KOMA-Script

a versatile LaTeX 2ε bundle

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

The Guide

KOMA-Script

Markus Kohm

2020-03-12

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

Legal Notes:

English translation of this manual by: Markus Kohm, Karl Hagen, DeepL, Kevin Pfeiffer, Gernot Hassenpflug, Krickette Murabayashi, Jens-Uwe Morawski, Jana Schubert, Jens Hühne, Harald Bongartz, Georg Grandke, Raimund Kohl, Stephan Hennig, Alexander Willand, Melvin Hendrix, and Arndt Schubert.

Free screen version without any optimization of paragraph and page breaks

To all my friends all over the world!

Preface to KOMA-Script 3.28

The KOMA-Script 3.28 manual, — not only the German version — once again benefits from the fact that a new edition of the print version [Koh20a] and the eBook version [Koh20b] will be published at almost the same time as this version. This has led to many improvements which also affect the free manual, in both the German and the English version.

In KOMA-Script 3.28 there are also some significant changes. In some cases, compatibility with earlier versions has been waived. Thus a recommendation from the ranks of *The LaTeX Project Team* regarding `\if...` statements is complied with. If you use such statements, you should refer to the manual again.

It is not just about the manual that I now receive little criticism. I conclude from this fact that KOMA-Script has reached the level that it fulfils all desires. At the same time, the project has — not only starting with the current release — reached a scale that makes it almost impossible for a single person to accomplish

- the search for and elimination of errors,
- the development and implementation of new functions,
- the observation of changes in other packages and the LaTeX kernel with regard to effects on KOMA-Script,
- the rapid response to such changes,
- the maintenance of the guides in two languages,
- help for beginners far beyond the functions of KOMA-Script down to the basic operation of a computer,
- assistance in the implementation of tricky solutions for advanced users and experts,
- moderation and participation in the maintenance of a forum for all kind of help around KOMA-Script.

While I am personally have most fun with the development of new functions, I consider troubleshooting in existing features, compatibility with new LaTeX kernel versions, and above all instructing users for the most important tasks. Therefore I will focus in the future on and new functions will be available only in exceptional cases. Therefore already in KOMA-Script 3.28 some experimental functions and packages have been removed. In future releases this should be continued.

This, of course, also reduces the effort for the documentation of new functions. Readers of this free, screen version, however, still have to live with some restrictions. So some information — mainly intended for advanced users or capable of turning an ordinary user into an

advanced one — is reserved for the printed book, which currently exists only in German. As a result, some links in this manual lead to a page that simply mentions this fact. In addition, the free version is scarcely suitable for making a hard-copy. The focus, instead, is on using it on screen, in parallel with the document you are working on. It still has no optimized wrapping but is almost a first draft, in which both the paragraph and page breaks are in some cases quite poor. Corresponding optimizations are reserved for the German book editions.

Another important improvement to the English guide has been accomplished by Karl Hagen, who has continued the translation of the entire manual. Many, many thanks to him! Everything that is fine in this English manual is because of him. Everything that is not good in this manual — like the translation of this preface — is because of me. Additional editors or translators, however, would still be welcome!

But the biggest thanks go to my family and above all to my wife. They absorb all my unpleasant experiences on the Internet. They have also tolerated it for more than 25 years, when I am again not approachable, because I am completely lost in KOMA-Script or some LaTeX problems. The fact that I can afford to invest an incredible amount of time in such a project is entirely thanks to my wife.

Markus Kohm, Neckarhausen in the foggy December of 2019.

Contents

List of Figures

List of Tables

Introduction

This chapter contains, among other things, 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 on how to install KOMA-Script and what to do if you encounter errors.

1.1. Preface

KOMA-Script is very complex. This is due to the fact that it consists of not just a single class or a single package but a bundle of many classes and packages. Although the classes are designed as counterparts to the standard classes, that does not mean they provide only the commands, environments, and settings of the standard classes, or that they imitate their appearance. The capabilities of KOMA-Script sometimes far surpass those of the standard classes. Some of them should be considered extensions to the basic capabilities of the LaTeX kernel.

The foregoing means that the documentation of KOMA-Script has to be extensive. In addition, KOMA-Script is not normally taught. That means there are no teachers who know their students and can therefore choose the teaching materials and adapt them accordingly. It would be easy to write documentation for a specific audience. The difficulty facing the author, however, is that the manual must serve all potential audiences. I have tried to create a guide that is equally suitable for the computer scientist and the fishmonger's secretary. I have tried, although this is actually an impossible task. The result is numerous compromises, and I would ask you to take this issue into account if you have any complaints or suggestions to help improve the current situation.

Despite the length of this manual, I would ask you to consult the documentation first in case you have problems. You should start by referring to the multi-part index at the end of this document. In addition to this manual, documentation includes all the text documents that are part of the bundle. See `manifest.tex` for a complete list.

1.2. Structure of the Guide

This manual is divided into several parts: There is a section for average users, one for advanced users and experts, and an appendix with further information and examples for those who want to understand KOMA-Script thoroughly.

part I is intended for all KOMA-Script users. This means that some information in this section is directed at newcomers to LaTeX. In particular, this part contains many examples that are intended to clarify the explanations. Do not hesitate to try these examples yourself and discover how KOMA-Script works by modifying them. That said, 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 [OPHS11] 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* [FAQ13]. Although the length of the *TeX Frequently Asked Questions on the Web* is considerable, you should get at least a rough overview of it and consult it in case you have problems, as well as this guide.

part II is intended for advanced KOMA-Script users, those who are already familiar with LaTeX or who have been working with KOMA-Script for a while and want to understand more about how KOMA-Script works, how it interacts with other packages, and how to perform more specialized tasks with it. For this purpose, we return to some aspects of the class descriptions from part I and explain them in more detail. In addition we document some commands that are particularly intended for advanced users and experts. This is supplemented by the documentation of packages that are normally hidden from the user, insofar as they do their work beneath the surface of the classes and user packages. These packages are specifically designed to be used by authors of classes and packages.

The appendix, which can only be found in the German book version, contains information beyond that which is covered in part I and part II. Advanced users will find background information on issues of typography to give them a basis for their own decisions. In addition, the appendix provides examples for aspiring package authors. These examples are not intended simply to be copied. Rather, they provide information about planning and implementing projects, as well as some basic LaTeX commands for package authors.

The guide's layout should help you read only those parts that are actually of interest. Each class and package typically has its own chapter. Cross-references to another chapter are thus usually also references to another part of the overall package. However, since the three main classes (scrbook, scrrprt, and scrartcl) largely agree, they are introduced together in chapter 3.

scrartcl Differences between the classes, e. g., for something that only affects the class scrartcl, are clearly highlighted in the margin, as shown here with scrartcl.

The primary documentation for KOMA-Script is in German and has been translated for your convenience; like most of the 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.

At this point you should know enough to understand the guide. It might, however, still be worth reading the rest of this chapter.

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.

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.

Special thanks go as well to the founders and members of 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 on the TEX newsgroups and mailing lists who answer questions and have helped me provide support for KOMA-Script.

My thanks also go to all those who have always encouraged me to go further and to implement this or that feature better, with fewer flaws, or simply as an extension. I would also like to thank the very generous donor who has given me the most significant amount of money I have ever been paid for the work done so far on KOMA-Script.

1.5. Legal Notes

KOMA-Script is 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 through their package management software. You should install and update KOMA-Script using these tools, if possible. Manual installation without using the package managers is described in the file `INSTALL.txt`, which is part of every KOMA-Script distribution. 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, packages, or another part of KOMA-Script, please do the following: First check on CTAN to see if a newer version of KOMA-Script has been released. If a newer version is available, install this new version and check if the problem persists.

If the bug still occurs and your installation is fully up to date, please provide a short LATEX file that demonstrates the problem. Such a file is known as a minimal working example (MWE). You should include only minimal text and use only the packages and definitions essential to demonstrate the problem. Avoid using any unusual packages as much as possible.

By preparing such an example it often becomes clear whether the problem is truly a KOMA-Script bug or caused by something else. To check if another package or class is actually causing the problem, you can also test your example with the corresponding standard class instead of a KOMA-Script class. If your problem still occurs, you should address your error report to the author of the appropriate package than to the author of KOMA-Script. Finally, you

should carefully review the instructions for the appropriate package, classes, and KOMA-Script component. A solution to your problem may already exist, in which case an error report is unnecessary.

If you think you have found a previously unreported error, or if for some other reason you need to contact the author of KOMA-Script, don't forget the following:

- Does the problem also occur if a standard class is used instead of a KOMA-Script class? In this case, the error is most likely not with KOMA-Script, and it makes more sense to ask your question in a public forum, a mailing list, or Usenet.
- Which KOMA-Script version do you use? For related information, see the `log` file of the LaTeX run of any document that uses a KOMA-Script class.
- Which operating system and which TeX distribution do you use? This information might seem rather superfluous for a system-independent package like KOMA-Script or LaTeX, but time and again they have certainly been shown to play a role.
- What exactly is the problem or the error? Describe the problem. It's better to be too detailed than too short. Often it makes sense to explain the background.
- What does a minimal working example look like? You can easily create one by commenting out content and packages from the document step by step. The result is a document that only contains the packages and parts necessary to reproduce the problem. In addition, all loaded images should be replaced by `\rule` statements of the appropriate size. Before sending your MWE,remove the commented-out parts, insert the command `\listfiles` in the preamble, and perform another LaTeX run. At the end of the `log` file, you will see an overview of the packages used. Add the MWE and the log file to the end of your description of the problem.

Do not send packages, PDF, PS, or DVI files. If the entire issue or bug description, including the minimal example and the `log` file is larger than a few tens of kilobytes, you're likely doing something wrong.

If you've followed all these steps, please send your KOMA-Script (only) bug report to komascript@gmx.info.

If you want to ask your question in a Usenet group, mailing list, or Internet forum, you should follow the procedures mentioned above and include a minimal working example as part of your question, but usually you don't need to provide the `log`-file. Instead, just add the list of packages and package versions from the `log`-file and, if your MWE compiles with errors, you should quote those messages from the `log` file.

Please note that default settings which are not typographically optimal do not represent errors. For reasons of compatibility, defaults are preserved whenever possible in new versions of KOMA-Script. Furthermore, typographical best practices are partly a matter of language and culture, and so the default settings of KOMA-Script are necessarily a compromise.

1.8. Additional Information

Once you become familiar with KOMA-Script, you may want examples that show how to accomplish more difficult tasks. Such examples go beyond the basic instructional scope of this manual and so are not included. However, you will find more examples on the website of the KOMA-Script Documentation Project [KDP]. These examples are designed for advanced LaTeX users and are not particularly suitable for beginners. The main language of the site is German, but English is also welcome.

Part I.

KOMA-Script for Authors

This part provides information for writers of articles, reports, books, and letters. The average user is probably less interested in how things are implemented in KOMA-Script and what pitfalls exist. Also, normal users aren't interested in obsolete options and instructions. They want to know how to achieve things using current options and instructions, and perhaps in some background information about typography.

The few passages in this part which contain extra information and explanations that may be of less interest for the impatient reader are set in a sans-serif typeface and can be skipped if desired. For those who are interested in more information about the implementation, side-effects with other packages, or obsolete options and instructions, please refer to part II beginning on page 331. That part of the KOMA-Script guide also describes all the features that were created specially for authors of packages and classes.

Calculating the Page Layout with typearea

Many LaTeX classes, including the standard classes, present the user with a largely fixed configuration of margins and page layout. In the standard classes, the choice is limited to selecting a font size. There are separate packages, such as geometry (see [Ume10]), which give the user complete control over, but also full responsibility for, setting the type area and margins.

KOMA-Script takes a somewhat different approach with the typearea package. Users are offered ways to adjust the design and algorithms based on established typographic standards, making it easier for them to make good choices.

2.1. Fundamentals of Page Layout

At first glance, a single page of a book or other printed material consists of the margins, a header, a body of text, and a footer. More precisely, there is also a space between the header area and the text body, as well as between the body and the footer. The text body is called, in the jargon of typographers and typesetters, the *type area*. The division of these areas, as well as their relations to each other and to the paper, is called the *page layout*.

Various algorithms and heuristic methods for constructing an appropriate type area have been discussed in the literature [Koh02]. These rules are known as the "canons of page construction." One approach often mentioned involves diagonals and their intersections. The result is that the aspect ratio of the type area corresponds to the proportions of the *page*. In a one-sided document, the left and right margins should have equal widths, while the ratio of the top and bottom margins should be 1:2. In a two-sided document (e. g. a book), however, the entire inner margin (the margin at the spine) should be the same size 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 emphasised *the page*. It is often mistakenly thought that the format of the page is the same as the format of the paper. However, if you look at a bound document, you can see that part of the paper disappears in the binding and is no longer part of the visible page. For the type area, however, it is not the format of the paper which is important; it is the impression of the visible page to the reader. Thus, it is clear that the calculation of the type area 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* but not the visible inner margin.

The binding correction depends on the production process and cannot be defined in general terms. It is therefore a parameter that must be redefined for each project. In professional printing, this value plays only a minor role, since printing is done on larger sheets of paper and then cropped to the right size. The cropping is done so that the above relations for the visible, two-sided page are maintained.

So now we know how the individual parts of a page relate to each other. However, we do not yet know how wide and high the type area is. Once we know one of these two dimensions, we can calculate all the other dimensions from the paper format and the page format or the binding correction.

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

The values *left margin* and *right margin* only exist in a one-sided document while *half inner margin* and *outer margin* only exist in a two-sided document. We use *half inner margin* in these equations, since the full inner margin is an element of the whole two-page spread. Thus, only half of the inner margin, *half inner margin*, belongs to a single page.

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

- the size, width, and type of font used,
- the line spacing,
- the word length,
- the available space.

The importance of the font becomes clear once you realize what serifs are for. Serifs are small strokes that finish off the lines of letters. Letters with vertical lines touching the text baseline disturb the flow rather than keeping the eye on the line. It is precisely with these letters that the serifs lie horizontally on the baseline and thus enhance the horizontal effect of the font. The eye can better follow the line of text, not only when reading the words but also when jumping back to the beginning of the next line. Thus, the line length can actually be slightly longer for a serif font than for a sans serif font.

Leading refers to the vertical distance between individual lines of text. In LaTeX, the leading is set at about 20% of the font size. With commands like `\linespread`, or better, packages like setspace (see [TF11]), you can change the leading. A wider leading makes it easy for the eye to follow the line. A very wide leading, however, disturbs reading because the eye has to travel long

distances between the lines. In addition, the reader becomes uncomfortable because of the visible striped effect. The uniform grey value of the page is thereby spoiled. Nevertheless, the lines can be longer with a wider leading.

The literature gives different values for good line lengths, depending on the author. To some extent, this is related to the author's native language. Since the eye usually jumps from word to word, short words make this task easier. Across 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 many hours a day reading. But even then, line lengths beyond 80 characters are unacceptable. In each case, the leading must be appropriately chosen. An extra 5% to 10% is recommended as a good rule of thumb. For typefaces like Palatino, which require more than 5% leading for normal line lengths, even more can be required.

Before looking at the actual construction of the page layout, there are a few minor points you should know. LaTeX does not start the first line in the text area of a page at the upper edge of the text area but sets the baseline at a defined distance from the top of the text area. Also, LaTeX recognizes 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 — even more when the end of the page coincides with headings, figures, tables, or the like. In two-sided documents that is usually undesirable. The second command, `\flushbottom`, makes sure that the last line is always at the lower edge of the text area. To achieve this vertical compensation, LaTeX may have to stretch vertical glue beyond what is normally allowed. Paragraph skip is such a stretchable, vertical glue, even when set to zero. To avoid stretching on normal pages where paragraph spacing is the only stretchable glue, the height of the text area should be a multiple of the height of the text line, including the distance of the first line from the top of the text area.

This concludes the fundamentals. In the following two sections, the methods of construction offered by KOMA-Script are presented in detail.

2.2. Constructing the Type Area by Division

The easiest way to make sure that the text area has the same ratio as the page is as follows:

- First, subtract the *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 of equal width.
- Then, take the uppermost row as the upper margin and the two lowermost rows as the lower margin. If you are printing two-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.

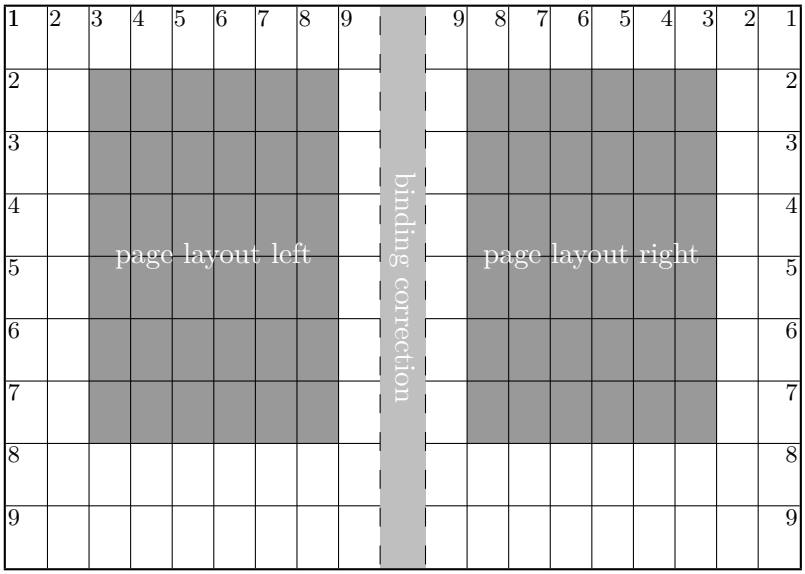


Figure 2.1.: Two-sided layout with the box construction of the classical nine-part division, after subtracting a binding correction

What remains within the page is the text area. The width and 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 greater than three. In order that the text area occupy at least twice as much space as the margins, *DIV* should really be at least nine. With this value, the design is also known as the *classical nine-part division* (see figure 2.1).

In KOMA-Script, this kind of design is implemented with the typearea package, where the bottom margin may drop any fractions of a line in order to comply with the constraint for the height of the type area mentioned in the previous paragraph and thereby reduce the problem mentioned with `\flushbottom`. For A4 paper, *DIV* is predefined according to the font size (see table 2.2, page 37). If there is no binding correction ($BCOR = 0\,\mathrm{pt}$), the results roughly match the values of table 2.1, page 36.

In addition to the predefined values, you can specify *BCOR* and *DIV* as options when loading the package (see section 2.4, starting on page 34). There is also a command to calculate the type area explicitly by providing these values as parameters (see also section 2.4, page 40).

The typearea package can automatically determine the optimal value of *DIV* for the font and leading used. Again, see section 2.4, page 37.

2.3. Constructing the Type Area by Describing a Circle

In addition to the construction method for the type area described above, there is an even more traditional, or even medieval, method found in the literature. The aim of this method is not just to have the same ratios between page size and type area; it is considered optimal when the height of the text area corresponds to the width of the page. This means that a circle can be drawn

that will touch both the sides of the page and the top and bottom of the text area. The exact procedure can be found in [Tsc87].

A disadvantage of this late-medieval canon of page construction is that the width of the text area no longer depends on the font. One no longer chooses the text area to match the font. Instead, the author or typesetter must choose the appropriate font for the text area. This should be considered mandatory.

In the typearea package, this construction is modified to determine the *DIV* value by selecting a special (normally meaningless) *DIV* value or a special, symbolic indication of the *DIV* value so that the resulting type area comes as close as possible to the late-medieval page canon. Hence it relies in turn on the method of constructing the type area by division.

2.4. Early or Late Selection of Options

This section introduces a special feature of KOMA-Script which, in addition to typearea, is also relevant to other KOMA-Script packages and classes. This section appears in nearly identical form in several chapters, so you can find all the information about a single package or class in the relevant chapter. Users who are interested not just in a particular package or class but in getting an overview of KOMA-Script as a whole only need to read this section in one of the chapters and can then skip it as they study the guide.

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

LaTeX allows users to pass class options as a comma-separated list of keywords in the optional argument to `\documentclass`. In addition to being passed to the class, these options are also passed on to all packages that can understand them. Users can also pass a similar comma-separated list of keywords in the optional argument of `\usepackage`. KOMA-Script extends the option mechanism for the KOMA-Script classes and some packages with further options. Thus most KOMA-Script options can also take a value, so an option does not necessarily take the form *option*, but can also take the form *option=value*. Except for this difference, `\documentclass` and `\usepackage` in KOMA-Script function as described in [Tea05b] or any introduction to LaTeX, for example [OPHS11].

v3.00

When using a KOMA-Script class, you should not specify options when loading the typearea or scrbase packages. The reason for this restriction is that the class already loads these packages without options, and LaTeX refuses to load a package multiple times with different option settings.

Setting the options with `\documentclass` has one major disadvantage: unlike the interface described below, the options in `\documentclass` are not robust. So commands, lengths, counters, and similar constructs may break inside the optional argument of this command. For example, with many non-KOMA-Script classes, using a LaTeX length in the value of an option results in an error before the value is passed to a KOMA-Script package and it can take

control of the option execution. So if you want to use a LaTeX length, counter, or command as part of the value of an option, you should use \KOMAoptions or \KOMAoption. These commands will be described next.

\KOMAoptions{*option list*}
\KOMAoption{*option*}{*value list*}

v3.00 KOMA-Script also provides the ability to change the values of most class and package options even after loading the class or package. You can use the \KOMAoptions command to change the values of a list of options, as in \documentclass or \usepackage. Each option in the *option list* has the form *option*=*value*.

Some options also have a default value. If you do not specify a value, that is if you give the option simply as *option*, then this default value will be used.

Some options can have several values simultaneously. For such options, it is possible, with the help of \KOMAoption, to pass a list of values to a single *option*. The individual values are given as a comma-separated *value list*.

KOMA-Script uses the commands \FamilyOptions and \FamilyOption with the family "KOMA" to implement this ability. Advanced users will find more on these instructions in section 12.2, page 336.

Options set with \KOMAoptions or \KOMAoption will reach both the KOMA-Script class and any previously loaded KOMA-Script packages that recognise these options. If an option or a value is unknown, scrbase will report it as an error.

2.5. Compatibility with Earlier Versions of KOMA-Script

Those who produce their documents from source code typically attach the utmost importance to the fact that future LaTeX runs will yield exactly the same result. In some cases, however, improvements and bug fixes to the package will result in changes of behaviour, especially to the layout. This, however, may be undesirable.

version=*value*
version=first
version=last

v3.01b Since Version 3.01b, typearea has been able to choose whether the source file should, as much as possible, continue to produce exactly the same result within a LaTeX run or should be formatted according to the modifications of the latest version. You can specify the version with which you want your file to be compatible by using the version option. Compatibility with the oldest supported KOMA-Script version can be achieved with version=first or version=2.9 or version=2.9t. Setting *value* to an unknown release number will result in a warning message and selects version=first for safety.

v3.01a

With `version=last`, you can select the latest version. In this case, you give up backwards compatibility. If the option is used without a value, `last` is assumed. This also corresponds to the default setting, as long as you do not use any deprecated options.

If you use a deprecated option of KOMA-Script 2, KOMA-Script 3 will switch to `version=first` automatically. This will also result in a warning message that explains how to prevent this switch. Alternatively, you can choose a different setting for `version` with the desired compatibility after the deprecated option.

Compatibility is primarily a question of line and page breaks (wrapping). If you choose compatibility with an older version, new options that do not affect wrapping are still available. The `version` option does not affect any wrapping changes that are the result of fixing unambiguous errors. If you need unconditional wrapping compatibility even in the case of bugs, you should physically save the old KOMA-Script version you need together with your document.

Note that you cannot change the `version` option after loading the typearea package. Setting this option with `\KOMAoptions` or `\KOMAoption` will therefore cause an error.

2.6. Adjusting the Type Area and Page Layout

The typearea package offers two different user interfaces to influence the construction of the type area. The most important method is to specify options when loading the package. For information on how to setup options with KOMA-Script, please refer to section 2.4.

In this section the classes used in the examples are not existing KOMA-Script classes but hypothetical ones. This guide assumes that ideally an appropriate class is available for each task.

`BCOR=`*`correction`*

v3.00

Use the `BCOR=`*`correction`* option to specify the absolute value of the binding correction, i. e. the width of the area lost from the paper during the binding process. This value is then automatically taken into account when constructing the page layout and is added back to the inner (or left) margin during output. In the value of the *`correction`*, you can specify any measurement unit understood by TEX.

Example: Suppose you create a financial report. The whole thing should be printed out one-sided on A4 paper and then stapled in a binder folder. The clip of the folder covers 7.5 mm. The stack of pages is very thin, so at most another 0.75 mm will be lost from bending and the sheets themselves. Therefore, you can write:

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

with `BCOR=8.25mm` as an option to typearea or

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

when using BCOR=8.25mm as a global option.

When using a KOMA-Script class, you do not need to load the typearea package explicitly:

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

You can omit the a4paper option with scrreprt, since this is the default for all KOMA-Script classes.

If you want to set the option to a new value later, you can, for example, use the following:

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

Defaults are initialized when the scrreprt class is loaded. Changing a setting with the \KOMAoptions or \KOMAoption commands will automatically calculate a new type area with new margins.

Note you must pass this option as a class option when loading one of the KOMA-Script classes, as in the example above, or via \KOMAoptions or \KOMAoption after loading the class. When you use a KOMA-Script class, you should not load the typearea package explicitly with \usepackage, nor should you specify it as an optional argument when loading the package if you are using another class. If the option is changed with \KOMAoptions or \KOMAoption after loading the package, the type area and margins are automatically recalculated.

DIV=*factor*

v3.00

The DIV=*factor* option specifies the number of strips into which the page is divided horizontally and vertically during the construction of the type area. The exact construction method is found in section 2.2. It's important to realise 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*. Note, however, that large values can cause violations in the constraints on the margins of the type area, depending on how you set other options. In extreme cases, the header may fall outside of the page. When you use the DIV=*factor* option, you are responsible for complying with the margin constraints and for choosing a typographically pleasing line length.

In table 2.1, you will find the sizes of the type areas for several DIV factors for the A4 page with no binding correction. In this case, the other constraints that are dependent on the font size are not taken into account.

Example: Suppose you are writing up the minutes of a meeting using the minutes class. The whole thing should be two-sided. Your company uses 12 pt Bookman font. This

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

	Type area		Margins	
DIV	width	height	top	inner
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

(all lengths in mm)

font, which is one of the standard PostScript fonts, is enabled in LaTeX with the command \usepackage{bookman}. Bookman is a very wide font, meaning that the individual characters are relatively wide compared to their height. Therefore, the default setting for DIV in typearea is too small. After thoroughly studying this entire chapter, you conclude that a value of 15, instead of 12, is most suitable. The minutes will not be bound but punched and kept in a folder, and thus no binding correction is necessary. So you write:

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

When you're done, you become aware that the minutes will from now on be collected and bound together as a book at the end of the quarter. The binding is to be a simple glue binding because this is only being done to conform to ISO 9000 and nobody is actually going to read them. The binding, including space lost in folding the pages, requires an average of 12 mm You change the options of the typearea package accordingly and use the class for minutes that conform to ISO 9000 regulations:

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

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

```
\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.: *DIV* defaults for A4

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

in all KOMA-Script classes.

Note that when using this option with one of the KOMA-Script classes, as in the example above, it must be passed either as a class option, or via \KOMAoptions or \KOMAoption after loading the class. When using a KOMA-Script class, the typearea package should not be loaded explicitly with \usepackage, 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 type area and margins are automatically recalculated.

DIV=calc
DIV=classic

v3.00

As already mentioned in section 2.2, there are fixed defaults for DIV when using A4 paper. These can be found in table 2.2. However, such fixed values have the disadvantage that they do not take into account the letter spacing of the font used. With A4 and fairly narrow fonts, this can quickly lead to an unpleasantly high number of characters per line. See the considerations in section 2.1. If you choose a different paper size, typearea will calculate an appropriate DIV value for you. Of course, you can also apply this same calculation to A4. To do so, simply use DIV=calc in place of DIV=*factor*. Of course, you can also specify this option explicitly for all other paper sizes. If you want automatic calculation, this specification is useful, as it is possible to set different preferences in a configuration file (see section 20.3). Explicitly specifying the DIV=calc option overrides such configuration settings.

You can also select the traditional page layout mentioned in section 2.3, the medieval page canon. Instead of the DIV=*factor* or DIV=calc option, simply use the DIV=classic option. A DIV value which is as close as possible to the medieval page canon is then chosen.

Example: In the example using the Bookman font and the DIV=*factor* option, the problem was to select a DIV value that better matched the font. Modifying that example, you can simply leave the calculation of this value to typearea:

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

Note that when using this option with one of the KOMA-Script classes, as in the example above, it must be passed either as a class option, or via \KOMAoptions or \KOMAoption after loading the class. When using a KOMA-Script class, the typearea package should not be loaded explicitly with \usepackage, nor should the option be given as an optional argument. If the

option is changed via \KOMAoptions or \KOMAoption after loading the package, the type area and margins are automatically recalculated.

```
DIV=current
DIV=last
```

v3.00 If you've been following the examples closely, you already know how to calculate a DIV value based on the font you chose when using a KOMA-Script class together with a font package.

The difficulty with doing so 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 specify 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 type area and margins would be calculated for the standard LaTeX font and not for the font loaded later.

However, it is possible to recalculate the type area and margins after loading the font with the aid of \KOMAoptions{DIV=calc} or \KOMAoption{DIV}{calc}. The option DIV=calc will then request a DIV value for an appropriate line length.

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

v3.00 The option DIV=*current* recalculates the type area and margins using the current DIV value. This is less important for recalculating the type area after loading a different font. Instead, it is useful if, for example, you change the leading while keeping the DIV value the same and want to ensure the margin constraint that \textheight minus \topskip is a multiple of \baselineskip.

v3.00 The option DIV=*last* will recalculate the type area and margins using exactly the same settings as the last calculation.

Example: Let's suppose again that we need to calculate an appropriate line length for a type area using the Bookman font. At the same time, a KOMA-Script class is used. This is very easy with the symbolic value last and the command \KOMAoptions:

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

If you decide later that you need a different DIV value, just change the setting of the optional argument to \documentclass.

For a summary of all possible symbolic values for the DIV option, see table 2.3. Note that the use of the fontenc package may also cause LaTeX to load a different font.

Frequently, the type area must be recalculated in combination with a change in the line spacing (*leading*). Since the type area should be calculated in such a way that a whole number

Table 2.3.: Available 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 medieval 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.

of lines fits in the text block, a change in the leading normally requires a recalculation of the type area.

Example: Suppose that you require a 10 pt font and a spacing of 1.5 lines for a dissertation. By default, LaTeX sets the leading for 10 pt fonts at 2 pt, in other words 1.2 lines. Therefore, you must use an additional stretch factor of 1.25. Suppose also that you need a binding correction of 12 mm. Then the solution to the problem might look like this:

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

Since typearea always executes the \normalsize command itself when calculating a new type area, it is not strictly necessary to set the chosen leading with \selectfont after \linespread, since this will already be done in the recalculation.

When using the setspace package (see [TF11]), the same example would appear as follows:

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

As you can see from the example, the setspace package saves you from needing to know the

correct stretch value. However, this only applies to the standard font sizes 10 pt, 11 pt, and 12 pt. For all other font sizes, the package uses an approximate value.

At this point, note that the line spacing for the title page should be reset to the normal value, and the indexes should be set with the normal line spacing as well.

Example: Here is a complete example:

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

Also see the notes in section 2.8. The \AfterTOCHead command is described in chapter 15 of part II on page 381.

Note also that changing the line spacing can also affect the page's header and footer. For example, if you are using the scrlayer-scrpage package, you have to decide for yourself whether you prefer to have the normal or the changed leading. See the singlespacing option in chapter 17, page 435.

Note that when using this option with one of the KOMA-Script classes, as in the example above, it must be passed either as a class option, or via \KOMAoptions or \KOMAoption after loading the class. When using a KOMA-Script class, the typearea package should not be loaded explicitly with \usepackage, 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 type area and margins are automatically recalculated.

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

If the DIV option or the BCOR option is set after loading the typearea package, the \typearea command will be called internally. When setting the DIV option, the symbolic value current is used internally for *BCOR*, which for reasons of completeness is also found in table 2.4. When setting the BCOR option, the symbolic value last is used internally for *DIV*. If instead you want

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

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

the type area and margins to be recalculated using the symbolic value current for *DIV*, you can use \typearea[current]{current} directly.

If you change both *BCOR* and *DIV*, you should use \typearea, since then the type area and margins are recalculated only once. With \KOMAoptions{DIV=*factor*,BCOR=*correction*} the type area 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 in the middle of a document. However, several assumptions about the structure of the LaTeX kernel are made, and internal definitions and sizes of the kernel are changed. Since changes are only made to the LaTeX kernel to fix bugs, there is a high likelihood, though no guarantee, that this will still work in future versions of LaTeX 2ε. When used within the document, a page break will result.

Since \KOMAoption{DIV}{last}, \KOMAoptions{DIV=last}, or \typearea[current]{last} is frequently needed to recalculate the type area and margins, there is a convenience command, \recalctypearea.

v3.00

Example: If you find the notation

```
\KOMAoptions{DIV=last}
```

or

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

too cumbersome for recalculating text area and margins because of the many special characters, you can simply use

```
\recalctypearea
```

twoside=*simple switch*
twoside=semi

As explained in section 2.1, the distribution of the margins depends on whether the document is to be printed one-sided or two-sided. For one-sided printing, the left and right margins are the same width, whereas for two-sided printing the inner margin of one page is only half as wide as the corresponding outer margin. To invoke two-sided printing, you must give the typearea package the twoside option. For the *simple switch*, you can use any of the standard values for simple switches in table 2.5. If the option is passed without a value, the value true is assumed, so two-sided printing is enabled. Deactivating the option leads to one-sided printing.

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

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

v3.00 v3.12

In addition to the values in table 2.5, you can also use the value `semi`. This value results in two-sided printing with one-sided margins and one-sided, that is non-alternating, marginal notes. Beginning with KOMA-Script version 3.12, binding corrections (see BCOR, page 34) will be part of the left margin on odd pages but part of the right margin on even pages. But if you switch on compatibility with a prior version of KOMA-Script (see section 2.5, page 33), the binding correction will be part of the left margin on both pages while using `twoside=semi`.

The option can also be passed as class option in `\documentclass`, as a package option with `\usepackage`, or even after loading typearea with `\KOMAoptions` or `\KOMAoption`. Using this option after loading typearea automatically results in the recalculation of the type area using `\recalctypearea` (see page 40). If the two-sided mode was active before the option was set, a page break is made to the next odd page before the recalculation.

`twocolumn=`*`simple switch`*

To compute an appropriate type area with the help of DIV=calc, it is useful to know in advance if the document is to be typeset in one or two columns. Since the considerations about line length in section 2.1 apply to each column, the type area in two-column documents can be up to twice as wide as in one-column documents.

To make this distinction, you must tell typearea if the document is to be set with two columns using the `twocolumn` option. Since this is a *`simple switch`*, any of the standard values for simple switches from table 2.5 are valid. If the option is passed without a value, the value `true` is used, i. e. the two-column setting. Deactivating the option returns you to the default one-column setting.

The option can also be passed as a class option in `\documentclass`, as a package option to `\usepackage`, or even after loading typearea with `\KOMAoptions` or `\KOMAoption`. Using this option after loading typearea will automatically recalculate the type area using `\recalctypearea` (see page 40).

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 body of the text. But one important question has not been answered: What exactly are *the margins*?

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

For the top and bottom margins, the question becomes how to handle headers and footers. Do these two belong to the text body or to their respective margins? This question is not easy to answer. Clearly an empty footer or header belongs to the margins, since it cannot be distinguished from the rest of the margins. A footer that contains only the pagination looks more like a margin and should therefore be counted as such. It is irrelevant for the visual effect whether headers or footers are easily recognized as such when reading or skimming. The decisive factor is how a well-filled page appears when viewed *out of focus*. For this purpose, you could, for example, steal the glasses of a far-sighted grandparent and hold the page about half a meter from the tip of your nose. If you lack an available grandparent, you can also adjust your vision to infinity and look at the page with one eye only. Those who wear glasses have a clear advantage here. If the footer contains not only the pagination but also other material like a copyright notice, it looks more like a slightly detached part of the body of the text. This needs to be taken into account when calculating the type area.

For the header, this is even more complicated. The header often contains running heads. If you use the current chapter and section titles in your running head and these titles are long, the header itself will necessarily be very long. In this case, the header again acts like a detached part of the text body and less like an empty margin. This effect is reinforced if the header contains not only the chapter or section title but also the pagination. With material on the right and left side, the header no longer appears as an empty margin. It is more difficult if the pagination is in the footer and the length of the running titles varies, so that the header may look like part of the margin on one page and part of the text body on another. Under no circumstances should you treat the pages differently. That would lead to vertically jumping headers, which is not suitable even for a flip book. In this case it is probably best to count the header as part of the text body.

The decision is easy when the header or footer is separated from the actual text body by a line. This will give a "closed" appearance and the header or footer should be calculated as 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 determine on its own whether to count headers and footers as part of the text body or the margin. The `headinclude` and `footinclude` options cause the header or footer to be counted as part of the text. These options, being *simple switch*es, accept the standard values for simple switches in table 2.5. You can use the options without

v3.00

specifying a value, in which case the value `true` is used for the *simple*, i. e. the header or footer is counted as part of the text.

If you are unsure what the correct setting should be, reread the explanations above. The default is usually `headinclude=false` and `footinclude=false`, but this can change in the KOMA-Script classes or in other KOMA-Script packages depending on the options used (see section 3.1 and chapter 5).

Note that these options must be passed as class options when using one of the KOMA-Script classes, or after loading the class with `\KOMAoptions` or `\KOMAoption`. Changing these options after loading the typearea package does not automatically recalculate the type area. Instead, the changes only take effect the next time the type area is recalculated. For recalculation of the type area, see the `DIV` option with the values `last` or `current` (see page 38) or the `\recalctypearea` command (see page 40).

`mpinclude=`*simple switch*

v2.8q

In addition to documents where the header and footer are more likely to be part of the text body than the margins, there are also documents where marginal notes should be considered part of the text body as well. The option `mpinclude` does exactly this. The option, as a

v3.00

simple switch, accepts the standard values for simple switches in table 2.5. You can also pass this option without specifying a value, in which case `true` is assumed.

The effect of `mpinclude=true` is that a width-unit is removed from the main text body and used as the area for marginal notes. With the `mpinclude=false` option, which is the default setting, part of the normal margin is used for marginal notes. The width of that area is one or one-and-a-half width units, depending on whether you have chosen one-sided or two-sided printing. The `mpinclude=true` option is mainly for experts and so is not recommended.

In most cases where the option `mpinclude` makes sense, you also require a wider area for marginal notes. Often, however, only a part of the marginal note's width should be part of the text area, not the whole width, for example if the margin is used for quotations. Such quotations are usually set as unjustified text, with the flush edge against the text area. Since the unjustified text gives no homogeneous optical impression, these lines can protrude partially into the margin. You can accomplish that by using the option `mpinclude` and by increasing the 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 increased depends on the individual situation and it requires a certain amount of sensitivity. This is another reason the `mpinclude` option is primarily intended for experts. Of course you can specify, for example, that the marginal notes should project a third of the way into the normal margin by using the following:

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

Currently there is no option to enlarge the space for marginal notes within the text area. There is only one way to accomplish this: first, either omit the `mpinclude` option or set it to `false`, and then, after the type area has been calculated, reduce `\textwidth` (the width of the text

body) and increase \marginparwidth (the width of the marginal notes) by the same amount. Unfortunately, this procedure cannot be combined with automatic calculation of the *DIV* value. In contrast, mpinclude is taken into account with DIV=calc (see page 37).

Note that these options must be passed as class options when using one of the KOMA-Script classes, or after loading the class with \KOMAoptions or \KOMAoption. Changing these options after loading the typearea package does not automatically recalculate the type area. Instead, the changes only take effect the next time the type area is recalculated. For recalculation of the type area, see the DIV option with the values last or current (see page 38) or the \recalctypearea command (see page 40).

headlines=*number of lines*
headheight=*height*

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

v3.00

The **headlines** option specifies the number of lines of text in the header. The typearea package uses a default of 1.25. This is a compromise: large enough for underlined headers (see section 3.12) and small enough that the relative weight of the top margin is not affected too much when the header is not underlined. Thus the default value will usually be adequate. In special cases, however, you may need to adjust the header height more precisely to your actual requirements.

Example: Suppose you want to create a two-line header. Normally this would result in LaTeX issuing the warning "overfull \vbox" for each page. To prevent this from happening, you tell the typearea package to calculate an appropriate type area:

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

If you use a KOMA-Script class, you should pass this option directly to the class:

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

Commands that can be used to define the contents of a two-line header can be found 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. This is accomplished with the alternative option **headheight**. All lengths and sizes that LaTeX understands are valid for *height*. Note, however, that if you use a LaTeX length such as \baselineskip, its value is not fixed at the time the option is set. The value that will be used will be the one current at the time the type area and margins are calculated. Also, LaTeX lengths like \baselineskip should never be used in the optional argument of \documentclass or \usepackage.

Please be sure to note that these options must be passed as class options when using one of the KOMA-Script classes, or after loading the class with \KOMAoptions or \KOMAoption. Changing these options after loading the typearea package does not automatically recalculate the type area. Instead, the changes only take effect the next time the type area is recalculated. For recalculation of the type area, see the DIV option with the values last or current (see page 38) or the \recalctypearea command (see page 40).

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

v3.12 Like the header, the footer also requires an indication of how high it should be. But unlike the height of the header, the LaTeX kernel does not provide a length for the height of the footer. So typearea defines a new length, \footheight, if it does not already exist. Whether this length will be used by classes or packages to design the headers and footers depends on the individual classes and packages. The KOMA-Script package scrlayer-scrpage incorporates \footheight and actively cooperates with typearea. The KOMA-Script classes, on the other hand, do not recognize \footheight because without the help of packages they offer only page styles with single-line page footers.

You can use footlines to set the number of lines in the footer, similar to headlines for the number of lines in the header. By default the typearea package uses 1.25 footer lines. This value is a compromise: large enough to accommodate an overlined or underlined footer (see section 3.12), and small enough that the relative weight of the bottom margin is not affected too much when the footer lacks a dividing line. Thus the default value will usually be adequate. In special cases, however, you may need to adjust the footer height more precisely to your actual requirements.

Example: Suppose you need to place a two-line copyright notice in the footer. Although there is no test in LaTeX itself to check the space available for the footer, exceeding the designated height will likely result in unbalanced distribution of type area and margins. Moreover, a package such as scrlayer-scrpage, which can be used to define such a footer, performs the appropriate test and will report any overruns. So it makes sense to specify the required footer height when calculating of the type area:

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

Again, if you use a KOMA-Script class, you should pass this option directly to the class:

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

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

In some cases it is useful to be able to specify the footer height not in lines but directly as a length. This is accomplished with the alternative option `footheight`. All lengths and sizes that LaTeX understands are valid for *height*. Note, however, that if you use a LaTeX length such as `\baselineskip`, its value is not fixed at the time the option is set. The value that will be used will be the one current at the time the type area and margins are calculated. Also, LaTeX lengths like `\baselineskip` should never be used in the optional argument of `\documentclass` or `\usepackage`.

Please be sure to note that these options must be passed as class options when using one of the KOMA-Script classes, or after loading the class with `\KOMAoptions` or `\KOMAoption`. Changing these options after loading typearea does not automatically recalculate the type area. Instead, the changes only take effect the next time the type area is recalculated. For recalculation of the type area, see the `DIV` option with the values `last` or `current` (see page 38) or the `\recalctypearea` command (see page 40).

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

So far, we have seen how to create a nice type area for standard situations and how the typearea package makes it easier to accomplish this while still giving the freedom to adapt the layout. However, there are cases where the text body has to adhere precisely to specific dimensions. At the same time, the margins should be distributed as nicely as possible and, if necessary, a binding correction should be taken into account. The typearea package offers the command `\areaset` for this purpose. This command takes as parameters the width and height of the text body, as well as the binding correction as an optional parameter. The width and position of the margins are then calculated automatically, taking account of the options `headinclude`, `headinclude=false`, `footinclude` and `footinclude=false` where needed. On the other hand, the options `headlines`, `headheight`, `footlines`, and `footheight` are ignored! For more information, see `\areaset` on page 470 of section 20.1.

The default for *BCOR* is 0 pt. If you want to preserve the current binding correction, for example the value set by option `BCOR`, you can use the symbolic value `current` at an optional argument.

Example: Suppose a text on A4 paper needs a width of exactly 60 characters in a typewriter font and a height of exactly 30 lines per page. You can accomplish this with the following preamble:

