the header or footer area. With the package options presented here, this specification can be modified for all page styles defined with scrpage2. The option `ilines` sets the justification in such a way that the lines align to the inside edge. The option `clines` behaves like the default justification, and `olines` aligns at the outside edge.

Example: The next example illustrates the influence of the option `ilines`. Please compare to the example for `\setfootsepline` on page 205.

```
\usepackage[ilines]{scrpage2}
\setfootbotline{2pt}
\setfootsepline[text]{.4pt}
\setfootwidth[0pt]{textwithmarginpar}
```

The mere use of the option `ilines` leads to the different result shown below:



In contrast to the default configuration, the separation line between text and footer is now left-aligned, not centered.

`automark`
`manualmark`

These options set at the beginning of the document whether to use running headings or manual ones. The option `automark` switches the automatic updating on, `manualmark` deactivates it. Without the use of one of the two options, the setting which was valid when the package was loaded is preserved.

Example: You load the package scrpage2 directly after the document class scrreprt without any package options:

```
\documentclass{scrreprt}
\usepackage{scrpage2}
```

Since the default page style of scrreprt is `plain`, this page style is also now still active. Futhermore, `plain` means manual headings. If one now activates the page style `scrheadings` with

```
\pagestyle{scrheadings}
```

then the manual headings are nevertheless still active.

If you instead use the document class scrbook, then after

```
\documentclass{scrbook}
\usepackage{scrpage2}
```

the page style `headings` is active and the running headings are updated automatically. Switching to the page style `scrheadings` keeps this setting active. The marking commands of scrbook continue to be used.

However, the use of

```
\usepackage[automark]{scrpage2}
```

activates running headings independently of the used document class. The option does not of course affect the used page style `plain` of the class scrreprt. The headings are not visible until the page style is changed to `scrheadings`, `useheadings` or another user-defined page style with headings.

`autooneside`

This option ensures that the optional parameter of `\automark` will be ignored automatically in one-sided mode. See also the explanation of the command `\automark` in section 5.1.2, page 200.

`komastyle`
`standardstyle`

These options determine the look of the predefined page styles `scrheadings` and `scrplain`. The option `komastyle` configures a look like that of the KOMA-Script classes. This is the default for KOMA-Script classes and can in this way also be set for other classes.

The option `standardstyle` configures a page style as it is expected by the standard classes. Furthermore, the option `markuppercase` will be activated automatically, but only if option `markusedcase` is not given.

`markuppercase`
`markusedcase`

In order to achieve the functionality of `\automark`, the package scrpage2 modifies internal commands which are used by the document structuring commands to set the running headings. Since some classes, in contrast to the KOMA-Script classes, write the headings in uppercase letters, scrpage2 has to know how the used document class sets the headings.

Option markuppercase shows scrpage2 that the document class uses uppercase letters. If the document class does not set the headings in uppercase letters, then the option markusedcase should be given. These options are not suitable to force a representation; thus, unexpected effects may occur if the given option does not match the actual behaviour of the document class.

nouppercase

In the previous paragraph dealing with markuppercase and markusedcase, it has been already stated that some document classes set the running headings in uppercase letters using the commands \MakeUppercase or \uppercase. Setting the option nouppercase allows disabling both these commands in the headers and footers. However, this is valid only for page styles defined by scrpage2, including scrheadings and its corresponding plain page style.

The applied method is very brutal and can cause that desired changes of normal letters to uppercase letters do not occur. Since these cases do not occur frequently, the option nouppercase usually affords a useful solution.

Example: Your document uses the standard class book, but you do not want the uppercase headings but mixed case headings. Then the preamble of your document could start with:

```
\documentclass{book}
\usepackage[nouppercase]{scrpage2}
\pagestyle{scrheadings}
```

The selection of the page style scrheadings is necessary, since otherwise the page style headings is active, which does not respect the settings made by option nouppercase.

In some cases not only classes but also packages set the running headings in uppercase letters. Also in these cases the option nouppercase should be able to switch back to the normal mixed case headings.

5.2. Defining Own Page Styles

5.2.1. The Interface for Beginners

Now one would not like to remain bound to only the provided page styles, but may wish to define one's own page styles. Sometimes there will be a special need, since a specific *Corporate Identity* may require the declaration of its own page styles. The easiest way to deal with this is:

\deftripstyle{*name*}[*LO*][*LI*]{*HI*}{*HC*}{*HO*}{*FI*}{*FC*}{*FO*}

The individual parameters have the following meanings:

name – the name of the page style, in order to activate it using the command `\pagestyle{`*name*`}`

LO – the thickness of the outside lines, i. e., the line above the header and the line below the footer (optional)

LI – the thickness of the separation lines, i. e., the line below the header and the line above the foot (optional)

HI – contents of the inside box in the page header for two-sided layout or left for one-sided layout

HC – contents of the centered box in the page header

HO – contents of the outside box in the page header for two-sided layout or right for one-sided layout

FI – contents of the inside box in the page footer for two-sided layout or left for one-sided layout

FC – contents of the centered box in the page footer

FO – contents of the outside box in the page footer for two-sided layout or right for one-sided layout

The command `\deftripstyle` definitely represents the simplest possibility of defining page styles. Unfortunately, there are also restrictions connected with this, since in a page range using a page style defined via deftripstyle, no modification of the lines above and below header and footer can take place.

Example: Assume a two-sided layout, where the running headings are placed on the inside. Furthermore, the document title, here "Report", shall be placed outside in the header, the page number shall be centered in the footer.

```
\deftripstyle{TheReport}%
              {\headmark}{}{Report}%
              {}{\pagemark}{}
```

If moreover the lines above the header and below the footer shall be drawn with a thickness of 2 pt, and the text body be separated from header and footer with 0.4 pt lines, then the definition has to be extended:

```
\deftripstyle{TheReport}[2pt][.4pt]%
              {\headmark}{}{Report}%
              {}{\pagemark}{}
```

See figure 5.2 for the result.

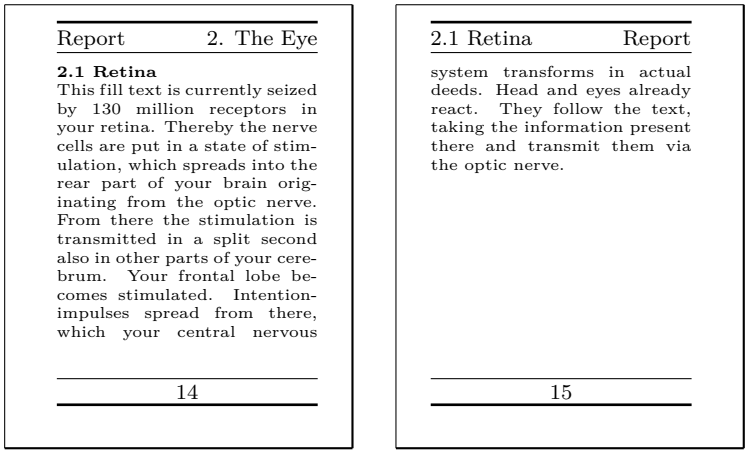


Figure 5.2.: example of a user defined, line dominated page style with a static and a running heading at the page header and a page number centered at the page footer

5.2.2. The Interface for Experts

Simple page styles, as they can be defined with `\deftripstyle`, are fairly rare according to experience. Either a professor requires that the thesis looks like his or her own — and who seriously wants to argue against such a wish? — or a company would like that half the financial accounting emerges in the page footer. No problem, the solution is:

```
\defpagestyle{name}{header definition}{footer definition}
\newpagestyle{name}{header definition}{footer definition}
\renewpagestyle{name}{header definition}{footer definition}
\providepagestyle{name}{header definition}{footer definition}
```

These four commands give full access to the capabilities of scrpage2 to define page styles. Their structure is indentical, they differ only in the manner of working.

`\defpagestyle` – defines a new page style. If a page style with this name already exists it will be overwritten.

`\newpagestyle` – defines a new page style. If a page style with this name already exists a error message will be given.

`\renewpagestyle` – redefines a page style. If a page style with this name does not exist a error message will be given.

`\providepagestyle` – defines a new page style only if there is no page style with that name already present.

Using `\defpagestyle` as an example, the syntax of the four commands is explained below.

name – the name of the page style for `\pagestyle{`*name*`}`

header definition – the declaration of the header, consisting of five element; elements in round parenthesis are optional:
(ALL,ALT){EP}{OP}{OS}(BLL,BLT)

footer definition – the declaration of the footer, consisting of five element; elements in round parenthesis are optional:
(ALL,ALT){EP}{OP}{OS}(BLL,BLT)

As can be seen, header and footer declaration have identical structure. The individual parameters have the following meanings:

ALL – above line length: (header = outside, footer = separation line)

ALT – above line thickness

EP – definition for *even* pages

OP – definition for *odd* pages

OS – definition for *one-sided* layout

BLL – below line length: (header = separation line, footer = outside)

BLT – below line thickness

If the optional line-parameters are omitted, then the line behaviour remains configurable by the commands introduced in section 5.1.3, page 203.

The three elements *EP*, *OP* and *OS* are boxes with the width of page header or footer, as appropriate. The corresponding definitions are set left-justified in the boxes. To set something left- *and* right-justified into the boxes, the space between two text elements can be stretched using `\hfill`:

```
{\headmark\hfill\pagemark}
```

If one would like a third text-element centered in the box, then an extended definition must be used. The commands `\rlap` and `\llap` simply write the given arguments, but for LaTeX they take up no horizontal space. Only in this way is the middle text really centered.

```
{\rlap{\headmark}\hfill centered text\hfill\llap{\pagemark}}
```

Example: This examples uses the document class scrbook, which means that the default page layout is two-sided. The package scrpage2 is loaded with options `automark` and `headsepline`. The first switches on the automatic update of running headings, the second determines that a separation line between header and text body is drawn in the `scrheadings` page style.

```
\documentclass{scrbook}
\usepackage[automark,headsepline]{scrpage2}
```

The expert interface is used to define two page styles. The page style `withoutLines` does not define any line parameters. The second page style `withLines` defines a line thicknes of 1 pt for the line above the header and 0 pt for the separation line between header and text.

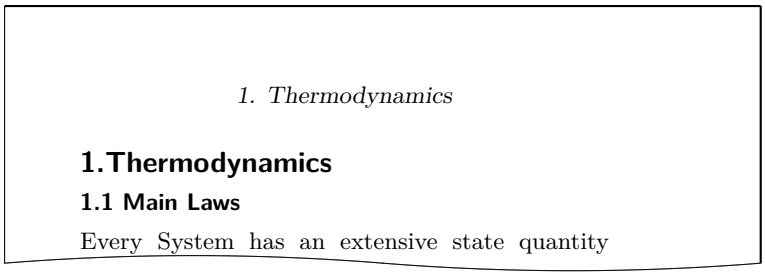
```
\defpagestyle{withoutLines}{%
  {Example\hfill\headmark}{\headmark\hfill without lines}
  {\rlap{Example}\hfill\headmark\hfill%
   \llap{without lines}}
}{%
  {\pagemark\hfill}
  {\hfill\pagemark}
  {\hfill\pagemark\hfill}
}

\defpagestyle{withLines}{%
  (\textwidth,1pt)
  {with lines\hfill\headmark}{\headmark\hfill with lines}
  {\rlap{\KOMAScript}\hfill \headmark\hfill%
   \llap{with lines}}
  (0pt,0pt)
}{%
  (\textwidth,.4pt)
  {\pagemark\hfill}
  {\hfill\pagemark}
  {\hfill\pagemark\hfill}
  (\textwidth,1pt)
}
```

Right at the beginning of the document the page style `scrheadings` is activated. The command `\chapter` starts a new chapter and automatically sets the page rstyle for this page to `plain`. Even though not a prime example, the command `\chead` shows how running headings can be created even on a plain page. However, in principle running headings on chapter start-pages are to be avoided, since otherwise the special character of the `plain` page style is lost. It is more important to indicate that a new chapter starts here than that a section of this page has a special title.

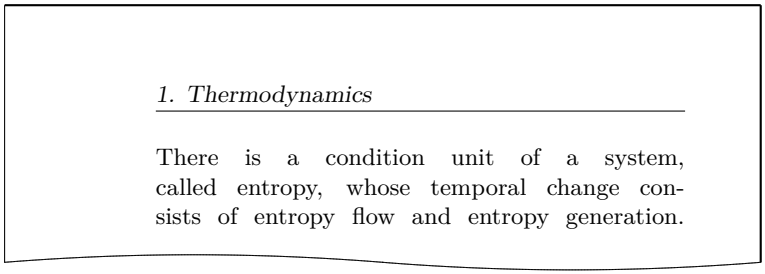
```
\begin{document}
\pagestyle{scrheadings}
\chapter{Thermodynamics}
\chead[\leftmark]{}
\section{Main Laws}
Every system has an extensive state quantity called
```

```
Energy. In a closed system the energy is constant.
```



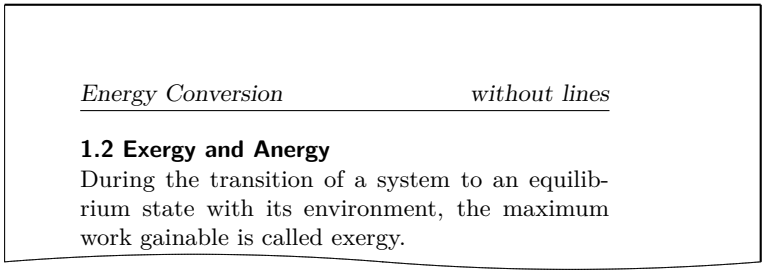
After starting a new page the page style `scrheadings` is active and thus the separation line below the header is visible.

```
There is a state quatity of a system, called entropy, whose temporal
change consists of entropy flow and entropy generation.
```



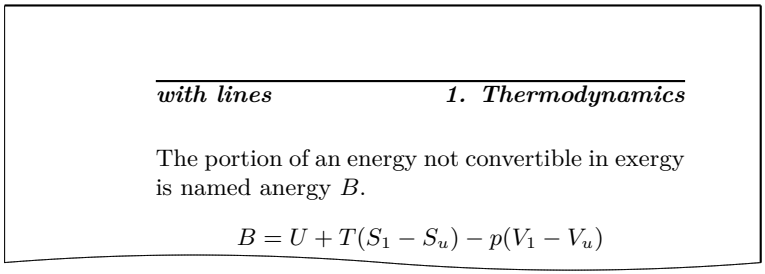
After switching to the next page, the automatic update of the running headings is disabled using `\manualmark`, and the page style `withoutLines` becomes active. Since no line parameters are given in the definition of this page style, the default configuration is used, which draws a separation line between header and text body because scrpage2 was called with `headsepline`.

```
\manualmark
\pagestyle{withoutLines}
\section{Exergy and Anergy}\markright{Energy Conversion}
During the transition of a system to an equilibrium state
with its environment, the maximum work gainable is called
exergy.
```



At the next page of the document, the page style `withLines` is activated. The line settings of its definition are taken in account and the lines are drawn accordingly.

```
\pagestyle{mitLinien}
\renewcommand{\headfont}{\itshape\bfseries}
The portion of an energy not convertible in exergy
is named anergy \Var{B}.
\[ B = U + T (S_1 - S_u) - p (V_1 - V_u)\]
\end{document}
```



5.2.3. Managing Page Styles

Before long the work with different page styles will establish a common set of employed page styles, depending on taste and tasks. In order to make the management of page styles easier and avoid time-consuming copy operations each time a new project is started, scrpage2 reads the file `scrpage.cfg` after initialisation. This file can contain a set of user-defined page styles which many projects can share.

Weekday and Time Using scrdate and scrtime

There are two packages included in KOMA-Script to improve and extend the handling of date and time over and above what is provided by the standard commands `\today` and `\date`. Like all the other packages from the KOMA-Script bundle these two packages may be used not only with KOMA-Script classes but also with the standard and many other classes.

v3.05a Since KOMA-Script 3.05a this packages use the common version number of KOMA-Script. This change has been done, because the packages need the corresponding version of scrkbase and scrbase.

6.1. 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`* doesn't 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 2012 is the year 12 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 doesn't 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 2 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 don't 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 Tuesday 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`
`\todaysname`
`\todaysnumber`

v3.05a

In the prior examples the current date have been given cumbersomely 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 `\todaysname` directly offers the name of the current day of the week. Command `\todaysnumber` 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 \todaysname.
```

This will result in, e. g.:

> I have done the LATEX run of this document on a Thursday.

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{\todaysname}
```

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 `\todaysname` 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 LaTeX run of this document on a workday.

There's no such command for changing the result of `\ISOToday` or `\IsoToday`.

Name of the day of the week in different languages

v3.08b

Currently the package scrdate knows the languages English (english, american, USenglish, UKenglish and british), German (german, ngerman, austrian, and naustrian), French (french), Italian (italian), Spanish (spanish), Croatian (croatian), Finnish (finnish), Norwegian (norsk), Swedish (swedish), and Danish (danish). If you want to configure it for other languages, see `scrdate.dtx`.

In the current implementation it does not matter whether you load scrdate before or after german, ngerman, babel or similar packages. 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.

6.2. Getting the Time with Package

The second problem is the question of the current time. The solution may be found using package scrtime.

\thistime[*delimiter*]
\thistime*[*delimiter*]

\thistime results in the current time. The delimiter between the values of hour, minutes and seconds can be given in the optional argument. The default symbol of the delimiter is ":".

\thistime* works in almost the same way as \thistime. The only difference is that unlike with \thistime, with \thistime* the value of the minute field is not preceded by a zero when its value is less than 10. Thus, with \thistime the minute field has always two places.

Example: The line

```
Your train departs at \thistime.
```

results, for example, in:

Your train departs at 9:49.

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 9 hours and 49 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 37. 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 27). The option has no effect on the results of \thistime and \thistime* if \settime is used.

Only for compatibility with former releases of scrtime 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

7.1. Overview

The package scraddr is a small extension to the KOMA-Script letter class. Its aim is to make access to the data of address files more flexible and easier. Basically, the package implements a new loading mechanism for address files which contain address entries in the form of `\adrentry` and newer `\addrentry` commands, as described in chapter 4 from page 192.

`\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 TEX 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 192 onwards. The division of the address file with the help of `\adrchar` or `\addrchar`, also discussed therein, has no meaning for scraddr and is simply ignored.

The commands for accessing the data are given by the name of the data field they are intended for.

```
\Name{key}
\FirstName{key}
\LastName{key}
\Address{key}
\Telephone{key}
\FreeI{key}
\FreeII{key}
\Comment{key}
\FreeIII{key}
\FreeIV{key}
```

These commands give access to data of your address file. The last parameter, i. e., parameter 8 for the `\adrentry` entry and parameter 9 for the `\addrentry` entry, is the identifier of an entry, thus the ***key*** has to be unique and non-blank. The ***key*** should only be composed of multiple uppercase letters out of the namespace of TEX macro names.

If the file contains more than one entry with the same *key* value, the last occurrence will be used.

7.2. Usage

First of all, we need an address file with valid address entries. In this example the file has the name `lotr.adr` and contains the following entries.

```
\addrentry{Baggins}{Frodo}%
          {The Hill\\ Bag End/Hobbiton in the Shire}{}%
          {Bilbo Baggins}{pipe-weed}%
          {the Ring-bearer}{Bilbo's heir}{FRODO}
\adrentry{Gamgee}{Samwise}%
          {Bagshot Row 3\\Hobbiton in the Shire}{}%
          {Rosie Cotton}{taters}%
          {the Ring-bearer's faithful servant}{SAM}
\adrentry{Bombadil}{Tom}%
          {The Old Forest}{}%
          {Goldberry}{trill queer songs}%
          {The Master of Wood, Water and Hill}{TOM}
```

The 4th parameter, the telephone number, has been left blank. If you know the story behind these addresses you will agree that a telephone number makes no sense here, and besides, it should simply be possible to leave them out.

The command `\InputAddressFile` is used to load the address file shown above:

```
\InputAddressFile{lotr}
```

With the help of the commands introduced in this chapter we can now write a letter to old Tom Bombadil. In this letter we ask him if he can remember two fellow-travelers from Elder Days.

```
\begin{letter}{\Name{TOM}\\\Address{TOM}}
   \opening{Dear \FirstName{TOM} \LastName{TOM},}

   or \FreeIII{TOM}, how your delightful \FreeI{TOM} calls you.  Can
   you remember Mr.\,\LastName{FRODO}, strictly speaking
   \Name{FRODO}, since there was Mr.\,\FreeI{FRODO} too.  He was
   \Comment{FRODO} in the Third Age and \FreeIV{FRODO} \Name{SAM},
   \Comment{SAM}, has attended him.

    Their passions were very worldly.  \FirstName{FRODO} enjoyed
    smoking \FreeII{FRODO}, his attendant appreciated a good meal with
    \FreeII{SAM}.

    Do you remember? Certainly Mithrandir has told you much
    about their deeds and adventures .
```

```
    \closing{``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.

7.3. 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 LaTeX run if `\FreeIV` has been used with an `\adrentry` entry.

`adrFreeIVempty` – the command `\FreeIV` will be ignored

`adrFreeIVshow` – "(entry FreeIV undefined at ***key***)" will be written as warning in the text

`adrFreeIVwarn` – writes a warning in the logfile

`adrFreeIVstop` – the LaTeX run will be interrupted with an error message

To choose the desired reaction, one of these options can be given in the optional argument of the `\usepackage` command. The default setting is `adrFreeIVshow`.

Creating Address Files from a Address Database

In former versions of KOMA-Script the package addrconv was a permanent part of the KOMA-Script system. The chief involvement with KOMA-Script was that with the help of addrconv it was possible from an address database in BibTeX format to create address files compatible with the KOMA-Script letter class or with the package scraddr.

```
@address{HMUS,
   name =       {Carl McExample},
   title =      {Dr.},
   city =       {Anywhere},
   zip =        01234,
   country =    {Great Britain},
   street =     {A long Road},
   phone =      {01234 / 5 67 89},
   note =       {always forget his birthday},
   key =        {HMUS},
}
```

From entries such as that given above, address files can be generated. For this addrconv employs BibTeX and various BibTeX styles. Additionally, there are some LaTeX files which can help to create various telephone and address lists for printing.

However, the package addrconv was actually an independent package, since besides what is required for KOMA-Script it includes several more interesting features. Therefore, the package addrconv has for some time already been removed from the KOMA-Script system. The package adrconv, with a single *d*, entirely replaces addrconv. If it is not included in your TeX distribution then it can be downloaded from [Kie99] and you can install it separately.

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 scrlttr2, the successor of scrlettr, 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 shouldn't 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.

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 10 from page 231. 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 scrlfile described in chapter 11 from page 248. 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 13 from page 261), that has been designed to be used by authors of classes and packages too. Because of this scrextend doesn't provide the features of this package. Nevertheless you may use tocbasic together with scrextend.

9.1. Early or late Selection of Options

All of what is described in section 2.4 is generally applicable.

9.2. Compatibility with Earlier Versions of KOMA-Script

It applies, mutatis mutandis, what is written in section 2.5.

9.3. 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 the conflict with features of the standard classes of often used packages.

Table 9.1.: overview of the optional available extended features of scrextend

title	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 9.7, from page 227)

`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 9.1.

9.4. Draft Mode

What is written in section 3.3 applies, mutatis mutandis.

9.5. Selection of the Document Font Size

What is described in section 3.5 applies, mutatis mutandis.

9.6. Text Markup

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 [OPHS99], [Tea05b], and [Tea05a].

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

The LaTeX-Kern already defines the command `\textsuperscript` to superscript text. Unfortunately, LaTeX 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 50.

`\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 [OPHS99], [Tea05b], or [Tea05a]. Color switching commands like \normalcolor (see [Car99d] 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 51. 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, that has been defined for the given *element*.

Example: Assumed, you want to print the document title in a serif font and with red color. 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 9.3, page 226).

9.7. Document Titles

It applies, mutatis mutandis, what is described in section 3.7. But there's a difference:

The document title capabilities of scrextend are part of the optional, advanced features. Therfore they are only available, if extendedfeature=title has been selected while loading the package (see section 9.3, page 226). 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.

9.8. Detection of Odd and Even Pages

What is described in section 3.11 applies, mutatis mutandis.

9.9. 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 doesn't 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 9.3, page 226). 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.

9.10. Interleaf Pages

What is described in section 3.13 applies, mutatis mutandis.

9.11. Footnotes

Package scrextend supports all the footnote features of KOMA-Script, that are described at section 3.14. Nevertheless, by default the footnotes are under full control of the used class. This changes as soon as command `\deffootnote` (see page 77) has been used.

9.12. Dicta

It applies, mutatis mutandis, what is described in section 3.17. 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.

9.13. Lists

What is described in section 3.18 applies, mutatis mutandis. 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.

9.14. Margin Notes

It applies, mutatis mutandis, what is described in section 3.21.

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 at Package scrbase

The package scrbase provides basic features, that are designed and implemented to be used by authors of packages and classes. Thereby it may not only used for wrapper classes, that use a KOMA-Script class. Authors of classes, that aren't related to anything else from KOMA-Script, may also benefit from the functionality of scrbase.

10.1. 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 opposite to this `\RequirePackageWithOptions` doesn't have an optional argument but passes all options given while 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 this 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 of those are build like `\scr@`*name*. With this those are internal commands. KOMA-Script really uses this internal commands internally. Authors of packages and classes may use those internal commands too, but should not redefine them. Because some of those commands may be useful for users too, they are also provided as `\`*name* normally. But eventually other packages provide commands with the same name but different syntax or different functionality. This would result in an conflict. So scrbase provides to suppress the definition of the user commands, `\`*name*, only. 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 shouldn't 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 doesn't want scrbase to define commands `\ifVTeX` and `\ifundefinedorrelax`. Because of this, user uses:

```
\documentclass%
  [internalonly=/ifVTeX/ifundefinedorrelax]%
  {foo}
```

to load the class. Class name foo has been used as an placeholder for any class in this example. The meanings of commands `\ifVTeX` and `\ifundefinedorrelax` and many more commands for conditional execution may be found in section 10.3.

10.2. Keys as Attributes of Families and their Members

As already mentioned in section 10.1, 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 knows also family members. Now, 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, of a family or of all family members.