```
\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}
```

The factor is 29 rather than 30 because the baseline of the topmost line of text is `\topskip` below the top margin of the type area, as long as the height of the topmost line is less than `\topskip`. So we don't need to add any height for the first line. The descenders of characters on the lowermost line, on the other hand, protrude below the dimensions of the type area.

To set a book of poetry with a square text area with a side length of 15 cm and a binding correction of 1 cm, the following is possible:

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

`DIV=areaset`

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

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

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

If you have concrete specifications for the margins, typearea is not suitable. In this case, you should use the geometry package (see [Ume10]).

2.7. Selecting the Paper Size

The paper size is a key feature of a document. As already mentioned in the description of the supported page layout constructions (see section 2.1 to section 2.3 starting on page 28), the layout of the page, and hence the entire document, depends on the paper size. Whereas the LaTeX standard classes are limited to a few formats, KOMA-Script supports even unusual paper sizes in conjunction with the typearea package.

`paper=`*`size`*
`paper=`*`orientation`*

v3.00 The `paper` option is the central element for paper-size selection in KOMA-Script. *`Size`* supports 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 lower case — `a4`.

Landscape orientations are supported by specifying the option one more time with the value v3.02c `landscape` or `seascape`. The only difference between `landscape` and `seascape` is that that the application `dvips` rotates `landscape` pages by -90 °, while it rotates `seascape` pages by

+90 °. Thus seascape is particularly useful whenever a PostScript viewer shows landscape pages upside-down. In order for the difference to have an effect, you must not deactivate the pagesize option described below.

v3.01b v3.22

Additionally, the *size* can also be specified either in the form *width*:*height* or in the form *height*:*width*. Which value is taken as the *height* and which as the *width* depends on the orientation of the paper. With paper=landscape or paper=seascape, the smaller value is the *height* and the larger one is the *width*. With paper=portrait, the smaller value is the *width* and the larger one is the *height*.

Note that until version 3.01a the first value was always the *height* and the second one the *width*. From version 3.01b through version 3.21, the first value was always the *width* and the second one the *height*. This is important if you use compatibility settings (see option version, section 2.5, page 33).

Example: Suppose you want to print an ISO-A8 index card in landscape orientation. The margins should be very small and 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:

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

By default, 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 these options must be passed as class options when using one of the KOMA-Script classes, or after loading the class with \KOMAoptions or \KOMAoption. Changing the

paper size or orientation with \KOMAoptions or \KOMAoption does not automatically recalculate the type area. Instead, the changes only take effect the next time the type area is recalculated. For recalculation of the type area, see the DIV option with the values last or current (see page 38) or the \recalctypearea command (see page 40).

pagesize=*output driver*

The above-mentioned mechanisms for choosing the paper format only affect the output insofar as internal LaTeX lengths are set. The typearea package then uses them in dividing the page into type area and margins. The specification of the DVI formats, however, does not include any indication of paper size. When outputting directly from the DVI format to a low-level printer language such as PCL[1] or ESC/P2[2] or ESC/P-R[3], this is usually not an issue, since with these formats the reference zero-position is at the top left, as in DVI. But nowadays, the output is normally translated into languages such as PostScript or PDF, in which the zero-position is at a different point, and in which the paper format should be specified in the output file, which is missing this information. To solve this problem, the corresponding driver uses a default paper size, which the user can change either by an option or by specifying it in the TeX source file. When using the DVI driver dvips or dvipdfm, the information can be given in the form of a \special command. When using pdfTeX, luaTeX, XeTeX or VTeX their paper-size lengths are set appropriately.

v3.17 With the pagesize option, you can 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. Using this option without providing a value is equivalent to pagesize=auto.

Example: Suppose a document should be available both as a DVI data file and in PDF format for on-line viewing. The preamble might begin as follows:

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

If the pdfTeX engine is used *and* PDF output is enabled, the lengths \pdfpagewidth and \pdfpageheight are set appropriately. If, however, a DVI data file is created — whether by LaTeX or by pdfLaTeX — then a \special is written at the start of this data file.

If you use an older version of typearea, you should always specify the pagesize option, because older versions of typearea did not set them by default. As a rule, the method without an *output driver* or with auto or automedia is convenient.

[1] PCL is a family of printer languages that HP uses for its inkjet and laser printers.
[2] ESC/P2 is the printer language that EPSON uses for its dot-matrix, and older inkjet or laser printers.
[3] ESC/P-R is the printer language that EPSON currently uses for inkjet and laser printers.

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

`auto` Uses output driver `pdftex` if the pdfTeX-specific lengths `\pdfpagewidth` and `\pdfpageheight` or the luaTeX-specific lengths `\pagewidth` and `\pageheight` are defined. In addition, the output driver `dvips` will also be used. This setting is in principle also suitable for XeTeX.
`automedia` Almost the same as `auto` but if the VTeX-specific lengths `\mediawidth` and `\mediaheight` are defined, they will be set as well.
`false`, `no`, `off` Does not set any output driver and does not send page size information to the output driver.
`dvipdfmx` v3.05a Writes the paper size into DVI files using `\special{pagesize=`*width*`,`*height*`}`. The name of the output driver is `dvipdfmx` because the application `dvipdfmx` handles such specials not just in the preamble but in the document body too.
`dvips` Using this option in the preamble sets the paper size using `\special {pagesize=`*width*`,`*height*`}`. Since the `dvips` driver cannot handle changes of paper size in the inner document pages, a hack is required to achieve such changes. Use changes of paper size after `\begin{document}` at your own risk, if you are using `dvips`!
`pdftex`, `luatex` v3.20 Sets paper size using the pdfTeX-specific lengths `\pdfpagewidth` and `\pdfpageheight` or the luaTeX-specific lengths `\pagewidth` and `\pageheight`. You can do this at any time in your document.

2.8. Tips

For theses and dissertations, many rules exist that violate even the most elementary rules of typography. The reasons for such rules include the typographical incompetence of those who issue them, but also the fact that they were originally meant for mechanical typewriters. With a typewriter or a primitive text processor from the early 1980s, it was not possible to produce typographically correct output without extreme effort. So rules were created that appeared to be easy to follow and were still accommodating to a proofreader. These include margins that lead to usable line lengths for one-sided printing with a typewriter. To avoid extremely short lines, which are made worse by unjustified text, the margins were kept narrow and the leading was increased to 1.5 lines to allow space for corrections. Before the advent of

modern text processing systems, single spacing would have been the only alternative — except 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 showed their playful side and tried to spice up their work by using an ornamental font to make their work look better than it really was. They did not consider that such fonts are often more difficult to read and therefore unsuitable for this purpose. Thus, two font families found their way into the regulations which are neither compatible nor particularly suitable for the job in the case of Times. Times is a relatively narrow typeface designed at the beginning of the 20th century for the narrow columns of British newspapers. Modern versions usually are somewhat improved. But still the Times font, which is often required, does not really fit the prescribed margins.

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

Often these typographically questionable rules are difficult to implement in LATEX. A fixed number of characters per line can be achieved only when a non-proportional font is used. There are very few good non-proportional fonts available. Hardly any 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 widths. This does not work and results in a fragmented and agitated-looking text. If you use LATEX for your thesis, some of these rules have to be either ignored or at least interpreted generously. For example, "60 characters per line" can be interpreted not as a fixed but as an average or maximum value.

As implemented, typesetting rules are usually intended to obtain a useful result even if the author does not know what needs to be considered. *Useful* frequently means readable and correctable. In my opinion the type area of a text set with LATEX and the typearea package meets these criteria well from the outset. So if you are confronted with regulations which deviate substantially from it, I recommend that you present a sample of the text to your advisor and ask whether you can submit the work despite deviations in the format. If necessary the type area can be adapted somewhat by changing the DIV option. I advise against using \areaset for this purpose, however. In the worst case, use the geometry package (see [Ume10]), which is not part of KOMA-Script, or change the page layout parameters of LATEX yourself. You can find the values as determined by typearea in the log file of your document. The usegeometry option, which you can find in part II, can also improve the interactions between typearea and geometry. This should allow modest adjustments. However, make sure that the proportions of the text area match those of the page, taking the binding correction into account.

If it is absolutely necessary to set the text with a line spacing of 1.5, do 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, use the \linespread command. I recommend the package setspace (see [TF11]), which is not part of KOMA-Script. You should also let typearea recalculate a new type area after changing

the line spacing. However, you should switch back to the normal line spacing for the title, and preferably for the table of contents and various lists — as well as the bibliography and the index. For details, see the explanation of DIV=current.

The typearea package, even with option DIV=calc, calculates a very generous text area. Many conservative typographers will find that the resulting line length is still excessive. The calculated DIV value may be found in the log file for each document. So you can easily choose a smaller value after the first LaTeX run.

Not infrequently I am asked why I dwell on type area calculations for an entire chapter, when it would be much easier just to provide a package with which you can adjust the margins as in a word processor. Often it is said that such a package would be a better solution in any case, since everyone knows how to choose appropriate margins, and that the margins calculated by KOMA-Script are not that good anyway. I would like to quote Hans Peter Willberg and Friedrich Forssmann, two of the most respected contemporary typographers [WF00]. (You can find the original German in the German guide.)

> *The practice of doing things oneself has long been widespread, but the results are often dubious because amateur typographers do not see what is wrong and cannot know what is important. This is how you get used to to incorrect and poor typography.* [...] *Now the objection could be made that typography is a matter of taste. When it comes to decoration, one could perhaps accept that argument, but since typography is primarily about information, not only can mistakes irritate, but they 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 letter class. 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 in accordance with table 3.1. For example, you can replace `\documentclass{book}` with `\documentclass{scrbook}`. Normally, LaTeX should process the document without errors, just as before the substitution. The layout, however, should be different. Additionally, the KOMA-Script classes provide new possibilities and options that are described in the following sections.

Let me say something before describing the classes. When beginning to write a document, you are often unsure which specific options to choose. Some settings, for instance the choice of paper size, may be fixed in advance. But even the question of the appropriate page layout could be difficult to answer initially. On the other hand, these settings should be nearly irrelevant, in the beginning, to the main business of an author: planning the document structure, writing the text, preparing figures, tables, lists, index, and other data. As an author, you should concentrate initially on the content. When that is done, you can take on the fine points of presentation. In addition to the choice of options, this includes correcting hyphenation, optimizing page breaks, and placing tables and figures.

3.1. Early or Late Selection of Options

The information in section 2.4 applies equally to this chapter. So if you have already read and understood section 2.4, you can skip ahead to section 3.2, page 56.

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

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

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

LaTeX allows users to pass class options as a comma-separated list of keywords in the optional argument to \documentclass. In addition to being passed to the class, these options are also passed on to all packages that can understand them. Users can also pass a similar comma-separated list of keywords in the optional argument of \usepackage. KOMA-Script extends the option mechanism for the KOMA-Script classes and some packages with further options. Thus most KOMA-Script options can also take a value, so an option does not necessarily take the form *option*, but can also take the form *option=value*. Except for this difference, \documentclass and \usepackage in KOMA-Script function as described in [Tea05b] or any introduction to LaTeX, for example [OPHS11].

v3.00

When using a KOMA-Script class, you should not specify options when loading the typearea or scrbase packages. The reason for this restriction is that the class already loads these packages without options, and LaTeX refuses to load a package multiple times with different option settings. In general, it is not necessary to load either one of these packages explicitly when using any KOMA-Script class.

Setting the options with \documentclass has one major disadvantage: unlike the interface described below, the options in \documentclass are not robust. So commands, lengths, counters, and similar constructs may break inside the optional argument of this command. For example, with many non-KOMA-Script classes, using a LaTeX length in the value of an option results in an error. So if you want to use a LaTeX length, counter, or command as part of the value of an option, you should use \KOMAoptions or \KOMAoption. These commands will be described next.

\KOMAoptions{*option list*}
\KOMAoption{*option*}{*value list*}

v3.00

KOMA-Script also provides the ability to change the values of most class and package options even after loading the class or package. You can use the \KOMAoptions command to change the values of a list of options, as in \documentclass or \usepackage. Each option in the *option list* has the form *option=value*.

Some options also have a default value. If you do not specify a value, that is if you give the option simply as *option*, then this default value will be used.

Some options can have several values simultaneously. For such options, it is possible, with the help of \KOMAoption, to pass a list of values to a single *option*. The individual values are given as a comma-separated *value list*.

KOMA-Script uses the commands \FamilyOptions and \FamilyOption with the family "KOMA" to implement this ability. See part II, section 12.2, page 336.

Options set with \KOMAoptions or \KOMAoption will reach both the KOMA-Script class and any previously loaded KOMA-Script packages that recognise these options. If an option or a value is unknown, scrbase will report it as an error.

3.2. Compatibility with Earlier Versions of KOMA-Script

The information in section 2.5 applies equally to this chapter. So if you have already read and understood section 2.5 you can skip ahead to page 56, page 56.

Those who produce their documents from source code typically attach the utmost importance to the fact that future LaTeX runs will yield exactly the same result. In some cases, however, improvements and bug fixes to the class will result in changes of behaviour, especially to the layout. This, however, may be undesirable.

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

v2.96a Since Version 2.96a, KOMA-Script has been able to choose whether the source file should, as much as possible, continue to produce exactly the same result within a LaTeX run or should be formatted according to the modifications of the latest version of the class. You can specify the version with which you want your file to be compatible by using the `version` option. Compatibility with the oldest supported KOMA-Script version can be achieved with `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`, you can select the latest version. In this case, you give up backwards v3.01a compatibility. If the option is used without a value, `last` is assumed. This also corresponds to the default setting, as long as you do not use any deprecated options.

If you use a deprecated option of KOMA-Script 2, KOMA-Script 3 will switch to `version=first` automatically. This will also result in a warning message that explains how to prevent this switch. Alternatively, you can choose a different setting for `version` with the desired compatibility after the deprecated option.

Compatibility is primarily a question of line and page breaks (wrapping). If you choose compatibility with an older version, new options that do not affect wrapping are still available. The `version` option does not affect any wrapping changes that are the result of fixing unambiguous errors. If you need unconditional wrapping compatibility even in the case of bugs, you should physically save the old KOMA-Script version you need together with your document.

Note that you cannot change the `version` option after loading the class. Setting this option with `\KOMAoptions` or `\KOMAoption` will therefore cause an error.

3.3. Draft Mode

Many classes and packages provide a draft mode in addition to the normal typesetting mode. The differences between these two are as diverse as the classes and packages that offer this distinction.

```
draft=simple switch
overfullrule=simple switch
```

v3.00 The `draft` option distinguishes between documents being drafted and finished documents. The *simple switch* can be one of the standard values for simple switches from table 2.5, page 42. If you activate this option, small black boxes will be output at the end of overly long lines. These boxes make it easier for the untrained eye to locate the paragraphs that require manual post-processing. By contrast, the default, `draft=false`, shows no such boxes. Incidentally, such lines often disappear when you use the microtype package [Sch13].

v3.25 Since the `draft` option can lead to all sorts of unwanted effects with various packages, KOMA-Script allows you to control this marking of overly long lines separately with the `overfullrule` option. If this option is enabled, the marker is again displayed.

3.4. Page Layout

Each page of a document consists of different layout elements, such as the margins, the header, the footer, the text area, the marginal note column, and the distances between these elements. KOMA-Script additionally distinguishes the entire page, also known as the paper, and the visible page. Without doubt, the separation of the page into these different parts is one of the basic features of a class. KOMA-Script delegates this work to the package typearea. This package can also be used with other classes. The KOMA-Script classes, however, load typearea on their own. Therefore, it's neither necessary nor sensible to load the package explicitly with `\usepackage` while using a KOMA-Script class. See also section 3.1, page 54.

Some settings of KOMA-Script classes affect the page layout and vice versa. Those effects are documented at the corresponding settings.

For more information about the choice of paper format, the division of the page into margins and type area, and the choice between one- and two-column typesetting, see the documentation for the typearea package. You can find it in chapter 2, starting on page 28.

```
\flushbottom
\raggedbottom
```

In two-sided documents especially, it is preferable to have the same visual baseline not only for the first lines of each text area in a two-page spread but also for the last lines. If a page consists only of text without paragraphs or headings, this is generally the result. But a paragraph spacing of half a line would be enough to prevent you from achieving this goal if the number of paragraphs on each page of the two-page spread differs by an odd number. In this case, at least some of the paragraph distances need to be stretched or shrunk to reach the target again. TEX defines stretchable and shrinkable distances for this purpose, and LATEX lets you perform this kind of *vertical adjustment* automatically.

Using two-sided printing with the `twoside` option (see section 2.4, page 41) or two-column formatting with the `twocolumn` option (see page 42) also activates this vertical adjustment.

v3.17 But this does not apply with a compatibility setting for a KOMA-Script version prior to 3.17 (see section 3.2, page 56, option version) if you use \KOMAoption or \KOMAoptions to change the setting of these options.

You can also explicitly request vertical adjustment at any time starting with the current page by using \flushbottom. \raggedbottom has the opposite effect, switching off vertical adjustment starting with the current page. This corresponds to the default for one-sided printing.

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

3.5. Choosing the Document Font Size

The main font and its size are central elements in the design of a document. As stated in chapter 2, the division of the page into the text area and the margins fundamentally depends on them. The main font is the one that is used for most of the text in a document. All variations, whether in shape, thickness, slant, or size, are related to the main font.

`fontsize=`*size*

While the standard classes support only a very limited number of font sizes, KOMA-Script provides the ability to specify any *size* for the main font. You can also use any known TEXunit as a unit for the *size*. If the *size* is specified without a unit, it is assumed to be pt.

If you set the option within the document, the main font size and the dependent font sizes of the commands \tiny, \scriptsize, \footnotesize, \small, \normalsize, \large, \Large, \LARGE, \huge and \Huge are changed. This can be useful, for example, if you want the appendix to be set in a smaller font size.

Note that using this option after loading the class does not automatically recalculate the type area and margins (see \recalctypearea, section 2.6, page 40). However, if this recalculation is performed, it will be based on the current main font size. The effects of changing the main font size upon other loaded packages or the class used depends on these packages and on the class. This means that you can encounter errors which are not the fault of KOMA-Script, and even the KOMA-Script classes themselves do not recalculate all lengths if the main font size changes after loading the class.

This option should by no means be misinterpreted as a substitute for \fontsize (see [Tea05a]). Also, you should not use it in place of one of the font size commands that are relative to the main font, from \tiny to \Huge.

The default for scrbook, scrreprt, and scrartcl is fontsize=11pt. In contrast, the default size in the standard classes is 10pt. You may need to account for this difference 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. In addition to the choice of the font, this includes commands for choosing the font size and orientation. For more information about the standard font facilities, see [OPHS11], [Tea05b], and [Tea05a].

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

v2.8p

With the help of the `\setkomafont` and `\addtokomafont` commands, you can attach particular font styling *commands* that change the appearance of a given *element*. Theoretically, all statements, including literal text, can be used as *commands*. You should, however, limit yourself to those statements that really change font attributes only. These are usually commands like `\rmfamily`, `\sffamily`, `\ttfamily`, `\upshape`, `\itshape`, `\slshape`, `\scshape`, `\mdseries`, `\bfseries`, `\normalfont`, as well as the font size commands `\Huge`, `\huge`, `\LARGE`, `\Large`, `\large`, `\normalsize`, `\small`, `\footnotesize`, `\scriptsize`, and `\tiny`. You can find these commands explained in [OPHS11], [Tea05b], or [Tea05a]. Colour switching commands like `\normalcolor` (see [Car17] and [Ker07]) are also acceptable. The use of other commands, in particular those that redefine things or or lead to output, is not supported. Strange behaviour is possible in these cases and does not represent a bug.

The command `\setkomafont` provides an element with a completely new definition of its font styling. In contrast, the `\addtokomafont` command merely extends an existing definition. You should not use either command inside the document body but only in the preamble. For examples of their use, refer to the sections for the respective element. The name and meaning of each element are listed in table 3.2. The default values can be found in the corresponding sections.

With the `\usekomafont` command, the current font style can be changed to the one defined for the specified *element*.

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

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

You can find other examples in the explanation of each element.

Table 3.2.: Elements whose font style can be changed in scrbook, scrreprt or scrartcl with \setkomafont and \addtokomafont

author
v3.12 author of the document in the title, i. e., the argument of \author when \maketitle is used (see section 3.7, page 67)

caption
text of a figure or table caption (see section 3.20, page 129)

captionlabel
label of a figure or table caption; applied in addition to the caption element (see section 3.20, page 129)

chapter
title of the sectioning command \chapter (see section 3.16, page 100)

chapterentry
table of contents entry for the sectioning command \chapter (see section 3.9, page 75)

chapterentrydots
v3.15 optional points connecting table-of-content entries for the \chapter level, differing from the chapterentry element, \normalfont and \normalsize (see section 3.9, page 75)

chapterentrypagenumber
page number of the table of contents entry for the sectioning command \chapter, differing from the element chapterentry (see section 3.9, page 75)

chapterprefix
label, e. g., “Chapter”, appearing before the chapter number in both chapterprefix=true and appendixprefix=true (see section 3.16, page 95)

date
v3.12 date of the document in the main title, i. e., the argument of \date when \maketitle is used (see section 3.7, page 67)

dedication
v3.12 dedication page after the main title, i. e., the argument of \dedication when \maketitle is used (see section 3.7, page 70)

...

Table 3.2.: Elements whose font style can be changed (*continued*)

`descriptionlabel`
: labels, i. e., the optional argument of `\item` in the `description` environment (see section 3.18, page 119)

`dictum`
: dictum or epigraph (see section 3.17, page 115)

`dictumauthor`
: author of a dictum or epigraph; applied in addition to the element `dictum` (see section 3.17, page 115)

`dictumtext`
: alternative name for `dictum`

`disposition`
: all sectioning command titles, i. e., the arguments of `\part` down to `\subparagraph` and `\minisec`, including the title of the abstract; applied before the element of the respective unit (see section 3.16, page 94)

`footnote`
: footnote text and marker (see section 3.14, page 89)

`footnotelabel`
: marker for a footnote; applied in addition to the element `footnote` (see section 3.14, page 89)

`footnotereference`
: footnote reference in the text (see section 3.14, page 89)

`footnoterule`
v3.07
: horizontal rule above the footnotes at the end of the text area (see section 3.14, page 92)

`labelinglabel`
: labels, i. e., the optional argument of `\item` in the `labeling` environment (see section 3.18, page 120)

`labelingseparator`
: separator, i. e., the optional argument of the `labeling` environment; applied in addition to the element `labelinglabel` (see section 3.18, page 120)

...

Table 3.2.: Elements whose font style can be changed (*continued*)

`minisec`
: title of `\minisec` (see section 3.16 ab page 105)

`pagefoot`
: only used if package scrlayer-scrpage has been loaded (see chapter 5, page 261)

`pagehead`
: alternative name for `pageheadfoot`

`pageheadfoot`
: the header and footer of a page (see section 3.12 from page 79)

`pagenumber`
: page number in the header or footer (see section 3.12)

`pagination`
: alternative name for `pagenumber`

`paragraph`
: title of the sectioning command `\paragraph` (see section 3.16, page 100)

`part`
: title of the `\part` sectioning command, without the line containing the part number (see section 3.16, page 100)

`partentry`
: table of contents entry for the sectioning command `\part` (see section 3.9, page 75)

`partentrypagenumber`
: page number of the table of contents entry for the sectioning command `\part`; applied in addition to the element `partentry` (see section 3.9, page 75)

`partnumber`
: line containing the part number in a title of the sectioning command `\part` (see section 3.16, page 100)

`publishers`
v3.12
: publishers of the document in the main title, i. e., the argument of `\publishers` when `\maketitle` is used (see section 3.7, page 67)

`section`
: title of the sectioning command `\section` (see section 3.16, page 100)

Table 3.2.: Elements whose font style can be changed (*continued*)

`sectionentry`
: table of contents entry for sectioning command `\section` (only available in scrartcl, see section 3.9, page 75)

`sectionentrydots`
v3.15
: optional points connecting table-of-content entries for the `\section` level, differing from the `sectionentry` element, `\normalfont` and `\normalsize` (only available in scrartcl, see section 3.9, page 75)

`sectionentrypagenumber`
: page number of the table of contents entry for the sectioning command `\section`; applied in addition to element `sectionentry` (only available in scrartcl, see section 3.9, page 75)

`sectioning`
: alternative name for `disposition`

`subject`
: topic of the document, i. e., the argument of `\subject` on the main title page (see section 3.7, page 67)

`subparagraph`
: title of the sectioning command `\subparagraph` (see section 3.16, page 100)

`subsection`
: title of the sectioning command `\subsection` (see section 3.16, page 100)

`subsubsection`
: title of the sectioning command `\subsubsection` (see section 3.16, page 100)

`subtitle`
: subtitle of the document, i. e., the argument of `\subtitle` on the main title page (see section 3.7, page 67)

`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 67)

`titlehead`
v3.12
: heading above the main title of the document, i. e., the argument of `\titlehead` when `\maketitle` is used (see section 3.7, page 67)

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

v3.12

Sometimes, although this is not recommended, the font setting of an element is used for settings that are not actually related to the font. If you want to apply only the font setting of an element but not those other settings, you can use `\usefontofkomafont` instead of `\usekomafont`. This will activate the font size and baseline skip, the font encoding, the font family, the font series, and the font shape of an element, but no further settings as long as those further settings are local.

You can also switch to a single one of those attributes using one of the other commands. Note that `\usesizeofkomafont` uses both the font size and the baseline skip.

However, you should not take these commands as legitimizing the insertion of arbitrary commands in an element's font setting. To do so can lead quickly to errors (see section 21.5, page 476).

3.7. Document Titles

In general, we distinguish two kinds of document titles. First, there are title pages. These include title of the document, together with additional information such as the author, on a separate page. In addition to the main title page, there may be several other title pages, such as the half-title or bastard title, publisher data, dedication, and so on. Second, there is the in-page title. This kind of title appears at the top of a new page, usually the first, and is specially emphasized. It too may be accompanied by additional information, but it will be followed by more material on the same page, for example by an abstract, the table of contents, or even a section.

```
titlepage=simple switch
titlepage=firstiscover
\coverpagetopmargin
\coverpageleftmargin
\coverpagerightmargin
\coverpagebottommargin
```

v3.00

This option determines whether to use document title pages or in-page titles when using `\maketitle` (see page 66). Any value from table 2.5, page 42 can be used for *simple switch*.

With the `titlepage=true` or `titlepage` option, invoking `\maketitle` creates titles on separate pages. These pages are set inside a `titlepage` environment, and they normally have

neither header nor footer. Compared to standard LaTeX, KOMA-Script significantly expands the handling of the titles. These additional elements can be found on the following pages.

In contrast, with the `titlepage=false` option, invoking `\maketitle` creates an *in-page* title. This means that the title is specially emphasized, but it may be followed by more material on the same page, for instance an abstract or a section.

v3.12 The third choice, `titlepage=firstiscover` not only activates title pages but also prints the first title page of `\maketitle`, i. e. either the half-title or the main title, as a cover page. Any other setting of the `titlepage` option will cancel this setting. The margins of the cover page are given by `\coverpagetopmargin`, `\coverpageleftmargin`, `\coverpagerightmargin`, and `\coverpagebottommargin`. The defaults of these depend on the lengths of `\topmargin` and `\evensidemargin` and can be changed with `\renewcommand`.

The default of the scrbook and scrreprt classes is to use title pages. The scrartcl class, on the other hand, uses in-page titles by default.

`\begin{titlepage}`...`\end{titlepage}`

The standard classes and KOMA-Script set all title pages in a special environment: the `titlepage` environment. This environment always starts a new page — in two-sided printing a new right-hand page — and in single-column mode. For this page, the style is changed to `\thispagestyle{empty}`, so that neither page number nor running head is 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`, described next, you should design a new one with the help of this environment.

Example: Suppose 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 standard classes produce at most one title page that can have three items (title, author, and date), with KOMA-Script `\maketitle` can produce up to six pages. In contrast to the standard classes, `\maketitle` in KOMA-Script accepts an optional numeric argument. If it is used, this number is the page number of the first title page. This page number is not output, but it affects the subsequent numbering. You should definitely choose an odd number, because otherwise the whole count gets mixed up. In my opinion, there are only two useful applications for the optional argument. On the one hand, you could give the the logical page number -1 to the half-title in order to give the full title page the number 1. On the other hand, you could use it to start at a higher page number, for example, 3, 5, or 7, to accommodate other title pages added by the publishing house. The optional argument is ignored for *in-page* titles. You can change the page style of such a title page by redefining the `\titlepagestyle` macro (see section 3.12, page 83).

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`. Note also that `\maketitle` should not be used inside a `titlepage` environment. As shown in the examples, you should use either `\maketitle` or `titlepage`, but not both.

The following commands only define the contents of the title. Therefore they must be used before `\maketitle`. It is, however, not necessary and, when using the babel package not recommended, to include these in the preamble before `\begin{document}` (see [BB13]). You can find examples in the descriptions of the other commands in this section.

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

In earlier times the inner book was often not protected from dirt by a cover. This function was then assumed by the first page of the book, which usually had just a short title, known as the *half-title*. Nowadays the extra page often appears before the real main 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 verso of the half-title is the frontispiece. The half-title is set on its own page even when in-page titles are used. The output of the half-title defined with `\extratitle` takes place as part of the title produced by `\maketitle`.

v3.25

Example: Let's return to the previous example and suppose that the Spartan "Me" is the half-title. The full title should still follow the half-title. You can proceed as follows:

```
\documentclass{scrbook}
\begin{document}
  \extratitle{\textbf{\Huge Me}}
```

```
  \title{It's me}
  \maketitle
\end{document}
```

You can centre 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}
```

There are seven elements available for the content of the main title page. The main title page is output as part of the title pages created by \maketitle, while the definitions given here only apply to the respective elements.

The *title head* is defined with the command \titlehead. It occupies the entire text width, at the top of the page, in normal justification, and it can be freely designed by the user. It uses the font element with same name (see table 3.4, page 68).

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

v2.8p The *title* is set in a very large font size. Along with the font size, the font element title is applied (see table 3.4, page 68).

v2.97c The *subtitle* is set just below the title using the font element of the same name (see table 3.4, page 68).

Below the *subtitle* appears the *author*. Several authors can be specified in the argument of \author. They should be separated by \and. The output uses the font element of the same name. (see table 3.4, page 68).

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

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

Below the author or authors appears the date in the font of the element of the same name. The default value is the current date, as produced by \today. The \date command accepts arbitrary information — even an empty argument. The output uses the font element of the same name (see table 3.4, page 68).

Finally comes the *publisher*. Of course this command can also be used for any other information of minor importance. If necessary, the \parbox command can be used to typeset this information over the full page width like a regular paragraph instead of centring it. It should then be considered equivalent to the title head. Note, however, that this field is placed above any existing footnotes. The output uses the font element of the same name (see table 3.4, page 68).

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, such as in the *author* argument of the command \author.

v3.12 For the output of the title elements, the font can be set using the \setkomafont and \addtokomafont command (see section 3.6, page 59). The defaults are listed in table 3.3.

With the exception of *title head* and any footnotes, all output is centred horizontally. These details are briefly summarized in table 3.4.

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

Element	Command	Font	Alignment
Title head	\titlehead	\usekomafont{titlehead}	justified
Subject	\subject	\usekomafont{subject}	centred
Title	\title	\usekomafont{title}\huge	centred
Subtitle	\subtitle	\usekomafont{subtitle}	centred
Authors	\author	\usekomafont{author}	centred
Date	\date	\usekomafont{date}	centred
Publishers	\publishers	\usekomafont{publishers}	centred

Note that for the main title, \huge will be used after the font switching element title. So you cannot change the size of the main title using \setkomafont or \addtokomafont.

Example: Suppose 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 including author and submission date should be given. The adviser must also be indicated, together with the fact that the document is a dissertation. You can do this 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 common misconception concerns the function of the full title page. It is often erroneously assumed to be the cover or dust jacket. Therefore, it is frequently expected that the title page will not follow the normal layout for two-sided typesetting but will have equally large left and right margins.

But if you pick up a book and open it, you will quickly find at least one title page inside the cover, within the so-called book block. Precisely these title pages are produced by \maketitle.

As is the case with the half-title, the full title page belongs to the book block, and therefore should have the same page layout as the rest of the document. A cover is actually something that you should create in a separate document. After all, it often has a very distinct 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, such as cardboard, and possibly with another printer.

Nevertheless, since KOMA-Script 3.12 the first title page issued by \maketitle can be formatted as a cover page with different margins. Changes to the margins on this page do not affect the other margins. For more information about this option, see titlepage=firstiscover on page 64.

\uppertitleback{*titlebackhead*}
\lowertitleback{*titlebackfoot*}

In two-sided printing, the standard classes leave the back (verso) of the title page empty. However, with KOMA-Script the back of the full title page can be used for other information. There are exactly two elements which the user can freely format: *titlebackhead* and *titlebackfoot*. The header can extend to the footer and vice versa. Using this guide as an example, the legal disclaimer was set with the help of the \uppertitleback command.

\dedication{*dedication*}

v3.12

KOMA-Script offers its own dedication page. This dedication is centred and set by default with a slightly larger font. The exact font setting for the dedication element, which is taken from table 3.3, page 68, can be changed with the \setkomafont and \addtokomafont commands (see section 3.6, page 59).

Example: Suppose you have written a book of poetry and want to dedicate it to your spouse. 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{Self-mockery Publishers}
\dedication{To my treasured hazel-hen\\
  in eternal love\\
  from your dormouse.}
\maketitle
\end{document}
```

Please use your own favourite pet names to personalize it.

3.8. Abstract

Particularly with articles, more rarely with reports, there is an abstract, or summary, directly beneath the title and before the table of contents. When using an in-page title, this abstract is normally a kind of left- and right-indented block. In comparison, the abstract appears as a chapter or section when using title pages.

`abstract=`*simple switch*

scrreprt, scrartcl

In the standard classes, the `abstract` environment sets the text "Abstract" centred before the abstract text. This used to be the normal practice. Since then, reading newspapers has trained us to recognize a suitably highlighted 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 confusing if, of all things, this title appears small and centred. KOMA-Script offers the option to include or exclude the abstract's title with the `abstract` option. For *simple switch*, you can use any value from table 2.5, page 42. The default for KOMA-Script is `false`.

v3.00

Books typically use a different kind of summary. There, you usually place an appropriate chapter at the beginning or the end of the work. This chapter is often combined with either the introduction or a description of a larger prospectus. Therefore, the scrbook class has no `abstract` environment. A summary chapter is also recommended for reports in a wider sense, such as a Master's thesis or Ph.D. dissertation. See the commands `\chapter*`, `\addchap`, and `\addchap*` documented in section 3.16, from page 104.

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

scrartcl, scrreprt

Some LaTeX classes provide a special environment for this summary: the `abstract` environment. This is output directly, so it is not part of the title created with `\maketitle`. Please note that `abstract` is an environment, not a command. Whether the abstract has a heading or not is determined by the `abstract` option (see above).

For books, the abstract is usually part of the introduction or a separate chapter at the end of the document. Therefore scrbook does not provide an `abstract` environment. When using the scrreprt class, it is definitely worth considering whether to proceed in the same way. See the commands `\chapter*` and `\addchap`, or `\addchap*` in section 3.16 from page 104 for more on this.

When using an in-page title (see option `titlepage`, section 3.7, page 64), the abstract is set internally using the `quotation` environment (see section 3.18, page 123). This way paragraphs will be set with the first line indented. If the first paragraph of the abstract should not be indented, you can suppress this indent by using `\noindent` just after `\begin{abstract}`.

3.9. Table of Contents

The title and optional abstract are normally followed by a table of contents. Often you also find additional lists of the floating environments, such as tables and figures, after the table of contents (see section 3.20).

In addition to the options documented in this section, the tocbasic package style selected and configured with `\DeclareTOCStyleEntry` (see page 386) also has a significant impact on the appearance of the table of contents. Similarly, the commands `\DeclareSectionCommand`, `\ProvideSectionCommand`, `\DeclareNewSectionCommand` and `\RedeclareSectionCommand` documented in section 21.8, page 479 can also affect the table of contents.

`toc=`*`setting`*

It is becoming increasingly common to to include lists of tables and figures, the bibliography, and sometimes even the index in the table of contents. This is surely related to the recent trend of putting lists of figures and tables at the end of the document. Both lists are similar to the table of contents in structure and intention. I'm therefore sceptical of this evolution. Since it makes no sense to include only the list of tables or that of figures in the table of contents without the other, there is only one *`setting`* `listof`, which 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 from Version 1.2e (see [Lin01]) or floatrow (see [Lap08]). None of these lists are generally given a chapter number. If you want to ignore this principle, use the *`setting`* `listofnumbered`.

v3.00

The `toc=index` option 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 sectioning levels. Despite the author's concerns, KOMA-Script does support deviating from this principle with `toc=indexnumbered`.

v3.18

The bibliography is a slightly different kind of listing. It does not list the contents of the present document but refers instead to external sources. For that reason, it could be argued that it qualifies as a chapter (or section) and, as such, should be numbered. The `toc=bibliographynumbered` option has this effect, and puts the appropriate entry in the table of contents. However, I think that this reasoning would lead us to consider even a classic, annotated source list to be a separate chapter. Moreover, the bibliography is ultimately not something that you wrote yourself. Therefore the bibliography merits, at best, an unnumbered entry in the table of contents, and you can achieve this achieved with `toc=bibliography`.

v2.8q

The table of contents is normally formatted so that different levels of sectioning commands have different indentations. The number for each level is set left-justified in a fixed-width field. This default set-up is selected with the `toc=graduated` option.

v3.00

If the sectioning level which appears in the table of contents is too deep, the number for that level can be so wide that the space reserved for the number is insufficient. The German FAQ [Wik] suggests redefining the table of contents in such a case. KOMA-Script offers an alternative format that avoids the problem completely. If you use the `toc=flat` option, no graduated indentation is applied to the headings of the sectioning levels. Instead, a table-like organisation is used, where all sectioning numbers and headings are set in a left-justified column. The space necessary for the section numbers is thus determined automatically.

You can find an overview of all available values for the *`setting`* of `toc`. in table 3.5.

Table 3.5.: Available values for the `toc` option to set the format and contents of the table of contents

`bibliography`, `bib`
: The bibliography has an unnumbered entry in the table of contents.

`bibliographynumbered`, `bibnumbered`, `numberedbibliography`, `numberedbib`
: The bibliography has a numbered entry in the table of contents.

`chapterentrywithdots`, `chapterentrydotfill`
v3.15
: The chapter entries for the scrbook and scrreprt classes also use dots to separate the heading text from the page numbers.

`chapterentrywithoutdots`, `chapterentryfill`
v3.15
: The chapter entries of the scrbook and scrreprt classes use white space to separate the heading text from the page numbers. This corresponds to the default setting.

`flat`, `left`
: The table of contents is set in table form. The numbers of the headings are in the first column, the heading text in the second column, and the page number in the third column. The amount of space needed for the numbers of the headings is determined by the required amount of space detected during the previous LaTeX run.

`graduated`, `indent`, `indented`
: The table of contents is set in hierarchical form. The amount of space for the heading numbers is limited. This corresponds to the default setting.

`indenttextentries`, `indentunnumbered`, `numberline`
v3.12
: The `numberline` property (see section 15.2, page 383) is set for the table of contents. As a result, unnumbered entries are left aligned with the text of numbered entries of the same level.

`index`, `idx`
: The index has an unnumbered entry in the table of contents.

`indexnumbered`, `idxnumbered`, `numberedindex`, `numberedidx`
v3.18
: The index has a numbered entry in the table of contents.

`leftaligntextentries`, `leftalignunnumbered`, `nonumberline`
v3.12
: The `numberline` property (see section 15.2, page 383) is deleted for the table of contents. This places unnumbered entries left-aligned with the number of numbered entries of the same level. This corresponds to the default setting.

...

Table 3.5.: Available values for the `toc` option (*continued*)

`listof`	The lists of floating environments, e. g. figures and tables, have unnumbered entries in the table of contents.
`listofnumbered`, `numberedlistof`	The lists of floating environments, e. g. figures and tables, have numbered entries in the table of contents.
`nobibliography`, `nobib`	The bibliography does not have an entry in the table of contents. This corresponds to the default setting.
`noindex`, `noidx`	The index does not have an entry in the table of contents. This corresponds to the default setting.
`nolistof`	The lists of floating environments, e. g. figures and tables, do not have entries in the table of contents. This corresponds to the default setting.
`sectionentrywithdots`, `sectionentrydotfill`	v3.15 The section entries of the scrartcl class also use dots to separate the heading text from the page numbers.
`sectionentrywithoutdots`, `sectionentryfill`	v3.15 The section entries of the scrartcl class use white space to separate the heading text from the page number. This corresponds to the default setting.

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

v3.15 scrbook, scrreprt scrartcl

These options configure a dotted connecting line between the text and page number of the chapter entries for the scrbook and scrreprt classes, or for the section entries of the scrartcl class, in the table of contents. For the *`simple switch`*, you can use any value from table 2.5, page 42. The default is `false`. It selects an empty gap instead of dots.

If a dotted line is used, you can change its font using the element `chapterentrydots` or `sectionentrydots` (see also `\setkomafont` and `\addtokomafont`, section 3.6, page 59, as well as table 3.2, page 60). The defaults of the elements are shown in table 3.6, from page 75. Note that the dots of all entries will be equally spaced only if all dots use the same font. Because of this the base font is always `\normalfont\normalsize` and only the colour of `chapterentry`

Table 3.6.: Default font styles for the elements of the table of contents

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

or sectionentry is also used for the dots.

\tableofcontents

The table of contents is output by the \tableofcontents command. To get correct values in the table of contents requires at least two LaTeX runs after every change. The toc option described above can also affect the extent and format of the table of contents. After changing the settings of this option, at least two LaTeX runs are needed again.

Entries for \chapter with scrbook and scrreprt, or \section with scrartcl, and the sectioning level \part are not indented. Additionally, there are no dots between the text of this heading and the page number. The typographical logic for this behaviour is that the font is usually distinct and appropriate emphasis is desirable. However, you can change this behaviour with the previously documented options. The table of contents of this guide is created with the default settings and serves as an example.

v3.15

v2.97c

The font style of the top two levels in the table of contents is also affected by the settings for the partentry element, as well as by the chapterentry element for the scrbook and scrreprt classes, and by the sectionentry element for the scrartcl class. You can set the font style of the page numbers separately from these elements using partentrypagenumber and chapterentrypagenumber — for scrbook and scrreprt — or sectionentrypagenumber — for scrartcl — (see \setkomafont and \addtokomafont in section 3.6, page 59, or table 3.2, page 60). If you use dotted lines connecting the heading entries (chapter or section depending on the class) to the page numbers using the toc chapterentrydots or sectionentrydots option, you can change their font style using the chapterentrydots and sectionentrydots elements. The defaults for these elements are found in table 3.6.

v3.15

```
tocdepth
\parttocdepth
\sectiontocdepth
\subsectiontocdepth
\subsubsectiontocdepth
\paragraphtocdepth
\subparagraphtocdepth
```

Normally, the sectioning divisions included in the table of contents are all those from \part to \subsection for the scrbook and scrreprt classes, or from \part to \subsubsection for the scrartcl class. Whether or not to include a sectioning level in the table of contents is controlled by the tocdepth counter. This has the value -1 for \part, 0 for \chapter, and so on. By incrementing or decrementing the counter, you can choose the lowest sectioning level to include in the table of contents. Incidentally, the standard classes work the same way. Unlike with the standard classes, with KOMA-Script you do not need to remember these values. KOMA-Script defines a *level*tocdepth command for each sectioning level with the appropriate value which you can use to set tocdepth.

v3.15

scrartcl Please note that in scrartcl, the values of tocdepth and secnumdepth (see section 3.16, page 112) for \part are not the same. This behaviour was copied from the standard article class for compatibility. Thus, for example, you should not use \partnumdepth to set the value of tocdepth.

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

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

Thus you set the tocdepth counter to the value of the \subsectiontocdepth command. That value is normally 2, but this way, you do not have to remember it.

If instead you simply want to include one less level in the table of contents than you normally would, you can simply subtract one from the default value of tocdepth:

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

The value that you need to add to or subtract from tocdepth is listed in the table of contents after at least two LaTeX runs.

3.10. Marking Paragraphs

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 guide to the paragraph break. In extreme cases, it is very difficult to tell whether a line is full or not. Furthermore, typographers find that a signal given at the paragraph's end is easily forgotten by the start of the next line. A signal 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 a new one begins. Also, at the top of a new page, it might be necessary to look at the previous page to determine if a new paragraph has been started or not. All these problems disappear when using indentation. A combination of indentation and vertical inter-paragraph spacing is redundant and therefore should be avoided. Indentation alone is sufficient. The only drawback of indentation is that it shortens the line length. The use of inter-paragraph spacing is therefore justified when using short lines, such as in a newspaper.

`parskip=`*`method`*

Once in a while you may require a document layout with vertical inter-paragraph spacing instead of indentation. The KOMA-Script classes provide several ways to accomplish this with the `parskip` option. The *`method`* consists of two elements. The first element is either `full` or `half`, where `full` stands for a paragraph spacing of one line and `half` stands for a paragraph spacing of half a line. The second element consists of one of the characters "`*`", "`+`", or "`-`" and can be omitted. Without the second element, the final line of a paragraph will end with a white space of at least 1 em. With the plus character as the second element, the white space will be at least one third — and with the asterisk one fourth — the width of a normal line. With the minus variant, no provision is made for white space in the last line of a paragraph.

v3.00

You can change the setting at any time. If you change it inside the document, the `\selectfont` command will be called implicitly. Changes to paragraph spacing within a paragraph will not be visible until the end of the paragraph.

v3.08

In addition to the resulting eight combinations for *`method`*, you can use the values for simple switches shown in table 2.5, page 42. Activating the option corresponds to using `full` with no second element and therefore results in inter-paragraph spacing of one line with at least 1 em white space at the end of the last line of each paragraph. Deactivating the option re-activates the default indentation of 1 em at the first line of the paragraph instead of paragraph spacing. A summary of all possible values for *`method`* are shown in table 3.7.

Table 3.7.: Available values of option `parskip` to select how paragraph are distinguished

`false`, `off`, `no`
: Paragraphs are identified by indentation of the first line by 1em. There is no spacing requirement at the end of the last line of a paragraph.

`full`, `true`, `on`, `yes`
: Paragraphs are identified by a vertical space of one line between paragraphs. There must be at least 1 em of free space at the end of the last line of the paragraph.

`full-`
: Paragraphs are identified by a vertical space of one line between paragraphs. There is no spacing requirement at the end of the last line of a paragraph.

`full+`
: Paragraphs are identified by a vertical space of one line between paragraphs. There must be at least a third of a line of free space at the end of a paragraph.

`full*`
: Paragraphs are identified by a vertical space of one line between paragraphs. There must be at least a quarter of a line of free space at the end of a paragraph.

`half`
: Paragraphs are identified by a vertical space of half a line between paragraphs. There must be at least 1 em free space at the end of the last line of a paragraph.

`half-`
: Paragraphs are identified by a vertical space of half a line between paragraphs. There is no spacing requirement at the end of the last line of a paragraph.

`half+`
: Paragraphs are identified by a vertical space of half a line between paragraphs. There must be at least a third of a line of free space at the end of a paragraph.

`half*`
: Paragraphs are identified by a vertical space of half a line between paragraphs. There must be at least a quarter of a line of free space at the end of a paragraph.

`never`
v3.08
: No inter-paragraph spacing will be inserted even if additional vertical spacing is needed for vertical adjustment with `\flushbottom`.

All eight `full` and `half` option values also change the spacing before, after, and inside list

environments. This prevents these environments or the paragraphs inside them from having a larger separation than that 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 is `parskip=false`. In this case, there is no spacing between paragraphs, only an indentation of the first line by 1 em.

3.11. Detecting Odd and Even Pages

In two-sided documents we distinguish left and right pages. Left pages always have an even page number, and right pages always have an odd page number. Identifying right and left pages is equivalent to identifying even- or odd-numbered pages, and so we normally refer to them as even and odd pages in this guide.

In one-sided documents, the distinction between left and right pages does not exist. Nevertheless, there are still pages with even and odd page numbers.

`\Ifthispageodd{`*true part*`}{`*false part*`}`

v3.28

If you want to determine whether text appears on an even or odd page, KOMA-Script provides the `\Ifthispageodd` command. The *true part* argument is executed only if you are currently on an odd page. Otherwise the *false part* argument is executed.

Example: Suppose 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.
```

This results 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 each change to the text. Only then will the decision be correct. In the first run, a heuristic is used to make the initial choice.