```
\DefineFamily{family}
\DefineFamilyMember[family member]{family}
```

scrbase needs to know the members of a family for different reasons. So it's necessary first to define a new family using `\DefineFamily`, that produces also an empty member list. If the family has already been defined nothing would happen. Nothing means also, that an already existing member list would not be overwritten.

A new member may be added to the family using `\DefineFamilyMember`. If the family doesn't exist, this would result in an error message. If the member already exists, nothing happens. If the member is omitted, the member won't stay empty, but "`.\@currname.\@currext`" will be assumed. While loading a class or package `\@currname` and `\@currext` together represent the file name of the class or package.

Theoretically is would be possible, to define a member without a name using an empty optional argument *family member*. But this would be same like the family itself. It is recommended to use only letters and digits at the *family* an start the *family member* with another char like a period. Otherwise it could happen, that members of one family are same like members of another family.

scrbase itself defines family "KOMA" and adds member "`.scrbase.sty`" to is. Generally family "KOMA" is reserved to KOMA-Script. For your own packages it is recommended to use the name of the bundle as *family* and the name of the package as *family member* of that *family*.

Example: Assumed you're writing a bundle "master butcher". Within that bundle you have packages `salami.sty`, `liversausage.sty`, and `kielbasa.sty`. Therefor you decide to use family name "`butcher`" and add the lines

```
\DefineFamily{butcher}
\DefineFamilyMember{butcher}
```

to each of the package files. While loading the three packages this will all 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*`}`

This command defines a ***key***. If a ***family member*** is given, the ***key*** will become an attribute of that member of the also given ***family***. If no ***family member*** is given, the member "`.\@currname.\@currext`" will be assumed. If later a value will be assigned to the ***key***, the ***action*** will be executed and the value will be an argument of this. So inside of ***action*** "`#1`" would be that value. If the value will be omitted, the ***default*** will be used instead. If there's no ***default***, the ***key*** can be used only with value.

At least

`\DefineFamilyKey[`*member*`]{`*family*`}{`*key*`}`
`[`*default*`]{`*action*`}`

will result in a call of

`\define@key{`*family member*`}{`*key*`}`
`[`*default*`]{`*action*`}`

with `\define@key` is provided by package keyval (see [Car99b]).

Example: Assumed, each of the three packages from the previous example should get a key `coldcuts`. If this is used, a switch should be set at each of the packages. For package salami this may be, e. g.,

```
\newif\if@Salami@Aufschnitt
\DefineFamilyKey{butcher}%
                {coldcut}[true]{%
  \expandafter\let\expandafter\if@salami@coldcut
  \csname if#1\endcsname
}
```

Available values for the key are `true` or `false` in this case. There's no test on inappropriate values in this example. If the key will be used later, this has to be done 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.

`\FamilyProcessOptions[`*family member*`]{`*family*`}`

Generally the extension of keys of families to keys of families and family members was mentioned to use keys or key-value settings as class or package options. This command therefor

is an extension of `\ProcessOption*` from LaTeX kernel (see [Tea06]. Thereby the command processes not only options that has been declared using `\DeclareOption`. It processes even all keys of the given family member. If the optional argument *family member* is omitted, family member "`.\@currname.\@currext`" will be used.

Somehow 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 will be set also and 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 `coldcut` while loading the package. If the option will be 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 will be loaded later.

Please note that packages process global options always before local options, that has been assigned locally to the package. In opposite to unknown options while processing of global options, that will only result in an information in the `log`-file, unknown options assigned to the package result in error messages.

`\FamilyProcessOptions` may be interpreted either as an extension of `\ProcessOption*` or as an extension of the *key=value* mechanism of keyval. Finally *key=value* pairs become options with help of `\FamilyProcessOptions`.

`\FamilyExecuteOptions[`*family member*`]{`*family*`}{`*options list*`}`

This command is an extension of `\ExecuteOptions` from the LaTeX kernel (see [Tea06]). Thereby the command processes not only options, that has been defined using `\DeclareOption`. Also all keys of the given family member will be processed. If the optional argument `\family member` is omitted, then "`.\@currname.\@currext`" is used.

Somehow 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 will be set also and before the keys of the family members.

Example: Assumed, option `coldcut` should be set by default already in the previous example. In this case only line

```
\FamilyExecuteOptions{butcher}{coldcut}
```

has to be added.

`\FamilyOptions{`*family*`}{`*options list*`}`
`\Family@Options{`*family*`}{`*options list*`}{`*error action*`}`

Thereby *options list* is like:

key=value,*key=value*...

whereby the value assignment may be omitted for *key*s, that have a defined default.

In opposite to average options, that has been defined using `\DeclareOption`, the *key*s may also be set after loading a class or package. For this the user uses `\FamilyOptions`. Thereby the *key*s of all members of the given family will be set. If a *key* also exists as an family attribute, then the family key will be set first. After this the member keys will follow in the order, the members has been defined. If a given *key* does exist neither in the family nor in any member of the family, then `\FamilyOptions` will result in an error. In opposite to this with `\Family@Options` the user may declare his own *error action*. But this internal command is reserved for authors of classes and packages.

Example: You extend your butcher project by a package sausagesalad. If this package has been loaded, all sausage package should generate cold cut:

```
\ProvidesPackage{sausagesalad}%
                 [2008/05/06 nonsense package]
\DefineFamily{butcher}
\DefineFamilyMember{butcher}
\FamilyProcessOptions{butcher}\relax
\FamilyOptions{butcher}{coldcut}
```

If currently non of the sausage packages has been loaded, then an error message would result because of undefined option "`coldcut`". Dies may be avoided changing the last line of the previous code into:

```
\Family@Options{butcher}{coldcut}{}
```

Nevertheless, sausage packages, that will be loaded after sausagesalad, won't produce cold cut. This may be changed additionally, by changing the last line again:

```
\AtBeginDocument{%
  \Family@Options{butcher}{coldcut}{%
    \PackageWarning{sausagesalad}{%
      sausage salad needs at least
      one sausage package}%
  }%
}%
```

This will also throw a warning message, if non of the sausage packages will be loaded.

`\FamilyOption{`*family*`}{`*option*`}{`*values list*`}`
`\Family@Option{`*family*`}{`*option*`}{`*values list*`}{`*error action*`}`

Beside options that have concurrently excluding values, there may be options, that have several values at the same time. Using `\FamilyOptions` for that kind of options 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:

value,*value*...

By the way please note, that usage of a comma inside a value may be done only, if the value is put into braces. The general functionality of these commands is the same like that of the previous commands `\FamilyOptions` and `\Family@Options`.

Example: Package sausagesalad should have one mire option, to add additional ingredients. Each of the ingredients will set a switch like done before for the cold cut.

```
\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” has been defined. An error message for not defined ingredients doesn’t 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}
```

`\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 `coldcut` is was necessary, that the user assigns either value `true` or value `false`. No error message existed on wrong value assignment. Because of such boolean switches are an often needed feature, scrbase provides `\FamilyBoolKey` for definition of such options. Thereby the arguments *family member*, *family*, and *key* are same like used by `\DefineFamilyKey` (see page 233). Argument *switch name* is the name of the switch without the prefix `\if`. If a switch with this name doesn’t exist already, `\FamilyBoolKey` will define it and initialize it to *false*. Internally `\FamilyBooKey` uses `\FamilySetBool` as *action* of `\DefineFamilyKey`. The *default* for those options is always `true`.

`\FamilySetBool` on the other side 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*. This will normally result in an error message about unknown value assignment (see also page 239).

Example: The key `coldcut` 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 cut.

```
\FamilyBoolKey{butcher}{coldcut}%
                          {@coldcut}
```

A test, whether or not to produce cold cut, may be:

```
\if@coldcut
   ...
\else
   ...
\fi
```

This would be the same in each of the three sausage packages. With this the attribute "coldcut" may be defined as a family option:

```
\@ifundefined{if@coldcut}{%
  \expandafter\newif\csname if@coldcut\endcsname
}{}%
\define@key{butcher}{coldcut}[true]{%
  \FamilySetBool{butcher}{coldcut}%
                           {@coldcut}%
                           {#1}%
}
```

or shorter:

```
\FamilyBoolKey[]{butcher}{coldcut}%
                           {@coldcut}
```

using the additional information at page 233, that's 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 opposite to switches that may be either true or false, also key exists, that accept several values. For example an alignment may not only be left or not left, but, e. g., left, centered, or right. Internally such differentiations are often made using `\ifcase`. This TEX 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},...

So the *values list* does not only define the supported values to the *key*. For each of the *value*s it also gives the *definition* of macro *command*. 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 while the assignment.

Example: The sausage may be cut different. For example the cold cut may stay uncut or will by 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 like 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 wouldn't 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 this:

```
\ifcase\cuthow
  % no cut
\or
  % rough cut
\else
  % thin cut
```

```
\fi
```

may be used.

Internally `\FamilyNumericalKey` uses `\FamilySetNumerical` as *action* of `\DefineFamilyKey`. If a unknown value is assigned to such a key, `\FamilySetNumerical` will call `\FamilyUnkownKeyValue` with the arguments *family*, *key* and *value*. This will normally result in an error message about assigning an unknown 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 their is no optional argument for the *default*, `\FamilyStringKey` is the same like

`\DefineFamilyKey[`*family member*`]{`*family*`}{`*key*`}`
`{\def`*command*`{#1}}`.

If an optional argument *default* has been used, `\FamilyStringKey` is the same like

`\DefineFamilyKey[`*family member*`]{`*family*`}{`*key*`}`
`[`*default*`]{\def`*command*`{#1}}`.

If *command* hasn't been defined already, 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}
```

That may be done, because the default weight has been set as default at the definition of the option above.

In this case there are no 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.

`\FamilyUnkownKeyValue{`*family*`}{`*key*`}{`*value*`}{`*values list*`}`
`\FamilyElseValues`

The command `\FamilyUnknownKeyValue` throws and 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*’ ...

Additional allowed values may be set by `\FamilyElseValues` in the form:

, ‘*value*’, ‘*value*’ ...

These will also be shown in the error message. `\FamilySetBool` as well as `\FamilySetNumerical` will reset `\FamilyElseValue` to an empty definition automatically — independent of whether or not a error has been thrown.

Example: Now, for the cold cut 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@coldcut}{%
  \expandafter
  \newif\csname if@coldcut\endcsname}{}
\DefineFamilyKey{butcher}%
                {coldcut}[true]{%
  \ifstr{#1}{thin}{%
    \@coldcuttrue
    \@thincuttrue
  }{%
    \@thincutfalse
    \def\FamilyElseValue{, ‘thin’}%
    \FamilySetBool{butcher}{coldcut}%
                          {@coldcut}%
                          {#1}%
  }%
}%
```

Command `\ifstr` used at the example code above will be described at page 242 in section 10.3.

10.3. Conditional Execution

The package scrbase provides several commands for conditional execution. Thereby not the TEX syntax of conditionals, e. g.,

```
\iftrue
  ...
\else
  ...
\fi
```

but the LaTeX syntax also known from LaTeX commands like `\IfFileExists`, `\@ifundefined`, `\@ifpackageloaded`, and many others will be used. Nevertheless, some package authors prefer to use the TeX syntax even for users at the LaTeX interface level. In fact the conditionals of scrbase are very basic conditionals, this could result in conditionals with the same name but different syntax and though in naming conflicts. Because of this scrbase firstly defines these conditionals as internal commands with prefix `\scr@`. Additional user commands without this prefix are additionally defined. But the definition of the user commands may be suppressed with option `internalonly` (see section 10.1, page 231).

Authors of packages and classes should use the internal commands like KOMA-Script itself does. Nevertheless, for completeness the user commands are described additionally.

`\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 opposite to `\@ifundefined \`*name* won't be defined to be `\relax` in the case it wasn't defined before. Using ε-TeX this case won't at least consume any hash memory.

`\scr@ifpdftex{`*then instructions*`}{`*else instructions*`}`
`\ifpdftex{`*then instructions*`}{`*else instructions*`}`

If pdfTeX has been used, the *then instructions* will be executed, otherwise the *else instructions*. Whether or not a PDF-file will be out, doesn't matter. Nevertheless, this pdfTeX test seems so make sense seldom only. Generally it's recommended to test for the wanted commands instead (see previous `\scr@ifundefinedorrelax` and `\ifundefinedorrelax`).

`\scr@ifVTeX{`*then instructions*`}{`*else instructions*`}`
`\ifVTeX{`*then instructions*`}{`*else instructions*`}`

If VTeX has been used, the *then instructions* will be executed, otherwise the *else instructions*. This test seems so make sense seldom only. Generally it's recommended to test for the wanted commands instead (see previous `\scr@ifundefinedorrelax` and `\ifundefinedorrelax`).

`\scr@ifpdfoutput{`*then instructions*`}{`*else instructions*`}`
`\ifpdfoutput{`*then instructions*`}{`*else instructions*`}`

While PDF-file generation the *then instructions* will be executed, otherwise the *else instructions*. Whether, e. g., pdfTeX or VTeX is used to generate the PDF-file doesn't matter.

```
\scr@ifpsoutput{then instructions}{else instructions}
\ifpsoutput{then instructions}{else instructions}
```

While PostScript-file generation the *then instructions* will be executed, otherwise the *else instructions*. VTEX provides direct PostScript generation, that will be recognized here. If VTEX is not used but a switch `\if@dvips` has been defined, the decision depends on that switch. KOMA-Script, e. g., provides `\if@dvips` in typearea.

```
\scr@ifdvioutput{then instructions}{else instructions}
\ifdvioutput{then instructions}{else instructions}
```

While DVI-file generation 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}
```

ε-TEX 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, the *else instructions* otherwise. There's no corresponding internal command for this.

```
\ifstr{string}{string}{then instructions}{else instructions}
```

Both *string* arguments will be expanded and afterward compared. If the expansions are same, the *then instructions* will be executed, otherwise the *else instructions*. There's no corresponding internal command for this.

```
\ifnumber{string}{then instructions}{else instructions}
```

Note, that this does not compare numbers. The *then instructions* will be executed, if the expansion of *string* consists of digits only. Otherwise the *else instructions* will be used. There's no corresponding internal command for this.

```
\ifdimen{string}{then instructions}{else instructions}
```

Note, that this does not compare dimensions. The *then instructions* will be executed, if the expansion of *string* consists of digits and a valid unit of length. Otherwise the *else instructions* will be used. There's no corresponding internal command for this.

```
\if@atdocument the instructions \else else instructions \fi
```

This command exists intentionally as internal command only. In the document preamble `\if@atdocument` is same like `\iffalse`. After `\begin{document}` it's same like `\iftrue`. Authors of classes and packages may use this, if a command should work somehow different depending in whether it has been used in the preamble or inside the documents body. Please note, that this is a condition in TEX syntax not in LATEX syntax!

10.4. Definition of Language-Dependent Terms

Normally one has to change or define language-dependent terms like `\captionsenglish` in such a way that in addition to the available terms the new or redefined terms are defined. This is made more difficult by the fact that some packages like german or ngerman redefine those settings when the packages are loaded. This definitions unfortunately occurs in such a manner as to destroy all previous private settings. That is also the reason why it makes sense to delay own changes with `\AtBeginDocument` until `\begin{document}`, that is, after package loading is completed. The user can also use `\AtBeginDocument`, or redefine the language-dependent terms after `\begin{document}`, that is, not put them in the preamble at all. The package scrbase provides some additional commands for defining language-dependent terms.

```
\providecaptionname{language}{term}{definition}
\newcaptionname{language}{term}{definition}
\renewcaptionname{language}{term}{definition}
```

Using one of these commands, the user can assign a *definition* for a particular *language* to a *term*. The *term* is always a macro. The commands differ dependent on whether a given *language* or a *term* within a given *language* are already defined or not at the time the command is called.

If *language* is not defined, then `\providecaptionname` does nothing other than writes a message in the log file. This happens only once for each language. If *language* is defined but *term* is not yet defined for it, then it will be defined using *definition*. The *term* will not be redefined if the *language* already has such a definition; instead, an appropriate message is written to the log file.

The command `\newcaptionname` has a slightly different behaviour. If the *language* is not yet defined, then a new language command will be created and a message written to the log file. For *language* `USenglish`, e. g. this would be the language command `\captionsUSenglish`. If *term* is not yet defined in *language*, then it will be defined using *definition*. If *term* already exists in *language*, then this results in an error message.

The command `\renewcaptionname` again behaves differently. It requires an existing definition of *term* in *language*. If neither *language* nor *term* exist or *term* is unknown in a defined language then a error message will be given. Otherwise, the *term* for *language* will be redefined according to *definition*.

KOMA-Script itself employs `\providecaptionname` in order to define the commands in section 17.4.

Example: If you prefer “fig.” instead of “figure”, you may achieve this using:

```
\renewcaptionname{USenglish}{\figurename}{fig.}
```

Since only existing terms in available languages can be redefined using `\renewcaptionname`, you have to put the command after `\begin{document}` or delay the command by using

`\AtBeginDocument`. Furthermore, you will get an error message if there is no package used that switches language selection to, e. g., *`USenglish`* in the example above.

Language dependent terms usually defined by classes and language packages are listed and described in table 10.1.

Table 10.1.: Overview of language dependent terms of usual language packages

`\abstractname`
: heading of the abstract

`\alsoname`
: “see also” in additional cross references of the index

`\appendixname`
: “appendix” in the heading of an appendix chapter

`\bibname`
: heading of the bibliography

`\ccname`
: prefix heading for the distribution list of a letter

`\chaptername`
: “chapter” in the heading of a chapter

`\contentsname`
: heading of the table of contents

`\enclname`
: prefix heading for the enclosure of a letter

`\figurename`
: prefix heading of figure captions

`\glossaryname`
: heading of the glossary

`\headtoname`
: “to” in header of letter pages

`\indexname`
: heading of the index

. . .

Table 10.1.: Overview of usual language dependent terms (*continuation*)

`\listfigurename`	heading of the list of figures
`\listtablename`	heading of the list of tables
`\pagename`	“page” in the pagination of letters
`\partname`	“part” in the heading of a part
`\prefacename`	heading of the preface
`\proofname`	prefix heading of mathematical proofs
`\refname`	heading of the list of references
`\seename`	“see” in cross references of the index
`\tablename`	prefix heading at table captions

10.5. Identification of KOMA-Script

scrbase may be used independent of KOMA-Script with other packages and classes, nevertheless it's a KOMA-Script package. For this is also provides commands to identify KOMA-Script and itself as a KOMA-Script package.

`\KOMAScript`

This command only sets the word “KOMA-Script” with sans-serif font and a little bit tracking of the capitals. Bay the way: All KOMA-Script classes and packages define this command, if it hasn't 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. This document has been made, e. g., using KOMA-Script version "2012/07/05 v3.11a KOMA-Script".

10.6. Extension of the LaTeX Kernel

Sometimes the LaTeX kernel itself provides commands, but lacks of 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. Beside \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, to write 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, that provides several of such commands for different purpose (see [Leh11]).

10.7. Extension of the Mathematical Features of ε-TeX

ε-TeX, that is meanwhile used by LaTeX and needed by KOMA-Script, provides 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, command \XmodY 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 is has to be prefixed by `\the`.

Control Package Dependencies with scrlfile

The introduction of LaTeX 2$_\varepsilon$ in 1994 brought many changes in the handling of LaTeX extensions. Today the package author has many macros available to determine if another package or class is employed and whether specific options are used. The author can load other packages or can specify options in the 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.

11.1. About Package Dependencies

More and more frequently, different packages either newly define or redefine the same macro again. In such a case the order in which a package is loaded becomes very important. For the user it sometimes becomes very difficult to understand the behaviour, and in some cases the user wants only to react to the loading of a package. This too is not really a simple matter.

Let us take the simple example of loading the package longtable with a KOMA-Script document class. The longtable package defines table captions very well suited to the standard classes, but the captions are totally unsuitable for documents using KOMA-Script and also do not react to the options of the provided configuration commands. In order to solve this problem, the longtable package commands which are responsible for the table captions need to be redefined. However, by the time the longtable package is loaded, the KOMA-Script class has already been processed.

Until the present, the only way for KOMA-Script to solve this problem was to delay the redefinition until the beginning of the document with help of the macro `\AtBeginDocument`. If the user wants to change the definitions too, it is recommended to do this in the preamble of the document. However, this is impossible since later at `\begin{document}` KOMA-Script will again overwrite the user definition with its own. Therefore, the user too has to delay his definition with `\AtBeginDocument`.

Actually, KOMA-Script should not need to delay the redefinition until `\begin{document}`. It would be enough to delay exactly until the package longtable has been loaded. Unfortunately, the LaTeX kernel does not define appropriate commands. The package scrlfile 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 scrlfile allows this, too.

The employment of scrlfile is not limited to package dependencies only. Even dependencies on any other file can be considered. For example, the user can be warned if the not uncritical file `french.ldf` has been loaded.

Although the package is particularly of interest for package authors, there are of course applications for normal LaTeX users, too. Therefore, this chapter gives and explains examples for both groups of users.

11.2. Actions Prior to and After Loading

scrlfile can execute actions both before and after the loading of files. In the commands used to do this, distinctions are made 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 scrlfile redefines the well known LaTeX command `\InputIfFileExists`. If this macro does not have the expected definition then scrlfile issues a warning. This is for the case that in future LaTeX versions the macro can have a different definition, or that another package has already redefined it.

The command `\InputIfFileExists` is used by LaTeX every time a file is to be loaded. This is independent of whether the actual load command is `\include`, `\LoadClass`, `\documentclass`, `\usepackage`, `\RequirePackage`, or similar. Exceptionally, the command

```
\input foo
```

loads the file `foo` without utilizing `\InputIfFileExists`. Therefore, one should always use

```
\input{foo}
```

instead. Notice the parentheses surrounding the file name!

```
\BeforeClass{class}{instructions}
\BeforePackage{package}{instructions}
```

These two commands work in the same way as `\BeforeFile`. The only difference is that the document class *class* and the LaTeX 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 LaTeX 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 hasn't been finished yet. Nevertheless, *instructions* will be executed before the instructions of `\AtEndOfClass` or `\AtEndOfPackage` when loading of the class or package hasn't been finished already.

If a class uses `\AtEndOfClass` or a package uses `\AtEndOfPackage` to execute instructions after the class of 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

v3.09 version, `\AfterClass!` respectively `\AfterPackage!`.

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,

v3.09 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 LaTeX user. Suppose a document that should be available as a PS file, using LaTeX and dvips, as well as a PDF file, using pdfLaTeX. 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 pdfLaTeX method, since here hyperlinks can be broken across multiple lines. However, if a hyperref driver for dvips or hyperTeX 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{scrlfile}
\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 scrlfile 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 wasn't 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
}
```

wouldn't 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 to 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@filesw
    \immediate\write\@auxout{%
      \protect\writethistoaux%
    }%
  \fi
}
```

This would be successful even if the final `\clearpage` inside of `\end{document}` wouldn't 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`.

11.3. 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 scrlfile 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 LaTeX 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, scrlfile redefined `\InputIfFileExists`.

Example: You want LaTeX 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 is would also be possible to replace a package or class by another one. But LaTeX 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 LaTeX 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 scrlfile 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 `\ReplaceFile`. 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 doesn't 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 doesn't. 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 scrlfile has to be loaded before the class using `\RequirePackage` instead of `\usepackage`.