In section 21.1, page 473, advanced users can find more information about the problems of detecting left and right pages, or even and odd page numbers.

3.12. Headers and Footers Using Predefined Page Styles

One of the general characteristics of a document is the page style. In LaTeX this primarily consists of the contents of headers and footers.

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

v3.00

You can use these options to specify whether a horizontal rule appears beneath the header or above the footer. You can use any of the values for simple switches shown in table 2.5, page 42. Setting the **headsepline** option to true or invoking it with no value results in a line beneath the header. Similarly, activating the **footsepline** option results in a rule above the footer. Deactivating either option switches off the respective rule.

The **headsepline** option naturally has no effect with the empty and plain page styles, which are described below, because these styles explicitly dispense with a header. Typographically, such a line has the effect of making the header appear to be closer to the text. This does not mean that the header then needs to be moved farther away from the body of the text. Instead, the header should be considered as belonging to the text body for the purpose of calculating the type area. KOMA-Script takes this into account by passing the **headsepline** option to the typearea package, which then automatically executes the package option headinclude with the same value. The same applies to the footer separation line. Unlike **headsepline**, the **footsepline** option also affects the **plain** page style because **plain** prints a page number in the footer.

The options themselves do not automatically recalculate the type area. To recalculate it, use the DIV option with the values last or current (see page 38) or the \recalctypearea command (see page 40) in chapter 2.

The scrlayer-scrpage package (see chapter 5) offers further possibilities for adjusting lines in headers and footers.

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

There are usually four different page styles:

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

headings is the page style with running heads in the header. In this style, headings are automatically inserted into the header. With the classes scrbook and scrreprt, the headings of chapters and sections are repeated in the header for two-sided printing — on the outer side with KOMA-Script, on the inner side with the standard classes. KOMA-Script puts the page number on the outer side of the footer; the standard classes put it on the inner side of the header. In one-sided printing, KOMA-Script uses only the chapter headings, which are centred in the header, and puts the page numbers centred in the footer. scrartcl behaves similarly but starts one a level deeper in the sectioning hierarchy, with sections and subsections, because the chapter level does not exist in this case.

scrbook, scrreprt

scrartcl

While the standard classes automatically convert the running heads to upper-case letters, KOMA-Script uses the capitalisation found in the headings. There are several

Table 3.8.: Default values for page style elements

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

typographical reasons for this. Upper-case letters are actually far too massive as a text decoration. If you use them anyway, they should be set one point smaller and with slightly tighter spacing. The standard classes do not take these points into consideration.

In addition, the KOMA-Script classes support rules below the header and above the footer with the `headsepline` and `footsepline` options (see page 80).

myheadings mostly corresponds to the **headings** page style, but the running heads are not generated automatically — they have to be defined by the user. You can use the `\markboth` and `\markright` commands for that purpose (see page 82).

plain is the page style with no running head and only a page number in the footer. The standard classes always centre this page number in the footer. KOMA-Script puts the page number on the outer side of the footer in two-sided mode. KOMA-Script behaves like the standard classes in one-sided printing.

You can set the page style at any time with the help of the `\pagestyle` command, and this setting takes effect with the next page that is output. If you use `\pagestyle` just before a command that results in an implicit page break and if the new page style should be used on the resulting new page, a `\cleardoublepage` just before `\pagestyle` will be useful. But usually you set 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, use the `\thispagestyle` command. This occurs automatically at some points in the document. For example, the `\thispagestyle{\chapterpagestyle}` command is issued implicitly on the first page of a chapter.

Note that when you use the scrlayer-scrpage package, switching between automatic and manual running heads is no longer accomplished by changing the page styles but with special instructions. You should not use the **headings** and **myheadings** page styles with this package.

v2.8p To change the font style used for the header, the footer, or the page number, use the `\setkomafont` and `\addtokomafont` commands (see section 3.6, page 59). The same element, **pageheadfoot**, is used for the header and the footer. The element for the page number within the header or footer is called **pagenumber**. The **pagefoot** element, which is also provided by the KOMA-Script classes, is used only if you define a page style with the scrlayer-scrpage package in which the footer contains text (see chapter 5, page 261).

You can find the default settings in table 3.8.

Example: Suppose 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 in bold. Apart from the fact that the result will look horrible, you can do this as follows:

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

On the other hand, if you only want a smaller font to be used along with the default slanted text, you can use the following:

```
\addtokomafont{pagehead}{\small}
```

As you can see, the previous example uses the `pagehead` element. You can achieve the same result using `pageheadfoot` instead (see table 3.2 on page 60).

It is not possible to use these methods to force upper-case letters to be used automatically for the running heads. Although you can redefine `\MakeMarkcase`, in such cases you should instead use the scrlayer-scrpage package (see chapter 5, page 270).

If you define your own page styles, the commands `\usekomafont{pageheadfoot}`, `\usekomafont{pagenumber}`, and `\usekomafont{pagefoot}` can be useful. In particular, if you do not use the KOMA-Script package scrlayer-scrpage (see chapter 5) but use, for example, the fancyhdr package (see [vO04]), you can use these commands in your definitions. In this way you can maintain compatibility with KOMA-Script as much as possible. If you do not use these commands in your own definitions, changes such as those shown in the previous examples have no effect. The scrlayer-scrpage package tries to maintain maximum compatibility as long as, for example, `\thepage` is not used directly for the page number rather than the `\pagemark` which is provided for it.

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

The myheadings page style does not set the running head. Instead, you set it with the help of the `\markboth` and `\markright` commands. This way the *left mark* will normally be used in the header of even pages and *right mark* in the header of odd pages. With one-sided printing, only the *right mark* exists. With the scrlayer-scrpage package, the `\markleft` command is also available.

You can use these commands with other page styles too. However, when combined with automatic running heads, for example with the headings page style, the effect of the commands lasts only until the next time the respective marks are set automatically.

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

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

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

scrbook, scrreprt

On some pages, a different page style is chosen automatically with the help of the `\thispagestyle` command. Which page style this actually is, is defined by these four macros, of which `\partpagestyle` and `\chapterpagestyle` are found only with classes scrbook and scrreprt, and not in scrartcl. The default value for all four cases is `plain`. You can find the meaning of these macros in table 3.9. You can redefine the page styles with the `\renewcommand` macro.

Example: Suppose you do not want the pages with a `\part` heading to be numbered. You can use the following command in the preamble of your document:

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

As mentioned previously on page 80, the `empty` page style is exactly what is required in this example. Of course, you can also use a user-defined page style.

Suppose you have defined your own page style for initial chapter pages with the scrlayer (see section 17.4) or the scrlayer-scrpage package (see section 18.2). You have given this page style the fitting name of `chapter`. To actually use this style, you must redefine `\chapterpagestyle` in this way:

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

Suppose you do not want the table of contents of a book to have page numbers. Everything after the table of contents, however, should use the `headings` page style, including the `plain` page style for the first page of every chapter. You can use the following:

```
\clearpage
\pagestyle{empty}
```

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

You can also keep the redefinition local by using a group. This method has the advantage that you do not need to make any assumptions about the what the previous page style was in order to restore it after your local change:

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

Note, however, that you never should put a numbered sectioning command into a group. Otherwise you may get unpredictable results with commands like `\label`.

On page 381 in section 15.2, you will discover the `\AfterTOCHead` command, which makes a solution even easier:

```
\AfterTOCHead[toc]{%
  \thispagestyle{empty}%
  \pagestyle{empty}%
}
```

This takes advantage of the fact that if there are several `\thispagestyle` commands on the same page, the last one always wins.

You might think that you can put running heads on the first page of a chapter simply by using the

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

command. Before you try this, you should read the remarks on `\rightfirstmark` starting on page page 446 in chapter 18, part II .

`\pagenumbering{`*numbering style*`}`

This command works the same way in KOMA-Script as in the standard classes. Strictly speaking, it is a feature of neither the standard classes nor the KOMA-Script classes but of the LaTeX kernel. This command is used to change the *numbering style* of page numbers.

The changes take effect immediately, i. e., starting from the page that contains the command. If necessary, you should first close the current page with `\clearpage` or better `\cleardoubleoddpage`. You can find the available settings for *numbering style* in table 3.10.

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

Calling `\pagenumbering` always resets the page number. The current page becomes number 1 in the selected *numbering style*. In order that two-sided documents produce the correct results on an even page, so that the left-hand page is not missing, you should always add `\cleardoubleoddpage` before `\pagenumbering`. The next section provides more information about potentially inserted blank pages.

Let me say a word about a common mistake found in various templates circulating on the Internet. If you encounter lines like the following — without the initial comment naturally — this is an unmistakable sign that the creator did not read or understand the remark above:

```
% Attention! This example contains errors!
% Please note the explanation in the text!
\tableofcontents
\pagenumbering{arabic}
\setcounter{page}{1}
```

Since `\tableofcontents` outputs the table of contents but does not automatically issue a page break at the end, the page numbering is already changed on the last page of the table of contents. Because it lacks a `\cleardoubleoddpage` command before `\pagenumbering`, it receives a pagination of the Arabic number 1. Additionally, the final line which sets the page numbering to 1 is superfluous, since this is already done by `\pagenumbering`.

Sometimes — without the initial comment, naturally — you find:

```
% Attention! This example contains errors!
% Please note the explanation in the text!
\tableofcontents
\pagebreak
\pagenumbering{arabic}
\setcounter{page}{1}
```

Here the creator tried to solve the problem with the final page of the table of contents with the help of `\pagebreak`.

Unfortunately, this solution is not much better. Here there is a page break after the last page of the table of contents. This may cause entries on the last page of a two-sided document to have excess vertical spacing (see `\flushbottom`, page 57). `\pagebreak` is clearly the wrong command here.

Furthermore, `\newpage` or `\clearpage` would not be sufficient for a two-sided document. For

example, if the last page of the table of contents had the Roman numeral vii, the Arabic numbered right-side page 1 would immediately follow the Roman numeral right-side page. A left-side page between the two would be missing, which could cause serious problems with later printing.

My advice: Avoid using templates that contain errors with respect to such simple things. Incidentally, the correct way would be:

```
\tableofcontents
\cleardoubleoddpage
\pagenumbering{arabic}
```

scrartcl This also applies if scrartcl uses a class that usually does not start a new page after the table of contents. If you switch the page numbering, a new right-hand page must be started. If you do not want such a change, you should keep the numbering style of pages consistent throughout the document without changing it in between.

scrbook It is easier to change the numbering style when using scrbook. There you have the support of two commands, \frontmatter and \mainmatter, for the most commonly used switching. For more information, please see section 3.15, page 93.

3.13. Interleaf Pages

Interleaf pages are pages that are inserted between parts of a document. Traditionally, these pages are completely blank. LaTeX, however, sets them by default with the current page style. KOMA-Script provides several extensions to this functionality.

Interleaf pages are mostly found in books. Because book chapters commonly start on the right (recto) page of a two-page spread, an empty left (verso) page must be inserted if the previous chapter ends on a recto page. For this reason, interleaf pages really only exist for two-sided printing.

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

v3.00 With this option, you can define the page style of the interleaf pages created by the commands \cleardoublepage, \cleardoubleoddpage, or \cleardoubleevenpage to advance to the desired page. You can use any previously defined *page style* (see section 3.12 from page 79 and chapter 5 from page 252). In addition, cleardoublepage=current is also possible. This case corresponds to the default prior to KOMA-Script 2.98c and creates an interleaf page without changing the page style. Starting with KOMA-Script 3.00, the default follows the recommendation of most typographers and creates interleaf pages with the empty page style unless you switch compatibility to earlier KOMA-Script versions (see option version, section 3.2, page 56).

v3.00

Example: Suppose you want interleaf pages that are empty except for the pagination, so they are created with plain. You can achieve this, for example, with:

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

You can find more information about the `plain` page style in section 3.12, page 81.

```
\clearpage
\cleardoublepage
\cleardoublepageusingstyle{page style}
\cleardoubleemptypage
\cleardoubleplainpage
\cleardoublestandardpage
\cleardoubleoddpage
\cleardoubleoddpageusingstyle{page style}
\cleardoubleoddemptypage
\cleardoubleoddplainpage
\cleardoubleoddstandardpage
\cleardoubleevenpage
\cleardoubleevenpageusingstyle{page style}
\cleardoubleevenemptypage
\cleardoubleevenplainpage
\cleardoubleevenstandardpage
```

The LaTeX kernel provides the `\clearpage` command, which ensures that all pending floats are output and then starts a new page. There is also the `\cleardoublepage` command, which works like `\clearpage` but which starts a new right-hand page in two-sided printing (see the `twoside` layout option in section 2.4, page 41). An empty left-hand page in the current page style is output if necessary.

v3.00 With `\cleardoubleoddstandardpage`, KOMA-Script works as exactly in the way just described for the standard classess. The `\cleardoubleoddplainpage` command, on the other hand, additionally changes the page style of the empty left page to `plain` in order to suppress the running title. Likewise, the `\cleardoubleoddemptypage` command uses the `empty` page style to suppress both running title and page number on the empty left-hand side. The page is thus completely empty. If you want to specify your own *page style* for the interleaf page, this should be given as an argument of `\cleardoubleoddusingpagestyle`. You can use any previously defined *page style* (see chapter 5).

Sometimes you want chapters to start not on the right-hand but on the left-hand page. Although this layout contradicts classic typography, it can be appropriate if the double-page spread at the beginning of the chapter very specific contents. For this reason, KOMA-Script provides the `\cleardoubleevenstandardpage` command, which is equivalent to the `\cleardoubleoddstandardpage` command except that the next page is a left page. The same applies to the `\cleardoubleevenplainpage`, `\cleardoubleevenemptypage`, and `\cleardoubleevenpageusingstyle` commands, the last of which expects an argument.

The \cleardoublestandardpage, \cleardoubleemptypage, and \cleardoubleplainpage commands, and the single-argument \cleardoublepageusingstyle command, as well as the standard \cleardoublepage command, depend on the open option explained in section 3.16, page 94 and, depending on that setting, correspond to one of the commands explained in the preceding paragraphs.

Example: Suppose you want to insert a double-page spread into your document with a picture on the left-hand page and a new chapter starting on the right-hand page. If the previous chapter ends with a left-hand page, an interleaf page has to be added, which should be completely empty. The picture should be the same size as the text area without any header or footer.

At the relevant place in your document, write:

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

The first of these lines switches to the next left-hand page. If needed it also adds a completely blank right-hand page. The second line makes sure that the following left-hand page will also be set using the empty page style. The third through sixth lines load an image file named picture and scale it to the desired size without distorting it. This command requires the graphicx package (see [Car17]). The last line starts a new chapter on the next page, which will be a right-hand one.

In two-sided printing, \cleardoubleoddpage always moves to the next left-hand page and \cleardoubleevenpage to the next right-hand page. The style of the interleaf page to be inserted if necessary is defined with the cleardoublepage option.

3.14. Footnotes

Unlike the standard classes, KOMA-Script offers the ability to configure the format of the footnote block.

Table 3.11.: Available values for the `footnotes` option to configure footnotes

`multiple`	Consecutive footnote marks will be separated by `\multfootsep`.
`nomultiple`	Consecutive footnote marks will be handled like single footnotes and not separated from each other.

`footnotes=`*`setting`*
`\multfootsep`

v3.00 Footnotes are marked by default in the text with a small superscript number. If several footnotes appear in succession at the same point, it gives the impression that there is one footnote with a large number rather than multiple footnotes (e. g. footnote 12 instead of footnotes 1 and 2). With `footnotes=multiple`, footnotes that follow each other directly are separated with a delimiter instead. The default delimiter in `\multfootsep` is defined as a comma without a space:

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

This can be redefined.

The whole mechanism is compatible with the footmisc package, version 5.3d to 5.5b (see [Fai11]). It affects footnote markers placed using `\footnote`, as well as those placed directly with `\footnotemark`.

You can switch back to the default `footnotes=nomultiple` at any time using the `\KOMAoptions` or `\KOMAoption` command. However, if you encounter any problems using another package that alters the footnotes, you should not use this option, nor should you change the *`setting`* anywhere inside the document.

A summary of the available *`setting`* values of `footnotes` can be found in table 3.11.

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

Footnotes in KOMA-Script are produced, as they are in the standard classes, with the `\footnote` command, or alternatively the pair of commands `\footnotemark` and `\footnotetext`. As in the standard classes, it is possible for a page break to occur 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 move the footnote to the next page. Unlike the standard classes, KOMA-Script can recognize and separate consecutive footnotes automatically. See the previously documented option `footnotes`.

v3.00

If instead you want to place this delimiter manually, you can do so by calling `\multiplefootnoteseparator`. However, users should not redefine this command, as it contains not only the delimiter but also the delimiter's formatting, for example the font size selection and the superscript. The delimiter itself is stored in the previously described `\multfootsep` command.

Example: Suppose you want to put two footnotes after a single word. First you try

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

Let's assume that the footnotes are numbered 1 and 2. Since the two footnote numbers follow each other directly, it creates the impression that the word has only one footnote numbered 12. You can change this behaviour by using

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

to enable the automatic recognition of footnote sequences. Alternatively, you can use

```
word\footnote{Footnote 1}%
\multiplefootnoteseparator
\footnote{Footnote 2}
```

This should give you the desired result even if automatic detection fails or cannot be used for some reason.

Now suppose you also want the footnote numbers to be separated not just by a comma, but by a comma and a space. In this case, write

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

in the preamble of your document. `\nobreakspace` was used here instead of a normal space to avoid paragraph or page breaks within the sequence of footnotes.

`\footref{`*reference*`}`

v3.00

Sometimes you have a footnote in a document to which there are several references in the text. An inconvenient way to typeset this would be to use `\footnotemark` to set the number directly. The disadvantage of this method is that you need to know the number and manually set every `\footnotemark` command. And if the number changes because you add or remove an earlier footnote, you will have to change each `\footnotemark`. KOMA-Script thefore offers the `\label` mechanism to handle such cases. After placing a `\label` inside the footnote, you can use `\footref` to set all the other marks for this footnote in the text.

Example: You are writing a text in which you must create a footnote each time a brand name occurs, indicating that it is a registered trademark. You can write, for example,

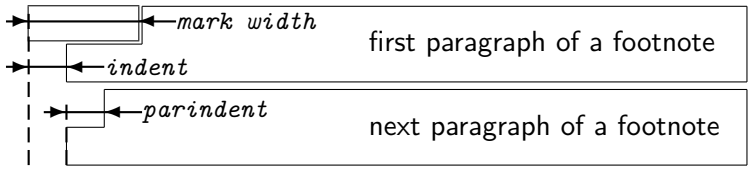


Figure 3.1.: Parameters that control the footnote layout

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

When setting footnote marks with the \label mechanism, any changes to the footnote numbers will require at least two LaTeX runs to ensure correct numbers for all \footref marks.

Note that statements like \ref or \pageref are fragile and therefore you should put \protect in front of them if they appear in moving arguments such as headings.

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

The KOMA-Script classes set footnotes slightly differently than the standard classes do. As in the standard classes, the footnote mark in the text is rendered with small, superscript numbers. The same formatting is used in the footnote itself. The mark in the footnote is typeset right-justified in a box with a width of *mark width*. The first line of the footnote follows directly.

All subsequent 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, the first line of each paragraph is indented by the value of *parindent*.

figure 3.1 shows the different parameters again. The default configuration of the KOMA-Script classes is as follows:

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

\textsuperscript controls both the superscript and the smaller font size. The command \thefootnotemark contains the current footnote mark without any formatting.

v2.8q The footnote, including the footnote mark, uses the font specified in the footnote element. You can change the font of the footnote mark separately using the \setkomafont

and \addtokomafont commands (see section 3.6, page 59) for the footnotelabel element. See also table 3.2, page 60. The default setting is no change to the font. Please don't misuse this element for other purposes, for example to set the footnotes ragged right (see also \raggedfootnote, page 93).

The footnote mark in the text is defined separately from the mark in front of the actual footnote. This is done with \deffootnotemark. The default setting is:

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

v2.8q With this default, the font for the footnotereference element is used (see table 3.2, page 60). Thus, the footnote marks in the text and in the footnote itself are identical. You can change the font with the commands \setkomafont and \addtokomafont (see section 3.6, page 59).

Example: One feature that is often requested is footnote marks which are neither in superscript nor in a smaller font. They should not touch the footnote text but be separated by a small space. You can accomplish this as follows:

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

This will set the footnote mark and subsequent space right-aligned in a box of width 1 em. The lines of the footnote text that follow are also indented by 1 em from the left margin.

Another layout that is often requested is footnote marks that are left-aligned. You can obtain them with the following definition:

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

If, however you want to change the font for all footnotes, for example to sans serif, this can easily be done with the commands \setkomafont and \addtokomafont (see section 3.6, page 59):

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

As the examples show, KOMA-Script allows a wide variety of different footnote formats with this simple user interface.

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

v3.06 Generally, a horizontal rule is set between the text area and the footnote area, but normally this rule does not extend the full width of the type area. With \setfootnoterule, you can set the exact thickness and length of the rule. In this case, the parameters *thickness* and *length* are only evaluated when setting the rule itself. If the optional argument *thickness* has been omitted, the thickness of the rule will not be changed. Empty arguments for *thickness* or

length are also allowed and do not change the corresponding parameters. Using absurd values will result in warning messages both when setting and when using the parameters.

v3.07 You can change the colour of the rule with the `footnoterule` element using the `\setkomafont` and `\addtokomafont` commands (see section 3.6, page 59). The default is no change of font or colour. In order to change the colour, you must also load a colour package like xcolor.

`\raggedfootnote`

v3.23 By default KOMA-Script justifies footnotes just as in the standard classes. But you can also change the justification separately from the rest of the document by redefining `\raggedfootnote`. Valid definitions are `\raggedright`, `\raggedleft`, `\centering`, `\relax` or an empty definition, which is the default. The alignment commands of the ragged2e package are also valid (see [Sch09]).

Example: Suppose you are using footnotes only to provide references to very long links, where line breaks would produce poor results if justified. You can use

```
\let\raggedfootnote\raggedright
```

in your document's preamble to switch to ragged-right footnotes.

scrbook

3.15. Book Structure

Sometimes books are loosely divided into *front matter*, *main matter*, and *back matter*. KOMA-Script also provides this capability for scrbook.

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

The front matter includes all the material which appears before the main text begins, including title pages, preface, and table of contents. It is initiated with `\frontmatter`. In the front matter, Roman numerals are used for the page numbers, and chapter headings in the header are not numbered. However, section headings are numbered consecutively, starting from chapter 0. This typically does not matter, as the front matter is used only for the title pages, table of contents, lists of figures and tables, and a preface or foreword. The preface can thus be created as a normal chapter. A preface should be as short as possible and never divided into sections. The preface thus does not require a deeper level of structure than the chapter.

v2.97e If you see things differently and want 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 56). It is explicitly noted that this creates confusion

with chapter numbers! The use of \addsec and \section* (see section 3.16, page 104 and page 105) are thus, in the author's opinion, greatly preferable.

v2.97e As of version 2.97e the numbering of floating environments, such as tables and figures, and equation numbers in the front matter also contains no chapter-number part. To take effect, this too requires the corresponding compatibility setting (see the version option, section 3.2, page 56).

The part of the book with the main text is introduced with \mainmatter. If there is no front matter, you can omit this command. The default page numbering in the main matter uses Arabic numerals and (re)starts the page count at 1 at the start of the main matter.

The back matter is introduced with \backmatter. Opinions differ as to what belongs in 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 can use the \pagenumbering command in section 3.12, page 84.

3.16. Document Structure

The structure refers to dividing a document into parts, chapters, sections, and additional levels of structure.

open=*method*

scrbook, scrreprt The KOMA-Script classes scrbook and scrreprt give you the choice of where to start a new chapter with two-sided printing. By default scrreprt starts a new chapter on the next page. This is equivalent to *method* any. However, scrbook starts new chapters at the next right-hand page. This is equivalent to *method* right and is usually used in books. But sometimes chapters should start on the left-hand page of a two-page spread. You can accomplish this with the *method* left. You can find a summary of the available values in table 3.12. The table also describes the effects of \cleardoublepage, \cleardoublepageusingstyle, \cleardoublestandardpage, \cleardoubleplainpage, and \cleardoubleemptypage (see section 3.13, page 87).

v3.00

Since LaTeX does not differentiate between left-hand and right-hand pages in one-sided printing, the option has no effect in that case.

In the scrartcl class, the section is the first structural element below the part. For this reason, scrartcl does not support this option.

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

any	Parts, chapter, index, and back matter use \clearpage, not \cleardoublepage; \cleardoublepageusingstyle, \cleardoublestandardpage, \cleardoubleplainpage, \cleardoubleemptypage, and \cleardoublepage behave the same as using open=right.
left	Part, chapter, index, and back matter use \cleardoublepage; the 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	Part, chapter, index, and back matter use \cleardoublepage; the 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.

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

scrbook, scrreprt

With the standard classes book and report, a chapter heading consists of a line with the word "Chapter"[1] followed by the chapter number. The heading itself is set left-justified on the following line. The same effect is obtained in KOMA-Script with the chapterprefix option. You can use any value from table 2.5, page 42 as the *simple switch*. The default, however, is chapterprefix=false, the opposite behaviour of that of the standard classes, which corresponds to chapterprefix=true. These options also affect the automatic running heads in the headers (see section 3.12, page 80).

Sometimes you may want to use the simplified chapter headings produced by chapterprefix=false but at the same time to have the heading of an appendix preceded by a line with "Appendix" followed by the appendix letter. This is achieved by using the appendixprefix option (see table 2.5, page 42). Since this results in an inconsistent document layout, I advise against using this option. Ultimately, this option changes the chapterprefix option automatically at the beginning of the \appendix.

v3.18

You can execute code depending on the current setting for the chapter preceding line us-

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

ing `\IfChapterUsesPrefixLine`. If `chapterprefix` is currently active, the *`then code`* is executed; otherwise, the *`else code`* is executed.

You can change the font style of the chapter number line that uses `chapterprefix=true` or `appendixprefix=true` by using the `chapterprefix` element with the `\setkomafont` and `\addtokomafont` commands (see section 3.6, page 59). The default is to use the `chapter` element (see page 100, as well as table 3.15, page 103).

v2.96a

You can find additional settings for chapter headings in the explanation for `\RedeclareSectionCommand` and the commands `\chapterlineswithprefixformat` and `\chapterlinesformat` in section 21.8, part II.

`headings=`*`setting`*

Headings of sectioning levels normally use a relatively large font size in both the standard classes and KOMA-Script. This choice does not appeal to everyone and is especially problematic for small paper sizes. Consequently, KOMA-Script provides, in addition to the large headings defined by the `headings=big` option, the options `headings=normal` and `headings=small`, which allow for smaller headings. The font sizes resulting from these options for scrbook and scrreprt are shown in table 3.15, page 103. Specifically, all three settings reset the elements `chapter`, `section`, `subsection`, `subsubsection`, `paragraph`, and `subparagraph` to the corresponding defaults. scrartcl generally uses slightly smaller headings. The spacing before and after chapter headings is also reset by these options.

v3.00

scrbook, scrreprt

Chapter headings also have the two options `headings=twolinechapter` and `headings=onelinechapter`, which correspond to `chapterprefix=true` and `chapterprefix=false` explained above. For the appendix, `appendixprefix=true` and `appendixprefix=false` serve as alternatives for the `headings=twolineappendix` and `headings=onelineappendix` options. Of course, these options do not exist with scrartcl.

scrbook, scrreprt

The `headings=standardclasses` option adjusts the font sizes of the headings to those of the standard classes. In addition, the font for the `disposition` element is set to `\bfseries`. It therefore no longer uses a sans-serif font for the headings. If you use scrbook or scrreprt, `headings=twolinechapter` is also set and the spacing between the chapter headings is adjusted to that of the standard classes.

v3.12

scrbook, scrreprt

You can set the method to specify the page on which new chapters begin with `headings=openany`, `headings=openright`, and `headings=openleft`, or alternatively with the `open` option, which takes the values `any`, `right`, and `left` (see above).

Another special feature of KOMA-Script is the handling of the optional argument of the sectioning commands `\part`, `\chapter`, etc. down to `\subparagraph`. You can change its function and meaning with the options `headings=optiontohead`, `headings=optiontotoc` and `headings=optiontoheadandtoc`.

v3.10

See table 3.13 for a summary of all available settings for the `headings` option. The explanations of the sectioning commands below contain examples using some of these settings.

Table 3.13.: Available values for the `headings` option to format section headings

`big`
: Use large fonts in the headings for each of the default sectioning levels with wide spacing above and below the titles.

`normal`
: Use medium-sized fonts in the headings with medium spacing above and below the titles.

`onelineappendix`, `noappendixprefix`, `appendixwithoutprefix`, `appendixwithoutprefixline`
: Chapter headings in the appendix are set like other headings.

`onelinechapter`, `nochapterprefix`, `chapterwithoutprefix`, `chapterwithoutprefixline`
: Chapter titles are set like other headings.

`openany`
: The commands \cleardoublepageusingstyle, \cleardoublestandardpage, \cleardoubleplainpage, \cleardoubleemptypage, and \cleardoublepage generate a page break and insert an interleaf page if needed to reach the next right-hand page in two-sided printing, the same as in `headings=openright`. Parts, chapter, the index, and back matter use \clearpage instead of \cleardoublepage.

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

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

...

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

`optiontohead`

v3.10 The advanced functionality of the optional argument of the sectioning commands \part down to \subparagraph is activated. By default, the optional argument is used only for the running head.

`optiontoheadandtoc`, `optiontotocandhead`

v3.10 The advanced functionality of the optional argument of the sectioning commands \part down to \subparagraph is activated. By default, the optional argument is used for the running head and the table of contents.

`optiontotoc`

v3.10 The advanced functionality of the optional argument of the sectioning commands \part down to \subparagraph is activated. By default, the optional argument is used only for the table of contents.

`small`

Use small fonts in the headings with small spacing above and below the titles.

`standardclasses`

v3.12 Reset the font settings for each of the standard sectioning levels and use headings with the sizes of the standard classes. For chapter headings, scrbook und scrreprt set `headings=twolinechapter`.

`twolineappendix`, `appendixprefix`, `appendixwithprefix`, `appendixwithprefixline`

Chapter titles in the appendix are set with a number line whose format is determined by \chapterformat.

`twolinechapter`, `chapterprefix`, `chapterwithprefix`, `chapterwithprefixline`

Chapter titles are set with a number line whose format is determined by \chapterformat.

`numbers=`*`setting`*

According to DUDEN, if only Arabic numerals are used to number section headings, the German practice is to have no point at the end (see [DUD96, R 3]). On the other hand, if Roman numerals or letters appear in the numbering, then a point should appear at the end of the numbering (see [DUD96, R 4]). KOMA-Script has a mechanism that tries to automate this somewhat complex rule. The result is that normally after the sectioning commands \part and \appendix the numbering switches to using a final point. This information is saved in the `aux` file and takes effect on the next LaTeX run.

Table 3.14.: Available values of the `numbers` option to configure the final points of the numbers for section headings

`autoendperiod`, `autoenddot`, `auto`	KOMA-Script decides whether to set the point at the end of sectioning command numbers. If there are only Arabic numerals, the point will be omitted. If a letter or Roman numeral is found, the point is set. However, references to these numbers are set without a final point.
`endperiod`, `withendperiod`, `periodatend`, `enddot`, `withenddot`, `dotatend`	All numbers of sectioning commands and their subordinate numbers are set with a final point. Only references will be set without the final point.
`noendperiod`, `noperiodatend`, `noenddot`, `nodotatend`	All numbers of sectioning commands are set without a final point.

Sometimes the mechanism for placing or omitting the final point may fail. Sometimes other languages have different rules. Therefore you can force the use of the final point with the option `numbers=endperiod` or to prohibit it with `numbers=noendperiod`.

Note that this mechanism only takes effect on the next LaTeX run. Therefore, before you try to use these options to force the correct numbering format, you should always perform another LaTeX run without modifying the document.

You can find a summary of the available values for the *setting* of *numbers* in table 3.14. Unlike most other settings, this option can only be set in the document preamble, i. e. before `\begin{document}`.

`chapteratlists`
`chapteratlists=`*value*

scrbook, scrreprt

v2.96a

As mentioned in section 3.20, page 141, every chapter that is created with `\chapter` normally inserts a vertical space in the lists of floating environments (e. g., tables and figures). Since version 2.96a, this also applies to the `\addchap` command unless you choose a compatibility setting for an earlier version (see the `version` option in section 3.2, page 56).

In addition, you can use the `chapteratlists` option to change the vertical spacing by specifying the desired distance as the *value*. The default with `listof=chaptergapsmall` is 10 pt (see the`version` option in section 3.2, page 56).

If you use `chapteratlists=entry` or `chapteratlists` without specifying a value, instead of a vertical space, the chapter entry itself will be entered into the lists. Note that such an entry occurs even if the chapter does not contain a floating environment. A method where only chapters with floating environments are displayed in the respective list can be found in the German-language KOMA-Script forum at [Koh15].

Please note that changes to this option only takes effect after two additional 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 the same way as those in the standard classes. Thus, you can specify an alternative text for the table of contents and running heads as an optional argument to the sectioning commands.

v3.10 However, with the `headings=optiontohead` option, KOMA-Script only uses the optional argument *short version* in the running head, not the table of contents. Of course, this text will only appear if you use a page style that puts the corresponding sectioning level in the running head. See section 3.12 and chapter 5. With the `headings=optiontotoc` option, KOMA-Script uses the optional argument *short version* exclusively for the table of contents and not the running head. However, the entry will be shown only if the `tocdepth` counter is great enough (see section 3.9, page 76). With the `headings=optiontoheadandtoc` option, KOMA-Script uses the optional argument *short version* in both the table of contents and the running head. These three options all activate the extended interpretation of the optional argument *short version*, which is not active by default.

v3.10 The extended interpretation of the optional argument checks to see if there is an equals sign in *short version*. If so, the optional argument will be interpreted as an *option list*. Three options — `head=`*running head*, `tocentry=`*table of contents entry*, `reference=`*reference title*, and `nonumber=`*simple switch* — are supported with this format. To use commas or equals signs within the values of those options, you must enclose them in braces.

v3.22

Please note that this mechanism only works as long as KOMA-Script controls the sectioning commands. If you use package that redefines the KOMA-Script's or the internal LaTeX kernel's sectioning commands, KOMA-Script can no longer provide this extended mechanism. This also applies to a KOMA-Script extension that is always active: sectioning commands with no heading text do not create entries in the table of contents. If you really want an entry with empty heading text, you can use an invisible entry like `\mbox{}`.

Example: Suppose you have a document with very long chapter headings. These headings should appear in the table of contents, but you want to limit the running head to short, single-line headings. You can do this with the optional argument of `\chapter`.

```
\chapter[short version of chapter heading]
        {The Sectioning Command for Chapters
          Supports not only the Heading Text
          Itself but also a Short Version Whose
          Use can be Controlled}
```

A little later you realize that the line breaks for this long heading are very inappropriate. You therefore want to choose the breaks yourself. Nevertheless, you still want automatic line breaking in the table of contents. With

```
\chapter[head={short version of chapter heading},
         tocentry={The Sectioning
           Command for Chapters Supports not
           only the Heading Text Itself but
           also a Short Version Whose Use
           can be Controlled}]
        {The Sectioning\\
             Command for Chapters\\
          Supports not only\\
          the Heading Text Itself\\
          but also\\
          a Short Version Whose\\
          Use can be Controlled}
```

you create separate entries for the table of contents, running head, and chapter heading itself. The arguments of the options `head` and `tocentry` have been enclosed in braces so their contents can be arbitrary.

The need for braces in the example above is best illustrated by another example. Suppose you chose the `headings=optiontotoc` option and set the title this way:

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

This results in the entry "Option head=*value*" in the table of contents but "*value*" in the running head. But surely you wanted the entry "head=*value*" in the table of contents and the complete heading text in the running head. You can do this using braces:

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

A similar case concerns the comma. Using the same `headings` option as before

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

results in an error because the comma is interpreted as the separator between the individual options of the option list "`0, 1, 2, 3, \dots`". But writing

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

makes "`0, 1, 2, 3, \dots`" the argument of the `head` option.

v3.22 Like setting the title of a running head with the `head` option, or setting the title of a table-of-contents entry with the `tocentry` option, you can set the title of a reference with

the nameref or titleref packages, or with the titleref module of the zref package, using the `reference` option. Note that the support for the titleref package is rather rudimentary, since that package's performance is poor compared to the other two, and it is not fully compatible with hyperref.

v3.27 You can deactivate the numbering using `nonumber=true` in the extended optional argument. In contrast to the starred versions of the sectioning commands explained below, the titles will still be added to the table of contents and, if applicable, used for the running head. For `\part`, `\chapter`, and `\section`, using `nonumber=true` this essentially corresponds to the `\addpart`, `\addchap`, and `\addsec` commands, which are explained on page 105.

The part-level title (`\part`) differs from other sectioning levels by being numbered independently. This means that the chapter level (in scrbook or scrreprt), or the section level (in scrartcl) is numbered consecutively over all parts. Furthermore, for the scrbook and scrreprt classes, the title of the part level together with the corresponding preamble (see `\setpartpreamble`, page 113) is set on a separate page.

scrbook, scrreprt

scrbook, scrreprt

The `\chapter` command only exists in the book and report classes, that is, in book, scrbook, report and scrreport, but not in the article classes article and scrartcl. Furthermore, the `\chapter` command in KOMA-Script differs substantially from the version in the standard classes. 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. KOMA-Script replaces this overpowering style with a simple chapter number before the chapter title, but you can restore the original behaviour with the `chapterprefix` option (see page 95).

scrbook, scrreprt

Please note that `\part` and `\chapter` in the scrbook and scrreprt classes change the page style for one page. The page style applied in KOMA-Script is defined in the macros `\partpagestyle` and `\chapterpagestyle` (see section 3.12, page 83).

v2.8p You can change the font style for all headings with the `\setkomafont` and `\addtokomafont` commands (see section 3.6, page 59). In doing so, the element `disposition` is applied first, followed by the specific element for each sectioning level (see table 3.2, page 60). There is a separate elements, `partnumber`, for the number of the part heading, and `chapterprefix`, for the optional prefix line of chapter headings. The initial definition for the `disposition` element is `\normalcolor\sffamily\bfseries`. The default font sizes for the specific elements depends on the options `headings=big`, `headings=normal`, and `headings=small` (see page 96). They are listed in table 3.15.

Example: Suppose you use the `headings=big` class option and notice that the very large headings of the 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

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

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

"Appendices". A much more convenient 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:

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

The second version is preferred because it gives you the correct result even if you

change the disposition element, for instance:

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

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

I strongly advise against using the ability to switch fonts in order to mix fonts, font sizes, and font attributes wildly. Picking the right font for the job is a matter for experts and has almost nothing to do with the personal tastes of non-experts. See the citation at the end of section 2.8, page 53 and to the following explanation.

It is possible to use different font types for different sectioning levels in KOMA-Script. Non-experts in typography should absolutely avoid doing so for excellent typographical reasons.

A rule of typography states that you should mix as few fonts as possible. Using sans serif for headings already seems to violate this rule. However, you should realize that large, bold, serif letters are much too heavy for headings. Strictly speaking, you should then use a normal instead of a bold or semi-bold font. However, in deeper levels of the sectioning, a normal font may then appear too light. On the other hand, sans serif fonts have a very pleasant appearance in headings, and almost solely in headings. There is, therefore, good reason why sans serif is the default in KOMA-Script.

Greater variety should, however, be avoided. Font mixing is something for professionals. For this reason, if you want to use fonts other than the standard TEX fonts — regardless of whether these are CM, EC, or LM fonts — you should consult an expert about the compatibility of the sans serif and serif fonts, or redefine the element disposition as a precautionary measure. The author considers the commonly encountered combinations of Times and Helvetica or Palatino with Helvetica to be awkward.

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

The starred variants of all sectioning commands produce unnumbered headings which do not appear in the table of contents or in the running head. The absence of a running head often has an unwanted side effect. If, for example, a chapter set using \chapter* spans several pages, then the running head of the previous chapter suddenly reappears. KOMA-Script offers a solution for this problem, described below. \chapter* only exists in book and report classes, that is, book, scrbook, report and scrreport, not in the article classes article and scrartcl.

scrbook, scrreprt

Please note that \part and \chapter change the page style for one page. While the standard classes use the plain page style, KOMA-Script applies the style defined in the \partpagestyle

and \chapterpagestyle macros (see section 3.12, page 83).

v2.8p The possibilities for switching fonts described above for the unstarred variants apply without change. The elements use the same names since they do not indicate variants but structuring levels.

```
\addpart[short version]{heading}
\addpart*{heading}
\addchap[short version]{heading}
\addchap*{heading}
\addsec[short version]{heading}
\addsec*{heading}
```

In addition to the commands of the standard classes, KOMA-Script offers the new commands \addpart, \addsec and \addchap. They are similar to the standard commands \part, \chapter and \section except that they are unnumbered. They thus produce both a running head and an entry in the table of contents which take into account the headings option settings, especially the handling of the optional argument. However, enabling or disabling the nonumber switch will have no effect.

The starred variants \addchap* and \addsec* are similar to the standard commands \chapter* and \section* except for a small but important difference: the running heads are deleted. This eliminates the side effect of obsolete headers mentioned above. Instead, the running heads on subsequent pages remain empty. \addchap and \addchap* only exist, of course, in book and report classes, namely book, scrbook, report and scrreport, not in the article classes article and scrartcl.

book, scrreprt

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

Please note that \addpart and \addchap and their starred variants change the page style for one page. The particular page style is defined in the macros \partpagestyle and \chapterpagestyle (see section 3.12, page 83).

v2.8p The possibilities for switching fonts described above for the unstarred variant of the sectioning commands apply without change. The elements have the same names since they do not designate variants but sectioning levels.

```
\minisec{heading}
```

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

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

v2.96a

You can change the font of the \minisec command using the disposition and minisec element (see table 3.2, page 60). The default of the minisec element is empty, so by default only the disposition element is used.

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:

```
\documentclass{scrartcl}

\begin{document}

    \title{DIY Projects}
    \author{Two Left Thumbs}
    \date{\today}
    \maketitle

    \section{Mousetrap}

    The first project is suitable for beginners and only requires
    a few components that should be found in every household.

  \minisec{Material Required}

  \begin{flushleft}
    1 board ($100\times 50 \times 12$)\\
    1 swing-top cap of a beer-bottle\\
    1 ballpoint pen\\
    1 push pin\\
    2 screws\\
    1 hammer\\
    1 knife
  \end{flushleft}

  \minisec{Assembly}
  First, find the mouse hole and put the push pin directly behind
  the hole so that the mouse cannot escape during the following
  actions.

  Next tap the swing-top cap into the mouse hole with the hammer.
  If the cap is not big enough to block the hole completely and
  permanently, take the board instead and screw it to the front
  of the mouse hole using the two screws and the knife. Of
  course, you can use a screwdriver instead of a knife. The
   ballpoint pen has fallen victim to animal welfare.
\end{document}
```

The main part, starting with the heading “Material Required” will look like this:

Material Required
1 board ($100 \times 50 \times 12$)
1 swing-top cap of a beer-bottle
1 ballpoint pen
1 push pin
2 screws
1 hammer
1 knife

Assembly
First, find the mouse hole and put the push pin directly behind the hole so that the mouse cannot escape during the following actions.

Next tap the swing-top cap into the mouse hole with the hammer. If the cap is not big enough to block the hole completely and permanently, take the board instead and screw it to the front of the mouse hole using the two screws and the knife. Of course, you can use a screwdriver instead of a knife. The ballpoint pen has fallen victim to animal welfare.