11.4. 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 scrlfile again provides a solution.

`\PreventPackageFromLoading{`*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 XeLaTeX oder LuaLaTeX the first time. In this case loading of fontenc wouldn't 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{scrlfile}
\PreventPackageFromLoading{fontenc,lmodern}
\documentclass{firmenci}
```

The example above also shows, that package scrlfile may be loaded before the class. In this case `\RequirePackage` has to be used, because `\usepackage` before `\documentclass` is not permitted.

`\StorePreventPackageFromLoading{\command}`
`\ResetPreventPackageFromLoading`

v3.08 `\StorePreventPackageFromLoad` defines `\command` to be the current list of packages, that should be prevented from being loaded. In opposite to this, v3.08 `\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.

Spare 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. Seams 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 .
\ch@ck ...\else \errmessage {No room for a new #3}
                                                \fi
```

Last but not least, immediately opening one more write handle for every table of contents, list of figures, list of tables, and so on has one more disadvantage. These are not only set by the corresponding commands, they also couldn't not be set once more, because their helper files are empty after the corresponding commands until the end of the document.

Package scrwfile provides a general change of the LATEX kernel, that may solve both problems.

12.1. General Modifications of the LATEX Kernel

LATEX classes use the LATEX kernel command `\@starttoc` for output and opening of a new table of contents, list of float, e. g., `\listoffigures` or `\listoftables`, or similar. Thereby LATEX not only reads the helper file with the contents of the directory, but opens the helper file for writing also. Nevertheless, if afterwards new entries to these lists of floats will be made using `\addcontentsline`, then this will not write immediately to the helper file. Instead of LATEX writes `\@writefile` commands into the `aux`-file. Only while reading the `aux`-file while the end of the document, those `\@writefile` commands generates real write operations into the helper files. LATEX even doesn't close the helper files explicitly. Instead of LATEX counts on TEX to close all open files at the end.

This procedure means, that all the helper files are open for writing from the output of the contents, i. e. at the front matter of the document until the end of the document, though their content is written after the end of the document. scrwfile's basic approach is exactly this: redefinition of `\@starttoc` and `\@writefile`.

Surely, changes of the LATEX kernel always may potentially result in incompatibilities with other packages. In case of scrwfile this may be happen with all packages, that redefine `\@starttoc` or `\@writefile` too. Sometimes changing the order of loading the packages may help.

In fact, such problems with scrwfile haven't been reported currently, though several users have tested the package for more than a year until first release. Nevertheless, if you find such

a problem, please contact the KOMA-Script author.

12.2. The Single File Feature

Just while loading the package using, e. g.,

```
\usepackage{scrwfile}
```

scrwfile will redefine `\@starttoc` to not longer allocate a file handle or open any file for writing. Immediately before closing the `aux`-file in `\end{document}` it will redefine `\@writefile` to not 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 LaTeX 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, after loading scrwfile one more write file handle will be available, 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` won't be limited any longer.

12.3. 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 longer, this may 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. Because of this, 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*, too.

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 `\listof`*destination* 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, too.

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}
\AtBeginDocument{%
  % aux first opened at \begin{document}. So wait until this:
  \addtocontents{toc}{% first toc entry:
    \protect\addcontentsline{stoc}{% write to Short Contents
      \protect\tableofcontents% Contents
    }%
  }%
}
\begin{document}
\setcounter{tocdepth}{1}% show chapters only
\listofstoc% Write short table of contents
\setcounter{tocdepth}{6}% show all levels
\tableofcontents% Write table of contents
```

You need at least three LaTeX runs to get a short table of contents and a detailed table of contents. The detailed one produces an entry at the short one but this entry will not be part of the detailed table of contents.

12.4. State of Development Note

Though this package has been tested by several users and even is in productivity usage several times, development is not finished yet. Because of this especially internal functionality may be changed in future. Most likely the package will be extended. Some code for extensions is already in the package. Nevertheless, currently user haven't make requests for such extension. So there's currently no user documentation for this.

Management of Tables and Lists of Contents Using tocbasic[1]

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 controll about these. For examples package tocbasic delegates language control of all these tables and lists of contents to package babel(see [Bra01]). So automatic change of language will be provides 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.

13.1. Basic Commands

Basic commands are used to handle a list of all extensions known for files representing a table of contents or list of something. Entries to such files are written using `\addtocontents` or `\addcontentsline` typically. There are also commands to do something for all known extensions. And there are commands to set or unset features of an 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*`}{`*true part*`}{`*false part*`}`

This command may be used to ask, wether or not a *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 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%
  }{%
    Each extension may be used only
    once.\MessageBreak
    The class or another package already
```

[1] This chapter has been generated from the source of the package. It’s not a 1-to-1 translation of the German manual. Currently translation has not been finished

```
    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 at 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 you 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 list of figure and list of table will also handle the lists of this package, that are already added to the known extensions, when the option is used. This is because KOMA-Script already registers file extension “`lof`” for the list of figures and file extension “`lot`” for the list of tables with owner “`float`” and sets options for this owner.

`\AtAddToTocList[`*owner*`]{`*instructions*`}`

This command adds the *instructions* to a internal list of instructions, that will be processed, whenever a file extension with the given *owner* will be added to the list of known extensions using `\addtotoclist`. The optional argument is handled in the same kind as with 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`, `\@currext` will be set to the extension of the currently added extension.

Example: tocbasic itself uses

```
\AtAddToTocList[]{%
  \expandafter\tocbasic@extend@babel
  \expandafter{\@currext}%
}
```

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 13.3, page 270 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 extensions, but also needs some additional effort. With `\doforeachtocfile`you'll win for this. The command provides to processes *instructions* for every known file extension of the given *owner*. While processing the *instructions* `\@currext` is the extension of the current file. If you omit the optional argument, `[`*owner*`]`, every known file extensions independent from the owner will be used. If the optional argument is empty, only file extensions with an empty owner will be processed.

Example: If you want to type out all known extensions, you may simply write:

```
\doforeachtocfile{\typeout{\@currext}}
```

and if only the extensions of owner "`foo`" should be typed out:

```
\doforeachtocfile[foo]{\typeout{\@currext}}
```

`\tocbasicautomode`

This command redefines LaTeX 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 13.3, page 271 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 partial under control of tocbasic. Whether or not this will make the wanted result, depends on the individual table of contents and lists of something. At least the babel control extension for all those tables of contents and lists of something 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.

13.2. Creating a Table of Contents or List of Something

At the previous section you've seen commands to handle a list of known 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 see commands to handle the file corresponding with an extension or the list of known extensions.

`\addtoeachtocfile[`*owner*`]{`*content*`}`

This command writes *content* to the files of every known file extension of *owner* using LaTeX kernel command `\addtocontents`. If you omit the optional argument, *content* it written to the files of every known file extension. Bay the way: The practical file name is build from `\jobname` and the file extension. While writing the *content*, `\@currext` 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`.

`\addcontentslinetoeachtocfile[`*owner*`]{`*level*`}{`*contentsline*`}`

This command is something like `\addcontentsline` from LaTeX kernel. In difference to that it writes the *contentsline* not only into one file, but into all files of all known file extensions or of all known file extensions of a given owner.

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.

```
\addcontentslinetoeachtocfile{chapter}{%
  \protect\numberline{\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 `\@currext` is the file extension of the file into which *contentsline* will be written.

`\listoftoc[`*list of title*`]{`*extension*`}`
`\listoftoc*{`*extension*`}`
`\listofeachtoc[`*owner*`]`
`\listof`*file-extension*`name`

These commands may be used to set the "list of" corresponding to file 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 LaTeX kernel macro `\@starttoc`.

The version without star, sets 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

\listof*extension*name was defined, that will be used. If that is also not defined, a standard ersatz 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 \listof*file-extension*name should be defined to get the correct titles.

It is recommended to define \listof*file-extension*name for all used file 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 268 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 LaTeX 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 12) may solve this problem.

\BeforeStartingTOC[*extension*]{*instructions*}
\AfterStartingTOC[*extension*]{*instructions*}

Sometimes it's useful, to process *instructions* immediately before reading the helper file of a list of something. 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 individuel one and the general after hook will be called after the individuel one. While calling the hooks \@currext is the extension of the toc-file and should not be changed.

An example for usage of \AfterStartingTOC may be found in section 12.3 at ??.

\BeforeTOCHead[*file extension*]{*instructions*}
\AfterTOCHead[*file extension*]{*instructions*}

This commands may be used to process *instructions* before or after setting the title of a list of something corresponding to given *file 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 individuel one and the general after hook will be called after the individuel one. While calling the hooks \@currextIndexCmd@currext 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 scrpage2 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 shouldn't 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 doesn't 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.

`\deftocheading{`*file extension*`}{`*definition*`}`

The package tocbasic contains a standard definition for typesetting headings of tables of contents or lists of something. 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 `\deftocheading`. Thereby *file extension* is the file extension of the corresponding helper 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{`*file extension*`}{`*feature list*`}`
`\unsettoc{`*file extension*`}{`*feature list*`}`

This commands set up and unset features bound to an *file extension*. The *feature list* is a comma seperated 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 `\deftocheading`, that user is responsible for the evaluation of the feature. The KOMA-Script classes set this feature using option `listof=leveldown`important`listof=leveldown` for all file extensions of the owner `float`.

nobabel prevents usage of the language switch of babel at the helper file with the corresponding *file extension*. This feature should be used only for helper files, that contain text in one language only. Changes of the language inside of the document will not longer regarded at the helper file. Package scrwfile uses this feature also for clone destinations, because those will get the language change from the clone source already.

v3.10 **noprotrusion** prevents disabling character protrusion at the lists of something. Character protrusion at the lists will be disabled by default if package microtype or another package, that supports `\microtypesetup`, was loaded. So if you want protrusion at the lists, you have to set this feature. But note, that with character protrusion entries at the list may be set wrong. This is a known issue of character protrusion.

numbered uses a numbered heading for the table of contents or list of something 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 extensions of the owner `float`.

v3.01 **onecolumn** typesets the corresponding table of contents or list of something with internal one column mode of `\onecolumn`. This will be done only, if the corresponding table of contents or list of something doesn't use feature `leveldown`. The KOMA-Script classes scrbook and scrreprt activate this feature with `\AtAddToTocList` (see section 13.1, page 263) for all lists of something 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 recognized or changed by this option.

totoc writes the title of the corresponding table of contents or the list of something 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 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 list 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 121 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 extension`* "`load`" 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 10.3, page 241).

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 recognize 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{`*file extension*`}{`*feature*`}{`*true-instructions*`}{`*false-instructions*`}`

This command may be used, to test, if a *feature* was set for *file 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 `\deftocheading` but want to support the features `totoc`, `numbered` or `leveldown`.

13.3. 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{`*file extension*`}`

The Package babel (see [Bra01]) respectively a LaTeX kernel that has been extended by the language management of babel writes instructions to change the language inside of the files with the file 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 extensions too. Argument *file 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 extension* that will be added to the list of known extensions using `\addtotoclist`. The may be suppressed using feature `nobabel` (see `\setuptoc`, section 13.2, page 268). For the file 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 by lists of something, that should not be under control of tocbasic, and to are not in tocbasic's list of known file extensions, but nevertheless should be handled by the language change mechanism of babel. The command may be used explicitly for those files. But pleas note, that this should be done only once per file extension!

`\tocbasic@starttoc{`*extension*`}`

This command is something like the LaTeX kernel macro `\@starttoc`. It's the command behind `\listoftoc*` (siehe section 13.2, page 265). Authors of classes or packages who want to participate from the advantages of tocbasic should at least use this command. Nevertheless it's 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, `\@currext` will be set to the file extension of the current helper file, so this will be available while the execution of the hooks, that will be done before and after reading the helper files.

Because of LaTeX will immediately open a new helper 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 12 for more information about that package.

`\tocbasic@@before@hook`
`\tocbasic@@after@hook`

The hook `\tocbasic@@before@hook` will be executed immediately before reading a helper file for a table of contents or list of something 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 a helper 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 favor those commands! These hooks shouldn't be used to generate any output!

If neither `\listofeachtoc` nor `\listoftoc` nor `\listoftoc*` are used for the output of a table of contents or list of something, 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 helper 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 table of contents or list of something, 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 list, either the default heading or the individually defined heading. If you define your own list command not using `\listoftoc` you may use `\tocbasic@listhead`. In this case you should define `\@currext` to be the file extension of the corresponding helper file before using `\tocbasic@listhead`.