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

In the standard classes, headings are set as justified text. That means that hyphenated words can occur and multi-line headings are stretched up to the text width. This approach is rather uncommon in typography. KOMA-Script therefore sets the headings left aligned with hanging indentation using `\raggedsection` with the default:

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

You can redefine this command 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 heading formatting which is very close to that of the standard classes. It will become even closer when you combine this change with the change to the `disposition` element mentioned above.

v3.15 Because some users want a different alignment for the `\chapter` level than for the other sectioning levels, you can change the `\chapter` justification separately by redefining `\raggedchapter`. By default, however, this command simply uses `\raggedsection`, so changing `\raggedsection` indirectly affects `\raggedchapter`.

By default, part headings (`\part`) are set horizontally centred rather than ragged right. This formatting is performed by the `\raggedpart` statement, which has the default definition

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

You can also redefine this command using `\renewcommand`.

Example: You want the headings for `\part` to be formatted the same as any other sectioning command. To do so, put

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

in the preamble of your document. In this case, and unlike in the example above, we did not use `\let` because `\let` would set `\raggedpart` to the underlying value of `\raggedsection`. Subsequent changes to `\raggedsection` would then not change the behaviour of `\raggedpart`. By redefining with `\renewcommand`, `\raggedpart` will use the current meaning of `\raggedsection` at the time it is used rather than when it was redefined.

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

KOMA-Script adds another logical layer above `\the`*`sectioning name`* to format the sectioning numbers. The counters for each heading are not merely output. They are formatted using the commands `\partformat`, `\chapterformat`, down to `\subparagraphformat`. Of course the `\chapterformat` command, like `\thechapter`, does not exist in the scrartcl class, but only in the scrbook and scrreprt classes.

v3.17

scrbook, scrreprt

As already explained for the `numbers` option at the beginning of this section (see page 98), KOMA-Script's handling of points in section numbers implements the rules given in [DUD96], which are standard in German-language typography, in the `\autodot` command. In all levels except for `\part`, a point is followed by a further `\enskip`. This corresponds to a horizontal skip of 0.5 em.

v3.17

Since KOMA-Script 3.17, the command `\othersectionlevelsformat` is used only in rare circumstances, i. e., if the corresponding format command to a section command does not exist or is `\relax`. This should not happen for any section commands defined by KOMA-Script itself. Therefore the command is no longer officially documented. Nevertheless, if you select a compatibility level prior to 3.17 (see option `version`, section 3.2, page 56), commands `\sectionformat` down to `\subparagraphformat` are ignored by KOMA-Script. Instead, these layers will continue to use `\othersectionlevelsformat`.

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

```
\newcommand*{\partformat}{\partname~\thepart\autodot}
\newcommand*{\chapterformat}{%
  \mbox{\chapappifchapterprefix{\nobreakspace}\thechapter
    \autodot\IfUsePrefixLine{}{\enskip}}}
\newcommand*{\sectionformat}{\thesection\autodot\enskip}
\newcommand*{\subsectionformat}{%
  \thesubsection\autodot\enskip}
\newcommand*{\subsubsectionformat}{%
  \thesubsubsection\autodot\enskip}
\newcommand*{\paragraphformat}{\theparagraph\autodot\enskip}
\newcommand*{\subparagraphformat}{%
  \thesubparagraph\autodot\enskip}
\newcommand*{\othersectionlevelsformat}[3]{%
  #3\autodot\enskip}
```

v3.17 Because it uses \IfUsePrefixLine, \chapterformat should not be used outside of \chapter. \IfUsePrefixLine is only valid inside KOMA-Script sectioning commands. Within those commands, it executes the *then code* if a prefix line for the number is used and the *else code* otherwise.

Please also remember to replace \newcommand with \renewcommand if you want to redefine one of the commands above.

Example: Suppose you do not want the word "Part" written in front of the part number when you use \part. You can put the following command in the preamble of your document:

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

In fact, you could do without \autodot here and insert a fixed point instead. Since \part is numbered with Roman numerals, it must be followed by a point according to [DUD96]. However, by using \autodot you retain the ability to use the numbers option numbers=endperiod and thus deviate from the rule. You can find more details concerning class options on page 98.

Another possibility is to place the section numbers in the left margin in such a way that the heading text is left aligned with the surrounding text. You can accomplish this with:

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

```
    \thesubsubsection\autodot\enskip}}
\renewcommand*{\paragraphformat}{%
  \makebox[0pt][r]{\theparagraph\autodot\enskip}}
\renewcommand*{\paragraphformat}{%
  \makebox[0pt][r]{%
    \thesubparagraph\autodot\enskip}}
```

The optional arguments of the \makebox command require LaTeX to create a zero-width box with right-justified content. As a result, the contents of the box are output to the left of the current position. You can find more about the optional arguments of \makebox in [Tea05b].

For formatting changes in the headings that go beyond merely formatting the unit number, please refer to \partlineswithprefixformat, \chapterlineswithprefixformat and \chapterlinesformat, as well as \sectionlinesformat and its \sectioncatchphraseformat format in section 21.8, starting from page 491.

\chapappifchapterprefix{*additional text*}
\chapapp

v2.8o
scrbook, scrreprt

These two commands are used internally by KOMA-Script and also made available to the user. Later, you will see how to use them, for example to redefine other commands.

If you use the layout option chapterprefix=true (see page 95), \chapappifchapterprefix outputs the word "Chapter" in the body of the text in the current language, followed by *additional text*. In the appendix, the word "Appendix" in the current language is output instead, followed by *additional text*. If the option maincls=chapterprefixfalse is set, then nothing is output.

The \chapapp command always outputs the word "Chapter" or "Appendix", regardless of the setting of the chapterprefix option.

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

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

As mentioned in section 3.12, the `headings` page style works with automatic running heads. For this, the commands `\chaptermark` and `\sectionmark`, or `\sectionmark` and `\subsectionmark`, are defined accordingly. Every sectioning command (`\chapter`, `\section`, etc.) automatically executes the corresponding `\...mark` command. The parameter passed contains the text of the section heading. The corresponding section number is added automatically in the `\...mark` command. The formatting is done according to the sectioning level with one of the three commands `\chaptermarkformat`, `\sectionmarkformat`, or `\subsectionmarkformat`.

Note that the running heads of `\addchap` and `\addsec` are also based on `\chaptermark` and `\sectionmark`. However, the `secnumdepth` counter is set locally to a value that switches off the numbering of chapters or sections. You should consider this, for example, if you redefine `\chaptermark` or `\sectionmark` (see `\Ifnumbered` on page 112). The starred variants `\addchap*` and `\addsec*` use additional commands `\addchapmark` and `\addsecmark` that are also defined based on `\chaptermark` and `\sectionmark` with local changes of `secnumdepth`.

scrartcl

Although there is no `\chaptermark` or `\chaptermarkformat` command in scrartcl, there are two commands, `\subsectionmark` and `\subsectionmarkformat`, which exist only in scrartcl. However using the scrlayer-scrpage package changes this (see chapter 5).

Just as numbers in the sectioning-command headers are formatted with `\partformat` down to `\subparagraphformat`, `\chaptermarkformat`, `\sectionmarkformat`, and `\subsectionmarkformat` 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 the word “Chapter” to precede the chapter number in the running head. You could put the following definition in the preamble of your document:

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

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

```
secnumdepth
\partnumdepth
\chapternumdepth
\sectionnumdepth
\subsectionnumdepth
\subsubsectionnumdepth
\paragraphnumdepth
\subparagraphnumdepth
```

Normally, the scrbook and scrreport classes number the section headings from \part down to \subsection and the scrartcl class numbers them from \part down to \subsubsection. The actual depth to which headings are numbered is controlled by the LaTeX counter secnumdepth.

v3.12 To save users the trouble of having to remember abstract numbers, the commands \partnumdepth to \subparagraphnumdepth return the appropriate number for the section level in their name.

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

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

No provision is made for redefining these commands. Doing so could lead to unexpected results, not only with KOMA-Script but also with third party packages. Thus, you should never redefine them.

Do not confuse the secnumdepth and tocdepth counters (see section 3.9, page 76). Depending on the class you are using, the meaning of the values of the secnumdepth and tocdepth counters may differ for the same section level.

```
\Ifnumbered{section level}{then code}{else code}
\Ifunnumbered{section level}{then code}{else code}
```

v3.28 The commands \Ifnumbered and \Ifunnumbered determine which section-level headings are numbered, using the technique described above, and execute code depending on whether a *section level* is numbered or not. If a *section level* is numbered with the current settings, \Ifnumbered executes the *then code*. If the section level is unnumbered, the *else code* is executed. The \Ifunnumbered command behaves in exactly the opposite manner, executing the *then code* if the current level is unnumbered and the *else code* if it is. The *section level* parameter is simply the LaTeX name of a section like

part, chapter, section, subsection, subsubsection, paragraph oder subparagraph.part, chapter, section, subsection, subsubsection, paragraph, or subparagraph.

KOMA-Script itself uses these tests, for example, in the definition of \chaptermark in the headings page style. This indirectly ensures that headings inserted by \addchap do not set a number inside the running head (see also \addchapmark, page 111).

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

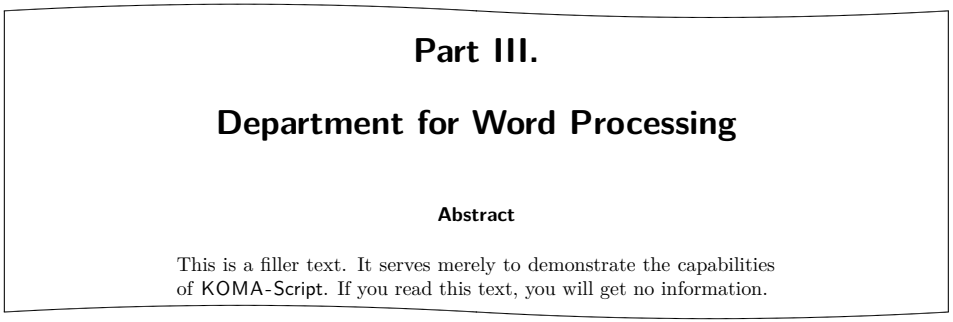
scrbook, scrreprt

Parts and chapters in KOMA-Script can be given a *preamble*. This is particularly useful when you are using a two-column format with the class option twocolumn, since the heading and the *preamble* are always set in a one-column layout. The *preamble* can contain more than one paragraph. The command to set the *preamble* must come 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. Each 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 96) and the options for the abstract environment (see section 3.8, page 71), the result will look something like this:



Please note that *you* are responsible for the spacing between the heading, preamble, and the following text. Note also that there is no abstract environment in the scrbook class (see section 3.8, page 71).

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 vertical placement there are two possibilities at present:

- `o` – above the heading
- `u` – below the heading

You can therefore put one preamble above and another below a heading. For horizontal placement you have three options:

- `l` – left-aligned
- `r` – right-aligned
- `c` – centred

This setting does not affect the alignment of the text in the *preamble*. Instead, it aligns the box that contains the preamble. The width of this box is determined by the second optional argument, *width*. If you omit this second argument, the box uses the full text width. In that case, the option for horizontal positioning has no effect. You can combine exactly one letter from the vertical with one letter from the horizontal positioning.

A typical use for `\setchapterpreamble` would be something like an epigraph, a wise saying, or a dictum. The `\dictum` command, which you can use for this purpose, is described in the next section. You will also find an example there.

Please note that a *preamble* placed above the heading is set within the existing vertical space above the heading. The heading will not be moved down. You are therefore responsible for ensuring that the *preamble* is not too large and that the space above the heading is sufficient. See also the `beforeskip` setting for `\RedeclareSectionCommand` in section 21.8, table 21.3, page 482.

3.17. Dicta

A common element in a document is an epigraph or quotation that is set above or below a chapter or section heading, along with a reference to the source and its own formatting. KOMA-Script refers to such an epigraph as a *dictum*.

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

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

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

You can set such a saying with the help of the \dictum command. This macro can be included in the mandatory argument of either the \setchapterpreamble or the \setpartpreamble command. However, this is not obligatory.

The dictum, along with an optional *author*, is inserted in a \parbox (see [Tea05b]) of width \dictumwidth. However, \dictumwidth is not a length which can be set with \setlength. It is a macro that can be redefined using \renewcommand. The default is 0.3333\textwidth, which is one third of the text width. The box itself is aligned with the command \raggeddictum. The default is \raggedleft, that is, right justified. \raggeddictum can be redefined with the help of \renewcommand.

You can align the *dictum* within the box using \raggeddictumtext. The default is \raggedright, that is, left justified. You can also redefine this macro with \renewcommand.The output uses the default font setting for the element dictum, which can be changed with the commands \setkomafont and \addtokomafont (see section 3.6, page 59). Default settings are listed in table 3.16.

If an *author* is defined, it is separated from the *dictum* by a horizontal rule spanning the full width of the \parbox. This rule is defined in \dictumrule as a vertical object with

v3.10

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

The \raggeddictumauthor command defines the alignment for the rule and the *author*. The default is \raggedleft. This command can also be redefined using \renewcommand. The format is defined with \dictumauthorformat. This macro expects the *author* text as its argument. By default \dictumauthorformat is defined as

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

Thus the *author* is set enclosed in rounded parentheses. For the dictumauthor element, you can define a different font than that used for the dictum element. The default settings are listed in table 3.16. Changes can be made using the \setkomafont and \addtokomafont commands (see section 3.6, page 59).

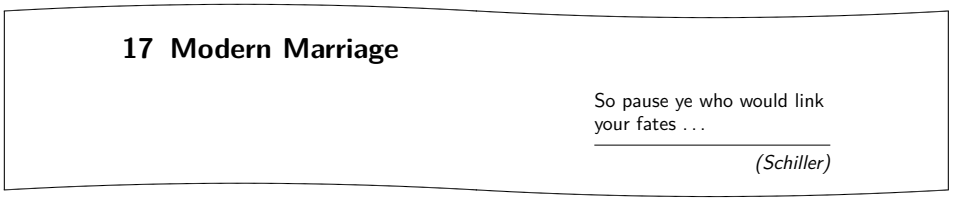
If you use \dictum within the \setchapterpreamble or \setpartpreamble macro, \textwidth is not the width of the whole text body but the current text width. If \dictumwidth is set to .5\textwidth and \setchapterpreamble has an optional width of .5\textwidth too, you will get a box with a width one quarter of the text width. The horizontal positioning of the dictum inside the box is always done with \raggeddictum. The optional argument for horizontal positioning which is implemented for these two commands has no effect to the \text. If you use \dictum you should 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, which is easily accomplished using the \bigskip command.

Example: You are writing a chapter about modern marriage, and you want to place a dictum in the preamble to the chapter heading. You write:

```
\setchapterpreamble[u]{%
  \dictum[Schiller]{So pause ye who would link your fates~\dots}}
\chapter{Modern Marriage}
```

The output would look as follows:



If you want the dictum to span only a quarter of the text width rather than a third, you can redefine \dictumwidth this way:

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

3.18. Lists

Both LaTeX and the standard classes offer different environments for lists. Naturally, KOMA-Script also offers all these environments, though slightly modified or extended in some cases. In general, all lists — even those of different kinds — can be nested up to four levels deep. From a typographical view, anything more would make no sense, as lists of more than three levels cannot easily be apprehended. In such cases, I recommend that you split a large list into several smaller ones.

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

The simplest form of a list is the itemized list, `itemize`. Depending on the level, KOMA-Script classes use the following marks: "•", "–", "∗", and "·". The definition of these symbols is specified in the macros `\labelitemi`, `\labelitemii`, `\labelitemiii`, and `\labelitemiv`, all of which you can redefine 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 in the game}
\begin{itemize}
  \item aeroplanes
  \begin{itemize}
    \item biplane
    \item transport planes
    \begin{itemize}
      \item single-engine
      \begin{itemize}
        \item jet propelled
        \item propeller driven
      \end{itemize}
      \item twin-engine
      \begin{itemize}
            \item jet propelled
                \item propeller driven
          \end{itemize}
    \end{itemize}
    \item helicopters
  \end{itemize}
  \item motorcycles
  \item automobiles
  \begin{itemize}
    \item racing cars
    \item passenger cars
    \item lorries
  \end{itemize}
  \item bicycles
\end{itemize}
```

As output you get:

Vehicles in the game

- aeroplanes
  - biplanes
  - transport planes
    - single-engine
      - jet-propelled
      - propeller-driven
    - twin-engine
      - jet propelled
      - propeller driven
  - helicopters
- motorcycles
- automobiles
  - racing cars
  - passenger cars
  - lorries
- bicycles

```
\begin{enumerate}
  \item ...
  ⋮
\end{enumerate}
\theenumi
\theenumii
\theenumiii
\theenumiv
\labelenumi
\labelenumii
\labelenumiii
\labelenumiv
```

The numbered list is also very common and already provided by the LaTeX kernel. The numbering differs according to the level, with Arabic numbers, small letters, small Roman numerals, and capital letters, respectively. The style 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 right parenthesis, the values of all other levels are followed by a dot. Every item is introduced with `\item`.

Example: Let's shorten the previous example, using an `itemize` environment instead of the `enumerate` environment:

Vehicles in the game

1. aeroplanes
   a) biplanes
   b) transport planes
      i. single-engine
         A. jet-propelled
         B. propeller-driven
      ii. twin-engine
2. motorcycles
   a) historically accurate
   b) futuristic, not real

Within the list, you can set labels in the normal way with `\label` and then reference then with `\ref`. In the example above, a label was set after the jet-propelled, single-engine transport planes with "`\label{xmp:jets}`". The `\ref` value is then "1(b)iA".

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

v2.8p

Another list form is the description list. It primarily serves to describe individual items or keywords. The item itself is specified as an optional parameter in `\item`. The font used to format the keyword can be changed with the `\setkomafont` and `\addtokomafont` commands (see section 3.6, page 59) for the `descriptionlabel` element (see table 3.2, page 60). The default is `\sffamily\bfseries`.

Example: You want the keywords to be printed bold and in the normal font instead of bold and sans serif. 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 (abbreviated) source is:

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

This 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}
```

Another form of description list is only available in the KOMA-Script classes: the `labeling` environment. Unlike the `description` described above, you can specify a pattern for `labeling` whose length determines the indentation of all items. Furthermore, you can put an optional *delimiter* between the item and its description. The font used to format the item and the separator can be changed with the `\setkomafont` and `\addtokomafont` commands (see section 3.6, page 59) for the element `labelinglabel` and `labelingseparator` (see table 3.2, page 60).

v3.02

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

```
\setkomafont{labelinglabel}{\ttfamily}
\setkomafont{labelingseparator}{\normalfont}
\begin{labeling}[~--]{myheadings}
  \item[empty]
    Page style without header or 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}
```

The result is this:

`empty` – Page style without header or footer

`plain` – Page style for chapter beginnings without headings

`headings` – Page style for running headings

`myheadings` – Page style for manual headings

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

Originally, this environment was implemented for things like "Premise, Evidence, Proof", or "Given, Find, Solution" that are often used in lecture handouts. These days, however, the environment has very different applications. For example, the environment for examples in this guide was defined with the `labeling` environment.

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

The `verse` environment is not normally perceived as a list environment because you do not work with `\item` commands. Instead, fixed line breaks are used within the `flushleft` environment. Internally, however, both the standard classes as well as KOMA-Script implement it as a list environment.

In general, the `verse` environment is used for poetry. 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 can, as usual, insert `\\*` instead of `\\`.

Example: As an example, Emma Lazarus's sonnet from the pedestal of Liberty Enlightening the World[2]:

```
\begin{verse}
  Not like the brazen giant of Greek fame\\*
  With conquering limbs astride from land to land\\*
  Here at our sea-washed, sunset gates shall stand\\*
  A mighty woman with a torch, whose flame\\*
  Is the imprisoned lightning, and her name\\*
  Mother of Exiles. From her beacon-hand\\*
  Glows world-wide welcome; her mild eyes command\\*
  The air-bridged harbor that twin cities frame.\\*
  ‘‘Keep, ancient lands, your storied pomp!’’ cries she\\*
  With silent lips. ‘‘Give me your tired, your poor,\\*
  Your huddled masses yearning to breathe free,\\*
  The wretched refuse of your teeming shore.\\*
  Send these, the homeless, tempest-tossed to me:\\*
  I lift my lamp beside the golden door.’’
\end{verse}
```

The result is as follows:

[2] The lines from Roald Dahl's poem "Little Red Riding Hood and the Wolf", which was used in former releases, has been replaced, because in these times certain politicians around the world really seem to need "The New Colossus" as urgent reminder.

Not like the brazen giant of Greek fame
With conquering limbs astride from land to land
Here at our sea-washed, sunset gates shall stand
A mighty woman with a torch, whose flame
Is the imprisoned lightning, and her name
Mother of Exiles. From her beacon-hand
Glows world-wide welcome; her mild eyes command
The air-bridged harbor that twin cities frame.
"Keep, ancient lands, your storied pomp!" cries she
With silent lips. "Give me your tired, your poor,
Your huddled masses yearning to breathe free,
The wretched refuse of your teeming shore.
Send these, the homeless, tempest-tossed to me:
I lift my lamp beside the golden door."

However, if you have very long lines of verse where a line break occurs within a line of verse:

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

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

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

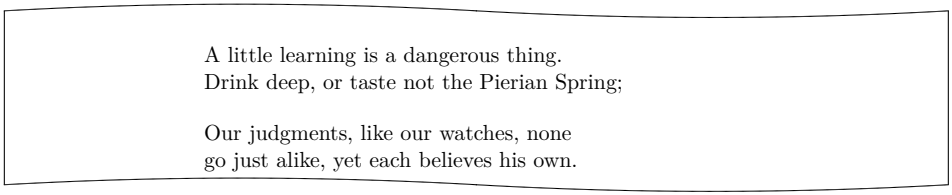
in this case `\\*` can not prevent a page break occurring within a verse at such a line break. To prevent such a page break, a change of `\interlinepenalty` would have to be inserted at the beginning of the environment:

```
\begin{verse}\interlinepenalty 10000
  Both the philosopher and the house-owner
  always have something to repair.\\
  \bigskip
  Don't trust a man, my son, who tells you
  that he has never lied.
\end{verse}
```

Here are two sayings that should always be considered when confronted with seemingly strange questions about LaTeX or its accompanying explanations:

```
\begin{verse}
  A little learning is a dangerous thing.\\*
  Drink deep, or taste not the Pierian Spring;\\
```

```
  \bigskip
  Our judgments, like our watches, none\\*
  go just alike, yet each believes his own.
\end{verse}
```



A little learning is a dangerous thing.
Drink deep, or taste not the Pierian Spring;

Our judgments, like our watches, none
go just alike, yet each believes his own.

Incidentally, `\bigskip` was used in these examples to separate two sayings.

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

These two environments are also set internally as list environments and can be found in both the standard and the KOMA-Script classes. Both environments use justified text which is indented on both the left and the right side. Often they are used to separate longer quotations from the main text. The difference between the two lies in in the manner in which paragraphs are typeset. While `quote` paragraphs are distinguished by vertical space, in `quotation` paragraphs, the first line is indented. This also applies to the first line of a `quotation` environment, unless it is preceded by `\noindent`.

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*}`

Like `quote` and `quotation`, the `addmargin` environment changes the margin. However, unlike the first two environments, `addmargin` lets the user change the width of the indentation. Apart from this change, 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 the value *left indentation* is used for the left margin instead of *indentation*.

The starred variant `addmargin*` differs from the normal version only in the two-sided mode. Furthermore, the difference only occurs if the optional argument *inner indentation* is used. In this case, the value of *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 allow negative values for all parameters. This can be done so that the environment protrudes 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 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 cannot be determined reliably on the first LaTeX run. For details please refer to the explanation of the commands `\Ifthispageodd` (section 3.11, page 79) and `\ifthispagewasodd` (section 21.1).

The interplay of environments such as lists and paragraphs gives rise to frequent questions. Therefore, you can find further explanation in the description of the `parskip` option in section 21.1.

3.19. Mathematics

KOMA-Script classes do not provide their own environments for formulas, systems of equations, or similar mathematical elements. Instead, KOMA-Script relies fully on the maths features of the LaTeX kernel. This also applies to the the options `leqno` and `fleqn`.

You will not find a description of the maths environments of the LaTeX kernel here. If you want to use `displaymath`, `equation`, or `eqnarray` you should read an introduction to LaTeX like [OPHS11]. But if you want more than very simple mathematics, you should use the amsmath package (see [Ame02]).

`leqno`

Equations are normally numbered on the right. The `leqno` option loads the standard option file `leqno.clo`. The equations are then numbered on the left. You must use this option as an optional argument of `\documentclass`. Using it as an argument of `\KOMAoptions` or `\KOMAoption` is not supported. The latter would not make sense because the recommended maths package amsmath can only respond to this option at load time and does not react to run-time changes of the option.

`fleqn`

Displayed equations are normally centred. The standard option `fleqn` loads the standard option file `fleqn.clo`. Displayed equations are then left-justified. You must use this option as an optional argument of `\documentclass`. Using it as an argument of `\KOMAoptions` or `\KOMAoption` is not supported. The latter would not make sense because the recommended maths package amsmath can only respond to this option at load time and does not react to run-time changes of the option.

3.20. Floating Environments for Tables and Figures

With the floating environments, LaTeX offers a powerful and convenient mechanism to arrange figures and tables automatically. Frequently, beginners do not properly understand these floating environments. They often ask to specify the exact position of a table or figure within the text. However, this is usually unnecessary, since the text will contain references to these floating environments. 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, possibly leaving a huge empty space on the previous page.

Often a document will use the same optional argument to position every floating object. This also makes no sense. In such cases, you should instead change the default value globally. For more details, see [Wik].

One final, important note before starting this section: most of mechanisms described here, which extend the capabilities of the standard classes, no longer work correctly when used with packages that modify the appearance of figure and table captions. This should be self-evident, but it is often overlooked.

`captions=`*`setting`*

In the standard classes, the titles of floating environments, which are formatted with the `\caption` command (see below), are set off from the float with vertical spacing appropriate for putting the caption beneath the float, like a signature. They also distinguish between one-line and multi-line captions. One-line captions are centred while multi-line captions are left-justified.

For tables, however, you want the caption to appear as a heading instead of a signature. That's because tables can span multiple pages. With such tables, the reader wants to know the purpose of the table on the first page. Furthermore, tables are usually read row by row, from top to bottom. So there are normally at least two good reasons to provide all tables with headings. KOMA-Script therefore offers the `captions=tableheading` option, which changes the formatting of table captions for use above the table.

Note that multi-page tabulars cannot use a floating environment. To have an automatic page break in a tabular you need an additional package like longtable (see [Car04]) or supertabular (see [BJ04]).

You can switch back to the default caption formatting using `captions=tablesignature`. Note that these options change only the formatting, not the actual position of the caption. Whether the caption is placed above or below a float depends solely upon where you use the `\caption` command inside float environment. However, this can change when using the float package with the `\restylefloats` command (see [Lin01]).

v3.09 Of course, corresponding functions exist for figures using the options `captions=figureheading` and `captions=figuresignature`. However, figures such as photos tend to be viewed as a whole, and a diagram or graph will mostly be examined starting from the lower left. Therefore, it only rarely makes sense to change the caption format for figures alone from signatures to headings.

Sometimes, however, all floating environments should use headings. Therefore KOMA-
v3.09 Script provides options `captions=heading` and `captions=signature` to switch the format of every floating environment. These options can also be used inside a floating environment.

float Please note when using the float package that the settings for signatures or headings will no longer work once `\restylefloat` is applied to tables or figures. For details about the float package and `\restylefloat`, please refer to [Lin01]. This also applies to `\caption` within new floating environments defined with float. You can find additional support which KOMA-Script provides when using the float package in the explanation of `komaabove` (see page 136). As an alternative to float, you can also consult `\captionof` (see page 131) and `\DeclareNewTOC` (see page 404). Additionally, when using float, the scrhack package is expressly recommended (see chapter 16 from page 411 in part II).

Furthermore, KOMA-Script offers the option to disable the distinction between one-line and multi-line captions using the `captions=nooneline` option. This can be useful, for example, if you do not want one-line captions to be centred. The default, where one-line captions are centred, corresponds to `captions=oneline`.

Another special feature of KOMA-Script is the ability to put the caption beside the floating object rather than above or below it. For this, you need the environment `captionbeside`, which is explained starting on page 133. The settings for this environment can also be changed with the `caption` option. You can find the available values for the corresponding *`settings`* in table 3.17.

Table 3.17.: Available values for the `captions` option for setting formatting of captions as headings or signatures in floating environments

`bottombeside`, `besidebottom`
: Captions for the `captionbeside` environment (see section 3.20, page 133) are vertically aligned with the bottommost baseline of the contents of the floating environment.

`centeredbeside`, `besidecentered`, `middlebeside`, `besidemiddle`
: Captions for the `captionbeside` environment (see section 3.20, page 133) are vertically aligned with the center of the contents of the floating environment.

`figureheading`, `figureabove`, `abovefigure`, `topatfigure`
v3.09
: Captions for figures use heading format — possibly deviating from `captions=signature`.

`figuresignature`, `belowfigure`, `bottomatfiggure`
v3.09
: Captions for figures use signature format — possibly deviating from `captions=headings`.

`heading`, `above`, `top`
v3.09
: Captions for floating environments use formatting suitable for use in a heading. This setting does not control whether they are placed at the top or the bottom of the object. This option also implies `captions=tableheading` and `captions=figureheading`.

`innerbeside`, `besideinner`
: Captions for the `captionbeside` environment (see section 3.20, page 133) are placed inside of and next to the contents of the environment in two-sided printing. In one-sided printing, `captions=leftbeside` is used.

`leftbeside`, `besideleft`
: Captions for the `captionbeside` environment (see section 3.20, page 133) are placed to the left of the contents of the floating environment.

`nooneline`
: Single-line captions are handled the same as multi-line captions.

`oneline`
: Single-line captions are centred horizontally.

...

Table 3.17.: Available values for the captions option (*continued*)

`outerbeside`, `besideouter` Captions for the `captionbeside` environment (see section 3.20, page 133) are placed outside of and next to the contents of the environment in two-sided printing. In one-sided printing, `captions=rightbeside` is used.
`rightbeside`, `besideright` Captions for the `captionbeside` environment (see section 3.20, page 133) are placed to the right of the contents of the floating environment.
`signature`, `below`, `bot`, `bottom` v3.09 Captions for floating environments use signature format. This setting does not control whether they are placed at the top or the bottom of the object. This options also implies `captions=tablesignature` and `captions=figuresignature`.
`tableheading`, `tableabove`, `abovetable`, `abovetabular`, `topattable` Captions for tables use heading format — possibly deviating from `captions=signature`.
`tablesignature`, `belowtable`, `belowtabular`, `bottomattable` Captions for tables use signature format — possibly deviating from `captions=heading`.
`topbeside`, `besidetop` Captions for the `captionbeside` environment (see section 3.20, page 133) are vertically aligned to the baseline at the top of the floating environment.

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

In the standard classes, tables and figures are given captions with the `\caption` command placed below the table or figure. For figures, this is generally correct. For tables, opinions differ as to whether captions should be placed above the table or, consistent with captions of figures, below it. Therefore KOMA-Script, unlike the standard classes, offers `\captionbelow` for captions below and `\captionabove` for captions above tables or figures.

For tables and figures, or in general for all floating environments, you can control the behaviour of `\caption` with the `captions` option described at the beginning of this section. For compatibility reasons, the default behaviour of `\caption` for all floating environments is like `\captionbelow`. However, you should use the `captions=tableheading` option, which switches

the behaviour of `\caption` inside table environments to `\captionabove`. Alternatively, you can use `\captionabove` instead of `\caption` inside 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 this 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 `captions=tableheading` option instead (see page 127). 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

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

element	default
caption	\normalfont
captionlabel	\normalfont

\addtokomafont (see section 3.6, page 59). The respective elements for this are caption and captionlabel (see table 3.2, page 60). The font style for the element caption is applied to the element captionlabel before the font style of captionlabel itself is applied. The default settings are listed in table 3.18.

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

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

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

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

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

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

v3.05 Like the caption and capt-of packages, KOMA-Script offers the \captionof command, with which you can put a caption for a floating environment, together with an entry in the corresponding environment list, outside of the floating environment or even in a different floating environment. Unlike \caption, the type of floating environment must be specified as the first parameter.

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

v3.09a Of course \captionof takes into account the heading and signature settings of the captions option. But this feature may be lost if you load the capt-of or caption packages. When using caption, you must follow the instructions for that package (see [Som13])!

Example: Suppose you want to create a floating object with a table and a figure next to each other. Since there are no mixed floating environments, you primarily use a figure environment:

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

Two `minipage` environments were used to place the figure and the table side by side. The percent sign after the end of the first `minipage` is important. Without it, an additional space would appear between the `minipage` environments.

The figure caption was created with `\caption`. The table caption was created with `\captionaboveof` with `table` as the first argument. Because of this, KOMA-Script knows that this is a table caption even though it is inside the `figure` environment.

The optional argument of `\captionaboveof` creates an entry in the list of tables. Without the optional argument, the caption specified in the final mandatory argument would have been used for the list of tables too. Although this caption text is sufficient for the environment itself, it would not be very meaningful in the list of tables. Therefore, a different title is used for the list of tables using the optional argument. figure 3.3 shows the result of the example code.

You can produce a non-floating table with a caption in the same way as the table inside a figure environment in the example above. In such a case, a `minipage` environment should also be used to avoid page breaks between the caption and the table. In addition, you should embed the `minipage` environment in a `flushleft` environment both to achieve a pleasing separation between the surrounding text and to avoid the paragraph indentation of the `minipage` environment.

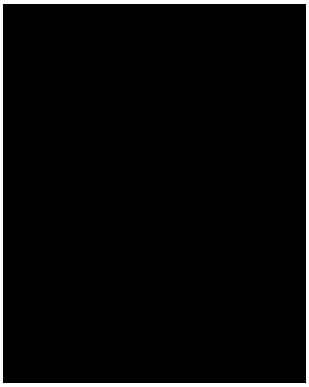


Table 3.19.: Rectangle size

Width:	4 cm
Height:	5 cm

Figure 3.2.: A rectangle

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

```
\begin{captionbeside}[short title]{caption text}[placement][width][offset]...\end
                     {captionbeside}
\begin{captionbeside}[short title]{caption text}[placement][width][offset]*...\end
                     {captionbeside}
```

v2.8q

In addition to captions above and below the figure, you will often find captions, in particular for small figures, which are placed beside the figure. The bottom edge of the caption is normally aligned with the bottom of the figure. Of course, you can achieve the same thing in the standard classes with some fiddling and the use of two `\parbox` commands. However, KOMA-Script offers a special environment for this which you can use within the floating environments. The first optional parameter, *short title*, and the required parameter *caption text* have the same meaning as the corresponding parameters of `\caption`, `\captionabove` or `\captionbelow`. The *caption text* is placed beside the content of the environment in this case.

The *placement* parameter can determine whether the *caption text* is placed on the left or the right. Exactly one of the following letters is allowed:

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

v3.00

The default placement is to the right of the content of the environment. You can change this default using the `captions` option (see page 126) 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.

Normally, the content of the environment and the *caption text* fill the entire available text width. However, you can specify a different width using the optional parameter *width*. This can be greater than the current text width.

When specifying a *width*, the width used is usually centred with respect to the body text. 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.

If you add a star to the optional *offset* parameter, the value represents a shift relative to the right margin on left-hand pages in a 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. This variant may require two LaTeX runs to achieve the correct offset.

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. You can change this setting using the captions option (see page 126) with the value `topbeside`, `centeredbeside`, or `bottombeside`. With the setting `topbeside`, the topmost base lines of the environment contents and caption will be aligned. With `centeredbeside`, they will be centred vertically. In this context, note that the base line of a figure is usually the bottom of the figure. You can change this using, for example, `\raisebox`.

Example: You can find an example using the `captionbeside` environment 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:\LabelBase.captionbeside}
\end{figure}
```

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

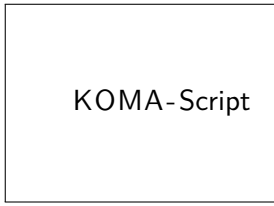


Figure 3.4.: A figure description which is neither above nor below, but beside the figure

Figure 3.5 shows a centred caption with:

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

This is certainly not a recommended solution.

In contrast, you can use the top-aligned format seen in figure 3.6. To illustrate how to shift the baseline using `\raisebox`, here is a complete example. You can apply this not only to a substitute figure, as previously shown, but also, for example, to `\includegraphics` (see [Car17]).

```
\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:\LabelBase.captionbesidetop}
```

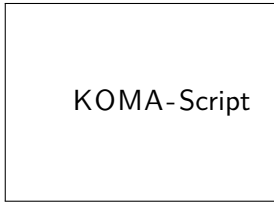
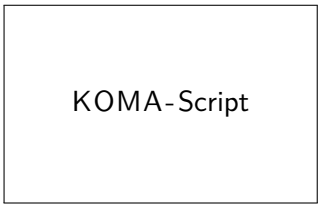


Figure 3.5.: A figure description which is neither above nor below, but centred beside the figure

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




```
\end{figure}
\end{document}
```

```
\begin{captionofbeside}{float type}[short title]{caption text}[placement][width]
                       [offset]
  ⋮
\end{captionofbeside}
\begin{captionofbeside}{float type}[short title]{caption text}[placement][width]
                       [offset]*
  ⋮
\end{captionofbeside}
```

v3.10 As is the case for \caption, there is a variant for \captionof where the *float type* is not defined by using it within a floating environment of this type, so you can specify a suitable environment for captionbeside with captionofbeside. In contrast to captionbeside, the *float type* must be specified as an additional, first 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 appear above or below. 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, you can activate these styles 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 various ways to change the formatting of the caption text. The definition of different font styles has already been explained above. The delimiter or delimiters between the label and actual caption text is specified by the macro \captionformat. In contrast to all other \...format commands, this is not the counter but only the items which follow it. The original definition is:

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

You can change this too with `\renewcommand`.

Example: For some inexplicable reason, you want a dash surrounded by spaces instead of a colon followed by a space as a label delimiter. You therefore define:

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

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

`\figureformat`
`\tableformat`

It has already been mentioned that `\captionformat` does not contain formatting for the label itself. You should not alter this by redefining the commands for the output of the `\thefigure` or `\thetable` counters. Such a redefinition would have unwanted side effects on the output of `\ref`, the table of contents, the list of figures, and the list of tables. Instead, KOMA-Script offers two `\...format` commands. These have the following defaults:

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

They can also be customised to your requirements with `\renewcommand`.

Example: From time to time, it is required to have captions without labels or delimiters. With KOMA-Script the following definitions suffice 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 especially important 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. In other words, 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.

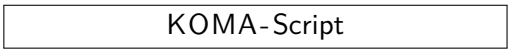


Figure 3.7.: With the default setting, like using `\setcaphanging`

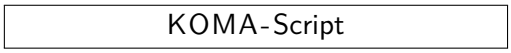


Figure 3.8.: With partially hanging indentation starting from the second line by using `\setcapindent{1em}`

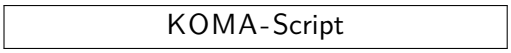


Figure 3.9.:
With hanging indentation starting from the second line and line break before the description using `\setcapindent*{1em}`

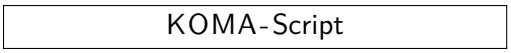


Figure 3.10.:
With indentation only in the second line and a line break before the description using `\setcapindent{-1em}`

You can change this behaviour, which corresponds to using `\setcaphanging`, at any time with the `\setcapindent` or `\setcapindent*` command. 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*`.

A negative value of *indent*, on the other hand, results in a line break before the caption text, and only the first line of the caption text, not subsequent lines, is indented by the absolute value of *indent*.

Whether one-line captions are set the same way as multi-line captions or are treated separately is specified with the option `captions`. For details please refer to the explanations of these option on page 127.

Example: The illustrations 3.7 to 3.10 show the effects of different settings. As you can see, using a fully hanging indentation with a narrow column width is awkward. 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 partially indented caption
           starting from the second line}
\end{figure}
```

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

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

v2.8q

Using these three commands, you can specify the width and justification of the caption text. Normally, the entire text or column width is available for the caption.

With the \setcapwidth command, you can decrease this *width*. The mandatory 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-justified

`c` – centred

`r` – right-aligned

`i` – aligned to the inner margin in two-sided printing

`o` – aligned to the outer margin in two-sided printing

Inside and outside justification corresponds to left-aligned and right-aligned justification, respectively, in one-sided printing. Within longtable tables, inside and outside justification 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 the longtable package.

v3.20

Note that \setcapwidth immediately sets the width to the value of the *width* parameter at the time of the assignment, as \setlength does. If you instead want to use the current value of *width* when the caption is set, you should use \setcapdynwidth. There can be significant differences in the result if, for example, you use lengths like \linewidth or other commands as *width* arguments.

With the \setcapmargin command, instead of specifying the width of the caption text, you can specify a *margin* to be set next to the caption text in addition to the normal text margin. If you want margins with different widths on the left and right sides, you can use the optional argument to specify a *left margin* that differs from *margin*. Instead of a *left margin*, the starred version \setcapmargin* defines an *inside margin* in a two-sided layout. The same problem arises here with with inside and outside justification inside longtable tables that occurs with \setcapwidth. Furthermore, using \setcapmargin or \setcapmargin* activates the captions=nooneline option (see page 127) for captions which are typeset with this margin setting.

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

Table 3.20.: Alignments for multi-line captions of floating environments

c	centred
j	fully justified
l	left justified
r	right justified
C	centred with ragged2e
J	fully justified with ragged2e
L	left justified with ragged2e
R	right justified with ragged2e

`\setcaptionalignment[`*float type*`]{`*alignment*`}`

v3.25

Normally, multi-line captions are fully justified. This corresponds to `\setcaptionalignment{j}`. Sometimes, however, you want a different alignment, for example ragged right. An appropriate change is possible at any time with `\setcaptionalignment`. You can specify exactly one of the letters listed in table 3.20 for the *alignment*. If you specify an unknown *alignment*, you will receive an error message.

The four possibilities with the ragged2e package are only available if that package was loaded before you use `\setcaptionalignment`. Otherwise, they are converted to the corresponding options without ragged2e. For safety reasons, a warning is issued in this case.

When using this command without the optional parameter, the result depends on whether the call occurs inside our outside of a floating environment. Within a floating environment, the alignment is then set for this floating environment. Outside, on the other hand, the optional parameter is assumed to be empty.

Using this command with an empty optional parameter, or outside a floating environment and also without any optional parameter, sets the general alignment. This is used whenever the current floating environment type does not define an alignment.

If you only want to set the alignment of a particular type of floating environment without changing the *alignment* for other types of floating environments, then the type of floating environment, e. g., `figure` or `table`, is given as the optional parameter *float type*.

Example: You want captions to be centred under the images even if they are multi-line. To text this for a single image, use:

```
\begin{figure}
  \centering
  \setcaptionalignment{c}
  \includegraphics{example-image}
  \caption{\blindtext}
\end{figure}
```

Since you are satisfied with the result, you move the

```
\setcaptionalignment{c}
```

command to the document preamble. Thereafter, however, you decide you do not like this change for table captions at all. Therefore, you use

```
\setcaptionalignment[figure]{c}
```

to limit the centring to figures.

A little later, you realize that the centring is not so suitable. Instead, you now prefer to have ragged right alignment. So you change the alignment again:

```
\setcaptionalignment[figure]{l}
```

However, you do not like the fact that the lines are very different in length. This is caused by the lack of hyphenation, causing long words to wrap completely onto the next line, leaving large gaps. You want to allow hyphenation as needed. This is easy to achieve with the help of the ragged2e package. However, it is not enough to use

```
\usepackage{ragged2e}
```

to load the package. Using the `newcommands` option when loading the package does not help. Instead, the *alignment* must also be changed:

```
\usepackage{ragged2e}
\setcaptionalignment[figure]{L}
```

Note the upper-case letter for the *alignment*.

`origlongtable`

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

`listof=`*setting*

v3.00

Normally lists of floating environments — like tables or figures — are neither numbered nor included in the table of contents. You can find more information about this behaviour in section 3.9. As an alternative to the settings `toc=nolistof`, `toc=listof`, and `toc=listofnumbered` mentioned there, you can also look at this behaviour from perspective of the lists themselves. Therefore you can achieve the same results with the settings `listof=notoc`, `listof=totoc`, and `listof=numbered`.

By default, the headings in the lists of floating environments use the topmost level below `\part`. This is the chapter level in scrbook and scrreprt and the section level in scrartcl.

v3.06

By contrast, `listof=leveldown` uses the next lower level to be used instead. `listof=standardlevel` switches back to the default sectioning level, if necessary.

Example: In a book, you want to put the list of figures and the list of tables as sub-lists into a common list named "Figures and Tables". Simply use:

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

to use the section instead of the chapter level for both lists, and put the following at the appropriate place in your document:

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

You can find more information about the \addchap* command in section 3.16 on page 105.

v2.8q Normally the lists of floating environments use a fixed-width space for the caption number of the entries. At the same time, all entries are somewhat indented. This behaviour corresponds to the listof=graduated setting.