`\tocbasic@listhead@`*extension*`{`*title*`}`

This command is used in `\tocbasic@listhead` to set the individual headings, optional toc-entry, and running head, if it was defined. If it was not defined it will be defined and used in `\tocbasic@listhead` automatically.

13.4. 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 own 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. LaTeX 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 helper file with name `\jobname` and a file extension.

```
\newcommand*{\ext@remarkbox}{lor}
```

The file extension of the helper file for the list of `remarkbox`es 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 lists of something also to the new one.

Next we define a title or heading for the list of `remarkbox`es:

```
\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 figure. This would be the easiest solution. A more explicit would be, e. g.,

```
\newcommand*{\l@remarkbox}{\@dottedtocline{1}{1em}{1.5em}}
```

Additionally you may want structure the list of remarks depending on chapters.

```
\setuptoc{lor}{chapteratlist}
```

The KOMA-Script classes provide that feature and may 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 simply

```
\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 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 extension `lor`.

13.5. 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 is may be even easier.

`\DeclareNewTOC[`*options*`]{`*extension*`}`

v3.06

This command declares in one step only a new list of something, the heading of that list, the term used for the entries to the list, 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 helper file, that represents the list of something. See section 13.1 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 13.1.

v3.06

Table 13.1.: Options for command `\DeclareNewTOC`

`atbegin=`*instructions*
: The *instructions* will be executed at the begin of the floating or non-floating environment.

`atend=`*instructions*
: The *instructions* will be executed at the end of the floating or non-floating environment.

`counterwithin=`*LaTeX 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 LaTeX 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 where already defined before.

`hang=`*length*
: The amount of the hanging indent of the entries for that list. If not given, 1.5 em will be used like standard classes use for entries to list of figures or list of tables.

...

Table 13.1.: Options for command `\DeclareNewTOC` (continuation)

`indent=`*`length`*
: The indent value for the entries of that list. If not given, 1 em will be used like standard classes use for entries to list of figures or list of tables.

`level=`*`number`*
: The level of the entries of that list. If not given level 1 will be used like standard classes use for entries to list of figures or list of tables.

`listname=`*`string`*
: The name of the list of foo. If not given the value of `types` with upper case first char 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 postfix "`-`".

`owner=`*`string`*
: The owner as described in the sections before. If no owner was given owner `float` will be used.

`type=`*`string`*
: sets the type of the new declared list. The type will be used e. g. to defined a `\listof`*`string`*. If no type is set up the extension from the mandatory argument will be used.

`types=`*`string`*
: the plural of the type. If no plural was given the value of `type` with addendum "s" will be used.

Example: Using `\DeclareNewTOC` reduces the example from section 13.4 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.

Hacks for Third-Party Packages by Package scrhack

Some packages from other authors may 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 may prevent scrhack from redefining macros of individual packages.

14.1. 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 couldn't do the needed patch. Contrary, in extreme cases the patch may 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 couldn't be guaranteed forever.

14.2. Early or late Selection of Options

All of what is described in section 2.4 is generally applicable.

14.3. 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 extention. 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 furthor 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 may result in an error or break the functionality of the commands.

To clear all this disadvantages and problems, KOMA-Script officially doesn't support this old interface any more. Instead KOMA-Script warns, if the old interface is used. At the same time package tocbasic (see chapter 13) 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 has been alluded above.

Though the effort using that package is very small, the authors of most of the packages, that are using the old interface, haven't 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 may be found in section 13.2.

If the modification for any of the packages is not wanted or causes problems, then is may be deactivated selectively with option `float=false`, `floatrow=false`, or `listings=false`. Please note that changing these options after loading the corresponding package would do it!

14.4. 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 wasn't patient enough to wait more than a year for the change of hyperref, a corresponding patch has been added to scrhack. This may 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.

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 [KM08] only. This shouldn't be a problem, because the average user, who only want to use the package, won't 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 shouldn't use them any longer.

15.1. Expert Commands

This section describes commands, that aren't 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 LaTeX lengths, whenever the type area is newly calculated inside of the document, iė., 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*`}`

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}
\rule{\textwidth}{\textheight}
```

```
\storeareas\MySavedValues
\KOMAoptions{paper=landscape,DIV=current}
\rule{\textwidth}{\textheight}

\clearpage
\MySavedValues
\rule{\textwidth}{\textheight}
\end{document}
```

Command \clearpage before calling \MySavedValues is important to restore the saved values at the next page.

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.

```
\AfterCalculatingTypearea{instructions}
\AfterCalculatingTypearea*{instructions}
\AfterSettingArea{instructions}
\AfterSettingArea*{instructions}
```

These commands manage *hooks*. \AfterCalculatingTypearea and it's 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 it's 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.

15.2. 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 [KM08] only.

15.3. 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 [KM08] 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 [KM08] only. This shouldn't be a problem, because the average user, who only want to use the package, won't 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 shouldn't use them any longer.

Sorry, currently additional information to this may be found at the same point of the German KOMA-Script book [KM08] only.

16.1. Additional Information to User Commands

Sorry, currently additional information to this may be found at the same point of the German KOMA-Script book [KM08] only.

16.2. 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 [KM08] only.

16.3. 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 isn't 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 don't 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 an authors of wrapper classes may redefined 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 live this wouldn't 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 85 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.

`\@fontsizefilebase`

The prefix `scrsize` for file names of font size files, that has been mentioned in section 16.1 at page 282 is only the default of the internal macro `\@fontsizefilebase`. This default is used only, if the macro hasn't 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 51). 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 51) 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 name*s may increase usability, because different users may find different names more or less intuitive. KOMA-Script itself defines a lot of *alias name*s for several *element*s.

```
\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.

`\raggedchapterentry`

This command is not really comparable with commands like `\raggedsection`. In opposite to those commands, that provide to print corresponding elements either justified or right- or left-aligned or centered, `\raggedchapterentry` currently only provides to print chapter entries at the table of contents justified or left-aligned. Default is justified. For left-aligned the command has to be redefined to be `\raggedright`. See also the limitation note in section 16.2, page 282 for this.

Sorry, currently additional information to this may be found at the same point of the German KOMA-Script book [KM08] only.

`\chapterheadstartvskip`
`\chapterheadendvskip`
`\partheadstartvskip`
`\partheadmidvskip`
`\partheadendvskip`
`\partheademptypage`

These 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 vertical distance before the chapter heading. Analogues `\chapterheadendvskip` is designed to be a command, that inserts vertical distance after the chapter heading.

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 therefor is already part of the default of `\partheadendvskip` in scrbook and scrreprt. Command `\partheademptypage` is used to produce the empty page after the part heading page of scrbook and scrreprt.

scrbook v3.02, scrreprt

scrbook, scrreprt, scrartcl

The scrbook and scrreprt default of the six commands are listed in table 16.1 and table 16.2. The scrartcl defaults of the three at this class defined commands are listed in table 16.3. An example, that redefines `\chapterheadstartvskip` and `\chapterheadendvskip` to print extra rules above and below the chapter heading, may be found at [KDP] (in German).

Table 16.1.: defaults of the commands for the vertical distances of chapter headings with scrbook and scrreprt

Mit `headings=big`:

command	default definition
`\chapterheadstartvskip`	`\vspace*{2.3\baselineskip}`
`\chapterheadendvskip`	`\vspace*{1.725\baselineskip plus .115\baselineskip minus .192\baselineskip}`

Mit `headings=normal`:

command	default definition
`\chapterheadstartvskip`	`\vspace*{2\baselineskip}`
`\chapterheadendvskip`	`\vspace*{1.5\baselineskip plus .1\baselineskip minus .167\baselineskip}`

Mit `headings=small`:

command	default definition
`\chapterheadstartvskip`	`\vspace*{1.8\baselineskip}`
`\chapterheadendvskip`	`\vspace*{1.35\baselineskip plus .09\baselineskip minus .15\baselineskip}`

Table 16.2.: defaults of the commands for the vertical distances of part headings with scrbook and scrreprt

command	default definition
`\partheadstartvskip`	`\null\vfil`
`\partheadmidvskip`	`\par\nobreak\vskip 20pt`
`\partheadendvskip`	`\vfil\newpage`
`\partheademptypage`	`\if@twoside\if@openright \null\thispagestyle{empty}% \newpage \fi\fi`

Table 16.3.: defaults of the commands for the vertical distances of part headings with scrartcl

command	default definition
`\partheadstartvskip`	`\addvspace{4ex}`
`\partheadmidvskip`	`\par\nobreak`
`\partheadendvskip`	`\vskip 3ex`

\appendixmore

There is a peculiarity within the \appendix command in the KOMA-Script classes. If the command \appendixmore is defined, this command is executed also by the \appendix command. Internally the KOMA-Script classes scrbook and scrreprt take advantage of this behaviour to implement the options appendixprefix (see section 3.16, page 80). 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.

scrbook, scrreprt

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 80). 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 would write in the preamble of your document:

```
\newcommand*{\appendixmore}{%
  \renewcommand*{\othersectionlevelsformat}[3]{%
    \ifthenelse{\equal{##1}{section}}{%
      \appendixname~}{}%
    ##3\autodot\enskip}
  \renewcommand*{\sectionmarkformat}{%
    \appendixname~\thesection\autodot\enskip}}
```

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

An alternative implementation would be:

```
\newcommand*{\appendixmore}{%
```

```
\renewcommand*{\othersectionlevelsformat}[3]{%
  \ifstr{##1}{section}{\appendixname~}{}%
  ##3\autodot\enskip}
\renewcommand*{\sectionmarkformat}{%
  \appendixname~\thesection\autodot\enskip}}
```

Thereby used command `\ifstr` is provided by KOMA-Script. See section 10.3, page 242.

Redefined commands are explained in more detail in section 3.16, page 92 and page 94.

`\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 don't 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 127) 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 [Mit00]) 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.

16.4. 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 [KM08] only.

Additional Information about the Letter Class scrlttr2

This chapter gives additional information about the KOMA-Script class scrlttr2. Some parts of the chapter are subject to the KOMA-Script book [KM08] only. This shouldn't be a problem, because the average user, who only want to use the package, won't 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 scrlttr2 class.

17.1. 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 scrlttr2 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 scrlttr2 are defined with macros instead of lengths. The drawback of this approach is that computations with macros is somewhat more complicated than with real lengths.

It can be pointed out that the now recommended LATEX installation with ε-TEX no longer suffers from the above-mentioned limitation. However, that improvement came too late for scrlttr2.

The pseudo-lengths defined and uses by scrlttr2 are listed in table 17.1. 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 17.1 at page 296. Beside the pseudo-lengths for the modifiable distances, also some lengths, that 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 17.1.: Pseudo-lengths provided by class scrlttr2

`backaddrheight`
: height of the return address at the upper edge of the address field (section 17.1.3, page 301)

`bfoldmarklength`
: length of the bottommost folding mark (section 17.1.1, page 297)

`bfoldmarkvpos`
: vertical distance of bottommost folding mark from top paper edge (section 17.1.1, page 297)

`firstfoothpos`
: 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 17.1.8, page 305)

`firstfootvpos`
: vertical distance of letter footer from top paper edge (section 17.1.8, page 305)

`firstfootwidth`
: width of letter footer (section 17.1.8, page 305)

`firstheadhpos`
: 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 17.1.2, page 299)

`firstheadvpos`
: vertical distance of letterhead from top paper edge (section 17.1.2, page 299)

`firstheadwidth`
: width of the letterhead (section 17.1.2, page 299)

`foldmarkhpos`
: horizontal distance of the horizontal folding marks from left paper edge (section 17.1.1, page 298)

`foldmarkvpos`
: vertical distance of the vertical folding marks from the top paper edge (section 17.1.1, page 298)

...

Table 17.1.: Pseudo-lengths provided by class scrlttr2 (*continued*)

`fromrulethickness`
: thickness of an optional horizontal rule in the letterhead (section 17.1.2, page 299)

`fromrulewidth`
: length of an optional horizontal rule in letterhead (section 17.1.2, page 299)

`lfoldmarkhpos`
: horizontal distance of the vertical folding mark from the left paper edge (section 17.1.1, page 298)

`lfoldmarklength`
: length of the vertical folding mark (section 17.1.1, page 298)

`locheight`
: height of the field with the extended sender's information, of the value is not zero; `toaddrheight` will be used instead of zero value (section 17.1.4, page 302)

`lochpos`
: 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 `toaddrhpos` will be used instead of zero value (section 17.1.4, page 302)

`locvpos`
: vertical distance of the field with the extended sender's information from the top paper edge, if the value is not zero; `toaddrvpos` will be used instead of zero value (section 17.1.4, page 302)

`locwidth`
: width of the field with the extended sender's information; for zero value width is calculated automatically with respect to option `locfield` that is described in section 4.10, page 168 (section 17.1.4, page 302)

`mfoldmarklength`
: length of the middle horizontal folding mark (section 17.1.1, page 298)

`mfoldmarkvpos`
: vertical distance of the middle horizontal folding mark from the top paper edge (section 17.1.1, page 297)

...