If the numbers become very wide, for example because you have many tables or figures, the space provided may at some point no longer be sufficient. KOMA-Script offers the setting listof=flat for lists of floating environments, comparable to toc=flat for the table of contents. The width required to print the number is determined automatically and the space is adjusted accordingly. See the toc=flat option, section 3.9, page 72 for more information about side effects and how it works. Note again that you need more than one LaTeX run before the lists of floating environments reach their final state.

v3.06 The listof=entryprefix setting automatically activates listof=flat too. Normally, it does not make sense to add a prefix such as "figure" or "table" to each entry in the lists of floating environments because, of course, only figures are included in the list of figures and only tables are included in the list of tables. Such a prefix provides no additional information and is thus omitted by default. However, you can add such prefixes using the listof=entryprefix option. With this, all entries in the same list will get the same prefix. The prefix depends on the file extension of the auxiliary file that is used for the corresponding list. For the list of figures, the file extension is "lof" and therefore \listoflofentryname is used. For the list of tables, the file extension is "lot" and \listoflotentryname is used.

scrbook, scrreprt For the scrbook and scrreprt classes, KOMA-Script adds a vertical space to the lists of floating environments whenever a new chapter starts. This behaviour, which also exists in the standard classes, groups the lists by chapters. In KOMA-Script, it corresponds to setting

v3.00 listof=chaptergapsmall. In this case, is uses a fixed vertical space of 10 pt. With the listof=chaptergapline option, you instead get a vertical space the height of one standard text line. With listof=nochaptergap, you can completely remove the vertical space. The listof=chapterentry option is a special feature. Instead of a vertical space, the table of contents entry for the chapter is inserted into the lists of floating environments. Note that this happens even if the chapter does not contain a floating environment. You can find a method where only chapters containing floating environments appear in the respective lists at [Koh15].

You can achieve a more direct control over what appears in the lists of floating environments with the `chapteratlists` option, which is explained in section 3.16, on page 99.

You can find an overview of all settings for the `listof` option in table 3.21.

Table 3.21.: Available values for the `listof` option for modifying the format and contents of the lists of floating environments

`chapterentry`, `withchapterentry`
: Indicates the beginning of chapters in the lists of floating environments with copies of their table-of-contents entries.

`chaptergapline`, `onelinechaptergap`
: Indicates the beginning of chapters in the lists of floating environments with a vertical space the height of one standard text line.

`chaptergapsmall`, `smallchaptergap`
: Indicates the beginning of chapters in the lists of floating environments with a small vertical space.

`entryprefix`
v3.06
: Adds a prefix before the number of each floating-environment list entry. The prefix is usually language-dependent, e. g., in English “Figure” is used for the list of figures and “Table” for the list of tables, each followed by a white space.

`flat`, `left`
: Prints the lists of floating environments in tabular form. The caption numbers are the first column, the caption texts the second column, and the page numbers the third column. The space reserved for the caption numbers depends on the previous LaTeX run.

`graduated`, `indent`, `indented`
: Prints the lists of floating environments in a hierarchical form. The space reserved for the caption numbers is limited.

`leveldown`
: Uses headings that are one level lower in the sectioning hierarchy than the default for lists of floating environments.

...

Table 3.21.: Available values for the `listof` option (*continued*)

`indenttextentries`, `indentunnumbered`, `numberline`

v3.12

The `numberline` property (see section 15.2, page 383) is set for the lists of floating environments such as figures and tables. As a result, unnumbered entries are left-aligned with the text of numbered entries of the same level. However, the KOMA-Script classes themselves do not put unnumbered entries in these lists. This option therefore affects only entries that are generated not by the KOMA-Script classes themselves but with the help of `\addxcontentsline` (see section 15.2, page 378).

`leftaligntextentries`, `leftalignunnumbered`, `nonumberline`

v3.12

The `nonumberline` property (see section 15.2, page 383) is set for the lists of floating environments. This will place unnumbered entries left-aligned with the number of numbered entries. However, the KOMA-Script classes themselves do not put unnumbered entries in these lists. This option therefore affects only entries that are generated not by the KOMA-Script classes themselves but with the help of `\addxcontentsline` (see section 15.2, page 378).

`nochaptergap`, `ignorechapter`

The beginnings of chapters are not marked in the lists of floating environments.

`notoc`, `nottotoc`, `plainheading`

The lists of floating environments do not generate entries in the table of contents.

`numbered`, `totocnumbered`, `tocnumbered`, `numberedtoc`, `numberedtotoc`

The lists of floating environments receive numbered entries in the table of contents.

`standardlevel`

The lists use the normal sectioning level.

`totoc`, `toc`, `notnumbered`

The lists of floating environment generate entries in the table of contents, but their headings are unnumbered.

`\listoftables`
`\listoffigures`

These commands generate a list of tables or figures. Changes affecting these lists will require two LaTeX runs to take effect. The layout of the lists can be influenced by the `listof` option with the values `graduated` or `flat` (see page 141). In addition, the `listof` and `listofnumbered` values of the `toc` option (see section 3.9), as well as the `totoc` and `numbered` values of the `listof` option described above indirectly affect these lists.

As a rule, you will find the lists of the floating environments immediately after the table of contents. In some documents, they go into the appendix. However, the author of this guide prefers them immediately after the table of contents.

3.21. Marginal Notes

In addition to the text area, which normally fills the type area, documents often contain a column for marginalia. You can set marginal notes in this area. This guide makes frequent use of them.

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

Marginal notes in LaTeX are usually inserted with the `\marginpar` command. They are placed in the outer margin. One-sided documents use the right border. Although you can specify a different marginal note for `\marginpar` in case it winds up in the left margin, marginal notes are always fully justified. However, experience has shown that many users prefer left- or right-justified marginal notes instead. For this purpose, KOMA-Script offers the `\marginline` command.

Example: In some parts of this guide, the class name scrartcl can be found in the margin. You can produce this 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 just a shorthand notation for the code above.

Advanced users will find notes about difficulties that can arise using `\marginpar` in section 21.1. These remarks also apply to `\marginline`. In addition, chapter 19 introduces a package that you can use to create note columns with their own page breaks.

3.22. Appendix

The appendix of a document mostly consists of supplements to the document. Typical parts of an appendix include a bibliography, an index, and a glossary. However you should not start an appendix for these parts alone because their format already distinguishes them from the main document. But if there are additional elements in the appendix, such as quoted third-party documents, endnotes, figures, or tabulars, the standard elements such as the bibliography should also be part of the appendix.

\appendix

The appendix is started in the standard as well as the KOMA-Script classes with \appendix. Among other things, this command changes the chapter numbering to upper-case letters while ensuring that the rules according to [DUD96] for numbering the sectioning levels are followed (for German-speaking regions). These rules are explained in more detail in the description of the numbers option in section 3.16, page 98.

scrbook, scrreprt

The format of the chapter headings in the appendix is influenced by the chapterprefix and appendixprefix options. See section 3.16, page 95 for more information.

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

3.23. Bibliography

The bibliography makes external sources accessible. As a rule, the bibliography is created from an external file with a database-like structure using the BibTeX program. You can use the BibTeX style to change both the format of the entries and their sorting. If you use an additional bibliography package like natbib, babelbib, or biblatex, KOMA-Script's influence over the bibliography disappears. In such cases, you must follow the manual of the relevant bibliography package. You can find general information about bibliographies in [OPHS11].

bibliography=*setting*

v3.00

For a start, you can select any predefined bibliography style in *setting*. There are two such bibliography styles predefined in KOMA-Script. You should not confuse them with the styles used by BibTeX, which you select with \bibstyle. While BibTeX determines both the sorting and the contents of the bibliography, KOMA-Script influences only some basic features of the bibliography and few properties of the entry format.

The bibliography=oldstyle option selects a compact format for bibliography entries. In this case, using the \newblock command results in only a small glue between the entries. The name comes from the fact that this is the most common classic form of a bibliography. In contrast, the bibliography=openstyle setting selects a more modern and open kind of bibliography. The name comes from the fact that the \newblock command inserts a paragraph break. The bibliography entries appear more structured. They are less compact and seem more relaxed or open. Information about defining new bibliography styles can be found in the description of the \newbibstyle command in section 21.9, page 500.

In addition to the formatting style, you can select one more feature using *setting*. The bibliography is a kind of list of contents. But instead of listing the contents of the document itself, it references external works. With this reasoning, you could argue that the bibliography is a separate chapter or section and therefore deserves a chapter or section number. The

Table 3.22.: Predefined values for the `bibliography` option for setting the bibliography format

`leveldown`	The bibliography uses a sectioning command one level lower than the default.
`notoc`, `nottotoc`, `plainheading`	The bibliography receives neither an entry in the table of contents nor a number.
`numbered`, `tocnumbered`, `totocnumbered`, `numberedtoc`, `numberedtotoc`	The bibliography receives an entry in the table of contents and a number.
`oldstyle`	The bibliography uses the classic, compact formation, where `\newblock` generates only a horizontal glue.
`openstyle`	The bibliography uses the structured, open format where `\newblock` generates a paragraph break.
`standardlevel`	The bibliography uses the standard sectioning command level.
`toc`, `totoc`, `notnumbered`	The bibliography receives an entry in the table of contents but no number.

`bibliography=numbered` setting does exactly that, including creating an entry in the table of contents. In my opinion, a traditional, annotated source list should by this reasoning be a separate chapter too. Moreover, the bibliography is ultimately nothing you've written yourself and so merits at most an unnumbered entry in the table of contents, which is achieved with the `bibliography=totoc` option. The default setting, where the bibliography is produced as an unnumbered chapter and does not receive an entry in the table of contents corresponds to `bibliography=nottotoc`. For more information, see the `toc` option in section 3.9, especially the `bibliographynumbered`, `bibliography`, and`nobibliography` values for this option on page 72.

v3.12 Sometimes a document made using scrbook or scrreprt will have a bibliography for every chapter rather than one bibliography for the whole document. In that case, it makes sense for the bibliography itself to be not a chapter but a section. You can achieve this using the `bibliography=leveldown` option. You can also use this if you want the bibliography to appear with other lists under a common heading, therefore this option is also available with scrartcl.

You can find a summary of available values for the `bibliography` option in table 3.22. Note, however, that you can define new values with `\newbibstyle`.

\setbibpreamble{*preamble*}

You can use the \setbibpreamble command to set a preamble for the bibliography. This preamble must be placed before the command for issuing the bibliography. However, it need not be directly in front of it. For example, it could be placed at the beginning of the document. Like the bibliography=totoc and bibliography=totocnumbered options, this command only works if you have not loaded a package which prevents this from happening by redefining the thebibliography environment. Although the natbib package uses undocumented, internal KOMA-Script macros, \setbibpreamble could still work with the current version of natbib (see [Dal10]).

Example: You want to point out that the bibliographical references are sorted alphabetically by the names of the authors. You therefore 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 ensures 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. With this instruction, you can insert a break into the bibliography. The *interruption code* will be expanded inside a group. Such a break may be, for example, a heading using \minisec. Unfortunately there is currently no way to have this command automatically generated, for example by using a special entry in the BibTeX database. Therefore, it can currently only be used by users who edit the bibliography directly. For this reason, its usefulness is very limited.

\AfterBibliographyPreamble{*code*}
\AtEndBibliography{*code*}

v3.00 In some cases, it may be useful to add some *code* after the bibliography preamble or just before the end of the bibliography. This is possible with the help of these two instructions.

Example: You want to set the bibliography not justified but ragged right. You can achieve this with:

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

You can put this instruction anywhere before the bibliography. However, it is recommended to do so in the preamble of the document or a separate package.

The functionality of this instruction depends on cooperation with packages modifying the bibliography, if you use such a package.

Table 3.23.: Available values for the `index` option

`leveldown` v3.18 The index is moved down by one sectioning level.
`notoc`, `nottotoc`, `plainheading` The index does not receive an entry in the table of contents.
`numbered`, `tocnumbered`, `totocnumbered`, `numberedtoc`, `numberedtotoc` v3.18 The index receives an entry in the table of contents and is numbered.
`standardlevel` v3.18 The index is at the usual sectioning level.
`toc`, `totoc`, `notnumbered` The index receives an entry in the table of contents without being numbered.

3.24. Index

For general information about making a keyword index, see [OPHS11], [Lam87], and [Keh97]. Using a package that redefines commands or environments for the index KOMA-Script's ability to influence the index may disappear. This applies, for example, when using the index package, but not when using the splitidx package (see [Koh14]).

`index=`*`setting`*

v3.00 By default or with `index=default`, the index is an unnumbered chapter (scrbook or scrreprt) or section (scrartcl) without an entry in the table of contents. Since the index usually comes last in a book or similar document, it does not actually need an entry in the table of contents. Nevertheless, if you want to do this, for example because you are working with a multi-index keyword dictionary such as the one that appears in this guide, you can create this with the v3.18 `index=totoc` option. You can even number the index using the `index=numbered` setting. See also the `toc` option with the `index` or `indexnumbered` values in section 3.9 starting at page 72.

For example, if you create multiple indexes with splitidx (see [Koh14]), it may be useful to v3.18 group them under a common heading. To make this possible, `index=leveldown` places the index one sectioning level deeper than usual. In scrbook and scrreprt, it is no longer a chapter v3.18 but a section; with scrartcl, a subsection. The `index=standardlevel` option is the counterpart to this and cancels any instance of `index=leveldown` used previously.

You can find a summary of the available values for the *`setting`* of `index` in table 3.23.

`\setindexpreamble{`*`preamble`*`}`

As with the bibliography, you can also provide a preamble to the index. This is often the case if you have more than one index or if you mark different kinds of references 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{All page numbers printed in \textbf{bold}
  refer to definitions of terms. Page numbers printed
  normally refer to pages where the term is used.\par\bigskip}
```

Note that the page style is changed for the first page of the index. The page style that is applied is defined in the macro `\indexpagestyle` (see section 3.12, page 83).

The usual LaTeX packages and additional programs are responsible for creating, sorting, and outputting the index. KOMA-Script, like the standard classes, provides only the basic macros and environments for them.

Letters with the scrlttr2 Class or the scrletter Package

Letters are quite different in many ways from articles, reports, books, and the like. That alone justifies a separate chapter on letters. But there are other reasons for a separate chapter on scrlttr2 and scrletter.

The scrlttr2 class was developed from scratch in 2002. It provides a completely new user interface, different from every other class I know. This new user interface may be unusual, but it offers benefits to both new and experienced KOMA-Script users.

v3.15 The scrletter package has supplemented KOMA-Script since Version 3.15. It also provides all the letter-based functionality of scrlttr2 to the other classes. I recommend you use one of the KOMA-Script classes — scrbook, scrreprt or scrartcl — which are explained in the previous chapter. With minor limitations, scrletter also works well with the standard classes.

The starting point for developing scrletter was, on the one hand, requests from users who also wanted to have elements such as section headings, floating environments, or a bibliography in letters. On the other hand, there were also requests to use scrlttr2 variables in the remaining KOMA-Script classes. You can achieve both by combining the desired KOMA-Script class with scrletter.

Compared to the letter class, the letter package has a few small changes that were necessary to avoid conflicts with other classes. These changes mainly affect the page styles and are explicitly documented (see section 4.13, starting at page 226). Where scrletter is not explicitly mentioned, everything that is documented for scrlttr2 applies without change.

4.1. Early or Late Selection of Options

The information in section 2.4 applies equally to this chapter. So if you have already read and understood section 2.4, you can skip ahead to section 4.2, page 152.

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

LaTeX allows users to pass class options as a comma-separated list of keywords in the optional argument to \documentclass. In addition to being passed to the class, these options are also passed on to all packages that can understand them. Users can also pass a similar comma-

v3.00 separated list of keywords in the optional argument of \usepackage. KOMA-Script extends the option mechanism for the KOMA-Script classes and some packages with further options. Thus most KOMA-Script options can also take a value, so an option does not necessarily take the form *option*, but can also take the form *option=value*. Except for this difference, \documentclass and \usepackage in KOMA-Script function as described in [Tea05b] or any introduction to LaTeX, for example [OPHS11].

When using a KOMA-Script class, you should not specify options when loading the typearea or scrbase packages. The reason for this restriction is that the class already loads these packages without options, and LaTeX refuses to load a package multiple times with different option settings. In general, it is not necessary to load either one of these packages explicitly when using any KOMA-Script class.

Setting the options with `\documentclass` has one major disadvantage: unlike the interface described below, the options in `\documentclass` are not robust. So commands, lengths, counters, and similar constructs may break inside the optional argument of this command. For example, with many non-KOMA-Script classes, using a LaTeX length in the value of an option results in an error. So if you want to use a LaTeX length, counter, or command as part of the value of an option, you should use `\KOMAoptions` or `\KOMAoption`. These commands will be described next.

`\KOMAoptions{`*option list*`}`
`\KOMAoption{`*option*`}{`*value list*`}`

v3.00

KOMA-Script also provides the ability to change the values of most class and package options even after loading the class or package. You can use the `\KOMAoptions` command to change the values of a list of options, as in `\documentclass` or `\usepackage`. Each option in the *option list* has the form *option*`=`*value*.

Some options also have a default value. If you do not specify a value, that is if you give the option simply as *option*, then this default value will be used.

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

KOMA-Script uses the commands `\FamilyOptions` and `\FamilyOption` with the family "`KOMA`" to implement this ability. See part II, section 12.2, page 336.

Options set with `\KOMAoptions` or `\KOMAoption` will reach both the KOMA-Script class and any previously loaded KOMA-Script packages that recognise these options. If an option or a value is unknown, scrbase will report it as an error.

4.2. Compatibility with Earlier Versions of KOMA-Script

The information in section 2.5 applies equally to this chapter. However, this feature has existed in scrlttr2 since version 2.9t, whereas scrletter does not offer it. So if you have already read and understood section 2.5 you can skip ahead to page 153, page 153.

Those who produce their documents from source code typically attach the utmost importance to the fact that future LaTeX runs will yield exactly the same result. In some cases, however, improvements and bug fixes to the class will result in changes of behaviour, especially to the layout. This, however, may be undesirable.

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

scrlttr2 Since Version 2.9t, scrlttr2 has been able to choose whether the source file should, as much as possible, continue to produce exactly the same result within a LaTeX run or should be formatted according to the modifications of the latest version of the class. You can specify the version with which you want your file to be compatible by using the `version` option. Compatibility with the oldest supported KOMA-Script version can be achieved with `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`, you can select the latest version. In this case, you give up backwards v3.01a compatibility. If the option is used without a value, `last` is assumed. This also corresponds to the default setting, as long as you do not use any deprecated options.

If you use a deprecated option of KOMA-Script 2, KOMA-Script 3 will switch to `version=first` automatically. This will also result in a warning message that explains how to prevent this switch. Alternatively, you can choose a different setting for `version` with the desired compatibility after the deprecated option.

Compatibility is primarily a question of line and page breaks (wrapping). If you choose compatibility with an older version, new options that do not affect wrapping are still available. The `version` option does not affect any wrapping changes that are the result of fixing unambiguous errors. If you need unconditional wrapping compatibility even in the case of bugs, you should physically save the old KOMA-Script version you need together with your document.

Example: The example letters in this chapter should use all the features of the latest version of KOMA-Script. For this, the we set the compatibility correspondingly when loading the class:

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

In this case the symbolic value `last` has been used to select the latest version.Here, the latest version was simply chosen with the symbolic value `last`.

Note that you cannot change the `version` option after loading the class. Setting this option with \KOMAoptions or \KOMAoption will therefore cause an error.

4.3. Draft Mode

scrlttr2 The information in section 3.3 applies equally to scrlttr2. So if you have already read and understood section 3.3, you can skip ahead to section 4.4 on page 154. The scrletter package does not provide a draft mode itself but relies upon the class you use.

Many classes and packages provide a draft mode in addition to the normal typesetting mode. The differences between these two are as diverse as the classes and packages that offer this distinction.

```
draft=simple switch
overfullrule=simple switch
```

scrlttr2

The `draft` option distinguishes between documents being drafted and finished documents. The *simple switch* can be one of the standard values for simple switches from table 2.5, page 42. If you activate this option, small black boxes will be output at the end of overly long lines. These boxes make it easier for the untrained eye to locate the paragraphs that require manual post-processing. By contrast, the default, `draft=false`, shows no such boxes. Incidentally, such lines often disappear when you use the microtype package [Sch13].

v3.25

Since the `draft` option can lead to all sorts of unwanted effects with various packages, KOMA-Script allows you to control this marking of overly long lines separately with the `overfullrule` option. If this option is enabled, the marker is again displayed.

4.4. Page Layout

Each page of a document consists of different layout elements, such as the margins, the header, the footer, the text area, the marginal note column, and the distances between these elements. KOMA-Script additionally distinguishes the entire page, also known as the paper, and the visible page. Without doubt, the separation of the page into these different parts is one of

scrlttr2

the basic features of a class. KOMA-Script delegates this work to the package typearea. This package can also be used with other classes. The KOMA-Script classes, however, load typearea on their own. Therefore, it's neither necessary nor sensible to load the package explicitly with `\usepackage` while using a KOMA-Script class. See also section 4.1, page 151.

Some settings of KOMA-Script classes affect the page layout and vice versa. Those effects are documented at the corresponding settings.

For more information about the choice of paper format, the division of the page into margins and type area, and the choice between one- and two-column typesetting, see the documentation for the typearea package. You can find it in chapter 2, starting on page 28.

For letters, it is normally not useful to distinguish one-sided and two-sided printing. Since letters are not usually bound, each page of a letter will be viewed on its own. This is also true even if both the letter is printed on both sides of the paper. Vertical adjustment usually does not matter for letters either. If you nevertheless need it, or want to understand what it is, please refer to the commands `\raggedbottom` and `\flushbottom` explained in section 3.4 on page 57.

4.5. Variables

In addition to options, commands, environments, counters, and lengths, chapter 3 introduced the concept of additional elements for KOMA-Script. A typical property of an element is its font style and the ability to change it (see section 4.9, page 179). In this section we introduce variables. Variables can have a label used to identify them when they are output in the document as well as their actual content. To avoid confusion with labels used for cross-references, this guide refers to such labels as the "description" of the variable. The content of a variable can be set independently of the time and place it is used the same way that the content of a command can be defined separately from its use. The main difference between a command and a variable is that a command usually triggers an action, whereas a variable usually consists of plain text which is then output by a command. In addition, a variable can also have a description which can be customised and output.

This section deliberately confines itself to introducing the concept of variables. The examples below have no special meaning. More detailed examples can be found in the explanation of predefined variables used in the class and the package. An overview of all defined variables is given in table 4.1.

Table 4.1.: Supported variables in scrlttr2 and scrletter

`addresseeimage`
: commands used to print the postpaid postmark for the `addrfield=backgroundimage` option or the postpaid address for the `addrfield=image` option (section 4.10, page 202)

`backaddress`
: return address for window envelopes (section 4.10, page 202)

`backaddressseparator`
: separator within the return address (section 4.10, page 202)

`ccseparator`
: separator between title of additional addresses (cc list) and additional addresses (section 4.7, page 175)

`customer`
: customer number (section 4.10, page 212)

`date`
: date (section 4.10, page 211)

...

Table 4.1.: Supported variables in scrlttr2 and scrletter (*continued*)

`emailseparator`
: separator between email name and email address (section 4.10, page 195)

`enclseparator`
: separator between title of enclosure and enclosures (section 4.7, page 175)

`faxseparator`
: separator between title of fax and fax number (section 4.10, page 195)

`firstfoot`
v3.08
: footer of the letterhead page (section 4.10, page 222)

`firsthead`
v3.08
: header of the letterhead page (section 4.10, page 200)

`fromaddress`
: sender's address without sender name (section 4.10, page 190)

`frombank`
: sender's bank details (section 4.10, page 223)

`fromemail`
: sender's e-mail (section 4.10, page 195)

`fromfax`
: sender's fax number (section 4.10, page 195)

`fromlogo`
: commands for inserting the sender's logo (section 4.10, page 199)

`frommobilephone`
v3.12
: sender's mobile telephone number (section 4.10, page 195)

`fromname`
: complete name of sender (section 4.10, page 190)

`fromphone`
: sender's telephone number (section 4.10, page 195)

`fromurl`
: URL of the sender, e. g. of the sender's homepage (section 4.10, page 195)

...

Table 4.1.: Supported variables in scrlttr2 and scrletter (*continued*)

`fromzipcode`
: ZIP code (postal code) of the sender for the postpaid postmark of the `addrfield=PP` option (section 4.10, page 202)

`invoice`
: invoice number (section 4.10, page 212)

`location`
: extra details of the sender (section 4.10, page 208)

`myref`
: sender's reference (section 4.10, page 212)

`nextfoot`
v3.08
: footer using page style `headings` or `myheadings` (section 4.13, page 230)

`nexthead`
v3.08
: header using page style `headings` or `myheadings` (section 4.13, page 230)

`phoneseparator`
: separator between title of telephone and telephone number (section 4.10, page 195)

`place`
: sender's location; used next to date (section 4.10, page 202)

`placeseparator`
: separator between location and date (section 4.10, page 212)

`PPdatamatrix`
: command to print the data array for the `addrfield=PP` option (section 4.10, page 202)

`PPcode`
: commands for the sender's identification code for the `addrfield=PP` option (section 4.10, page 202)

`signature`
: signature annotation beneath the closing text of the letter (section 4.10.7, page 219)

`specialmail`
: delivery method (section 4.10, page 202)

...

Table 4.1.: Supported variables in scrlttr2 and scrletter (*continued*)

`subject`
: letter's subject (section 4.10, page 216)

`subjectseparator`
: separator between subject title and subject (section 4.10, page 216)

`title`
: letter title (section 4.10, page 215)

`toaddress`
: address of recipient without recipient name (section 4.10, page 202)

`toname`
: complete name of recipient (section 4.10, page 202)

`yourmail`
: date of recipient's referenced mail (section 4.10, page 212)

`yourref`
: recipient's reference (section 4.10, page 212)

`zipcodeseparator`
: separator between the title of ZIP code (postal code) and the code itself (section 4.10, page 202)

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

The `\setkomavar` command sets the *`content`* of the *`name`* variable. Using the optional argument, you can change the *`description`* of the variable at the same time. In contrast, `\setkomavar*` sets only the *`description`* of the *`name`* variable.

Example: It is customary for letters to indicate the sender in the letterhead. First, KOMA-Script must know the name of the sender. For "Joe Public" that would be done with:

```
\setkomavar{fromname}{Joe Public}
```

The default for the description of the sender is "From". Assuming, however, that Mr Public wants to have "Sender" in the places where KOMA-Script outputs his name, he would have to add

```
\setkomavar*{fromname}{Sender}
```

or combine the two commands into one:

```
\setkomavar{fromname}[Sender]{Joe Public}
```

He thus kills two birds with one stone, so to speak.

By the way, you can delete the content of the variable by providing an empty *content* argument. Of course, you can delete the *description* of the variable in the same way, with an empty argument for the description.

Example: Suppose Mr Public wants to have no label for the name of the sender. He can either delete it for himself with

```
\setkomavar*{fromname}{}
```

or he could again kill two birds with one stone and use

```
\setkomavar{fromname}[]{Joe Public}
```

This will simultaneously set the contents of the variable and delete its description.

```
\usekomavar[command]{name}
\usekomavar*[command]{name}
```

v2.9i

In some cases, it is necessary to access the content or description of a variable and not to leave this solely to the class. This is especially important if 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 *name* variable, whereas the starred version `\usekomavar*` allows you to access the description or title. In section 22.1, page 502 you can find more information about defining your own variables.

```
\Ifkomavar{name}{then code}{else code}
```

v3.08

With this command, you can determine if a variable has already been defined. The *then code* will be executed only if the variable already exists. The variable's contents will not be examined and so can be empty. The *else code* will be executed if the variable does not exist. Such tests can be useful, for example, if your own variables are defined in one `lco` file (see section 4.20 starting at page 240) but used in another `lco` file only if they exist.

```
\Ifkomavarempty{name}{then code}{else code}
\Ifkomavarempty*{name}{then code}{else code}
```

v3.28

With these commands, you can determine whether either the content or the description of a variable is empty. The *then code* will be executed if the expanded content or the expanded description of the *name* variable is empty. Otherwise, the *else code* will be executed. The starred variant tests the variable's description, while the normal variant tests its contents.

4.6. Pseudo-lengths

LaTeX processes lengths with three commands: `\newlength`, `\setlength` and `\addtolength`. Many packages also use macros, which are commands, to store lengths. KOMA-Script extends this method with the ability to process such lengths stored in macros with commands similar to those used to handle real lengths. KOMA-Script calls lengths that are actually stored in macros *pseudo-lengths*.

Note that even though these pseudo-lengths are internally implemented as macros, the commands for pseudo-length management expect only the names of the pseudo-lengths not the macros representing the pseudo-lengths. The names of pseudo-lengths are written without the initial backslash, like the names of LaTeX counters and unlike macros or LaTeX lengths.

Historical 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 defined with `\newlength`, skip registers are used. Once all these registers are in use, you can not define any more lengths. Both scrlttr2 and scrletter would normally use more than 20 of these 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 these precious registers would already be in use. For this reason, in 2002 scrlttr2 stores letter-specific lengths in macros instead of lengths.

Anyone who wants to argue that the recommended LaTeX installation with ε-TeX, which is required for KOMA-Script anyway, no longer suffers from the above-mentioned limitation would be right. However, that improvement came too late for scrlttr2. With scrletter, the concept of psuedo-lengths was adopted for reasons of compatibility.

The pseudo-lengths defined and uses by KOMA-Script are listed in table 4.2, which also provides cross references to the detailed descriptions of each pseudo-lengths in the following sub-sections.

A schematic display of the most important distances of the letterhead page is shown in figure 4.1 on page 165. In addition to the pseudo-lengths for the configurable distances, some non-configurable lengths are also shown in light gray. For the sake of clarity, however, some rarely required pseudo-lengths have been omitted.

Table 4.2.: Pseudo-lengths provided by scrlttr2 and scrletter

backaddrheight	the height of the return address at the upper edge of the address field (section 4.10.3, page 207)
bfoldmarklength	the length of the bottommost fold mark (section 4.10.1, page 186)

...

Table 4.2.: Pseudo-lengths provided by scrlttr2 and scrletter (*continued*)

`bfoldmarkvpos`
: the vertical distance of the bottommost fold mark from the top edge of the paper (section 4.10.1, page 186)

`firstfoothpos`
: the horizontal distance of the letterhead page footer from the left edge of the paper; values greater than the width of the paper or less than the negative value of the width of the paper activate special handling (section 4.10.8, page 224)

`firstfootvpos`
: the vertical distance of letterhead page footer from the top edge of the paper (section 4.10.8, page 224)

`firstfootwidth`
: the width of the letterhead page footer (section 4.10.8, page 224)

`firstheadhpos`
: the horizontal distance of the letterhead from the left edge of the paper; values greater than the width of the paper or less than the negative value of the width of the paper activate special handling (section 4.10.2, page 189)

`firstheadvpos`
: the vertical distance of the letterhead from the top edge of the paper (section 4.10.2, page 189)

`firstheadwidth`
: the width of the letterhead (section 4.10.2, page 189)

`foldmarkhpos`
: the horizontal distance of the horizontal fold marks from the left edge of the paper (section 4.10.1, page 188)

`foldmarkvpos`
: the vertical distance of the vertical fold marks from the top edge of the paper (section 4.10.1, page 188)

`fromrulethickness`
: the thickness of an optional horizontal rule in the letterhead (section 4.10.2, page 194)

`fromrulewidth`
: the length of an optional horizontal rule in the letterhead (section 4.10.2, page 194)

...

Table 4.2.: Pseudo-lengths provided by scrlttr2 and scrletter (*continued*)

`lfoldmarkhpos`
: the horizontal distance of the vertical fold mark from the left edge of the paper (section 4.10.1, page 188)

`lfoldmarklength`
: the length of the vertical fold mark (section 4.10.1, page 188)

`locheight`
: the height of the field containing the additional sender information if the value is not 0; if it is 0, `toaddrheight` is used instead (section 4.10.4, page 209)

`lochpos`
: the horizontal distance of the field containing the additional sender information; if the value is positive, the distance is measured from the right paper edge; if negative, from the left paper edge; if 0, the negative value of `toaddrhpos` is used instead (section 4.10.4, page 209)

`locvpos`
: the vertical distance of the field containing the additional sender information from the top edge of the paper if the value is not 0; if it is 0, `toaddrvpos` is used instead (section 4.10.4, page 209)

`locwidth`
: the width of the field containing the additional sender information; if it is 0, the width is calculated automatically based on the `locfield` option described in section 4.10, page 208 (section 4.10.4, page 209)

`mfoldmarklength`
: the length of the middle horizontal fold mark (section 4.10.1, page 188)

`mfoldmarkvpos`
: the vertical distance of the middle horizontal fold mark from the top edge of the paper (section 4.10.1, page 186)

`pfoldmarklength`
: the length of the hole-punch mark (section 4.10.1, page 188)

`PPdatamatrixvskip`
: the vertical distance between the postpaid header and the data array with `addrfield=PP` (section 4.10.3, page 207)

...

Table 4.2.: Pseudo-lengths provided by scrlttr2 and scrletter (*continued*)

PPheadheight
: the height of the postpaid header (section 4.10.3, page 207)

PPheadwidth
: the width of the left postpaid field with addrfield=PP (section 4.10.3, page 207)

refaftervskip
: vertical skip below reference-field line (section 4.10.5, page 215)

refhpos
: the horizontal distance of reference-field line from the left edge of the paper; if the value is 0, the reference-field line is centred horizontally on the letterhead page (section 4.10.5, page 214)

refvpos
: the vertical distance of reference-field line from the top edge of the paper (section 4.10.5, page 214)

refwidth
: the width of the reference-field line (section 4.10.5, page 214)

sigbeforevskip
: the vertical skip between the closing and the signature (section 4.10.7, page 221)

sigindent
: the indentation of the signature with respect to the text body (section 4.10.7, page 221)

specialmailindent
: the left indentation of the delivery method within the address field (section 4.10.3, page 207)

specialmailrightindent
: the right indentation of the delivery method within the address field (section 4.10.3, page 207)

subjectaftervskip
: the vertical skip after the subject (section 4.10.6, page 219)

subjectbeforevskip
: additional vertical skip before the subject (section 4.10.6, page 219)

...

Table 4.2.: Pseudo-lengths provided by scrlttr2 and scrletter (*continued*)

`subjectvpos`	the vertical distance of the subject from the top edge of the paper; if it is 0, the position is calculated based on the `subject` option (section 4.10.6, page 219)
`tfoldmarklength`	the length of the topmost horizontal fold mark (section 4.10.1, page 188)
`tfoldmarkvpos`	the vertical distance of the topmost horizontal folding mark from the top edge of the paper (section 4.10.1, page 186)
`toaddrheight`	the height of the address field (section 4.10.3, page 206)
`toaddrhpos`	the horizontal distance of the address field from left edge of the paper, if the value is positive; if it is negative, the negative horizontal distance of the address field from the right edge of the paper (section 4.10.3, page 205)
`toaddrindent`	the left and right indentation of the address within the address field (section 4.10.3, page 206)
`toaddrvpos`	the vertical distance of the address field from the the top edge of the paper (section 4.10.3, page 205)
`toaddrwidth`	the width of the address field (section 4.10.3, page 206)

`\newplength{`*name*`}`

v3.26

This command defines a new pseudo-length. The new pseudo-length is uniquely identified by its *name*. Each name can therefore be assigned only once. If you attempt to redefine an existing pseudo-length, the commands exits with an error message.

Since the ordinary user does not normally need to define pseudo-lengths, this command was not a user instruction until KOMA-Script 3.26. Before then, `\@newplength` existed with the same functionality. This instruction still exists for package authors.

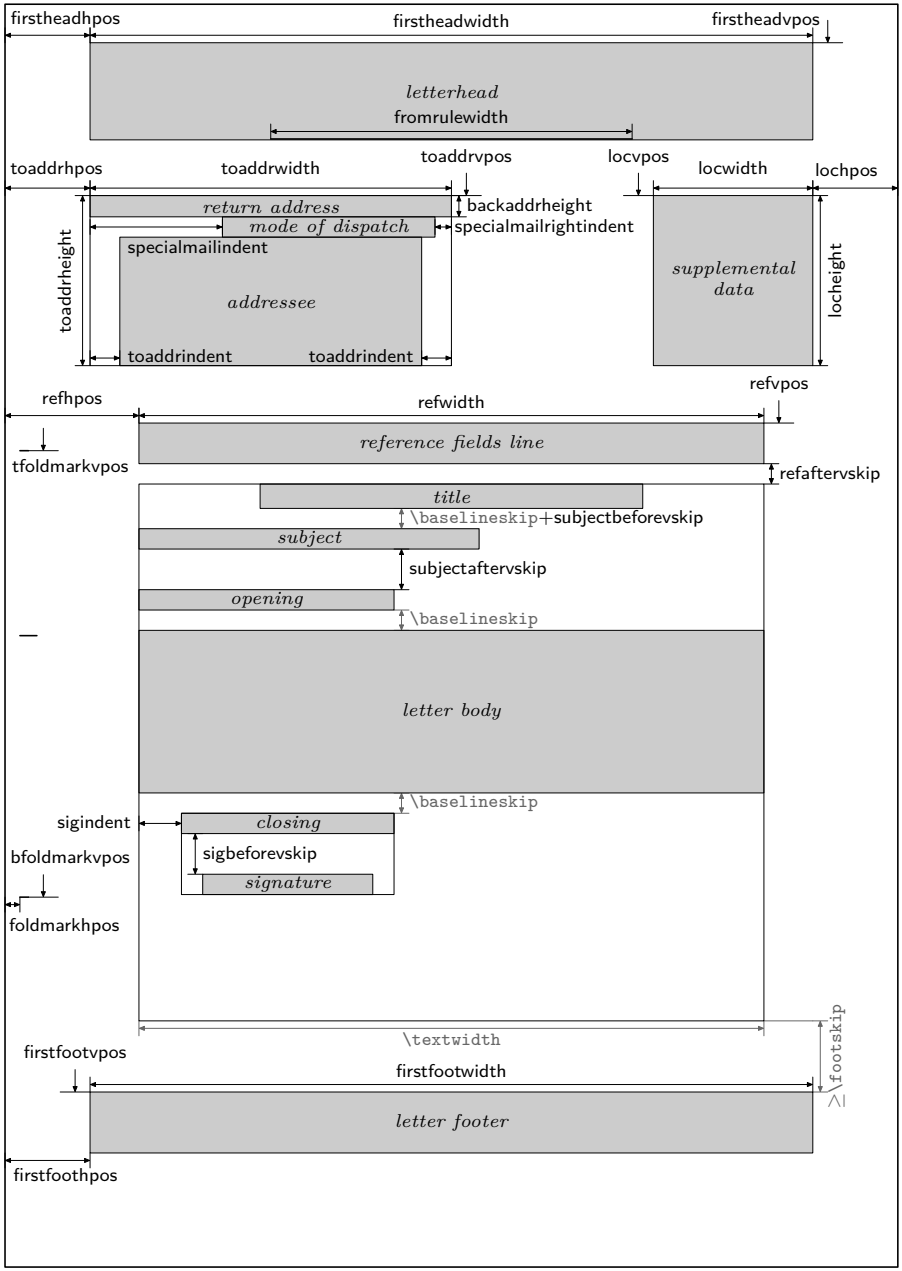


Figure 4.1.: Schematic of the pseudo-lengths for a letter

\Ifplength{*pseudo-length*}{*then-code*}{*else-code*}

v3.27 This command can be used to determine whether a *pseudo-length* has been defined. The *then-code* is executed if the *pseudo-length* is defined and not \relax. Otherwise the *else-code* is executed.

For reasons of consistency only, the internal command \if@plength, with the identical meaning, exists for the use of package authors.

\useplength{*name*}

Using this command you can access the value of the pseudo-length of 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 4.20 ab page 240).

\setplength[*factor*]{*pseudo-length*}{*value*}
\addtoplength[*factor*]{*pseudo-length*}{*value*}

Using \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 4.6, page 167).

With \addtoplength you can add the multiple of a *value* to a *pseudo-length*. Again, you can pass a *factor* as an optional argument.

To assign or to add the multiple of one *pseudo-length* to another pseudo-length, use the \useplength command (see section 4.6, page 166) within the *value*. To subtract the value of one *pseudo-length* from another *pseudo-length*, you use should use at the same time a minus sign or -1 as the *factor*.

Since the ordinary user does not normally need to define pseudo-lengths, these commands were not user instructions until KOMA-Script 3.26. Before then, \@setplength and \@addtoplength existed with the same functionality. These commands still exist for the use of package authors.

\setplengthtowidth[*factor*]{*pseudo-length*}{*content*}
\setplengthtoheight[*factor*]{*pseudo-length*}{*content*}
\setplengthtodepth[*factor*]{*pseudo-length*}{*content*}
\setplengthtototalheight[*factor*]{*pseudo-length*}{*content*}

v3.26 The first three commands essentially correspond with \settowidth, \settoheight, and \settodepth from the LaTeX kernel, but set *pseudo-length*s instead of lengths. Like \setplength, these commands extend their LaTeX kernel equivalents with an optional *factor*. They set a *pseudo-length* to the natural width, height or depth of the given *content*, multiplied by the optional *factor*. The additional command \setplengthtototalheight sets the *pseudo-length* to the sum of the height and depth of *content* multiplied by the optional *factor*.

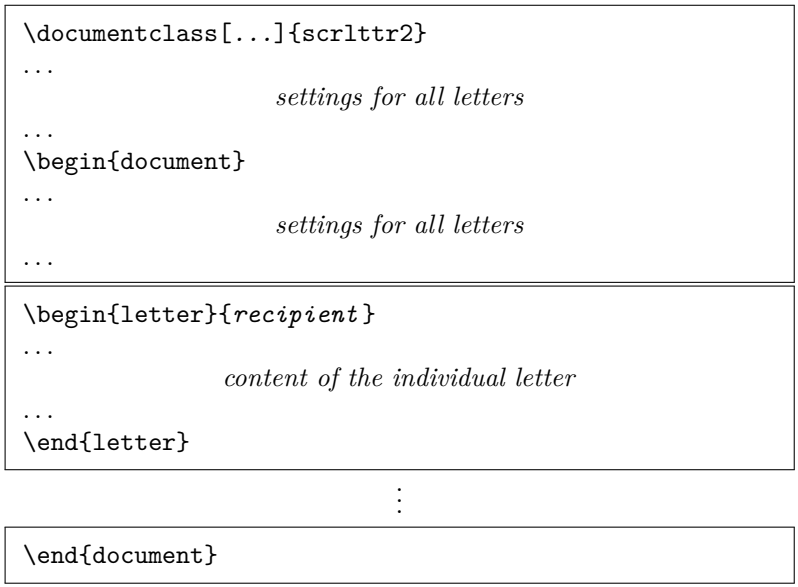


Figure 4.2.: General structure of a letter document containing several individual letters (the structure of a single letter is shown in figure 4.3)

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

With the `\setlengthtoplength` command, you can assign a multiple of a *pseudo-length* to a real *length*. The *factor* is given as an optional argument instead of directly preceding the *pseudo-length*. You should also use this command when you want to assign the negative of a *pseudo-length* to a *length*. In this case, you can use either a minus sign or `-1` as the *factor*. The `\addtolengthplength` command works very similarly. It adds the *pseudo-length* multiplied by the *factor* to the *length*.

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 usually contains only one book, a letter document can contain several letters. As illustrated in figure 4.2, a letter document consists of a preamble, the individual letters, and the closing.

The preamble contains all the settings that apply generally to all letters. Most of them can also be overwritten in the settings of the individual letters. The only setting which cannot currently be changed within a single letter is the version of scrlttr2 for which compatibility is required (see the `version` option in section 4.2, page 153).

If you use scrletter, the only difference is that another class is loaded, with `\usepackage{scrletter}` added before the settings for all letters. For setting options with scrletter, see section 4.1, on page 151.

```
\begin{letter}[options]{recipient}
...
                    settings for this letter
...
\opening{salutation}
```

```
...
                        letter text
...
```

```
\closing{concluding text}
\ps
...
                        postscript
...
\encl{enclosures}
\cc{additional recipients}
\end{letter}
```

Figure 4.3.: General structure of a single letter within a letter document (see also figure 4.2)

I recommended that you place only general settings such as loading packages and setting options before `\begin{document}`. You should initialise all variables or other textual features after `\begin{document}`, particularly when you use the babel package (see [BB13]) or change language-dependent variables of scrlttr2.

The closing usually consists only of `\end{document}`. Of course you can also add additional comments at this point.

As detailed in figure 4.3, individual letters each consist of an introduction, the body of the letter, and the closing. In the introduction, all settings pertaining to the current letter alone are defined. It is important that this introduction always ends with `\opening`. Similarly, the closing always starts with `\closing`. The *opening* and *closing* arguments of the two commands can be left empty, but both commands must be used and must have an argument.

Note that you can change additional settings between the individual letters. Such changes then apply to all subsequent letters. However, to keep your letter documents clear and maintainable, you should think carefully before actually placing further general settings of limited scope between the letters. I cannot recommend this practice. However, if you use scrletter2, there is nothing wrong with inserting additional parts of the document between or after letters that should not be in the same scope. For example, you can combine a cover letter and a CV in one document.

\begin{letter}[*options*]{*recipient*}...\end{letter}

The `letter` environment is one of the key environments of the letter class. A noteworthy feature of scrlttr2 and scrletter is that they can provide additional *options* for the `letter` environment. These *options* are executed internally using the `\KOMAoptions` command.

The *recipient*, or addressee, is a mandatory argument passed to the `letter` environment and includes both the name and the address of the recipient of the letter. Double backslashes serve to separate the individual parts of the address. These parts are output on individual lines in the address field. Nevertheless, you should not interpret the double backslash as a mandatory line break. Vertical material such as new paragraphs or vertical space is not permitted within the address. They can lead to unexpected results and error messages. Incidentally, this is the same for the standard letter class.

Example: Suppose you want to write a letter to Joanna Public. A minimalist letter document would look like this:

```
\documentclass[version=last]{scrlttr2}
\usepackage[british]{babel}
\begin{document}
\begin{letter}{Joanna Public\\
    1 Hillside\\
    SAMPLESTEAD\\
    WX12 3YZ}
\end{letter}
\end{document}
```

However, this would not result in any output. It would not even print the recipient on the letterhead page. Why this is the case is explained in the description of the `\opening` command on page 171.

v3.27 Letters are always printed in single-column mode and without vertical adjustment. You can use `\flushbottom`, explained in section 3.4 on page 57, together with `\AtBeginLetter` to force a vertical adjustment.

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

As mentioned in [Tea06], LaTeX lets the user declare additional *code* to be executed at certain points in a LaTeX run. For this purpose, the LaTeX kernel provides, for example, `\AtBeginDocument` and `\AtEndOfClass`. Such points are called *hooks*. The scrlttr2 class and the scrletter package provide two additional hooks. You can declare the *code* for these using v2.95 `\AtBeginLetter` and `\AtEndLetter`. Originally, hooks were intended for package and class authors, so they are documented only in [Tea06] and not in [Tea05b]. However, with letters there are useful applications at the user level for both new hooks, as the following example illustrates.

Example: Suppose you have several letters in a document that use their own commands to insert a questionnaire in the letters. The questions are numbered automatically using a counter. Since KOMA-Script is unaware of this counter, it would not be reset at the start of each new letter, unlike the page number. If each questionnaire contains ten questions, the first question in the fifth letter would get the number 41. You can solve this problem by telling KOMA-Script to reset this counter at the beginning of each 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 in this example requires the tabularx package (see [Car99b]).

```
letter
\thisletter
\letterlastpage
```

v3.19 If you have more than one letter in a document, it is useful to have a letter number. For this purpose, KOMA-Script has provided the `letter` counter, which is incremented by one at each `\begin{letter}`, since version 3.19.

Example: Let's return to the `\AtBeginLetter` example. Instead of resetting the counter explicitly at `\begin{letter}`, we can do so implicitly by defining counter `Question` to depend on counter `letter`:

```
\newcounter{Question}[letter]
\newcommand{\Question}[1]{%
  \refstepcounter{Question}\par
  \noindent\begin{tabularx}{\textwidth}{l@{}X}
    \theQuestion:~ & #1\\
  \end{tabularx}%
}%
```

Now the new counter will be reset at every start of each letter so that the first question in each letter will be number one.

If you want to display the current value of `letter`, this is possible, as usual, with `\theletter`. The counter can also be used for cross-references. So you can use `\label{`*name*`}` to generate a label immediately after `\begin{letter}` and reference it somewhere in the document using `\ref{`*name*`}`. Inside the same letter you can get the same result by simply using `\thisletter` without creating a label.

For labels in form letters, it is necessary to give them a unique name across all letters. Once again, you can use `\thisletter` for this purpose. KOMA-Script also uses `\thisletter` internally to put a label on the last page of each letter. This makes it possible to use `\letterlastpage` to reference the number of the last page of the current letter at any point within the letter. Since `\letterlastpage` uses `\label` and `\pageref`, it is only valid after several LaTeX runs — usually two or three. If you use `\letterlastpage`, pay attention to the *Rerun* messages in the terminal output or `log` file messages about labels that have been changed.

`\opening{`*salutation*`}`

This is one of the most important commands for letters. On the surface, it may seem that only the *salutation*, for example "Dear Mrs ...", is printed. Actually, this command also prints the fold marks, the letterhead, the address, the extra sender information, the reference line, the title, the subject, and the footer. In short, without `\opening` there is no letter. If, in fact, you want to print a letter without a salutation, you have to use an `\opening` command with an empty argument.

Example: Let's return to the example of page 169 and add a salutation:

```
\documentclass[version=last]{scrlttr2}
\usepackage[british]{babel}
\begin{document}
\begin{letter}{%
    Joanna Public\\
    1 Hillside\\
    SAMPLESTEAD\\
    WX12 3YZ%
  }
\opening{Dear Madam Chair,}
\end{letter}
\end{document}
```

This will result in the letterhead shown in figure 4.4.

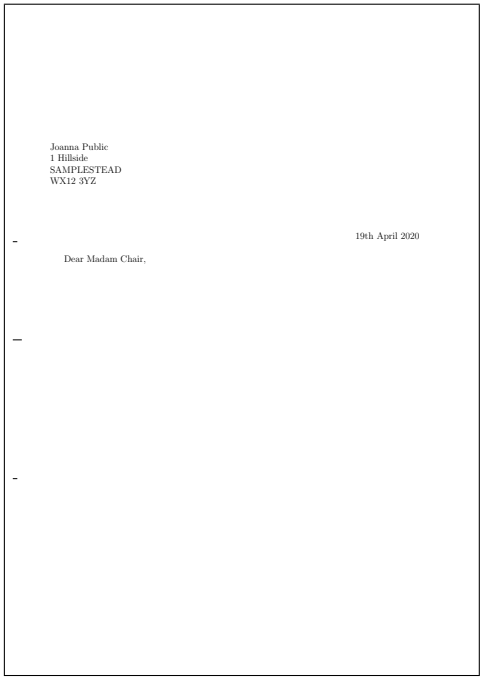


Figure 4.4.: result of a minimalist letter with recipient and salutation only (the date is set by default)

`\closing{`*`concluding text`*`}`

The main purpose of the command `\closing` is to typeset the *`concluding text`*. This can even consist of multiple lines. In that case, the individual lines should be separated by a double backslash. Paragraph breaks inside the *`concluding text`* are not allowed.

In addition, this command also prints the content of the `signature` variable. You can find more information about the signature and its configuration in section 4.10.7 on page 219.

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

```
\documentclass[version=last]{scrlttr2}
\usepackage[british]{babel}
\begin{document}
\begin{letter}{%
    Joanna Public\\
    1 Hillside\\
    SAMPLESTEAD\\
    WX12 3YZ%
  }
\opening{Dear Madam Chair,}
The last general meeting was more than a year ago.
I would like to remind you that the articles of our
club stipulate that one should be held every
```

Figure 4.5.: result of a short letter with recipient, opening, text, and closing (the date is set by default)

```
six months. For this reason, I call on the executive
board to arrange such a meeting immediately.
\closing{Anticipating an invitation}
\end{letter}
\end{document}
```

This will result in a the letter shown in figure 4.5.

\ps

This instruction merely switches to the postscript. To do so, a new paragraph begins, and a vertical gap — usually below the signature — is inserted. The `\ps` text can be followed by any text. If you want the postscript to be introduced with the acronym "PS:", which in most current usage is written without full stops, you have to type this yourself. This abbreviation is printed neither automatically nor optionally by the scrlttr2 class.

Example: The sample letter with the addition of a postscript

```
\documentclass[version=last]{scrlttr2}
\usepackage[british]{babel}
\begin{document}
\begin{letter}{%
    Joanna Public\\
```

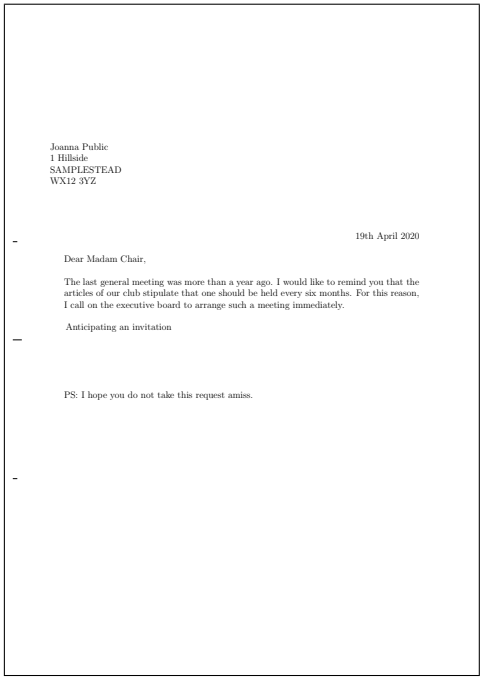


Figure 4.6.: result of a short letter with recipient, opening, text, closing, and postscript (the date is set by default)

```
    1 Hillside\\
    SAMPLESTEAD\\
    WX12 3YZ%
  }
\opening{Dear Madam Chair,}
The last general meeting was more than a year ago.
I would like to remind you that the articles of our
club stipulate that one should be held every
six months. For this reason, I call on the executive
board to arrange such a meeting immediately.
\closing{Anticipating an invitation}
\ps PS: I hope you do not take this request amiss.
\end{letter}
\end{document}
```

results in figure 4.6.

When letters were written still by hand, it was quite common to use a postscript because this was the only way to add information which had been forgotten in the main part of the letter. For letters written with LaTeX, of course, you can easily insert additional lines. Nevertheless, postscripts remain popular. They can be useful to emphasize important points once more, or even to mention the less important matters.

\cc{*distribution list*}
\setkomavar{ccseparator}[*description*]{*contents*}

You can print a *distribution list* with the \cc command. The command takes the *distribution list* as its argument. If the content of the variable ccseparator is not empty, the name and content of this variable are inserted before the *distribution list*. In this case, the *distribution list* will be indented appropriately. It's a good idea to set the *distribution list* \raggedright and to separate the individual entries with a double backslash.

Example: This time, the sample letter should go not only to the chairman but also to all club members:

```
\documentclass[version=last]{scrlttr2}
\usepackage[british]{babel}
\begin{document}
\begin{letter}{%
    Joanna Public\\
    1 Hillside\\
    SAMPLESTEAD\\
    WX12 3YZ%
  }
\opening{Dear Madam Chair,}
The last general meeting was more than a year ago.
I would like to remind you that the articles of our
club stipulate that one should be held every
six months. For this reason, I call on the executive
board to arrange such a meeting immediately.
\closing{Anticipating an invitation}
\ps PS: I hope you do not take this request amiss.
\cc{executive board\\all members}
\end{letter}
\end{document}
```

The result is shown in figure 4.7.

A vertical gap is inserted automatically before the distribution list.

\encl{*enclosures*}
\setkomavar{enclseparator}[*description*]{*contents*}

The *enclosures* have the same structure as the distribution list. The only difference is that the list of enclosures begins with the name and content of the enclseparator variable.

Example: To the sample letter we will attach an excerpt from the club's articles of association. These will be added as an enclosure. Because there is only one enclosure, we change the description title accordingly:

Figure 4.7.: result of a short letter with recipient, opening, text, closing, postscript, and distribution list (the date is set by default)

```
\documentclass[version=last]{scrlttr2}
\usepackage[british]{babel}
\begin{document}
\begin{letter}{%
    Joanna Public\\
    1 Hillside\\
    SAMPLESTEAD\\
    WX12 3YZ%
  }
\opening{Dear Madam Chair,}
The last general meeting was more than a year ago.
I would like to remind you that the articles of our
club stipulate that one should be held every
six months. For this reason, I call on the executive
board to arrange such a meeting immediately.
\closing{Anticipating an invitation}
\ps PS: I hope you do not take this request amiss.
\setkomavar*{enclseparator}{Enclosure}
\encl{Excerpt from the articles governing general
  meetings}
\cc{executive board\\all members}
\end{letter}
\end{document}
```

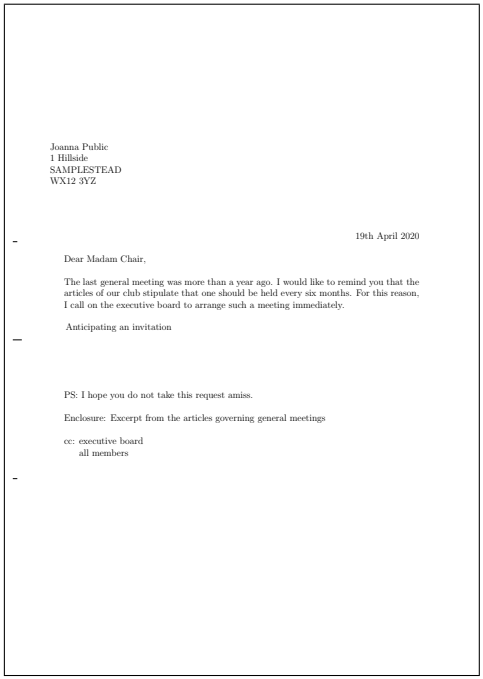


Figure 4.8.: result of a short letter with recipient, opening, text, closing, postscript, distribution list, and enclosure (the date is set by default)

This will result in figure 4.8.

4.8. Choosing the Document Font Size

scrlttr2 The information in section 3.5 applies equally to scrlttr2. By contrast, the scrletter package by itself does not offer font-size selection but depends completely on the class you use. So if you have already read and understood section 3.5, you can continue to page 178 at the end of this section. If you use scrletter, you can skip directly to section 4.9, page 179.

`fontsize=`*size*

scrlttr2 While the standard classes support only a very limited number of font sizes, scrlttr2 provides the ability to specify any *size* for the main font. You can also use any known TEXunit as a unit for the *size*. If the *size* is specified without a unit, it is assumed to be `pt`.

If you set the option within the document, the main font size and the dependent font sizes of the commands `\tiny`, `\scriptsize`, `\footnotesize`, `\small`, `\normalsize`, `\large`, `\Large`, `\LARGE`, `\huge` and `\Huge` are changed. This can be useful, for example, if you want another letter to be set in a smaller font size.

Note that using this option after loading the class does not automatically recalculate the type area and margins (see `\recalctypearea`, section 2.6, page 40). However, if this recalculation

is performed, it will be based on the current main font size. The effects of changing the main font size upon other loaded packages or the class used depends on these packages and on the class. You can encounter errors which are not the fault of KOMA-Script, and further, the scrlttr2 class itself does not recalculate all lengths if the main font size changes after loading the class.

This option should by no means be misinterpreted as a substitute for `\fontsize` (see [Tea05a]). Also, you should not use it in place of one of the font size commands that are relative to the main font, from `\tiny` to `\Huge`. For scrlttr2 the default is `fontsize=12pt`.

Example: Suppose the organization in the sample letter is the *"Friends of Noxious Font Sizes"*, for which reason it should be set in 14 pt instead of 12 pt. You can achieve this by making a small change to the first line:

```
\documentclass[version=last,fontsize=14pt]{scrlttr2}
\usepackage[british]{babel}
\begin{document}
\begin{letter}{%
    Joanna Public\\
    1 Hillside\\
    SAMPLESTEAD\\
    WX12 3YZ%
  }
\opening{Dear Madam Chair,}
The last general meeting was more than a year ago.
I would like to remind you that the articles of our
club stipulate that one should be held every
six months. For this reason, I call on the executive
board to arrange such a meeting immediately.
\closing{Anticipating an invitation}
\ps PS: I hope you do not take this request amiss.
\setkomavar*{enclseparator}{Enclosure}
\encl{Excerpt from the articles governing general
  meetings}
\cc{executive board\\all members}
\end{letter}
\end{document}
```

Alternatively, the option could be set as an optional argument to `letter`:

```
\documentclass[version=last]{scrlttr2}
\usepackage[british]{babel}
\begin{document}
\begin{letter}[fontsize=14pt]{%
    Joanna Public\\
    1 Hillside\\
    SAMPLESTEAD\\
```

```
    WX12 3YZ%
  }
\opening{Dear Madam Chair,}
The last general meeting was more than a year ago.
I would like to remind you that the articles of our
club stipulate that one should be held every
six months. For this reason, I call on the executive
board to arrange such a meeting immediately.
\closing{Anticipating an invitation}
\ps PS: I hope you do not take this request amiss.
\setkomavar*{enclseparator}{Enclosure}
\encl{Excerpt from the articles governing general
  meetings}
\cc{executive board\\all members}
\end{letter}
\end{document}
```

Since the text area is not recalculated in this late change of the font size, the two results differ in figure 4.9.

4.9. Text Markup

The information in in section 3.6 largely applies to this chapter. So if you have already read and understood section 3.6, you can limit yourself to examining table 4.3, page 181 and then skip ahead to section 4.10, page 183.

LaTeX offers different possibilities for logical and direct markup of text. For more information about the standard font facilities, see [OPHS11], [Tea05b], and [Tea05a].

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

With the help of the `\setkomafont` and `\addtokomafont` commands, you can attach particular font styling *commands* that change the appearance of a given *element*. Theoretically, all statements, including literal text, can be used as *commands*. You should, however, limit yourself to those statements that really change font attributes only. These are usually commands like `\rmfamily`, `\sffamily`, `\ttfamily`, `\upshape`, `\itshape`, `\slshape`, `\scshape`, `\mdseries`, `\bfseries`, `\normalfont`, as well as the font size commands `\Huge`, `\huge`, `\LARGE`, `\Large`, `\large`, `\normalsize`, `\small`, `\footnotesize`, `\scriptsize`, and `\tiny`. You can find these commands explained in [OPHS11], [Tea05b], or [Tea05a]. Colour switching commands like `\normalcolor` (see [Car17] and [Ker07]) are also acceptable. The use of other commands, in particular those that redefine things or or lead to output, is not supported. Strange behaviour is possible in these cases and does not represent a bug.

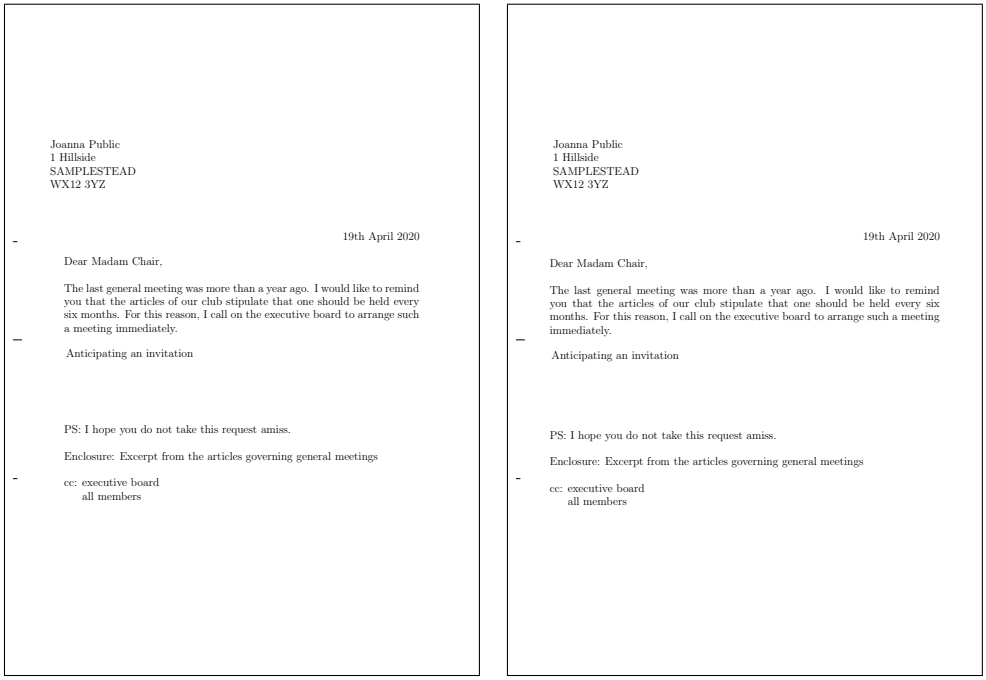


Figure 4.9.: result of a short letter with recipient, opening, text, closing, postscript, enclosures, distribution list, and a noxiously large font (the date is set by default): in the left-hand version, the font size has been defined by the optional argument of `letter`; in the right-hand one, the optional argument of `\documentclass` has been used

The command `\setkomafont` provides an element with a completely new definition of its font styling. In contrast, the `\addtokomafont` command merely extends an existing definition. You should not use either command inside the document body but only in the preamble. For examples of their use, refer to the sections for the respective element. The name and meaning of each element are listed in table 4.3 . The default values can be found in the corresponding sections.

The `\usekomafont` command can be used to switch the current font style to the specified *`Element`*.

You can find a general example that uses both `\setkomafont` and `\usekomafont` in section 3.6 on page 59.

Table 4.3.: Elements whose font style can be changed in the scrlttr2 class or the scrletter package with the `\setkomafont` and `\addtokomafont` commands

`addressee`
: recipient's name and address in the address field (section 4.10, page 202)

`backaddress`
: return address for a window envelope (section 4.10, page 202)

`descriptionlabel`
: label, i. e. the optional argument of `\item`, in a `description` environment (section 4.16, page 237)

`foldmark`
: fold mark on the letterhead page; allows change of line colour (section 4.10, page 184)

`footnote`
: footnote text and marker (section 4.15, page 233)

`footnotelabel`
: footnote marker; applied in addition to the `footnote` element (section 4.15, page 233)

`footnotereference`
: footnote reference in the text (section 4.15, page 233)

`footnoterule`
v3.07 : horizontal rule above the footnotes at the end of the text area (section 3.14, page 92)

`fromaddress`
: sender's address in the letterhead (section 4.10, page 190)

`fromname`
: sender's name in the letterhead, not including `fromaddress` (section 4.10, page 190)

`fromrule`
: horizontal rule in the letterhead; intended for colour changes (section 4.10, page 190)

`labelinglabel`
: labels, i. e. the optional argument of `\item` in the `labeling` environment (see section 4.16, page 237)

...

Table 4.3.: Elements whose font style can be changed (*continued*)

`labelingseparator`
: separator, i. e. the optional argument of the `labeling` environment; applied in addition to the `labelinglabel` element (see section 4.16, page 237)

`pagefoot`
: depending on the page style used after the `pageheadfoot` element for the footer (section 4.13, page 230)

`pagehead`
: depending on the page style used after the `pageheadfoot` element for the header (section 4.13, page 230)

`pageheadfoot`
: the header and footer of a page for all page styles that have been defined using KOMA-Script (section 4.13, page 230)

`pagenumber`
: page number in the header or footer (section 4.13, page 230)

`pagination`
: alternative name for `pagenumber`

`placeanddate`
v3.12
: place and date, if a date line will be used instead of a normal reference line (section 4.10, page 212)

`refname`
: description or title of the fields in the reference line (section 4.10, page 212)

`refvalue`
: content of the fields in the reference line (section 4.10, page 212)

`specialmail`
: delivery type in the address field (section 4.10, page 202)

`lettersubject`
v3.17
: subject in the opening of the letter (section 4.10, page 216)

`lettertitle`
v3.17
: title in the opening of the letter (section 4.10, page 215)

...

Table 4.3.: Elements whose font style can be changed (*continued*)

`toaddress`	variation of the `addressee` element to format the recipient's address, not including the name, in the address field (section 4.10, page 202)
`toname`	variation of the `addressee` element to format the recipient's name in the address field (section 4.10, page 202)

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

v3.12

Sometimes, although this is not recommended, the font setting of an element is used for settings that are not actually related to the font. If you want to apply only the font setting of an element but not those other settings, you can use `\usefontofkomafont` instead of `\usekomafont`. This will activate the font size and baseline skip, the font encoding, the font family, the font series, and the font shape of an element, but no further settings as long as those further settings are local.

You can also switch to a single one of those attributes using one of the other commands. Note that `\usesizeofkomafont` uses both the font size and the baseline skip.

However, you should not take these commands as legitimizing the insertion of arbitrary commands in an element's font setting. To do so can lead quickly to errors (see section 21.5, page 476).

4.10. Letterhead Page

The letterhead page is the first page of, and therefore the signpost for, each letter. In a business context, the paper for this page is often preprinted stationery containing elements such as a header with the sender's information and logo. This header itself is also known as a letterhead. KOMA-Script lets you position these elements freely, and so you can not only replicate the letterhead page directly but also fill in the required fields instantaneously. This free positioning is achieved with pseudo-lengths (see section 4.6 starting on page 160). You can find a schematic representation of the letterhead page and the variables used for it in figure 4.10. The names of the variables are printed in bold to better distinguish the commands from their arguments.

Table 4.4.: Combinable values for configuring fold marks with the `foldmarks` option

`B`	activate bottom horizontal fold mark on left paper edge
`b`	deactivate bottom horizontal fold mark on left paper edge
`H`	activate all horizontal fold marks on left paper edge
`h`	deactivate all horizontal fold marks on left paper edge
`L`	activate left vertical fold mark on upper paper edge
`l`	deactivate left vertical fold mark on upper paper edge
`M`	activate middle horizontal fold mark on left paper edge
`m`	deactivate middle horizontal fold mark on left paper edge
`P`	activate hole-punch or centre mark on left paper edge
`p`	deactivate hole-punch or centre mark on left paper edge
`T`	activate top horizontal fold mark on left paper edge
`t`	deactivate top horizontal fold mark on left paper edge
`V`	activate all vertical fold marks on upper paper edge
`v`	deactivate all vertical fold marks on upper paper edge

Subsequent pages should be distinguished from the letterhead page. For the purposes of this manual, subsequent pages are all pages of a letter except the first one.

4.10.1. Fold Marks

Fold marks, or folding marks, are short horizontal lines at the left edge, and short vertical lines at the upper edge of the paper. KOMA-Script currently supports three configurable horizontal and one configurable vertical fold marks. In addition, there is support for a hole-punch mark, or centre mark, which cannot be shifted vertically.

`foldmarks=`*setting*

The `foldmarks` option activates or deactivates fold marks for two, three, or four vertical divisions and one horizontal division. The individual parts do not have to be of equal size. The positions of three of the four horizontal marks and the single vertical mark are configurable via pseudo-lengths (see section 4.6, page 160).

With the `foldmarks` option, you can either use the default values for simple switches described in table 2.5, page 42 in order to activate or deactivate all configured fold marks on the left and upper edges of the paper at once, or you can configure the individual fold marks independently by specifying one or more letters, as listed in table 4.4. In the latter case, the fold marks are only shown if they have not been deactivated globally with `false`, `off`, or `no`. The exact position of the fold marks is depends on the user settings or the `lco` files (see section 4.20 starting on page 240). The default values are `true` and `TBMPL`.

v2.97e

Example: Suppose you want to deactivate all fold marks except the hole-punch mark. If the default has not already been changed, you can deactivate it as follows:

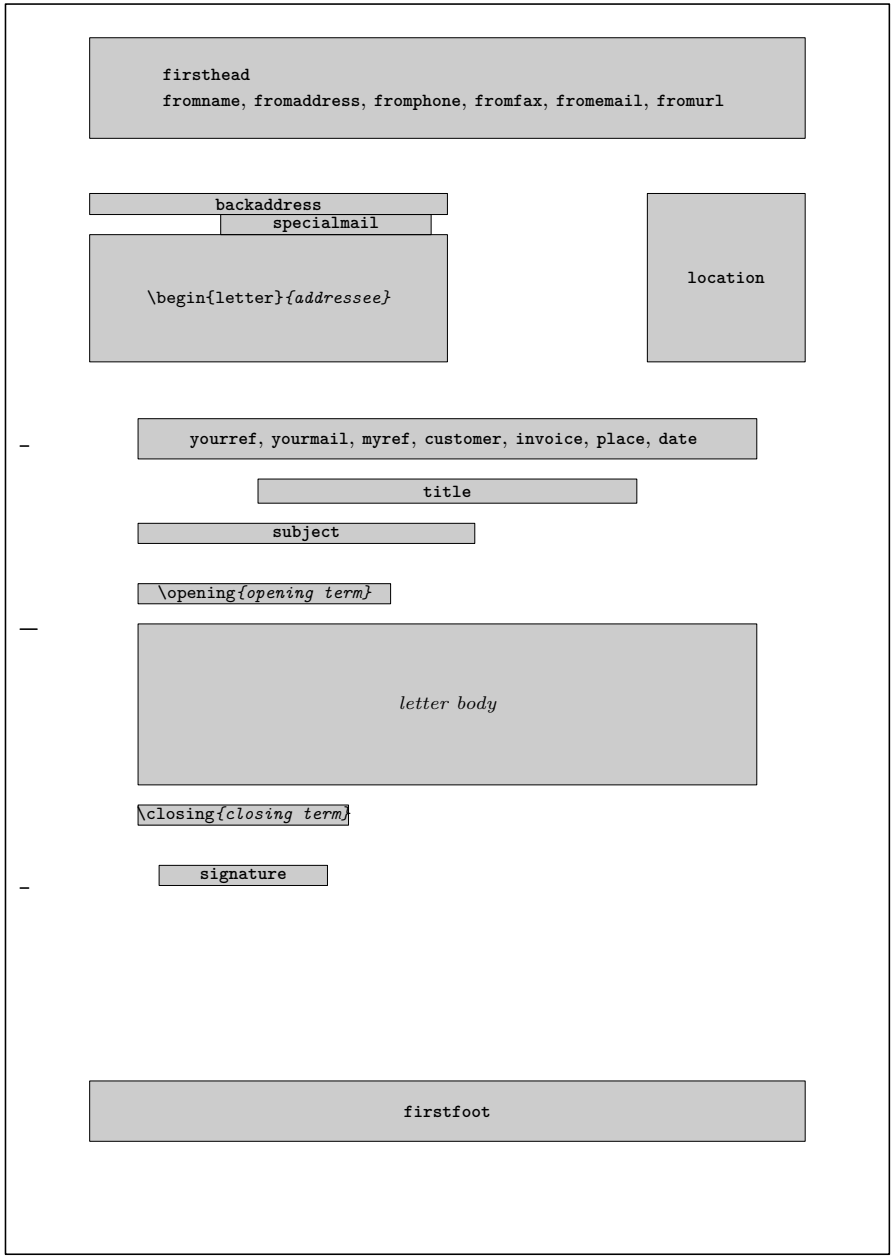


Figure 4.10.: schematic display of the letterhead page outlining the most important commands and variables

```
\KOMAoptions{foldmarks=blmt}
```

If there is a chance that the default has already been changed, you should use a safer method. This changes our example a little bit:

```
\documentclass[foldmarks=true,foldmarks=blmtP,
  version=last]{scrlttr2}
\usepackage[british]{babel}
\begin{document}
\begin{letter}{%
    Joanna Public\\
    1 Hillside\\
    SAMPLESTEAD\\
    WX12 3YZ%
  }
\opening{Dear Madam Chair,}
The last general meeting was more than a year ago.
I would like to remind you that the articles of our
club stipulate that one should be held every
six months. For this reason, I call on the executive
board to arrange such a meeting immediately.
\closing{Anticipating an invitation}
\ps PS: I hope you do not take this request amiss.
\setkomavar*{enclseparator}{Enclosure}
\encl{Excerpt from the articles governing general
  meetings}
\cc{executive board\\all members}
\end{letter}
\end{document}
```

The result is shown in figure 4.11.

v2.97c You can change the colour of the fold mark with the \setkomafont and \addtokomafont commands (see section 4.9, page 179) with the `foldmark` element. The default is no change.

```
\setplength{tfoldmarkvpos}{length}
\setplength{mfoldmarkvpos}{length}
\setplength{bfoldmarkvpos}{length}
```

KOMA-Script recognises a total of three fold marks whose vertical position can be configured. The distance of the top fold mark from the upper edge of the paper is determined by the v2.97e `tfoldmarkvpos` pseudo-length; the distance of the middle fold mark, by the `mfoldmarkvpos` pseudo-length; the distance of the bottommost fold mark, by `bfoldmarkvpos` pseudo-length. With the addition of the hole-punch or centre mark, there is yet a fourth horizontal mark. This one is however always placed at the vertical centre of the paper. There is no pseudo-length for this last mark because its vertical position is not configurable.

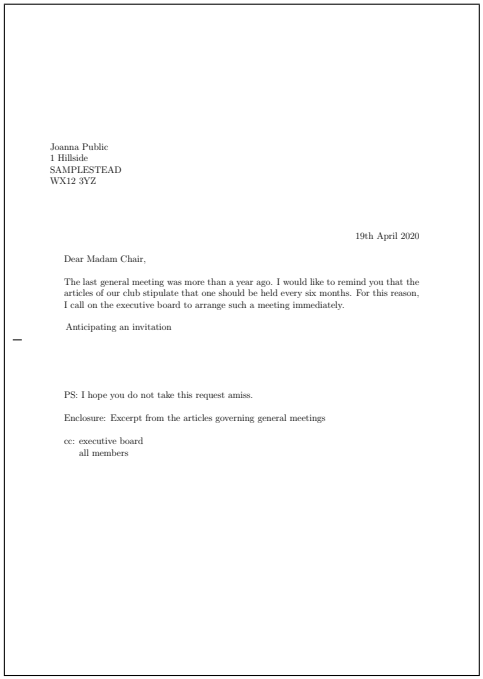


Figure 4.11.: result of a short letter with recipient, opening, text, closing, postscript, distribution list, enclosure, and hole-punch mark (the date is set by default)

The top and bottom fold marks do not serve to divide the paper exactly into equal thirds. Instead, the paper should be folded with their help such that the address field can be seen in a window envelope. The settings are therefore different depending on the `lco` file chosen. Several such files are available offering predefined formats. One format particularly worth noting is `DINmtext`. This format assumes an envelope format of C6/5 (also known as "C6 long"). Letters written with this option are typically not suited for C5 or C4 envelopes.

The middle fold mark is not normally required for Western letters. In Japan, however, a larger number of envelope formats exists, requiring one more fold mark (see the Japanese `lco` files). Note that the terms "top", "middle", and "bottom" fold marks only represent a naming convention. In fact, it is not required that `tfoldmarkvpos` must be smaller than `mfoldmarkvpos`, which in turn must be smaller than `bfoldmarkvpos`. If, though, one of the pseudo-lengths is zero, then the corresponding fold mark will not be set even if the `foldmarks` option (see section 4.10, page 184) is explicitly activated.

\setplength{tfoldmarklength}{*length*}
\setplength{mfoldmarklength}{*length*}
\setplength{bfoldmarklength}{*length*}
\setplength{pfoldmarklength}{*length*}

v2.97e These four pseudo-lengths determine the lengths of the four horizontal fold marks. One feature is particularly worth noting. If the length is given as zero, then the three vertically configurable pseudo-lengths `tfoldmarklength`, `mfoldmarklength` and `bfoldmarklength` are set to 2 mm. The length of the hole-punch mark, `pfoldmarklength`, however, is set to 4 mm.

\setplength{foldmarkhpos}{*length*}

This pseudo-length gives the distance of all horizontal fold marks from the left edge of the paper. Normally, this is 3.5 mm. You can change this value in your own `lco` file if you are using a printer that has a wider unprintable left margin. Whether the fold marks are typeset at all depends on the option `foldmarks` (see section 4.10, page 184).

\setplength{lfoldmarkhpos}{*length*}

v2.97e In addition to the horizontal fold marks, there is also a vertical fold mark. Its distance from the left margin is set via the `lfoldmarkhpos` pseudo-length. This fold mark is used, for example, in Japanese Chou- or You-format envelopes if you want to use them with A4 paper. It can also be useful for envelopes in C6 format.

\setplength{lfoldmarklength}{*length*}

v2.97e The `lfoldmarklength` pseudo-length determines the length of the vertical fold mark. Once again, a feature worth noting is that if the length is given as zero, a length of 4 mm is actually used.

\setplength{foldmarkvpos}{*length*}

v2.97e This pseudo-length determines the distance of all vertical fold marks from the upper edge of the paper. Normally this is 3.5 mm, but you can change the value in your own `lco` file in case your printer has a wider unprintable top margin. Whether or not the foldmarks are actually typeset depends on the `foldmarks` option (see section 4.10, page 184). At present there is only one vertical fold mark, called the left vertical fold mark.

\setplength{foldmarkthickness}{*length*}

v2.97c This pseudo-length determines the thickness of all fold marks. The default is 0.2 pt, in other words a very thin hairline. In particular, if the colour of the fold marks is changed, this may not be enough.

4.10.2. Letterhead

The term letterhead refers here to all of the information concerning the sender that appears above the recipient's address. Normally you would expect that this information would be set through the page-style settings. In fact, this was the case with the old letter class, scrlettr. But scrlttr2 and scrletter output the letterhead independently of the page style by means of the \opening command. The letterhead is positioned absolutely, so that it is 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.

firsthead=*simple switch*

v2.97e

The letterhead is usually the topmost element of the letterhead page. With the firsthead option, you can choose whether the letterhead will be typeset at all. The option accepts the standard values for simple switches given in table 2.5 on page 42. The default is to typeset the letterhead.

\setplength{firstheadvpos}{*length*}

The firstheadvpos pseudo-length gives the distance between the top edge of the paper and the start of the letterhead. This value is set differently in the various predefined lco files. A typical value is 8 mm.

\setplength{firstheadhpos}{*length*}

v3.05

A positive value of the firstheadhpos pseudo-length gives the distance between the left edge of the paper and the start of the letterhead. If the value is actually greater than or equal to the paper width, \paperwidth, the letterhead will be centred horizontally on the letterhead paper. A negative value gives the distance between the right edge of the paper and the right edge of the letterhead. If the value actually less than or equal to the negative value of the width of the paper, the letterhead is placed flush with the left edge of the type area.

The default value is typically \maxdimen, which is the maximum allowed value of a length. This results in horizontal centring.

\setplength{firstheadwidth}{*length*}

The firstheadwidth pseudo-length 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 recipient's address field, it was the width of the type area in KOMAold and has a fixed value of 170 mm in NF.

Table 4.5.: Available values for the `fromalign` option to define the position of the from address in the letterhead with scrlttr2

`center`, `centered`, `middle`	Sender information is centred inside the letterhead; a logo is placed at the beginning of the extra sender information, if applicable; letterhead extensions are activated.
`false`, `no`, `off`	The default design will be used for the sender information; the letterhead extensions are deactivated.
`left`	Sender information is left-aligned in the letterhead; a logo is placed right-aligned, if applicable; letterhead extensions are activated.
`locationleft`, `leftlocation`	Sender information is left-justified and uses the extra sender information; a logo is placed at the top of it, if applicable; the letterhead is automatically deactivated but can be reactivated using the `firsthead` option.
`locationright`, `rightlocation`, `location`	Sender information is right-justified and uses the extra sender information; a logo is placed at the top of it, if applicable; the letterhead is automatically deactivated but can be reactivated using the `firsthead` option.
`right`	Sender information is right-justified; a logo is placed left-justified, if applicable; letterhead extensions are activated

`fromalign=`*`method`*

v2.97e

The `fromalign` option determines where the sender information should be placed on the first page. In addition to the various placement options in the letterhead, you also have the ability to accommodate extra sender information. At the same time, this option serves as a central switch to activate or deactivate 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. Available values for `fromalign` are shown in table 4.5. The default value is `left`.

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

The sender's name is determined by the `fromname` variable. Its *`description`* (see also table 4.7, page 196) is not used in the default letterheads.

In the extended letterhead, you can create a horizontal rule below the sender's name with `fromrule=aftername`. Alternatively you can place this rule below the complete sender in-

Table 4.6.: Available values for the `fromrule` option for the position of the rule in the sender information with scrlttr2

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

formation with `fromrule=afteraddress`. A summary of all available settings for the rule position is shown in table 4.6. The length of this rule is determined by the `fromrulewidth` pseudo-length.

The default for the rule with the extended letterhead is `false`. But in 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 address *`description`* (see also table 4.7) is not used in the default letterheads

You can set the font used for the complete sender information with the `fromaddress` element. You can define modifications to this with the `fromname` element for the sender's name, and with the `fromrule` element for the rule created with the `fromrule` option. The default setting does not change the font. For the rule, font switching is mainly intended to change the colour of the rule, for example to use grey instead of black. See [Ker07] for information about colours.

Example: Now let's give the sender of our sample letter a name:

```
\documentclass[foldmarks=true,foldmarks=blmtP,
  fromalign=false,
  version=last]{scrlttr2}
\usepackage[british]{babel}
\begin{document}
\setkomavar{fromname}{Joe Public}
\setkomavar{fromaddress}{2 Valley\\
                         SAMPLEBY\\
                         ZY32 1XW}
\setkomavar{fromphone}{0\,12\,34~56\,78}
\setkomavar{fromemail}{joe@public.invalid}
\setkomavar{fromlogo}{\includegraphics{musterlogo}}
\begin{letter}{%
    Joanna Public\\
    1 Hillside\\
```

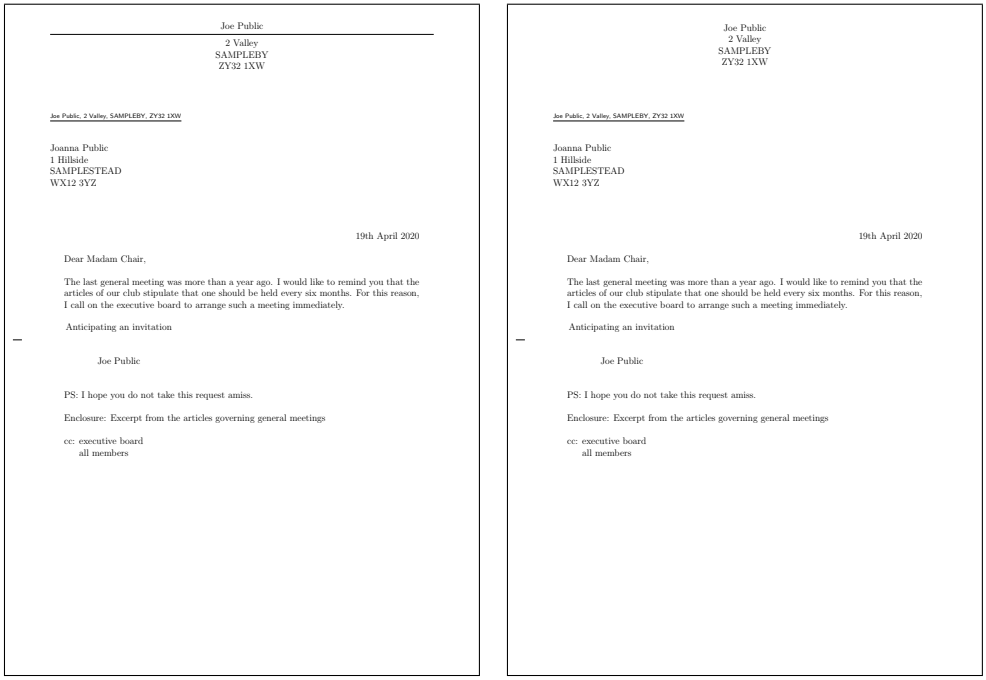


Figure 4.12.: result of a short letter with sender, recipient, opening, text, closing, postscript, distribution list, and enclosure (the date is set by default): on the left, the standard letterhead using `fromalign=false`; on the right, the extended letterhead using `fromalign=center`

```
    SAMPLESTEAD\\
    WX12 3YZ%
  }
\opening{Dear Madam Chair,}
The last general meeting was more than a year ago.
I would like to remind you that the articles of our
club stipulate that one should be held every
six months. For this reason, I call on the executive
board to arrange such a meeting immediately.
\closing{Anticipating an invitation}
\ps PS: I hope you do not take this request amiss.
\setkomavar*{enclseparator}{Enclosure}
\encl{Excerpt from the articles governing general
  meetings}
\cc{executive board\\all members}
\end{letter}
\end{document}
```

Initially, the standard rather than the extended letterhead is used. The result can

be seen in figure 4.12 on the left. For comparison, the same example is shown on the right with `fromalign=center` (that is, with the extended letterhead). You can see that this variation initially has no rule.

For the first time, figure 4.12 also shows a signature below the closing phrase. This is generated automatically from the sender's name. You can find more information about how to configure the signature in section 4.10.7, starting on page 219.

Next, the letter with the extended letterhead should use the `fromrule` option to print a rule below the sender's name:

```
\documentclass[foldmarks=true,foldmarks=blmtP,
  fromalign=center,fromrule=aftername,
  version=last]{scrlttr2}
\usepackage[british]{babel}
\begin{document}
\setkomavar{fromname}{Joe Public}
\setkomavar{fromaddress}{2 Valley\\
                         SAMPLEBY\\
                         ZY32 1XW}
\begin{letter}{%
    Joanna Public\\
    1 Hillside\\
    SAMPLESTEAD\\
    WX12 3YZ%
  }
\opening{Dear Madam Chair,}
The last general meeting was more than a year ago.
I would like to remind you that the articles of our
club stipulate that one should be held every
six months. For this reason, I call on the executive
board to arrange such a meeting immediately.
\closing{Anticipating an invitation}
\ps PS: I hope you do not take this request amiss.
\setkomavar*{enclseparator}{Enclosure}
\encl{Excerpt from the articles governing general
  meetings}
\cc{executive board\\all members}
\end{letter}
\end{document}
```

You can see the result on the right in figure 4.13. By comparison, the same example on the left uses the standard letterhead, which ignores the additional options.

An important note concerns the sender's address: within the sender's address, individual parts such as house number and street, city and postal code, etc., are separated with a double backslash. This double backslash is interpreted differently depending on how the sender's

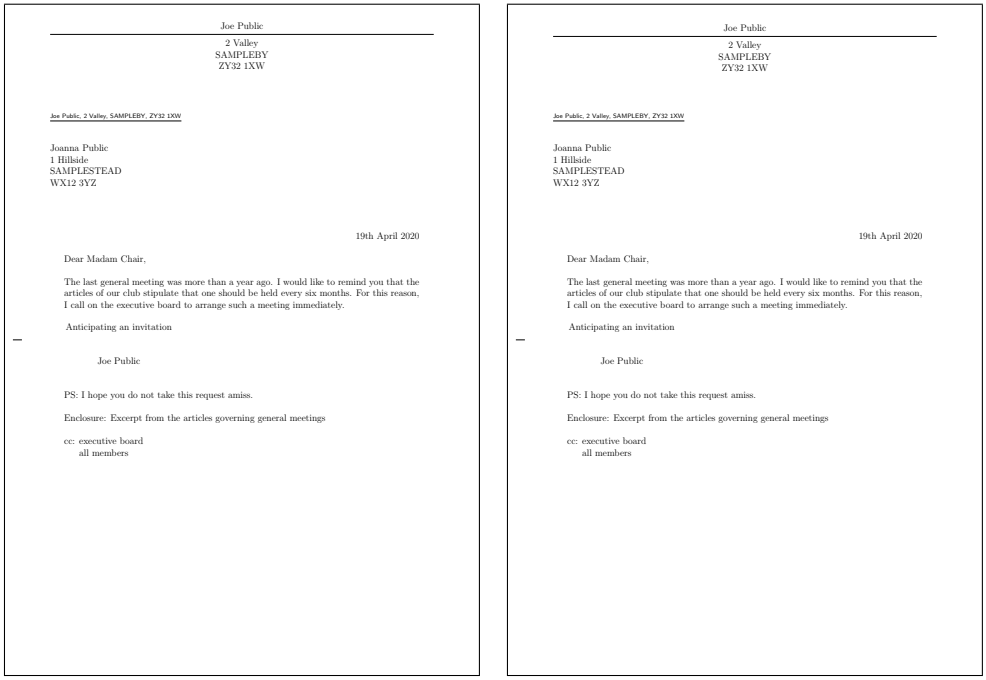


Figure 4.13.: result of a short letter with sender, rule, recipient, opening, text, closing, signature, postscript, distribution list, enclosure and hole-punch mark (the date is set by default): at left one the standard letterhead using `fromalign=false`, at right one the extended letterhead using `fromalign=center`

address is used and therefore is not necessarily a line break. Paragraphs, vertical spacing, and the like are usually not allowed within the sender's information. You have to know KOMA-Script very well to put such things into the sender information, if necessary. In addition, note that if you do so, you should definitely set the variables for return address (see `backaddress`, page 202) and signature (see `signature`, page 219) yourself.

`\setplength{fromrulethickness}{`*length*`}`
`\setplength{fromrulewidth}{`*length*`}`

As mentioned in the explanation of the `fromrule` option in section 4.10, page 190, you can put a horizontal rule within or below the sender's address in the predefined letterheads. If the `fromrulewidth` pseudo-length has a value of 0 pt, which is the default in the predefined `lco` files, the length of this rule is calculated automatically taking into account, for example, letterhead width or an optional logo. You can adjust rule length manually in your own `lco` files by setting this pseudo-length to positive values using `\setplength` (see page 166). The default thickness of the line, `fromrulethickness`, is 0.4 pt.