Table 17.1.: Pseudo-lengths provided by class scrlttr2 (*continued*)

`pfoldmarklength`
: length of the puncher hole mark (section 17.1.1, page 298)

`refaftervskip`
: vertical skip below reference fields or business line (section 17.1.5, page 303)

`refhpos`
: 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 17.1.5, page 303)

`refvpos`
: vertical distance of reference fields or business line from top paper edge (section 17.1.5, page 303)

`refwidth`
: width of reference fields line (section 17.1.5, page 303)

`sigbeforevskip`
: vertical skip between closing and signature (section 17.1.7, page 304)

`sigindent`
: indentation of signature with respect to text body (section 17.1.7, page 304)

`specialmailindent`
: left indentation of mode of dispatch within address field (section 17.1.3, page 301)

`specialmailrightindent`
: right indentation of mode of dispatch within address field (section 17.1.3, page 301)

`subjectaftervskip`
: vertical distance after the subject (section 17.1.6, page 304)

`subjectbeforevskip`
: additional vertical distance before the subject (section 17.1.6, page 304)

`subjectvpos`
: vertical distance of the subject from the top paper edge; zero value will set the subject depending on option `subject` (section 17.1.6, page 304)

`tfoldmarklength`
: length of the topmost horizontal folding mark (section 17.1.1, page 298)

. . .

Table 17.1.: Pseudo-lengths provided by class scrlttr2 (*continued*)

`tfoldmarkvpos`	vertical distance of the topmost horizontal folding mark from the top paper edge (section 17.1.1, page 297)
`toaddrheight`	height of address field (section 17.1.3, page 300)
`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 17.1.3, page 300)
`toaddrindent`	left and right indentation of address within address field (section 17.1.3, page 301)
`toaddrvpos`	vertical distance of the address field from the top paper edge (section 17.1.3, page 300)
`toaddrwidth`	width of address field (section 17.1.3, page 300)

`\@newplength{`*`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.

`\@setplength[`*`factor`*`]{`*`pseudo-length`*`}{`*`value`*`}`
`\@addtoplength[`*`factor`*`]{`*`pseudo-length`*`}{`*`value`*`}`

Using the command `\@setplength` 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 135).

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 135) is used within *`value`*. To subtract the

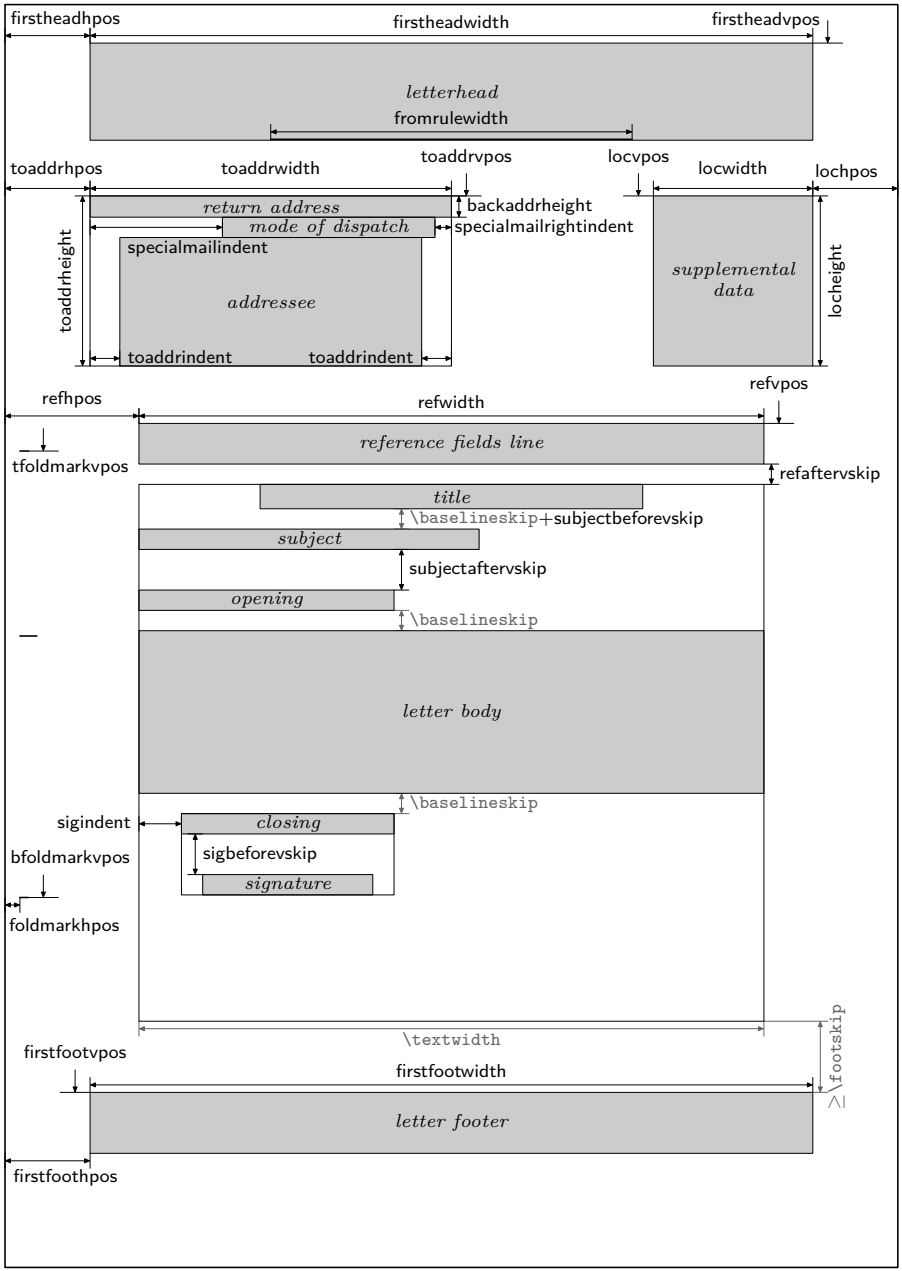


Figure 17.1.: Schematic of the pseudo-lengths for a letter

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.

17.1.1. 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.

`tfoldmarkvpos`
`mfoldmarkvpos`
`bfoldmarkvpos`

The letter class scrlttr2 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`, that of the middle foldmark by pseudo-length `mfoldmarkvpos`, and that 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.

v2.97e

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 150) is explicity activated.

`tfoldmarklength`
`mfoldmarklength`
`bfoldmarklength`
`pfoldmarklength`

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.

`foldmarkhpos`

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 150).

`lfoldmarkhpos`

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.

`lfoldmarklength`

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.

`foldmarkvpos`

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 150). At present there is only one vertical foldmark, which is designated the left vertical foldmark.

`foldmarkthickness`

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!

17.1.2. 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 scrlttr2, the letterhead is made independent of the page style setting, and is set by the command `\opening`. The position of the letterhead is absolute and independent of the type area. In fact, the first page of a letter, the page that holds the letterhead, is set using the page style `empty`.

`firstheadvpos`

The pseudo-length `firstheadvpos` gives the distance between the top edge of the paper and start of the letterhead. This value is set differently in the various predefined `lco`-files. A typical value is 8 mm.

`firstheadhpos`

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.

`firstheadwidth`

The pseudo-length `firstheadwidth` gives the width of the letterhead. This value is set differently in the various predefined `lco`-files. While this value usually depends on the paper width and the distance between the left edge of the paper and the adressee address field, it was the type area width in `KOMAold` and has a definite value of 170 mm in `NF`.

`fromrulethickness`
`fromrulewidth`

Depending on the class option `fromrule` (see section 4.10, page 154), 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-

v2.97c length to positive values using `\setplength` (see page 295). The default thickness of the line, `fromrulethickness`, is 0.4 pt.

17.1.3. 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.

`toaddrvpos`
`toaddrhpos`

These pseudo-lengths define vertical and horizontal position of the address field relative to the top-left corner of the paper. Values are set differently in the various predefined `lco`-files, according to standard envelope window measures. A special feature of `toaddrhpos` is that with negative values the offset is that of the right edge of the address field relative to the right edge of the paper. This can be found, for instance, in the case of `SN` 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 164).

`toaddrheight`

The pseudo-length `toaddrheight` defines the height of the address field, including the dispatch type. If no dispatch type is specified, then the address is vertically centered in the field. If a dispatch type is specified, then the address is set below the dispatch type, and vertically centered in the remaining field height.

`toaddrwidth`

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`.

`toaddrindent`

Additional indentation of the address within address field can be controlled by the pseudo-length `toaddrindent`. Its value applies to both left and right margin. Default value is 0 pt.

v3.03 With each of the settings `addrfield=PP`, `addrfield=image`, and `addrfield=backgroundimage` (see section 4.10, page 164) 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.

`backaddrheight`

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 the 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 5 mm 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 164) and `backaddress` (see section 4.10, page 164).

`specialmailindent`
`specialmailrightindent`

An optional dispatch type can be output within the address field between the return address and the addressee address, by setting the variable `specialmail`. Left and right alignment are determined by pseudo-lengths `specialmailindent` and `specialmailrightindent`, respectively. In the predefined `lco`-files provided by KOMA-Script, `specialmailindent` is set to rubber length `\fill`, while `specialmailrightindent` is set to 1 em. Thus the dispatch type is set 1 em from the address field's right margin.

`PPheadheight`
`PPheadwidth`

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 164) 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 scrlttr2 automatically changes pseudo-length `PPheadheight`'s usual default value from 0 mm into 20,74 pt and `PPheadwidth`'s default from 0 mm into 42 mm.

PPdatamatrixvskip

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 164). Class scrlttr2 automatically changes the default value 0 mm into 9 mm. The data matrix will be set right aligned with the Port-Payé head.

17.1.4. 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 is has been called *location field*.

locheight
lochpos
locvpos
locwidth

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 0 pt 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 168) 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 17.1.3, page 300).

17.1.5. 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 17.3, page 305.

`refvpos`

This pseudo-length gives the distance between the upper edge of the paper and the reference fields line. Its value is set differently in the various predefined `lco`-files. Typical values are between 80.5 mm and 98.5 mm.

`refwidth`
`refhpos`

This pseudo-length gives the width that is available for the reference fields line. The value is set typically to 0 pt in the predefined `lco`-files. This value has a special meaning: in no way does it determine that there is no available width for the business line; instead, this value means that the width will be calculated within the command `\opening`. Thus the calculated width depends on the determination of the options `refline` (see section 4.10, page 171). 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` isn't 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.

`refaftervskip`

This pseudo-length gives the vertical space that has to be inserted beneath the reference fields line. The value is set in the predefined `lco`-files. It directly affects the text height of the first page. A typical value lies between one and two lines.

17.1.6. 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.

`subjectvpos`

v3.01

If the value of this pseudo-length is 0 pt, the position of the subject is given by option `subject` (see section 4.10, page 174). 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]
\@setplength{subjectvpos}{\useplength{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.

`subjectbeforevskip`
`subjectaftervskip`

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.

17.1.7. 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.

`sigindent`
`sigbeforevskip`

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.

17.1.8. 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`.

`firstfootvpos`

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 154). 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 28) 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`.

v2.97e If the letter footer be switched off using option `firstfoot=false` (see section 4.10, page 176), then the setting of `firstfootvpos` is ignored, and instead `\paperheight` is applied. Thus, there remains a mimimum bottom margin of length `\footskip`.

`firstfoothpos`

v3.05 A positive value of pseudo-length `firstfoothpos` 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.

`firstfootwidth`

This pseudo-length gives the width of the letter's first page footer. The value is set equal to that of the pseudo-length `firstheadwidth` in the predefined `lco`-files.