```
symbolicnames=value
fromphone=simple switch
frommobilephone=simple switch
fromfax=simple switch
fromemail=simple switch
fromurl=simple switch
\setkomavar{fromphone}[description]{contents}
\setkomavar{frommobilephone}[description]{contents}
\setkomavar{fromfax}[description]{contents}
\setkomavar{fromemail}[description]{contents}
\setkomavar{fromurl}[description]{contents}
\setkomavar{phoneseparator}[description]{contents}
\setkomavar{mobilephoneseparator}[description]{contents}
\setkomavar{faxseparator}[description]{contents}
\setkomavar{emailseparator}[description]{contents}
\setkomavar{urlseparator}[description]{contents}
```

v3.12 v3.12 You can use the five options `fromphone`, `frommobilephone`, `fromfax`, `fromemail`, and `fromurl` to specify whether to include the phone number, mobile phone number, fax number, e-mail address, or URL should be as part of the sender's information. You can assign any standard value for simple switches from table 2.5, page 42 to these options. The default for all of them is `false`. The *contents* themselves are determined by the variables of the same name. You can find the defaults for the *description* or title of each variable in table 4.7. You can find the separators that will be inserted between the *description* and the *content* in table 4.8.

v3.12 You can change the defaults for describing the variables all at once with the `symbolicnames` option. This option understands the values for simple switches found in table 2.5, page 42.

v3.27 Activating the option corresponds to *value* `marvosym` and replaces the descriptions from the language-dependent labels of `\emailname`, `\faxname`, `\mobilephonename`, and `\phonename` with symbols from the marvosym package. At the same time, the colon is omitted when defining the separators. In this case, both the description and the content of the URL separator will be empty. With `symbolicnames=fontawesome` or `symbolicnames=awesome`, symbols of package fontawesome are used. In this case there is also a symbol for the URL. Note that you may need to load the marvosym or fontawesome package in your document preamble if you activate the option for the corresponding package for the first time after `\begin{document}`.

Example: Mr Public from our sample letter has a telephone and an e-mail address. He also wants to show these in the letterhead. At the same time, the separation rule should now be placed after the letterhead. So he uses the appropriate options and also sets the required variables:

```
\documentclass[foldmarks=true,foldmarks=blmtP,
  fromalign=false,fromrule=afteraddress,
```

Table 4.7.: Default descriptions of the letterhead variables (you can find the description and contents of the separator variables in table 4.8

fromemail \usekomavar*{emailseparator}\usekomavar{emailseparator}
fromfax \usekomavar*{faxseparator}\usekomavar{faxseparator}
frommobilephone v3.12 \usekomavar*{mobilephoneseparator}\usekomavar{mobilephoneseparator}
fromname \headfromname
fromphone \usekomavar*{phoneseparator}\usekomavar{phoneseparator}
fromurl \usekomavar*{urlseparator}\usekomavar{urlseparator}

```
  fromphone,fromemail,
  version=last]{scrlttr2}
\usepackage[british]{babel}
\begin{document}
\setkomavar{fromname}{Joe Public}
\setkomavar{fromaddress}{2 Valley\\
                         SAMPLEBY\\
                         ZY32 1XW}
\setkomavar{fromphone}{0\,12\,34~56\,78}
\setkomavar{fromemail}{joe@public.invalid}
\begin{letter}{%
    Joanna Public\\
    1 Hillside\\
    SAMPLESTEAD\\
    WX12 3YZ%
  }
\opening{Dear Madam Chair,}
The last general meeting was more than a year ago.
I would like to remind you that the articles of our
club stipulate that one should be held every
six months. For this reason, I call on the executive
board to arrange such a meeting immediately.
\closing{Anticipating an invitation}
\ps PS: I hope you do not take this request amiss.
\setkomavar*{enclseparator}{Enclosure}
\encl{Excerpt from the articles governing general
```

Table 4.8.: Default descriptions and contents of the letterhead separators without the `symbolicnames` option

variable name	description	content
`emailseparator`	`\emailname`	`:~`
`faxseparator`	`\faxname`	`:~`
`mobilephoneseparator`	`\mobilephonename`	`\usekomavaer{phoneseparator}`
`phoneseparator`	`\phonename`	`:~`
`urlseparator`	`\wwwname`	`:~`

```
  meetings}
\cc{executive board\\all members}
\end{letter}
\end{document}
```

The results on the left side of figure 4.14, however, are confounding: the options are ignored. That's because the additional variables and options are only used in the extended letterhead. So the `fromalign` option must be used, as it is in the right side of figure 4.14.

```
\documentclass[foldmarks=true,foldmarks=blmtP,
  fromalign=center,fromrule=afteraddress,
  fromphone,fromemail,
  version=last]{scrlttr2}
\usepackage[british]{babel}
\begin{document}
\setkomavar{fromname}{Joe Public}
\setkomavar{fromaddress}{2 Valley\\
                         SAMPLEBY\\
                         ZY32 1XW}
\setkomavar{fromphone}{0\,12\,34~56\,78}
\setkomavar{fromemail}{joe@public.invalid}
\begin{letter}{%
    Joanna Public\\
    1 Hillside\\
    SAMPLESTEAD\\
    WX12 3YZ%
  }
\opening{Dear Madam Chair,}
The last general meeting was more than a year ago.
I would like to remind you that the articles of our
club stipulate that one should be held every
six months. For this reason, I call on the executive
```

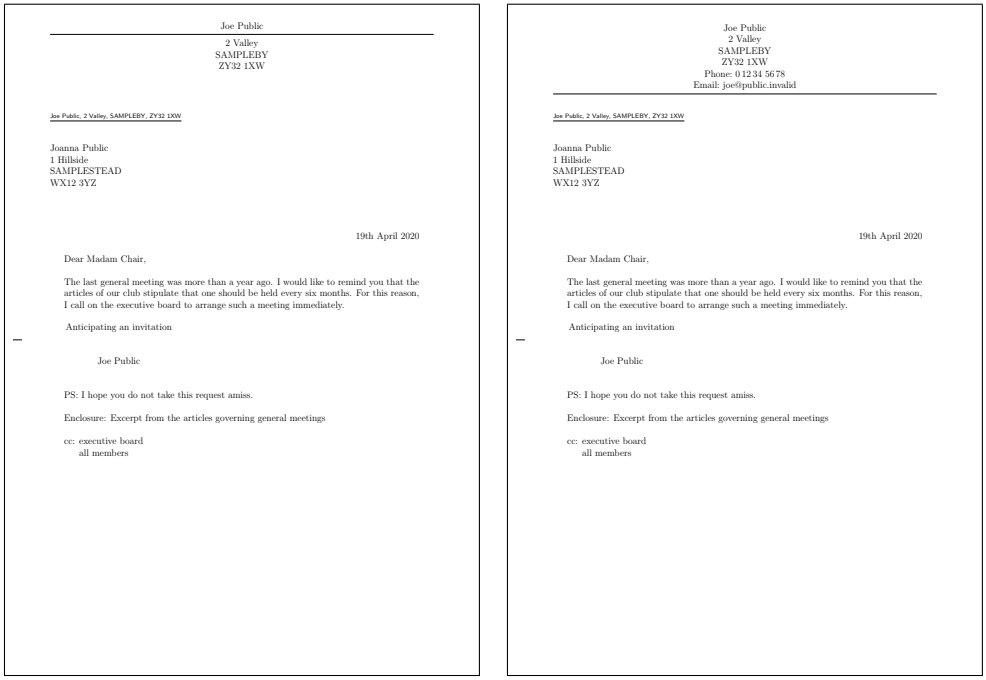


Figure 4.14.: result of a short letter with sender, rule, recipient, opening, text, closing, signature, postscript, distribution list, enclosure and hole-punch mark (the date is set by default): the left one uses the standard letterhead with `fromalign=false`; the right one uses the extended letterhead with `fromalign=center`

```
board to arrange such a meeting immediately.
\closing{Anticipating an invitation}
\ps PS: I hope you do not take this request amiss.
\setkomavar*{enclseparator}{Enclosure}
\encl{Excerpt from the articles governing general
  meetings}
\cc{executive board\\all members}
\end{letter}
\end{document}
```

You can compare two other alternatives with left-aligned sender information using `fromalign=left` and right-aligned sender information using `fromalign=right` in figure 4.15.

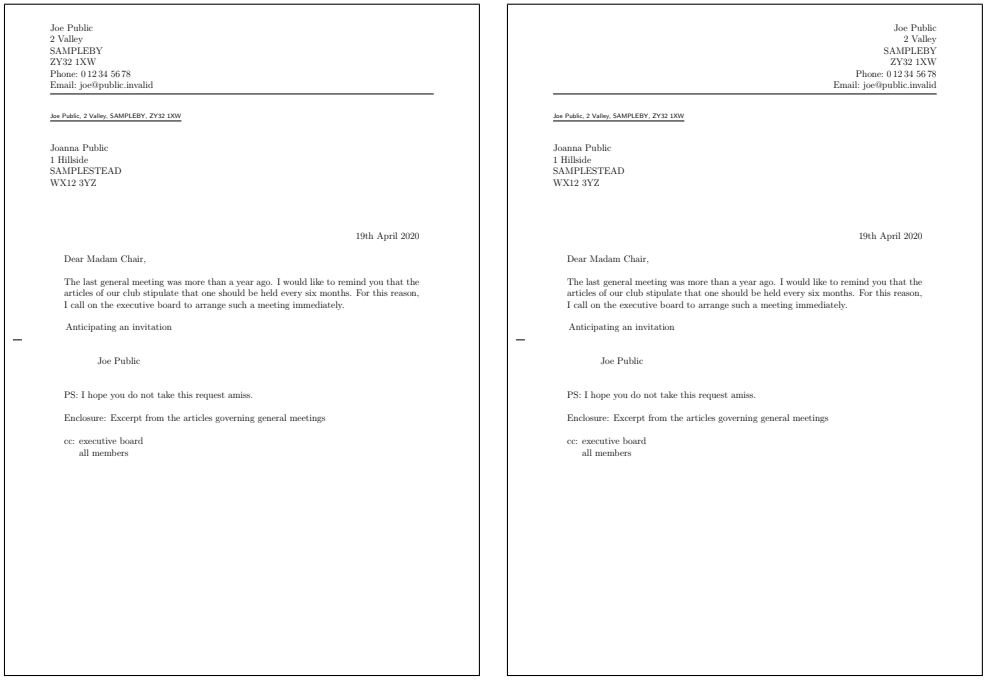


Figure 4.15.: result of a short letter with extra sender information, rule, recipient, opening, text, closing, signature, postscript, distribution list, enclosure and hole-punch mark (the date is set by default): the left one uses a left-aligned letterhead with `fromalign=left`; the right one uses a right-aligned letterhead using `fromalign=right`

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

You can use the `fromlogo` to configure whether to put a logo in the letterhead. You can use any of the default values from table 2.5, page 42 for the *`simple switch`*. The default is `false`, which means no logo. The logo itself is defined by the *`content`* of the `fromlogo` variable. The *`description`* of the logo is empty by default and KOMA-Script does not use it in the default letterhead pages.

Example: Mr Public finds it particularly stylish when he provides his letterhead with a logo. He has saved his logo as a graphics file, which he would like to load using `\includegraphics`. To do this, he loads the graphics package (see [Car17]).

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

```
\usepackage{graphics}
\begin{document}
\setkomavar{fromname}{Joe Public}
\setkomavar{fromaddress}{2 Valley\\
                         SAMPLEBY\\
                         ZY32 1XW}
\setkomavar{fromphone}{0\,12\,34~56\,78}
\setkomavar{fromemail}{joe@public.invalid}
\setkomavar{fromlogo}{\includegraphics{musterlogo}}
\begin{letter}{%
    Joanna Public\\
    1 Hillside\\
    SAMPLESTEAD\\
    WX12 3YZ%
  }
\opening{Dear Madam Chair,}
The last general meeting was more than a year ago.
I would like to remind you that the articles of our
club stipulate that one should be held every
six months. For this reason, I call on the executive
board to arrange such a meeting immediately.
\closing{Anticipating an invitation}
\ps PS: I hope you do not take this request amiss.
\setkomavar*{enclseparator}{Enclosure}
\encl{Excerpt from the articles governing general
  meetings}
\cc{executive board\\all members}
\end{letter}
\end{document}
```

You can see the result in the top left of figure 4.16. The other two images in this figure show the results with right-aligned and centred sender information.

`\setkomavar{firsthead}[`*description*`]{`*contents*`}`

In many cases, the capabilities that scrlttr2 offers with the foregoing options and variables will be sufficient to design a letterhead. In some cases, however, you may want even more flexibility. In those situations, you will have to do without the possibilities offered by the predefined letterhead, which you can select via the options described above. Instead, you must create your own letterhead from scratch. To do so, you must specify the desired structure as the *contents* of the `firsthead` variable. For example, you can set several boxes side by side or one beneath the other using the `\parbox` command (see [Tea05b]). Experienced users should thus be able to create their own a letterheads. Of course you can and should use other variables with `\usekomavar`. KOMA-Script does not use the *description* of variable `firsthead`.

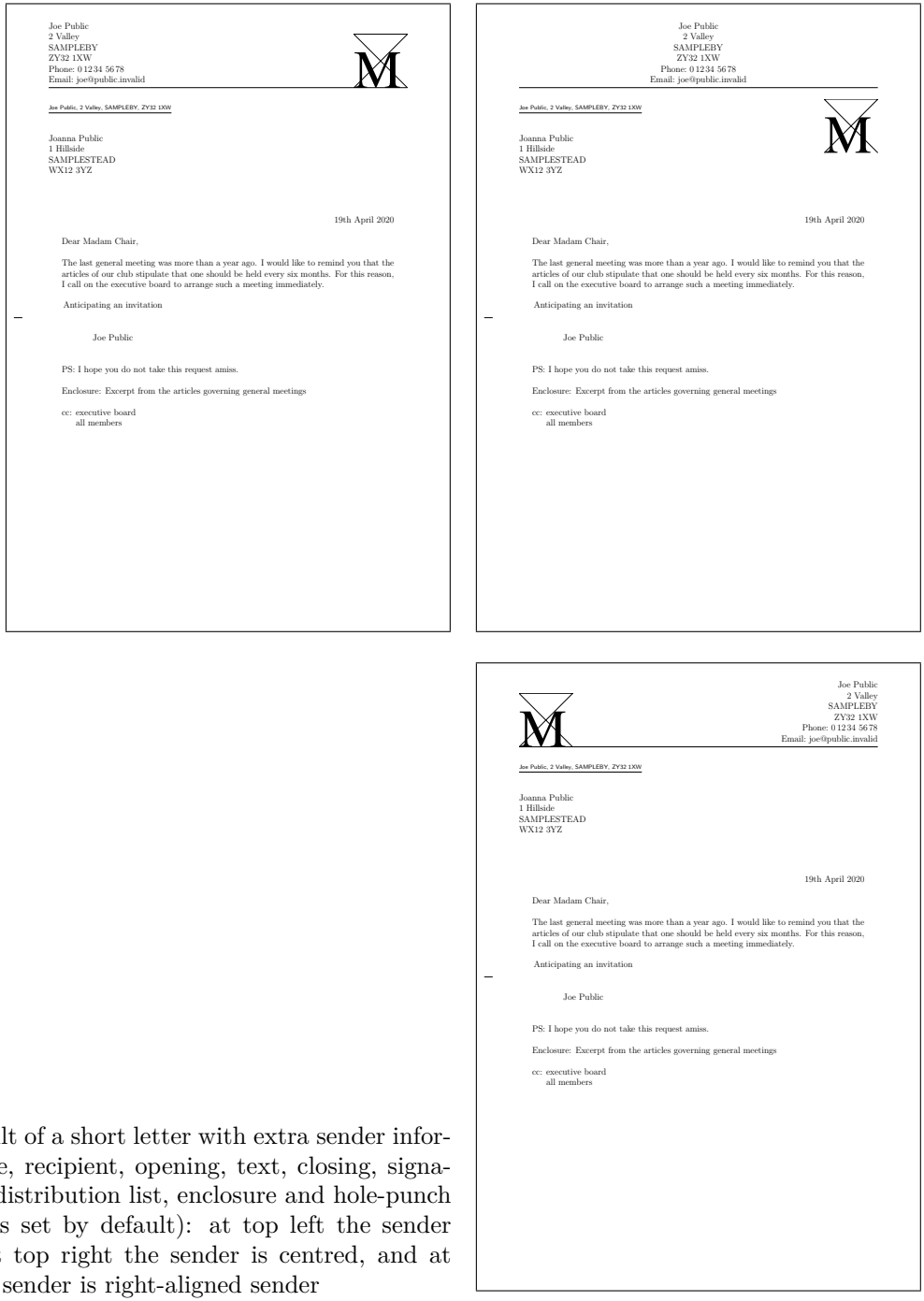


Figure 4.16.: result of a short letter with extra sender information, logo, rule, recipient, opening, text, closing, signature, postscript, distribution list, enclosure and hole-punch mark (the date is set by default): at top left the sender is left-aligned, at top right the sender is centred, and at bottom right the sender is right-aligned sender

4.10.3. Addressee

The term *addressee* normally refers only to the recipient's name and address, which are output in an address field. Additional information, however, can be placed within this address field, including the delivery method, for example registered mail or special delivery. For window envelopes, the return address also counts as part of the address field, as it will be displayed in the address window. The address field directly follows the letterhead.

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

v3.03 v3.17 The `addrfield` option determines whether or not to print an address field. The default is to include an address field. The option recognizes the formats described in table 4.9. With the values `true`, `topaligned`, `PP`, and `backgroundimage`, the name and address of the recipient are determined by the mandatory argument of the `letter` environment (see section 4.7, page 169). This information is also copied into the variables `toname` and `toaddress`.

v2.97c You can change the default font styles with the `\setkomafont` or `\addtokomafont` commands (see section 4.9, page 179). There are three elements. First, the `addressee` element is responsible for the overall style of the recipient's name and address. The elements `toname` and `toaddress` are responsible only for the name and the address of the recipient, respectively. You can use `toname` and `toaddress` to define modifications from the basic `addressee` configuration.

The default `addrfield=true` also prints an underlined return address in the address field. The `backaddress` option defines if and in what form a return address will be printed for window envelopes. On the one hand, this option accepts the standard values for simple switches listed in table 2.5, page 42. These values do not change style of the return address. On the other v2.96 hand, when the return address is enabled, you can select its format at the same time. For example, the `underlined` option value enables an underlined return address, while `plain` selects a style without underlining. The default is `underlined` and thus prints an underlined

Table 4.9.: Available values for the `addrfield` option to change the recipient format of scrlttr2

`backgroundimage`, `PPbackgroundimage`, `PPBackgroundImage`, `PPBackGroundImage`, `ppbackgroundimage`, `ppBackgroundImage`, `ppBackGroundImage` Prints an address with a background image stored in the `addresseimage` variable as the postpaid postmark, but without a return address or delivery type.
`false`, `off`, `no` Does not print an address.
`image`, `Image`, `PPimage`, `PPImage`, `ppimage`, `ppImage` Prints an image stored in the `addresseeimage` variable as an address with a postpaid postmark. Recipient information, return address, and delivery type or priority are ignored.
`PP`, `pp`, `PPexplicite`, `PPExplicite`, `ppexplicite`, `ppExplicite` Prints an address with a postage print impression (pospaid) header filled in with the variables `fromzipcode`, `place`, and `PPcode`, and when applicable with a priority and a data array defined by `PPdatamatrix`, but without a return address or delivery type.
v3.17 `topaligned`, `alignedtop` Prints an address including a return address and a delivery type or priority. The address is not centred vertically beneath the delivery type.
`true`, `on`, `yes` Prints an address field including a return address and delivery type or priority.

return address.

The return address itself is defined by the *content* of the `backaddress` variable. The default is the name specified in `fromname` and the address specified in `fromaddress`. The double backslash is also replaced with the *content* of the `backaddressseparator` variable. The default separator is a comma followed by a non-breaking space. KOMA-Script does not use the *description* of the `backaddress` variable. You can configure the font style of the return address via the `backaddress` element. The default is `\sffamily` (see table 4.10). Before applying the element's font style KOMA-Script switches to `\scriptsize`.

The default `addrfield=true`, centres the address vertically within the space available for
v3.17 the address. In contrast, `addrfield=topaligned` will not centre the address but place it flush with the top of the available space.

With the default `addrfield=true` setting, you can output an optional delivery type between the return address and the recipient address. This will be output only if the `specialmail`
v3.03 variable has non-empty content and `priority=manual` has been selected, which is the default. The scrlttr2 class itself does not use the *description* of the `specialmail` variable. The alignment is defined by the `specialmailindent` and `specialmailrightindent` pseudo-

Table 4.10.: Default font styles 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	

v2.97c lengths (see page 207). You can change the default font style of the specialmail element, which is listed in table 4.10, with the \setkomafont and \addtokomafont commands (see section 4.9, page 179).

v3.03 However, selecting an international priority with priority=A or priority=B (see table 4.11) together with addrfield=true, prints the priority as the delivery type. Using it together with addrfield=PP prints the priority at the corresponding position in the postpaid postmark (also known as the postage print impression or *port payé*). The priority element defines the basic font style, and prioritykey defines the modification of the basic font style for the priority keys "A" or "B". You can find the default font styles listed in table 4.10 and can change then using the \setkomafont and \addtokomafont commands (see section 4.9, page 179).

v3.03 With addrfield=PP, the postal code in the fromzipcode variable and the location from the place pariable will be output in the postpaid postmark. The postal code (that is, the *content* of fromzipcode) is preceded by the *description* of the fromzipcode variable and the *content* of zipcodeseparator. The default for the *description* depends on the lco file used (see section 4.20 starting on page 240). On the other hand, the default *content* of the

Table 4.11.: Available values for the priority option to select the international priority in 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 is printed instead of the delivery type.
A, a, priority, Priority, PRIORITY, A-PRIORITY, A-Priority, a-priority	Use international priority A-Priority. With addrfield=true, this is printed instead of the delivery type.

zipcodeseparator variable is a thin space followed by an en rule followed by another thin space (\,--\,).

v3.03 With addrfield=PP, a code is placed in the postpaid postmark that uniquely identifies the sender. This is stored in the PPcode variable. You can also print an additional data array to the right of the address, which is stored in the PPdatamatrix variable.

v3.03 The ZIP code (postal code), place, and sender code will be printed by default in an 8 pt font. The the font style of the PPdata is used to do so. The default is listed in table 4.10 and can be changed with the \setkomafont and \addtokomafont commands (see section 4.9, page 179).

For the postpaid logo "P.P.", however, the font style of the PPlogo element is used. Its default can also be found in table 4.10.

v3.03 With the two settings addrfield=backgroundimage and addrfield=image, an image is print inside the address window. This image is stored in the *content* of variable addresseeimage. KOMA-Script does not use the *description* of this variable. Although only the image is printed with the addrfield=image option, the recipient's name and address from the mandatory argument of the letter environment will be printed with the addrfield=backgroundimage option.

The position of the postpaid postmark and that of the postpaid address is defined by the toaddrindent pseudo-length (see page 206) as well as PPheadwidth and PPheadheight (see page 207). The position of the data array is defined by the PPdatamatrixvskip pseudo-length (see page 207).

Note that KOMA-Script cannot print any external graphics or pictures by itself. So if you want to put external picture files into variables like addresseeimage or PPdatamatrix, you must load an additional graphics package like graphics or graphicx and use, for instance, the \includegraphics command in the *content* of the variables.

\setplength{toaddrvpos}{*length*}
\setplength{toaddrhpos}{*length*}

These pseudo-lengths define the vertical and horizontal distance of the address field from the top-left corner of the paper. Values are set differently in the various predefined lco files, according to standard envelope window measures. For toaddrhpos, one property worth noting is that with negative values, the offset is the distance from the right edge of the address field to the right edge of the paper. You will find this, for instance, in SN or NF. The smallest value of toaddrvpos is found with DINmtext. With this setting, the letterhead can easily protrude into the address window. Whether the address field is output or not depends on the addrfield option (see section 4.10, page 202).

\setplength{toaddrheight}{*length*}

This pseudo-length defines the height of the address field, including the delivery method. Whether the name and address of the recipient are vertically centred in the address field, taking into account the presence or absence of the delivery method, depends on the addrfield option.

\setplength{toaddrwidth}{*length*}

This pseudo-length defines the width of the address field. The various predefined `lco` files use different settings according to the different standards for window envelopes. Typical values are between 70 mm and 100 mm.

Example: Suppose your printer has very wide, non-printable left and right margins of 15 mm. This means that the letterhead, the additional sender information, and the address field cannot be completely printed if you use the `SN` option. You therefore create a new `lco` file with the following content:

```
\ProvidesFile{SNmmarg.lco}
              [2002/06/04 v0.1 my 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`.

\setplength{toaddrindent}{*length*}

Sometimes you do not want the address field to extend the full width of the address window but to be indented a bit. You can set the amount of this indentation with the `toaddrindent` pseudo-length. Typically, the default value is 0 pt.

v3.03 For the addrfield=PP, addrfield=image, and addrfield=backgroundimage settings (see section 4.10, page 202) a value of 0 pt will be replaced by a value of 8 mm. If you really do not want any indentation, you can use a value of 1 sp to set a negligibly small indentation. Furthermore, `toaddrindent` is also used for the distance to the right edge of the address window with the these `addrfield` settings.

\setplength{backaddrheight}{*length*}

For window envelopes, the sender is often printed in a small font on one line above the addressee. This sender information is called the return address, because it is visible in the address window and will be used by the post office to return an undeliverable letter to the sender. This return address, therefore, requires only the information necessary for that purpose.

The height reserved for the return address within the address window is given by the `backaddrheight` pseudo-length. This value is typically 5 mm in the predefined `lco` fileslco file=`lco` file. Whether to print the return address at all depends on the `addrfield` (see section 4.10, page 202) and `backaddress` options (see section 4.10, page 202).

\setplength{specialmailindent}{*length*}
\setplength{specialmailrightindent}{*length*}

You can print an optional delivery method between the return address and the recipient's address. This field is printed only if the `specialmail` variable has non-empty contents. Its alignment is determined by the `specialmailindent` and `specialmailrightindent` pseudo-lengths, which specify the left and right indentation, respectively. In the predefined `lco` files, `specialmailindent` is set to rubber length `\fill`, while `specialmailrightindent` is set to 1 em. Thus the delivery method is set 1 em from the address field's right margin.

\setplength{PPheadheight}{*length*}
\setplength{PPheadwidth}{*length*}

v3.03

The `PPheadheight` pseudo-length specifies the height reserved for the postpaid header at the start of the address field for the two options `addrfield=PP` and `addrfield=backgroundimage`. The `PPheadwidth` pseudo-length is used only with `addrfield=PP` (see section 4.10, page 202) and gives the width of the left-hand field within the postpaid header, which contains the postpaid logo, the postal code, and the location. The width of the right-hand field containing the sender's code and the priority is the remaining width.

KOMA-Script automatically changes the default value of 0 mm for the `PPheadheight` pseudo-length to 20.74 pt, and `PPheadwidth`'s default from 0 mm to 42 mm.

\setplength{PPdatamatrixvskip}{*length*}

v3.03

This pseudo-length specifies the vertical distance between the postpaid header and the data array used with `addrfield=PP` (see section 4.10, page 202). KOMA-Script automatically changes the default value of 0 mm to 9 mm. The data matrix is flush right within the postpaid header.

Table 4.12.: Available values for the `locfield` option to set the width of the field for extra sender information with scrlttr2

`narrow`	narrow extra sender information field
`wide`	wide extra sender information field

4.10.4. Extra Sender Information

Often, especially with business letters, there is not enough space in the letterhead and footer to accommodate all the information about the sender that you want to include. For such additional information, you can use the space next to the addressee. In this manual, this field is called the *extra sender information*. In earlier versions of this manual, it was called the *sender's extension* or the *location field*.

`locfield=`*setting*

The content ot the field with extra sender attributes next to the address field can be anything you want, for example a location, a bank-account number, or other information. Depending on the `fromalign` option, it will also be used for the sender's logo. The width of this field can be specified in an `lco` file (see section 4.20). If the width is set to zero there, the `locfield` option can toggle between two defaults for the field width. This is the case for most of the `lco` files that come with KOMA-Script. See also the explanation on the `locwidth` pseudo-length in section 4.10.4, page 209. Available values for this option are shown in table 4.12. The default is `narrow`.

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

The contents of the extra sender information field, unless it contains the logo or the basic sender information itself, are specified by the `location` variable. You can use formatting commands like `\raggedright` within this variable's *content*. KOMA-Script does not use the *description* of this variable.

Example: Mr Public would like to provide some additional information about his membership. Therefore he uses the `location` variable:

```
\documentclass[foldmarks=true,foldmarks=blmtP,
  fromphone,fromemail,fromlogo,
  version=last]{scrlttr2}
\usepackage[british]{babel}
\usepackage{graphics}
\begin{document}
\setkomavar{fromname}{Joe Public}
```

```
\setkomavar{fromaddress}{2 Valley\\
                         SAMPLEBY\\
                         ZY32 1XW}
\setkomavar{fromphone}{0\,12\,34~56\,78}
\setkomavar{fromemail}{joe@public.invalid}
\setkomavar{fromlogo}{\includegraphics{musterlogo}}
\setkomavar{location}{\raggedright
  Club member no.~4711\\
  since 11.09.2001\\
  chairman 2003--2005}
\begin{letter}{%
    Joanna Public\\
    1 Hillside\\
    SAMPLESTEAD\\
    WX12 3YZ%
  }
\opening{Dear Madam Chair,}
The last general meeting was more than a year ago.
I would like to remind you that the articles of our
club stipulate that one should be held every
six months. For this reason, I call on the executive
board to arrange such a meeting immediately.
\closing{Anticipating an invitation}
\ps PS: I hope you do not take this request amiss.
\setkomavar*{enclseparator}{Enclosure}
\encl{Excerpt from the articles governing general
  meetings}
\cc{executive board\\all members}
\end{letter}
\end{document}
```

This will place the corresponding field beside the recipient's address, as shown in figure 4.17.

```
\setplength{locheight}{length}
\setplength{lochpos}{length}
\setplength{locvpos}{length}
\setplength{locwidth}{length}
```

v2.97d

The `locwidth` and `locheight` pseudo-lengths set the width and height of the extra-sender-information field. The `lochpos` and `locvpos` pseudo-lengths determine the distances from the top-right edge of the paper. These values are typically set to 0 pt in the predefined `lco` files. These zero-length values have a special meaning. They result in the actual values being set within `\opening` based on the paper width, the address-window width, the address window's

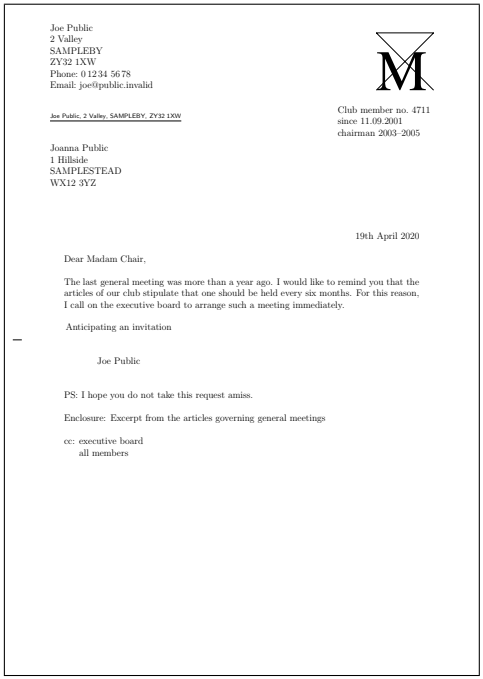


Figure 4.17.: result of a short letter with extended sender, logo, recipient, extra sender information, opening, text, closing, signature, postscript, distribution list, enclosure, and hole-punch mark (the date is set by default)

distance from the top-left edge of the paper, and the `locfield` option (see section 4.10, page 208). As is the case for `toaddrhpos`, negative values of `lochpos` also take on a special meaning. Instead of referring to the distance from the right edge of the paper, `lochpos` then refers to the distance from the left edge of the paper. The meaning is thus the opposite of that of `toaddrhpos` (see section 4.10.3, page 205).

4.10.5. Reference Line

The reference line can actually be longer than just one line. It is printed only if at least one of variables for the reference line is not empty. Only non-empty fields will be printed. To set a seemingly empty field, you can provide content for the variable that appears empty, such as `\mbox{}`. If the reference line is omitted, the description and contents of the `date` variable are printed in its place. You can find information about adding variables to or removing them from the reference line in section 22.1, page 502.

`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 date format. It should be defined by a package such as ngerman, babel, or isodate. The short, numerical representation, however,

Table 4.13.: Available values for the `refline` option to set the width of the reference line with scrlttr2

`dateleft`	
v3.09	The date is placed leftmost in the reference line.
`dateright`	
v3.09	The date is placed rightmost in the reference line.
`narrow`	The reference line is restricted to the type area.
`nodate`	
v3.09	The date is not placed automatically into the reference line.
`wide`	The width of the reference line depends on the positions of the address and extra sender information.

is produced by scrlttr2 itself. The option recognizes the standard values for simple switches as listed in table 2.5, page 42. The default is `false`, which results in standard date format.

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

The date in the selected format is stored as *content* of the `date` variable. Setting the `numericaldate` option has no effect if this variable is overridden. The date is usually output as part of the reference line. If all other elements of the reference line are empty, a date line consisting of the location and the date is printed instead. However in this case, the settings of the `refline` option described below also affect the date line. See the description of variable `place` on page 212 for more information about the location.

`refline=`*selection*

Business letters in particular often contain an area with information such as an identification code, phone extension, customer number, invoice number, or references to previous letters. This guide calls this area the *reference-field line*, or simply the *reference line*.

With scrlttr2 and scrletter, the header, footer, address, and extra sender information can extend left and right beyond the normal type area. The `refline=wide` option specifies that this should also apply to the reference-field line. Normally, the reference-field line contains at least the date, but it can hold additional data. Available values for this option are shown in
v3.09 table 4.13. The default is `narrow` and `dateright`.

Table 4.14.: Default descriptions of typical variables in the reference line using language-dependent commands

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

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

You can manage the typical contents of the reference-field line with five variables: yourref, yourmail, myref, customer and invoice. Their meanings are listed in table 4.1 on page 155. Each variable has also a predefined *description*, shown in table 4.14. How to add more variables to the reference-field line is explained in section 22.1 starting on page 502.

v2.97c You can change the font style and colour of the *description* and *content* of the fields in the reference line with the refname and refvalue elements using the \setkomafont and \addtokomafont commands (see section 4.9, page 179). The defaults for both elements are listed in table 4.15.

```
\setkomavar{placeseparator}[description]{contents}
```

If all variables in the reference-field line, with the exception of date, are empty, no actual reference line is output. Instead, a date line consisting only of a location and a date is output. The *content* of the place variable contains the location. The delimiter, which in this case is placed after the location, is specified by the *content* of the placeseparator variable. The default *content* of the delimiter is a comma followed by a non-breaking space. If the location is empty, the delimiter is not output. The default *content* of date is \today and depends on the setting of the numericaldate option (see page 210).

v3.09 Starting with version 3.09, the location and date are no longer required to be right-aligned. Instead, when a date line is used, its alignment is specified by the date setting of the refline option, as listed in table 4.13.

v3.12 If such a date line is output with a location rather than a reference-field line, the font setting of the refvalue element does not apply. In this case, the placeanddate element is used. You can change the empty default of this font element as usual with the \setkomafont and \addtokomafont commands (see section 4.9, page 179).

Example: Now Mr Public also sets the variable for the location:

Table 4.15.: Default font style settings for the elements in the reference line

element	default configuration
refname refvalue	\sffamily\scriptsize

```
\documentclass[foldmarks=true,foldmarks=blmtP,
  fromphone,fromemail,fromlogo,
  version=last]{scrlttr2}
\usepackage[british]{babel}
\usepackage{graphics}
\begin{document}
\setkomavar{fromname}{Joe Public}
\setkomavar{fromaddress}{2 Valley\\
                         SAMPLEBY\\
                         ZY32 1XW}
\setkomavar{fromphone}{0\,12\,34~56\,78}
\setkomavar{fromemail}{joe@public.invalid}
\setkomavar{fromlogo}{\includegraphics{musterlogo}}
\setkomavar{location}{\raggedright
  Club member no.~4711\\
  since 11.09.2001\\
  chairman 2003--2005}
\setkomavar{date}{29th February 2011}
\setkomavar{place}{Sampleby}
\begin{letter}{%
    Joanna Public\\
    1 Hillside\\
    SAMPLESTEAD\\
    WX12 3YZ%
  }
\opening{Dear Madam Chair,}
The last general meeting was more than a year ago.
I would like to remind you that the articles of our
club stipulate that one should be held every
six months. For this reason, I call on the executive
board to arrange such a meeting immediately.
\closing{Anticipating an invitation}
\ps PS: I hope you do not take this request amiss.
\setkomavar*{enclseparator}{Enclosure}
\encl{Excerpt from the articles governing general
  meetings}
\cc{executive board\\all members}
\end{letter}
\end{document}
```

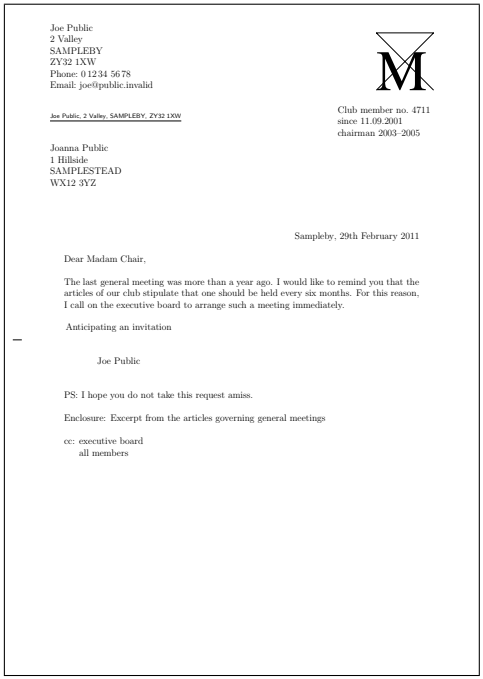


Figure 4.18.: result of a short letter with extended sender, logo, recipient, extra sender information, location, date, opening, text, closing, signature, postscript, distribution list, enclosure and hole-punch mark

As seen in figure 4.18, the location appears in front of the date, followed by the automatic delimiter. This date has been defined explicitly so that the original date is maintained in any later LaTeX run rather than using the date of the LaTeX run.

`\setplength{refvpos}{`*length*`}`

This pseudo-length specifies the distance from the upper edge of the paper to the reference line. Its value is set differently in the various predefined `lco` files. Typical values are between 80.5 mm and 98.5 mm.

`\setplength{refwidth}{`*length*`}`
`\setplength{refhpos}{`*length*`}`

The `refwidth` pseudo-length specifies the width available for the reference line. Its value is typically set to 0 pt in the predefined `lco` files. This value has a special meaning. In no way does it indicate that there is no available width for the reference line. Instead, it indicates that the width will be calculated within the `\opening` command. This calculated width then depends on the value of the `refline` options (see section 4.10, page 211). At the same time, `refhpos` will also be set according to this option. With `refline=wide`, the reference fields line is centred, while with `refline=narrow` it is aligned flush left with the type area.

If `refwidth` is not zero, the width of the reference line is not determined by the `refline` option, and so `refhpos` specifies the distance of the reference line from the left edge of the paper. If this distance is zero, the the reference line is placed so that the ratio between its distances from the left and right edges of the paper corresponds to the ratio of distance of the type area from the left and right edges of the paper. Thus, for a type area horizontally centred on the paper, the reference line will also be centred.

As a rule, these special cases are likely of little interest to the normal user. The simplest rule is as follows: either `refhpos` remains zero, and so the width and alignment of the reference line are determined with the `refline` option, or the user sets both `refwidth` and `refhpos`.

`\setplength{refaftervskip}{`*length*`}`

This pseudo-length specifies the vertical skip to be inserted beneath the reference line. The value is set in the predefined `lco` files. It directly affects the height of the text area on the first page. A typical value is between one and two lines.

4.10.6. Subject

Different countries have different conventions for placing the subject line of a letter. Some place it before the opening phrase; others place it after. Some professional groups even want it before the reference line.

`\setkomavar{title}[`*description*`]{`*contents*`}`

With KOMA-Script, you can also give a letter a title. The title is centred, using the `\LARGE` font size, and placed directly beneath the reference-field line.

You can change the font style for the `lettertitle` element with the `\setkomafont` and `\addtokomafont` commands (see section 4.9, page 179). Font size declarations are allowed. The `\LARGE` font size always precedes the font selection in KOMA-Script, and is therefore not part of the default setting `\normalcolor\sffamily\bfseries`. With scrlttr2, you can also use `title` as an alias for `lettertitle`. This is not possible when using scrletter with a KOMA-Script class because these classes already define a `title` element for the document title with different setting.

Example: Suppose you are writing a reminder letter. You set a corresponding title:

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

This way the recipient should recognize the reminder as such.

As shown in the example, the *content* of the variable defines the title. KOMA-Script does not use the *description*.

Table 4.16.: Default descriptions of variables for the subject

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

subject=*selection*
\setkomavar{subject}[*description*]{*contents*}
\setkomavar{subjectseparator}[*description*]{*contents*}

To include a subject, define the *content* of the subject variable accordingly. First of all, you can use the subject=true option, to choose whether the subject should be preceded with a *description* or not. If you use the *description* the *content* of the subjectseparator variable is output between the *description* and the *content* of the subject variable. The default *content* of *subjectseparator* consists of a colon followed by a space.

Additionally, you can use subject=afteropening to place the subject after the letter opening instead of the default subject=beforeopening. You can also specify other formatting for the subject with subject=underlined, subject=centered, or subject=right. The available values are listed in table 4.17. Note that with the subject=underlined option, the whole subject must fit onto one line. The defaults are subject=left, subject=beforeopening, and subject=untitled.

v2.97c

The subject line is set in a separate font. To change this, use the \setkomafont and \addtokomafont commands (see section 4.9, page 179). For the lettersubject element, the default font is \normalcolor\bfseries. With the scrlttr2 class, you can also use subject as an alias of lettersubject. This is not possible with the scrletter package when using a KOMA-Script class, because these classes already define a subject element for the document title with different settings.