17.2. 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}
\removereffields
\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*

won't be 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 171). 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 \removereffields 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 shouldn't 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 scrpage2. 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}
```

\ifkomavarempty{*name*}{*true*}{*false*}
\ifkomavarempty*{*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.

17.3. lco-Files for Experienced Users

Even though each paper size, that may be set up using package typearea, may be also used with scrlttr2, 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 their aren't any universal rules, to calculate, e. g., the 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.

17.3.1. Survey of Paper Size

At present there exist only parameter sets and lco-files for A4-sized or letter-sized paper. Nevertheless, class scrlttr2 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 their wouldn't 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{paper=a5}{a5}
\@setplength{tfoldmarkvpos}{\paperheight}
\@setplength{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`".

17.3.2. 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 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.

v3.04

`\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 51) 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 won't be 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 doesn't differ open and closed areas. Instead a minimal height will be used for open areas. The third available *visualization style* is `edge`. 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`, `\showUScommercia`, 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 51) 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 51) with element `measure`. Default is `\normalcolor` The dimensioning will be done in multiply of `\unitlength` with accuracy of 1/`\unitfactor`. Nevertheless accuracy of the TEX 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]{scrlttr2}
\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.

17.4. Language Support

The document class scrlttr2 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`

v3.09

v3.08

for American, and `british` and `UKenglish` for British), French, Italian, Spanish, Dutch, Croatian, Finnish, Norsk, and Swedish.

If the package babel (see [Bra01]) is used, one can switch between languages with the command `\selectlanguage{`*`language`*`}`. Other packages like german (see [Rai98a]) and ngerman (see [Rai98b]) also define this command. As a rule though, the language selection takes place already as a direct consequence of loading such a package.

There is one thing more to mention about language packages. The package french (see [Gau03]) redefines not only the terms of table 17.3, 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 scrlttr2 this is not the case, therefore the package french destroys the definition in scrlttr2 and does not work correctly with KOMA-Script. The author views this is a fault in the french package.

If one utilizes the babel package in order to switch to language `french` while the package french is simultaneously installed, then the same problems may likely occur, since babel employs definitions from the french package.

From babel version 3.7j this problem only occurs when it is indicated explicitly by means of an option that babel should use the french package. If it cannot be ascertained that a new version of babel is being used, it is recommended to use

```
\usepackage[...,frenchb,...]{babel}
```

in order to select french.

Other languages can possibly cause similar problems. Currently there are no known problems with the babel package for the german language and the various english language selections.

```
\captionsenglish
\captionsUSenglish
\captionsamerican
\captionsbritish
\captionsUKenglish
\captionsgerman
\captionsngerman
\captionsaustrian
\captionsnaustrian
\captionsfrench
\captionsitalian
\captionsspanish
\captionsdutch
\captionscroatian
\captionsfinnish
\captionsnorsk
\captionsswedish
```

If one switches the language of a letter then using these commands the language-dependent terms from table 17.3, page 314 are redefined. If the used language selection scheme does not support this then the commands above can be used directly.

```
\dateenglish
\dateUSenglish
\dateamerican
\datebritish
\dateUKenglish
\dategerman
\datengerman
\dateaustrian
\datenaustrian
\datefrench
\dateitalian
\datespanish
\datedutch
\datecroatian
\datefinnish
\datenorsk
\dateswedish
```

The numerical representation of the date (see option `numericaldate` in section 4.10, page 170) will be written depending on the selected language. Some examples can be found in table 17.2.

Table 17.2.: Language-dependent forms of the date

Command	Date example
\dateenglish	24/12/1993
\dateUSenglish	12/24/1993
\dateamerican	12/24/1993
\datebritish	24/12/1993
\dateUKenglish	24/12/1993
\dategerman	24. 12. 1993
\datengerman	24. 12. 1993
\dateaustrian	24. 12. 1993
\datefrench	24. 12. 1993
\dateitalian	24. 12. 1993
\datespanish	24. 12. 1993
\datedutch	24. 12. 1993
\datecroatian	24. 12. 1993.
\datefinnish	24.12.1993.
\datenorsk	24.12.1993
\dateswedish	24/12 1993

```
\yourrefname
\yourmailname
\myrefname
\customername
\invoicename
\subjectname
\ccname
\enclname
\headtoname
\headfromname
\datename
\pagename
\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 10.4. The definitions become active only at `\begin{document}`. Therefore they are not available in the LaTeX preamble and cannot be redefined there. In table 17.3 the default settings for `english` and `ngerman` can be found.

Table 17.3.: Default settings for language-dependent terms using languages `english` and `ngerman`, as long as language selection packages haven't been used

Command	`english`	`ngerman`
`\bankname`	Bank account	Bankverbindung
`\ccname`[1]	cc	Kopien an
`\customername`	Customer no.	Kundennummer
`\datename`	Date	Datum
`\emailname`	Email	E-Mail
`\enclname`[1]	encl	Anlagen
`\faxname`	Fax	Fax
`\headfromname`	From	Von
`\headtoname`[1]	To	An
`\invoicename`	Invoice no.	Rechnungsnummer
`\myrefname`	Our ref.	Unser Zeichen
`\pagename`[1]	Page	Seite
`\phonename`	Phone	Telefon
`\subjectname`	Subject	Betrifft
`\wwwname`	Url	URL
`\yourmailname`	Your letter of	Ihr Schreiben vom
`\yourrefname`	Your ref.	Ihr Zeichen

1 Normally these terms are defined by language packages like babel. In this case they are not redefined by scrlttr2 and may differ from the table above.

17.5. From Obsolete scrlettr to Current scrlttr2

With the June 2002 release of scrlttr2 (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 LaTeX 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 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 [KM08] only.

Japanese Letter Support for scrlttr2[1]

Since version 2.97e scrlttr2 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.

[1] This 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

[1] Because 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.
[2] Two slightly different sizes are sold as You 1; the smaller size (173 mm×118 mm) is the paper-industry standard size.
[3] Twice 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 scrlttr2 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`

[1] Names (acting as subtype information) are taken from the manufacturer catalog.
[2] Given as width by height in millimeters.
[3] Given as offset from left (l) or right (r), followed by offset from bottom (b) or top (t).
[4] The `lco` file, which provides support (see table A.7).
[5] In the absence of any other information, a numerical variation number for the subtype name is provided.
[6] Dimensions apply when envelope is held in portrait mode.

Table A.7.: `lco` files provided by scrlttr2 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

[1] Window 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, scrlttr2 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.

Bibliography

In the following you can find many references. All of them are referenced from the main text. In many cases the reference points to documents or directories which can be accessed via Internet. In these cases the reference includes a URL instead of a publisher. If the reference points to a LaTeX package then the URL is written in the form "CTAN://*destination*". The prefix "CTAN://" means the TeX archive on a CTAN server or mirror. For example, you can substitute the prefix with ftp://ftp.ctan.org/tex-archive/. For LaTeX packages it is also important to mention that we have tried to give a version number appropriate to the text that cites the reference. But for some packages is 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/.

[BCJ+05] Johannes Braams, David Carlisle, Alan Jeffrey, Leslie Lamport, Frank Mittelbach, Chris Rowley, and Rainer Schöpf:
The LaTeX2e Source, December 2005.

[Bra01] Johannes Braams:
Babel, a multilangual package for use with LaTeX's standard document classes, February 2001.
CTAN://macros/latex/required/babel/.

[Car99a] David Carlisle:
The ifthen package, September 1999.
CTAN://macros/latex/base/.

[Car99b] David Carlisle:
The keyval package, March 1999.
CTAN://macros/latex/required/graphics/.

[Car99c] David Carlisle:
The tabularx package, January 1999.
CTAN://macros/latex/tools.

[Car99d] David P. Carlisle:
Packages in the 'graphics' bundle, February 1999.
CTAN://macros/latex/required/graphics/.

[Car04] David Carlisle:
The longtable package, February 2004.
CTAN://macros/latex/required/tools/.

[Che11] Florent Chervet:
tabu and longtabu, February 2011.
CTAN://macros/latex/contrib/tabu/.

[Dal99] Patrick W. Daly:
Natural sciences citations and references, May 1999.
CTAN://macros/latex/contrib/natbib/.

[DUD96] DUDEN:
Die deutsche Rechtschreibung. DUDENVERLAG, Mannheim, 21st edition, 1996.

[Fai05] Robin Fairbairns:
footmisc — a portmanteau package for customising footnotes in LaTeX, March 2005.
CTAN://macros/latex/contrib/footmisc.

[FAQ11] *Tex frequently asked questions on the web*, April 2011.
http://www.tex.ac.uk/faq/.

[Gau03] Bernard Gaulle:
Les distributions de fichiers de francisation pour latex, December 2003.
CTAN://language/french/.

[KDP] *KOMA-Script Homepage*.
http://www.komascript.de.

[Keh97] Roger Kehr:
XINDY, A Flexible Indexing System, 1997.

[Ker07] Dr. Uwe Kern:
Extending LaTeX's color facilities: the xcolor package, January 2007.
CTAN://macros/latex/contrib/xcolor/.

[Kie99] Axel Kielhorn:
adrconv, November 1999.
CTAN://macros/latex/contrib/adrconv/.

[KM08] Markus Kohm and Jens Uwe Morawski:
KOMA-Script. Edition DANTE. Lehmanns Media, Berlin, 3rd edition, 2008, ISBN 978-3-86541-291-1.

[Koh02] Markus Kohm:
Satzspiegelkonstruktionen im Vergleich. Die TeXnische Komödie, 4:28–48, 2002. DANTE e. V.

[Koh03] Markus Kohm:
Moderne Briefe mit KOMA-Script. Die TEXnische Komödie, 2:32–51, 2003. DANTE e. V.

[Koh06] Markus Kohm:
Creating more than one index using splitidx and splitindex, June 2006.
CTAN://macros/latex/contrib/splitindex.

[Lam87] Leslie Lamport:
MakeIndex: An Index Processor For LaTeX, February 1987.
CTAN://indexing/makeindex/doc/makeindex.pdf.

[Lap06] Olga Lapko:
The floatrow package, July 2006.
CTAN://macros/latex/contrib/floatrow/.

[Leh11] Philipp Lehman:
The etoolbox package, January 2011.
CTAN://macros/latex/contrib/etoolbox/.

[Lin01] Anselm Lingnau:
An improved environment for floats, July 2001.
CTAN://macros/latex/contrib/float/.

[Mit00] Frank Mittelbach:
An environment for multicolumn output, July 2000.
CTAN://macros/latex/required/tools/.

[Obe10] Heiko Oberdiek:
The engord package, March 2010.
CTAN://macros/latex/contrib/oberdiek/.

[OPHS99] Tobias Oetker, Hubert Partl, Irene Hyna, and Elisabeth Schlegl:
The Not So Short Introduction to LaTeX 2ε, April 1999.
CTAN://info/lshort/.

[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/.

[Rai98b] Bernd Raichle:
ngerman package, July 1998.
CTAN://language/german/.

[Sch09] Martin Schröder:
The ragged2e package, June 2009.
CTAN://macros/latex/contrib/ms/.

[Sch10] R Schlicht:
The microtype package: An interface to the micro-typographic extensions of pdfTEX, January 2010.
CTAN://macros/latex/contrib/microtype.

[Tea05a] LaTeX3 Project Team:
LaTeX 2ε font selection, November 2005.
CTAN://macros/latex/doc/fntguide.pdf.

[Tea05b] LaTeX3 Project Team:
LaTeX 2ε for authors, November 2005.
CTAN://macros/latex/doc/usrguide.pdf.

[Tea06] LaTeX3 Project Team:
LaTeX 2ε for class and package writers, February 2006.
CTAN://macros/latex/doc/clsguide.pdf.

[Tob00] Geoffrey Tobin:
setspace LaTeX package, December 2000.
CTAN://macros/latex/contrib/setspace/.

[Tsc87] Jan Tschichold:
Ausgewählte Aufsätze über Fragen der Gestalt des Buches und der Typographie. Birkhäuser Verlag, Basel, 2nd edition, 1987.

[Ume00] Hideo Umeki:
The geometry package, June 2000.
CTAN://macros/latex/contrib/geometry/.

[vO00] Piet van Oostrum:
Page layout in LaTeX, October 2000.
CTAN://macros/latex/contrib/fancyhdr/.

[WF00] Hans Peter Willberg and Friedrich Forssman:
Erste Hilfe in Typografie. Verlag Hermann Schmidt, Mainz, 2000.

[Wik] Wiki:
Deutsche TEX-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

R

S

T

Index of Commands, Environments, and Variables

B

C

D

E

F

M

N

O

P

Q

R

S

T

Index of Lengths and Counters

B

F

L

M

P

Index of Elements with Capability of Font Adjustment

Index of Files, Classes, and Packages

A

B

C

E

F

G

I

K

L

M

Index of Class and Package Options

D

E

F

H

I

K

L

M

N

O

P

R

S

T

U

V