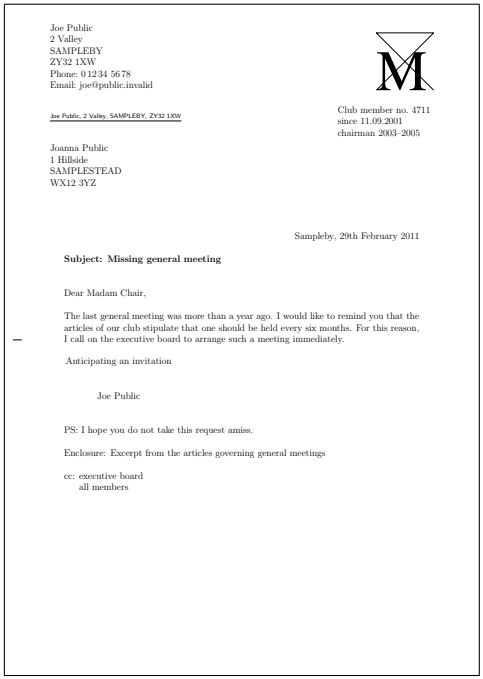
Example: Mr Public now includes a subject. As a traditionalist, he also wants the subject to be labelled accordingly and therefore sets the corresponding option:

```
\documentclass[foldmarks=true,foldmarks=blmtP,
  fromphone,fromemail,fromlogo,
  subject=titled,
  version=last]{scrlttr2}
\usepackage[british]{babel}
\usepackage{graphics}
\begin{document}
\setkomavar{fromname}{Joe Public}
\setkomavar{fromaddress}{2 Valley\\
                         SAMPLEBY\\
                         ZY32 1XW}
\setkomavar{fromphone}{0\,12\,34~56\,78}
```

Table 4.17.: Available values for the `subject` option for the position and formatting of the subject with scrlttr2

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

```
\setkomavar{fromemail}{joe@public.invalid}
\setkomavar{fromlogo}{\includegraphics{musterlogo}}
\setkomavar{location}{\raggedright
  Club member no.~4711\\
  since 11.09.2001\\
  chairman 2003--2005}
\setkomavar{date}{29th February 2011}
\setkomavar{place}{Sampleby}
\setkomavar{subject}{Missing general meeting}
\begin{letter}{%
    Joanna Public\\
    1 Hillside\\
    SAMPLESTEAD\\
    WX12 3YZ%
  }
\opening{Dear Madam Chair,}
The last general meeting was more than a year ago.
I would like to remind you that the articles of our
club stipulate that one should be held every
six months. For this reason, I call on the executive
board to arrange such a meeting immediately.
\closing{Anticipating an invitation}
\ps PS: I hope you do not take this request amiss.
\setkomavar*{enclseparator}{Enclosure}
\encl{Excerpt from the articles governing general
```


Joe Public
2 Valley
SAMPLEBY
ZY32 1XW
Phone: 0 12 34 56 78
Email: joe@public.invalid

Joe Public, 2 Valley, SAMPLEBY, ZY32 1XW

Club member no. 4711
since 11.09.2001
chairman 2003–2005

Joanna Public
1 Hillside
SAMPLESTEAD
WX12 3YZ

Sampleby, 29th February 2011

Subject: Missing general meeting

Dear Madam Chair,

The last general meeting was more than a year ago. I would like to remind you that the articles of our club stipulate that one should be held every six months. For this reason, I call on the executive board to arrange such a meeting immediately.

Anticipating an invitation

Joe Public

PS: I hope you do not take this request amiss.

Enclosure: Excerpt from the articles governing general meetings

cc: executive board
all members

Figure 4.19.: result of a short letter with extended sender, logo, recipient, extra sender information, place, date, subject, opening, text, closing, signature, postscript, distribution list, enclosure and hole-punch mark

```
  meetings}
\cc{executive board\\all members}
\end{letter}
\end{document}
```

The result is shown in figure 4.19.

`\setplength{subjectvpos}{`*length*`}`

v3.01

If the value of this pseudo-length is 0 pt, the `subject` option (see section 4.10, page 216) determines the position of the subject. Any other value defines the distance between the top edge of the paper and the subject. In this case, you should ensure that there is enough space available that overlapping with other elements is unlikely.

Example: A few professionals prefer to have the subject above the reference line. For this, you can specify the position as follows, which also changes the position of the reference line itself:

```
\ProvidesFile{lawsubj.lco}
              [2008/11/03 lawyers lco file]
\setplength{subjectvpos}{\useplength{refvpos}}
\addtoplength{refvpos}{3\baselineskip}
\endinput
```

If you want to leave at least one empty line between the subject and the reference, you have space for a maximum of two lines.

\setplength{subjectbeforevskip}{*length*}
\setplength{subjectaftervskip}{*length*}

v3.01

If the subject is placed not absolutely but before or after the salutation, you can insert additional vertical space before and after the subject. The space before the subject may interfere with other distances, such as the automatic distance of one line after the title. Therefore the default is to use no additional space here. The default of the class and the package for the space after the subject is two lines.

4.10.7. Closing

It has already been mentioned in section 4.7, page 172 that the letter's closing text is provided by \closing. Beneath the closing text, there is often a space for a handwritten signature, beneath which there can be a printed name, which serves as a kind of annotation to the actual signature.

\setkomavar{signature}[*description*]{*contents*}

The signature variable contains the printed name or annotation for the handwritten signature. Its default *content* is the \usekomavar{fromname}. This annotation can consist of multiple lines. In that case, you should separate the individual lines with double backslashes. Paragraph breaks in the signature annotation, however, are not permitted.

\raggedsignature

The closing phrase and the signature will be typeset in a box. The width of the box is determined by the length of the longest line in the closing phrase or signature.

The sigindent and sigbeforevskip pseudo-lengths determine exactly where this box is placed (see section 4.10.7, page 221). The \raggedsignature command defines the alignment inside the box. In the default 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, you can redefine the command in the same way as \raggedsection (see the example in section 3.16, page 107).

Example: Now Mr Public wants to make himself seem really important, and therefore he uses the signature to show once again that he was formerly a chairman himself. So he changes *contents* of the signature variable. He also wants the signature be aligned flush-left and so he also redefines \raggedsignature:

```
\documentclass[foldmarks=true,foldmarks=blmtP,
  fromphone,fromemail,fromlogo,
  subject=titled,
  version=last]{scrlttr2}
\usepackage[british]{babel}
\usepackage{graphics}
\begin{document}
\setkomavar{fromname}{Joe Public}
\setkomavar{signature}{Joe Public\\
  (former chairman)}
\renewcommand*{\raggedsignature}{\raggedright}
\setkomavar{fromaddress}{2 Valley\\
                         SAMPLEBY\\
                         ZY32 1XW}
\setkomavar{fromphone}{0\,12\,34~56\,78}
\setkomavar{fromemail}{joe@public.invalid}
\setkomavar{fromlogo}{\includegraphics{musterlogo}}
\setkomavar{location}{\raggedright
  Club member no.~4711\\
  since 11.09.2001\\
  chairman 2003--2005}
\setkomavar{date}{29th February 2011}
\setkomavar{place}{Sampleby}
\setkomavar{subject}{Missing general meeting}
\begin{letter}{%
    Joanna Public\\
    1 Hillside\\
    SAMPLESTEAD\\
    WX12 3YZ%
  }
\opening{Dear Madam Chair,}
The last general meeting was more than a year ago.
I would like to remind you that the articles of our
club stipulate that one should be held every
six months. For this reason, I call on the executive
board to arrange such a meeting immediately.
\closing{Anticipating an invitation}
\ps PS: I hope you do not take this request amiss.
\setkomavar*{enclseparator}{Enclosure}
\encl{Excerpt from the articles governing general
  meetings}
\cc{executive board\\all members}
\end{letter}
\end{document}
```

See figure 4.20 for the result.

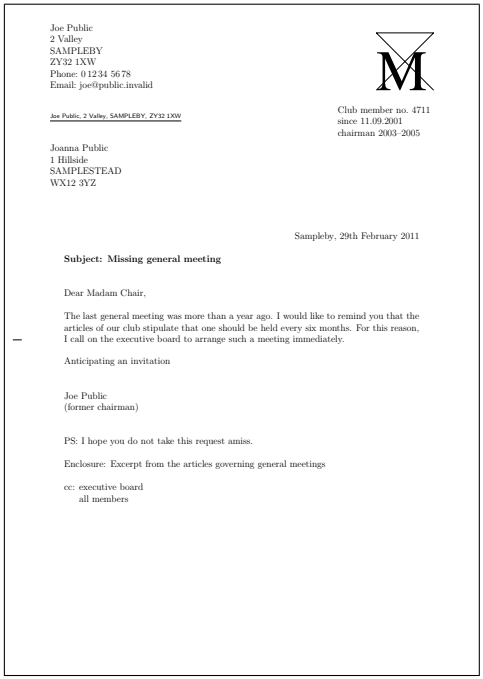


Figure 4.20.: result of a short letter with extended sender, logo, recipient, extra sender information, place, date, subject opening, text, closing, modified signature, postscript, distribution list, enclosure and hole-punch mark

The preceding example shows the most important, although not all possible, elements of a letter. It can, however, serve quite well as a general template.

```
\setplength{sigindent}{length}
\setplength{sigbeforevskip}{length}
```

The closing phrase and signature explanation are typeset in a box whose width is determined by the length of the longest line of the closing phrase or explanation.

The box will be indented by the distance specified in the `sigindent` pseudo-length. In the predefined `lco` files, this length is set to 0 mm.

Between the closing phrase and the signature explanation, a vertical skip is inserted whose height is defined in the `sigbeforevskip` pseudo-length. In the predefined `lco` files this value is set to two lines. In this space you can then write your signature.

4.10.8. Letterhead Page Footer

The first page of a letter, the letterhead page, contains not just its own header, the letterhead, but also its own footer. Just like the letterhead, it will be set not by the page style but directly within `\opening`.

enlargefirstpage=*simple switch*

The first page of a letter always uses a different page layout because of the many predefined elements such as the letterhead or the address. The scrlttr2 class provides a mechanism to calculate height and vertical alignment of header and footer of the first page independently of the subsequent pages. If, as a result, the footer of the first page would protrude into the text area, this text area of the first page is automatically reduced using `\enlargethispage`.

If the text area should become larger, assuming the footer on the first page permits this, you can use this option. At best, a little more text will then fit on the first page. See also the description of the `firstfootvpos` pseudo-length on page 224. This option takes the standard values for simple switches, as listed in table 2.5, page 42. The default is `false`.

firstfoot=*simple switch*

v2.97e

This option determines whether the footer for the letterhead page is output. Switching off the letterhead page footer with `firstfoot=false`, has an effect when the enlargefirstpage option (see page 222) is used at the same time, as this will logically extend the page downwards. In this case, the distance that is ordinarily between type area and the footer becomes the distance between the end of the type area and the bottom of the page.

The option recognizes the standard values for simple switches listed in table 2.5, page 42. The default is to include the letterhead page footer.

\setkomavar{firstfoot}[*description*]{*contents*}

v3.08

The footer of the first page is empty by default. However, you can specify a new footer in the *content* of the `firstfoot` variable. KOMA-Script does not use the *description* of the variable.

Example: You want to put the *content* of the `frombank` variable (i. e. the bank account details) in the footer of the first page. The double backslash should be replaced with a comma:

```
\setkomavar{firstfoot}{%
  \parbox[b]{\linewidth}{%
    \centering\def\\{, }\usekomavar{frombank}%
  }%
}
```

You can also define your own variable for the separator. I leave this as an exercise for the reader.

If you want to create a footer in order to counterbalance the letterhead, you can do so, for example, 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@{}}{Managing Director:}\\
        Lisa Mayer\\[1ex]
        \multicolumn{1}{@{}l@{}}{Court Jurisdiction:}\\
        Great Plains
      \end{tabular}%
      \Ifkomavarempty{frombank}{}{%
        \hfill
        \begin{tabular}[t]{l@{}}%
          \multicolumn{1}{@{}l@{}}{\usekomavar*{frombank}:}\\
          \usekomavar{frombank}
        \end{tabular}%
      }%
    }%
  }
```

This example originally came from Torsten Krüger. You should consider placing a definition intended for multiple use in different documents in a separate `lco` file.

```
\setkomavar{frombank}{Account No. 12345678\\
  Buffoon Bank\\
  Bank Code: 65-43-21}
```

can then be used to set the bank details in the document.

The previous example uses a multi-line footer. With a compatibility setting of version 2.9u (see `version` in section 4.2, page 153) the space is generally insufficient. In this case, you should reduce `firstfootvpos` (see page 224) appropriately.

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

The `frombank` variable used in the previous example occupies a special position. It is not currently used internally. However, you can use it to put the bank information in the extra sender information field or in the footer, as in the example (see the variable `location`, page 208).

\setplength{firstfootvpos}{*length*}

This pseudo-length gives the distance from the top of the paper to the footer of the letterhead page. It also ensures that the text area does not protrude into the footer. To do so, the height of the text area on the first page will be decreased, if necessary, using \enlargethispage. The enlargefirstpage option (see section 4.10, page 222) can also ensure that the height of the text area is increased, if necessary. Thus, the distance between text area and the letterhead page footer can be reduced to the value of the \footskip length.

v2.9t With the compatibility option set to versions up to 2.9t (see version in section 4.2, page 153) the footer is set independently of the type area in all predefined lco files (see section 4.20) except for KOMAold and NF. Thus the enlargefirstpage option has no effect. From version 2.9u on, the footer is placed at the bottom edge of the paper. Thus, the height of the letterhead page's type area may also depend on the enlargefirstpage option.

v2.97e If the letter footer is deactivated with the firstfoot=false option (see section 4.10, page 222), the setting of firstfootvpos is ignored, and instead \paperheight is applied. There remains then a minimum bottom margin of length \footskip.

\setplength{firstfoothpos}{*length*}

v3.05 A positive value of the firstfoothpos pseudo-length specifies the distance from the left edge of the paper to the letterhead page footer. If the value is greater than or equal to the paper width, \paperwidth, the footer is centred horizontally on the letterhead page. But if the value is less than or equal to the negative width of the paper, the footer is placed flush with the left edge of the typing area.

The typical default for this value is \maxdimen, which is the maximum possible value for a length. This results in horizontal centring.

\setplength{firstfootwidth}{*length*}

This pseudo-length specifies the width of the footer of the first page of the letter, that is the letterhead page. The value in the predefined lco files matches firstheadwidth.

4.11. Marking Paragraphs

The preliminaries of section 3.10, page 76 explain why paragraph indentation is preferred to paragraph spacing. But the elements to which this explanation refers, for example figures, tables, lists, equations, and even new pages, are rare in normal letters. Letters are usually not so long that an unrecognised paragraph will have serious consequences to the readability of the document. The arguments for indentation, therefore, are less consequential for standard letters. This may be one reason that you often find paragraphs in letters marked with vertical spacing. But two advantages of paragraph indentation remain. One is that such a letter stands out from the crowd. Another is that it maintains the *brand identity*, that is the

uniform appearance, of all documents from a single source. Apart from these suggestions, the information described in section 3.10 for the other KOMA-Script classes is valid for letters too. So if you have already read and understood section 3.10 you can skip ahead to section 4.12 on page 225. This also applies if you work not with the scrlttr2 class but with the scrletter package. The package does not provide its own settings for paragraph formatting but relies entirely on the class being used.

scrlttr2

`parskip=`*`method`*

In letters, you often encounter paragraphs marked not by indentation of the first line but by a vertical space between them. The KOMA-Script class scrlttr2 provides ways to accomplish this with the `parskip` option. The *`method`* consists of two elements. The first element is either `full` or `half`, where `full` stands for a paragraph spacing of one line and `half` stands for a paragraph spacing of half a line. The second element consists of one of the characters "`*`", "`+`", or "`-`" and can be omitted. Without the second element, the final line of a paragraph will end with a white space of at least 1 em. With the plus character as the second element, the white space will be at least one third — and with the asterisk one fourth — the width of a normal line. With the minus variant, no provision is made for white space in the last line of a paragraph.

scrlttr2

You can change the setting at any time. If you change it inside the document, the `\selectfont` command will be called implicitly. Changes to paragraph spacing within a paragraph will not be visible until the end of the paragraph.

v3.08

In addition to the resulting eight combinations for *`method`*, you can use the values for simple switches shown in table 2.5, page 42. Activating the option corresponds to using `full` with no second element and therefore results in inter-paragraph spacing of one line with at least 1 em white space at the end of the last line of each paragraph. Deactivating the option re-activates the default indentation of 1 em at the first line of the paragraph instead of paragraph spacing. A summary of all possible values for *`method`* are shown in table 3.7 at page 78.

All eight `full` and `half` option values also change the spacing before, after, and inside list environments. This prevents these environments or the paragraphs inside them from having a larger separation than that between the paragraphs of normal text. Several elements of the letterhead are always set without inter-paragraph spacing.

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

4.12. Detecting Odd and Even Pages

The information in section 3.11 applies equally to this chapter. So if you have already read and understood section 3.11, you can skip ahead to page 226, page 226.

In two-sided documents we distinguish left and right pages. Left pages always have an even page number, and right pages always have an odd page number. As a rule, letters will be set

one-sided. However, if you need to print letters using both sides of the paper or, in exceptional cases, are generating real two-sided letters, it may be useful to know whether you are currently on an even or odd page.

`\Ifthispageodd{`*true part*`}{`*false part*`}`

v3.28

If you want to determine whether text appears on an even or odd page, KOMA-Script provides the `\Ifthispageodd` command. The *true part* argument is executed only if you are currently on an odd page. Otherwise the *false part* argument is executed.

Example: Suppose 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.
```

This results in the output

> This page has an even page number.

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

In section 21.1, page 473, advanced users can find more information about the problems of detecting left and right pages, or even and odd page numbers.

4.13. Headers and Footers with the Default Page Style

One of the general properties of a document is its page style. In LaTeX this mostly consists of the contents of headers and footers. As already mentioned in section 4.10, the header and footer of the letterhead page are treated as elements of the letterhead page. Therefore they do not depend on the settings of the page style. So this section describes the page styles of the subsequent pages of a letter after the letterhead page. For one-sided letters, this is the page style of the second sheet. For two-sided letters, this is also the page style of all verso pages.

`\letterpagestyle`

v3.19

scrlttr2

The default page style for letters is specified by the contents of the `\letterpagestyle` command. By default, scrlttr2 leaves this command empty. This means that the page style of letters is the same as in the rest of the document. This is useful because scrlttr2 is intended for letter-only documents, and it is easier to define the page style for all letters globally, using `\pagestyle` as usual.

scrletter

Since both the `plain` and the `headings` page styles of other classes differs from the page style necessary for letters, the scrletter package defines `\letterpagestyle` with to be

plain.letter. This causes all letters to use the plain style of the letter page style pair, regardless of the page style of the rest of the document. See section 22.2 for more information about the characteristics of the page style of the scrletter package.

Example: You are using the scrletter package and want letters to use the same page style that was set for the rest of the document with \pagestyle. To do this, put the command

```
\renewcommand*{\letterpagestyle}{}
```

in your document preamble. Notice the star in \renewcommand*.

Of course, if you use \pagestyle or \thispagestyle inside a letter, this will take precedence over the page style set in \begin{letter} with \letterpagestyle.

headsepline=*simple switch*
footsepline=*simple switch*

scrlttr2 With these options scrlttr2 can select whether to put a separator rule below the header or above the footer, respectively, on subsequent pages. This option acccepts the standard values for simple switches listed in table 2.5, page 42. Activating the headsepline option enables a rule below the header. Activating the footsepline option enables a rule above the footer. Deactivating the options disables the corresponding rules.

Of course, the headsepline and footsepline options have no effect on the empty page style (see later in this section). This page style explicitly does not use headers or footers.

Typographically speaking, such a rule creates an optical effect of making the header or footer appear to be closer to the text area. This does not mean that the distance between the header or footer and the text should be increased. Instead, they should be seen as part of the text body when calculating the type area. To achieve this scrlttr2 uses the headsepline class option to automatically pass the headinclude package option to the typearea package. The same applies to footsepline for footinclude.

The options themselves do not automatically recalculate the type area. To recalculate the type area, see the DIV option with the last or current values (see page 38), or the \recalctypearea command (see page 40) in chapter 2.

The scrlayer-scrpage package (see chapter 5) provides further control over header and footer rules and you can also combine it with scrlttr2. The scrletter package automatically uses to define the letter and plain.letter page styles.

scrletter The letter and plain.letter page styles defined by scrletter follow the rules of scrlayer-scrpage. This particularly applies to setting the header and footer rules of the plain page style plain.letter. See in section 5.5, page 276 and page 276 the options headsepline and plainheadsepline. Also, settings like automark are of some importance for the letter page style.

Table 4.18.: Available values for the `pagenumber` option to position the page number in the `headings`, `myheadings`, and `plain` page styles with scrlttr2

`bot`, `foot`	page number in footer, horizontal position unchanged
`botcenter`, `botcentered`, `botmittle`, `footcenter`, `footcentered`, `footmiddle`	page number in footer, centred
`botleft`, `footleft`	page number in footer, left-justified
`botright`, `footright`	page number in footer, right-justified
`center`, `centered`, `middle`	page number centred horizontally, vertical position unchanged
`false`, `no`, `off`	no page number
`head`, `top`	page number in header, horizontal position unchanged
`headcenter`, `headcentered`, `headmiddle`, `topcenter`, `topcentered`, `topmiddle`	page number in header, centred
`headleft`, `topleft`	page number in header, left-justified
`headright`, `topright`	page number in header, right-justified
`left`	page number left, vertical position unchanged
`right`	page number right, vertical position unchanged

`pagenumber=`*position*

scrlttr2 scrletter

With this option you can select if and where a page number should be placed on subsequent pages. In scrlttr2 this option affects the page styles `headings`, `myheadings` and `plain`; in scrletter, it affects the `letter` and `plain.letter` page styles. It also affects the default page styles of the scrlayer-scrpage package, as long as it is set before that package is loaded (see chapter 5). It accepts values that influence only the horizontal position, only the vertical position, or both positions simultaneously. Available values are listed in table 4.18. The default is `botcenter`.

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

scrlttr2 scrletter Letters written with scrlttr2 have four different page styles available. In contrast, scrletter defines only two of its own page styles.

`empty` is the page style where the headers and footers of subsequent pages are completely empty. This page style is also automatically used for the first page of the letter because the header and footer of this page are set by other means, using `\opening` (see section 4.10, page 171).

scrlttr2 `headings` is, for scrlttr2, the page style for automatic running heads on subsequent pages. The sender's name from the `fromname` variable and the subject from the `subject` variable are used to set the marks (see section 4.10, page 190 and page 216). Where exactly these marks and page numbers are placed depends on the `pagenumber` described above and the *`content`* of the `nexthead` and `nextfoot` variables. An author can also change these marks manually after the `\opening` command. As usual, the `\markboth` and `\markright` commands are available, as is `\markleft` if you use scrlayer-scrpage (see section 5.5, page 267).

scrletter Since scrletter uses scrlayer-scrpage internally, any page style `headings` provided by the class are redefined as an alias of `scrheadings`. You can find more about this page style in chapter 5 on page 257.

scrletter `letter` Since the `headings` page style is generally already in use by the classes, the scrletter package instead defines the `letter` page style. This is accomplished with the help of scrlayer-scrpage in chapter 5, page 252. With the `automark=true` setting enabled, `letter` then assumes the role played by `headings` in scrlttr2. With `automark=false` set, `letter` assumes the role of `myheadings`.

When you use scrlayer-scrpage with scrletter, you cannot use the obsolete scrpage2 package or the fancyhdr package, which is incompatible with KOMA-Script.

scrlttr2 `myheadings` in scrlttr2 is the page style for manual running heads on subsequent pages. Unlike `headings`, you must set the marks yourself with the `\markboth` and `\markright` commands. When you use scrlayer-scrpage, `\markleft` is also available.

scrletter With the scrletter package, the `letter` page style also assumes the role of `myheadings`.

scrlttr2 `plain` is the default page style for scrlttr2, which does not use any headers on subsequent pages and only outputs a single page number. Where this is placed depends on the `pagenumber` option explained above.

scrletter Since scrletter uses scrlayer-scrpage internally, the `plain` page style is redefined as an alias of `plain.scrheadings`. You can find more about this page style in chapter 5 on page 257.

scrletter

plain.letter Since the **plain** page style is generally already in use by the classes, the scrletter defines a **plain.letter** page style instead. This is accomplished with the help of scrlayer-scrpage. When you use scrlayer-scrpage with scrletter, you cannot use the obsolete scrpage2 package or the fancyhdr package, which is incompatible with KOMA-Script.

The appearance of the page styles is also influenced by the **headsepline** and **footsepline** options described above. The page style beginning with the current page is changed 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.

To changing the font style of headers or footers, you should use the user interface described in section 3.6. The header and footer use the same element **pageheadfoot**. With package scrlayer-scrpage and therefore with scrletter the **pagehead** element is additionally responsible for the header. In scrlttr2 and without package scrlayer-scrpage the element is an alias of

v3.00

pageheadfoot only. The **pagefoot** element additionally controls the formatting of the footer. This element is applied after **pageheadfoot** in the **nextfoot** variable or when scrlayer-scrpage page styles are used (see chapter 5, page 261). The element for the page number within the header or footer is called **pagenumber**. The default settings are listed in table 3.8, page 81. Please also note the example from section 3.12, page 81.

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

In most cases, KOMA-Script's options and variables should be quite adequate for creating headers and footers for the subsequent pages, all the more so because, in addition to **\markboth** and **\markright**, you can change the sender information that scrlttr2 or scrletter puts in the header. You can, in particular, use the **\markboth** and **\markright** commands with the **myheadings** pagestyle. Of course, if you use the scrlayer-scrpage package, this is also valid for the **scrheadings** page style. In that case, the **\markleft** command is also available.

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

At times, however, you may want to the header or the footer of subsequent pages to more closely resemble the letterhead page. In these cases, you must dispense with the predefined options that can be selected with the **pagenumber** option described above. Instead, you can customise the header and footer of subsequent pages using the **headings** or **myheadings** page

scrlttr2

scrletter v3.00

styles with scrlttr2, and the **letter** page style with scrletter. To do so, you create the structure you want in the *content* of the **nexthead** or **nextfoot** variables. Within the *content* of **nexthead** and **nextfoot** you can, for example, use the **\parbox** command to place several boxes next to or beneath each other (see [Tea05b]). More advanced users should be able to create their own headers and footers. Of course, you can and should also make use of

additional variables using \usekomavar. KOMA-Script does not use the *description* for either variable.

4.14. Interleaf Pages

The information in section 3.13 applies equally to this chapter. So if you have already read and understood section 3.13, you can skip ahead to section 4.15, page 233.

Interleaf pages are pages that are inserted between parts of a document. Traditionally, these pages are completely blank. LaTeX, however, sets them by default with the current page style. KOMA-Script provides several extensions to this functionality.

Interleaf pages are mostly found in books. Because book chapters commonly start on the right (recto) page of a two-page spread, an empty left (verso) page must be inserted if the previous chapter ends on a recto page. For this reason, interleaf pages really only exist for two-sided printing.

Interleaf pages are unusual in letters. This is not least because two-sided letters are rare, as letters are usually not bound. Nevertheless, KOMA-Script also supports interleaf pages for two-sided letters. However, since the commands described here are seldom used in letters, you will not find any examples here. If necessary, please refer to the examples in section 3.13, starting on page 86.

cleardoublepage=*page style*
cleardoublepage=current

v3.00 With this option, you can define the page style of the interleaf pages created by the commands \cleardoublepage, \cleardoubleoddpage, or \cleardoubleevenpage to advance to the desired page. You can use any previously defined *page style* (see section 4.13 from page 226 and chapter 5 from page 252). In addition, cleardoublepage=current is also possible. This case corresponds to the default prior to KOMA-Script 2.98c and creates an interleaf v3.00 page without changing the page style. Starting with KOMA-Script 3.00, the default follows the recommendation of most typographers and creates interleaf pages with the empty page style unless you switch compatibility to earlier KOMA-Script versions (see option version, section 4.2, page 153).

```
\clearpage
\cleardoublepage
\cleardoublepageusingstyle{page style}
\cleardoubleemptypage
\cleardoubleplainpage
\cleardoublestandardpage
\cleardoubleoddpage
\cleardoubleoddpageusingstyle{page style}
\cleardoubleoddemptypage
\cleardoubleoddplainpage
\cleardoubleoddstandardpage
\cleardoubleevenpage
\cleardoubleevenpageusingstyle{page style}
\cleardoubleevenemptypage
\cleardoubleevenplainpage
\cleardoubleevenstandardpage
```

The LaTeX kernel provides the `\clearpage` command, which ensures that all pending floats are output and then starts a new page. There is also the `\cleardoublepage` command, which works like `\clearpage` but which starts a new right-hand page in two-sided printing (see the `twoside` layout option in section 2.4, page 41). An empty left-hand page in the current page style is output if necessary.

v3.00

With `\cleardoubleoddstandardpage`, KOMA-Script works as exactly in the way just described for the standard classess. The `\cleardoubleoddplainpage` command, on the other hand, additionally changes the page style of the empty left page to `plain` in order to suppress the page header. Likewise, the `\cleardoubleoddemptypage` command uses the `empty` page style to suppress both page header and page footer on the empty left-hand side. The page is thus completely empty. If you want to specify your own *`page style`* for the interleaf page, this should be given as an argument of `\cleardoubleoddusingpagestyle`. You can use any previously defined *`page style`* (see chapter 5).

The `\cleardoublestandardpage`, `\cleardoubleemptypage`, and `\cleardoubleplainpage` commands, and the single-argument `\cleardoublepageusingstyle` command, as well as the standard `\cleardoublepage` command, correspond to the commands previously explained for the scrlttr2 class. The remaining commands are defined in scrlttr2 for completeness only. You can find more information on them in section 3.13, page 87 if necessary.

In two-sided printing, `\cleardoubleoddpage` always moves to the next left-hand page and `\cleardoubleevenpage` to the next right-hand page. The style of the interleaf page to be inserted if necessary is defined with the `cleardoublepage` option.

4.15. Footnotes

The information in section 3.14 applies equally to this chapter. So if you have already read and understood section 3.14, you can skip ahead to page 236, page 236. If you do not use a KOMA-Script class, scrletter relies on the scrextend package. Therefore, see also section 10.11, page 301 when using scrletter.

scrletter

You can find the basic commands to set footnotes in any introduction to LaTeX, for example [OPHS11]. KOMA-Script provides additional features to change the format of the footnote block.

`footnotes=`*setting*
`\multfootsep`

v3.00

Footnotes are marked by default in the text with a small superscript number. If several footnotes appear in succession at the same point, it gives the impression that there is one footnote with a large number rather than multiple footnotes (e. g. footnote 12 instead of footnotes 1 and 2). With `footnotes=multiple`, footnotes that follow each other directly are separated with a delimiter instead. The default delimiter in `\multfootsep` is defined as a comma without a space:

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

This can be redefined.

The whole mechanism is compatible with the footmisc package, version 5.3d to 5.5b (see [Fai11]). It affects footnote markers placed using `\footnote`, as well as those placed directly with `\footnotemark`.

You can switch back to the default `footnotes=nomultiple` at any time using the `\KOMAoptions` or `\KOMAoption` command. However, if you encounter any problems using another package that alters the footnotes, you should not use this option, nor should you change the *setting* anywhere inside the document.

A summary of the available *setting* values of `footnotes` can be found in table 3.11, page 89.

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

Footnotes in KOMA-Script are produced, as they are in the standard classes, with the `\footnote` command, or alternatively the pair of commands `\footnotemark` and `\footnotetext`. As in the standard classes, it is possible for a page break to occur 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 move the footnote to the next page. Unlike the standard

v3.00

classes, KOMA-Script can recognize and separate consecutive footnotes automatically. See the previously documented option footnotes.

If instead you want to place this delimiter manually, you can do so by calling \multiplefootnoteseparator. However, users should not redefine this command, as it contains not only the delimiter but also the delimiter's formatting, for example the font size selection and the superscript. The delimiter itself is stored in the previously described \multfootsep command.

You can find examples and additional hints in section 3.14 from page 90.

\footref{*reference*}

v3.00

Sometimes you have a footnote in a document to which there are several references in the text. An inconvenient way to typeset this would be to use \footnotemark to set the number directly. The disadvantage of this method is that you need to know the number and manually set every \footnotemark command. And if the number changes because you add or remove an earlier footnote, you will have to change each \footnotemark. KOMA-Script thefore offers the \label mechanism to handle such cases. After placing a \label inside the footnote, you can use \footref to set all the other marks for this footnote in the text. When setting footnote marks with the \label mechanism, any changes to the footnote numbers will require at least two LaTeX runs to ensure correct numbers for all \footref marks.

You can find an example of how to use \footref in section 3.14 on page 90.

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

KOMA-Script sets footnotes slightly differently than the standard classes do. As in the standard classes, the footnote mark in the text is rendered with small, superscript numbers. The same formatting is used in the footnote itself. The mark in the footnote is typeset right-justified in a box with a width of *mark width*. The first line of the footnote follows directly.

All subsequent 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, the first line of each paragraph is indented by the value of *parindent*.

figure 3.1 on page 91 shows the different parameters. The default configuration of the KOMA-Script classes is as follows:

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

\textsuperscript controls both the superscript and the smaller font size. The command \thefootnotemark contains the current footnote mark without any formatting.

The footnote, including the footnote mark, uses the font specified in the footnote element. You can change the font of the footnote mark separately using the \setkomafont and

\addtokomafont commands (see section 4.9, page 179) for the footnotelabel element. See also table 4.3, page 181. The default setting is no change to the font. Please don't misuse this element for other purposes, for example to set the footnotes ragged right (see also \raggedfootnote, page 235).

The footnote mark in the text is defined separately from the mark in front of the actual footnote. This is done with \deffootnotemark. The default setting is:

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

With this default, the font for the footnotereference element is used (see table 4.3, page 181). Thus, the footnote marks in the text and in the footnote itself are identical. You can change the font with the commands \setkomafont and \addtokomafont (see section 4.9, page 179).

For examples, see section 3.14, page 92.

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

v3.06

Generally, a horizontal rule is set between the text area and the footnote area, but normally this rule does not extend the full width of the type area. With \setfootnoterule, you can set the exact thickness and length of the rule. In this case, the parameters *thickness* and *length* are only evaluated when setting the rule itself. If the optional argument *thickness* has been omitted, the thickness of the rule will not be changed. Empty arguments for *thickness* or *length* are also allowed and do not change the corresponding parameters. Using absurd values will result in warning messages both when setting and when using the parameters.

v3.07

You can change the colour of the rule with the footnoterule element using the \setkomafont and \addtokomafont commands (see section 4.9, page 179). The default is no change of font or colour. In order to change the colour, you must also load a colour package like xcolor.

\raggedfootnote

v3.23

By default KOMA-Script justifies footnotes just as in the standard classes. But you can also change the justification separately from the rest of the document by redefining \raggedfootnote. Valid definitions are \raggedright, \raggedleft, \centering, \relax or an empty definition, which is the default. The alignment commands of the ragged2e package are also valid (see [Sch09]).

Example: Suppose you are using footnotes only to provide references to very long links, where line breaks would produce poor results if justified. You can use

```
\let\raggedfootnote\raggedright
```

in your document's preamble to switch to ragged-right footnotes.

4.16. Lists

scrletter

The information in section 3.18 applies equally to this chapter. So if you have already read and understood section 3.18, you can skip ahead to section 4.17, page 239. The scrletter package does not define any list environments itself but leaves them to the class used. If this is not a KOMA-Script class, it will load scrextend. However, the scrextend package only defines the `labeling`, `addmargin`, and `addmargin*` environments. All other list environments are left to the responsibility of the class used.

Both LaTeX and the standard classes offer different environments for lists. Naturally, KOMA-Script also offers all these environments, though slightly modified or extended in some cases. In general, all lists — even those of different kinds — can be nested up to four levels deep. From a typographical view, anything more would make no sense, as lists of more than three levels cannot easily be apprehended. In such cases, I recommend that you split a large list into several smaller ones.

Because lists are standard elements of LaTeX, examples have been omitted in this section. Nevertheless, you can find examples either in section 3.18, page 116 or in any LaTeX tutorial.

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

scrlttr2

The simplest form of a list is the itemized list, `itemize`. Depending on the level, KOMA-Script classes use the following marks: “•”, “–”, “∗”, and “·”. The definition of these symbols is specified in the macros `\labelitemi`, `\labelitemii`, `\labelitemiii`, and `\labelitemiv`, all of which you can redefine using `\renewcommand`. Every item is introduced with `\item`.

```
\begin{enumerate}
  \item ...
  :
\end{enumerate}
\theenumi
\theenumii
\theenumiii
\theenumiv
\labelenumi
\labelenumii
\labelenumiii
\labelenumiv
```

scrlttr2

The numbered list is also very common and already provided by the LaTeX kernel. The numbering differs according to the level, with Arabic numbers, small letters, small Roman numerals, and capital letters, respectively. The style 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 right parenthesis, the values of all other levels are followed by a dot. Every item is introduced with `\item`.

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

scrlttr2

v2.8p

Another list form is the description list. It primarily serves to describe individual items or keywords. The item itself is specified as an optional parameter in `\item`. The font used to format the keyword can be changed with the `\setkomafont` and `\addtokomafont` commands (see section 4.9, page 179) for the `descriptionlabel` element (see table 4.3, page 181). The default is `\sffamily\bfseries`.

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

v3.02

Another form of description list is only available in the KOMA-Script classes and scrextend: the `labeling` environment. Unlike the `description` described above, you can specify a pattern for `labeling` whose length determines the indentation of all items. Furthermore, you can put an optional *`delimiter`* between the item and its description. The font used to format the item and the separator can be changed with the `\setkomafont` and `\addtokomafont` commands (see section 4.9, page 179) for the element `labelinglabel` and `labelingseparator` (see table 4.3, page 181).

Originally, this environment was implemented for things like “Premise, Evidence, Proof”, or “Given, Find, Solution” that are often used in lecture handouts. These days, however, the environment has very different applications. For example, the environment for examples in this guide was defined with the `labeling` environment.

scrlttr2

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

The `verse` environment is not normally perceived as a list environment because you do not work with `\item` commands. Instead, fixed line breaks are used within the `flushleft` environment. Internally, however, both the standard classes as well as KOMA-Script implement it as a list environment.

In general, the `verse` environment is used for poetry. 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 can, as usual, insert `\\*` instead of `\\`.

scrlttr2

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

These two environments are also set internally as list environments and can be found in both the standard and the KOMA-Script classes. Both environments use justified text which is indented on both the left and the right side. Often they are used to separate longer quotations from the main text. The difference between the two lies in in the manner in which paragraphs are typeset. While `quote` paragraphs are distinguished by vertical space, in `quotation` paragraphs, the first line is indented. This also applies to the first line of a `quotation` environment. If you want to prevent the indentation there, you must precede it with the `\noindent` command.

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

Like quote and quotation, the `addmargin` environment changes the margin. However, unlike the first two environments, `addmargin` lets the user change the width of the indentation. Apart from this change, 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 the value *left indentation* is used for the left margin instead of *indentation*.

The starred variant `addmargin*` differs from the normal version only in the two-sided mode. Furthermore, the difference only occurs if the optional argument *inner indentation* is used. In this case, the value of *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 allow negative values for all parameters. This can be done so that the environment protrudes into the margin.

Whether a page is going to be on the left or right side of the book cannot be determined reliably on the first LaTeX run. For details please refer to the explanation of the commands `\Ifthispageodd` (section 4.12, page 226) and `\ifthispagewasodd` (section 21.1).

4.17. Mathematics

The KOMA-Script classes do not provide their own maths environments. Instead, KOMA-Script relies completely on the maths features of the LaTeX kernel. However, since numbered equations and formulas are very unusual in letters, KOMA-Script does not actively support them. Therefore, the `leqno` and `fleqn` options for scrbook, scrreprt, and scrartcl, documented in section 3.19, are not available with scrlttr2.

You will not find a description here of the maths environments of the LaTeX kernel, that is `displaymath`, `equation`, and `eqnarray`. If you want to use them, consult an introduction to LaTeX like [OPHS11]. But if you want more than very simple mathematics, you should use the amsmath package (see [Ame02]).

4.18. Floating Environments for Tables and Figures

Floating environments for tables or figures are very unusual in letters. Therefore scrlttr2 does not provide them. If you still believe you need them, this often indicates a misuse of the letter class. In such cases, it is advisable to combine one of the KOMA-Script classes from chapter 3 with the scrletter package instead. In this case you, the floating environments documented for the class can also be used in letters. You could also define your own floating environments with help of tocbasic (see chapter 15).

4.19. Marginal Notes

The information in section 3.21 applies equally to this chapter. So if you have already read and understood section 3.21, you can skip ahead to section 4.20, page 240.

In addition to the text area, which normally fills the type area, documents often contain a column for marginalia. You can set marginal notes in this area. In letters, however, marginal notes are unusual and should be used sparingly.

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

Marginal notes in LaTeX are usually inserted with the `\marginpar` command. They are placed in the outer margin. One-sided documents use the right border. Although you can specify a different marginal note for `\marginpar` in case it winds up in the left margin, marginal notes are always fully justified. However, experience has shown that many users prefer left- or right-justified marginal notes instead. For this purpose, KOMA-Script offers the `\marginline` command.

For a detailed example, see section 3.21 at page 145.

Advanced users will find notes about difficulties that can arise using `\marginpar` in section 21.1. These remarks also apply to `\marginline`. In addition, chapter 19 introduces a package that you can use to create note columns with their own page breaks.

4.20. 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. The same applies to the letterhead and footer used on the first page. Therefore it makes sense to save these settings in a separate file. For this purpose, KOMA-Script offers `lco` files. The `lco` suffix is an abbreviation for l*etter class option*.

In an `lco` file, you can use all commands available to the document at the time the `lco` file is loaded. You can also use internal commands available to package writers. For scrlttr2 and scrletter, these are, in particular, the commands `\newplength`, `\setplength`, and `\addtoplength` (see section 4.6).

v3.04 KOMA-Script comes provided with some `lco` files. The `DIN.lco`, `DINmtext.lco`, `SNleft.lco`, `SN.lco`, `UScommercial9`, `UScommercial9DW`, and `NF.lco` files serve to adapt scrlttr2 and scrletter to different standards. They are well suited as templates for your own parameter sets as you become a KOMA-Script expert. The `KOMAold.lco` file, on the other hand, serves to improve compatibility with the old letter class scrlettr. This class was removed from KOMA-Script over fifteen years ago. It is therefore not discussed in any detail. Since this file also contains internal commands that are not released for 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 4.19, page 245.

If you have defined a parameter set for a letter standard that is not yet supported by KOMA-Script, you are explicitly requested to send this parameter set to the KOMA-Script support address. Please also provide permission for distribution under the KOMA-Script license (see the `lppl.txt` file). If you have the necessary measurements for an unsupported letter standard but are unable to write a corresponding `lco` file yourself, you can also contact the KOMA-Script author, Markus Kohm, directly. you can find further examples of `lco` files, some very complex, at [KDP] or in [Koh03]. Both sites are mainly in German.

\LoadLetterOption{*name*}
\LoadLetterOptions{*list of names*}

scrlttr2 With scrlttr2 you can load `lco` files with the `\documentclass` command. To do so, give the name of the `lco` file without the extension as an option. The `lco` file is then loaded directly after the class. The scrletter package does not offer this option. To load `lco` files, you must use `\LoadLetterOption` or `\LoadLetterOptions`. This is also recommended for scrlttr2. v3.14

You can also use `\LoadLetterOption` and `\LoadLetterOptions` after `\begin{document}`, or even from within another `lco` file. Both commands take the *name* of the `lco` file without the extension. While the argument of `\LoadLetterOption` should be exactly one `lco` file, `\LoadLetterOptions` accepts a comma-separated *list of names*. The corresponding `lco` files with those names are loaded in the order given by the list.

Example: Joe Public also writes a document containing several letters. For most of them, the default format, which follows the format of the German Institute for Standardisation, or *Deutsches Institut für Normung* (DIN), is sufficient. So he starts with:

```
\documentclass{scrlttr2}
```

However, he wants to send one letter in a C6/5 envelope, and so he uses the `DINmtext` variant, in which the address field appears higher on the page, so that more text fits on the first page. The fold marks are adjusted so that the address field still fits the address window in a C6/5 envelope. He achieves this as follows:

```
\begin{letter}{%
  Joanna Public\\
  1 Hillside\\
  SAMPLESTEAD\\
  WX12 3YZ}
  \LoadLetterOption{DINmtext}
  \opening{Hello,}
```

Since construction of the first page only really begins with the `\opening` command, you only need to load the `lco` file before this point. In particular, you do not need to load it before `\begin{letter}`. That way the changes made by loading the `lco` file are local to the corresponding letter.

If an `lco` file is loaded via `\documentclass`, then it must not have the same name as an option. v2.97

Example: Since Mr Public often writes letters with the same options and parameters, he finds it quite annoying to copy this information to each new letter. To simplify the effort of writing a new letter, he therefore creates an `lco` file:

```
\ProvidesFile{ich.lco}[2008/06/11 lco
```

```
  (Joe Public)]
\KOMAoptions{foldmarks=true,foldmarks=blmtP,
  fromphone,fromemail,fromlogo,subject=titled}
\setkomavar{fromname}{Joe Public}
\setkomavar{signature}{Joe Public\\
  (former chairman)}
\renewcommand*{\raggedsignature}{\raggedright}
\setkomavar{fromaddress}{2 Valley\\
                         SAMPLEBY\\
                         ZY32 1XW}
\setkomavar{fromphone}{0\,12\,34~56\,78}
\setkomavar{fromemail}{joe@public.invalid}
\setkomavar{fromlogo}{%
  \includegraphics{musterlogo}}
\setkomavar{location}{\raggedright
  Club member no.~4711\\
  since 11.09.2001\\
  chairman 2003--2005}
\setkomavar{place}{Sampleby}
\setkomavar{frombank}{Bank of Friendly Greetings}
```

With this, the size of his letter from the previous example shrinks considerably:

```
\documentclass[version=last,ich]{scrlttr2}
\usepackage[british]{babel}
\usepackage{graphics}
\begin{document}
\setkomavar{date}{29th February 2011}
\setkomavar{subject}{Missing general meeting}
\begin{letter}{%
    Joanna Public\\
    1 Hillside\\
    SAMPLESTEAD\\
    WX12 3YZ%
  }
\opening{Dear Madam Chair,}
The last general meeting was more than a year ago.
I would like to remind you that the articles of our
club stipulate that one should be held every
six months. For this reason, I call on the executive
board to arrange such a meeting immediately.
\closing{Anticipating an invitation}
\ps PS: I hope you do not take this request amiss.
\setkomavar*{enclseparator}{Enclosure}
\encl{Excerpt from the articles governing general
  meetings}
```

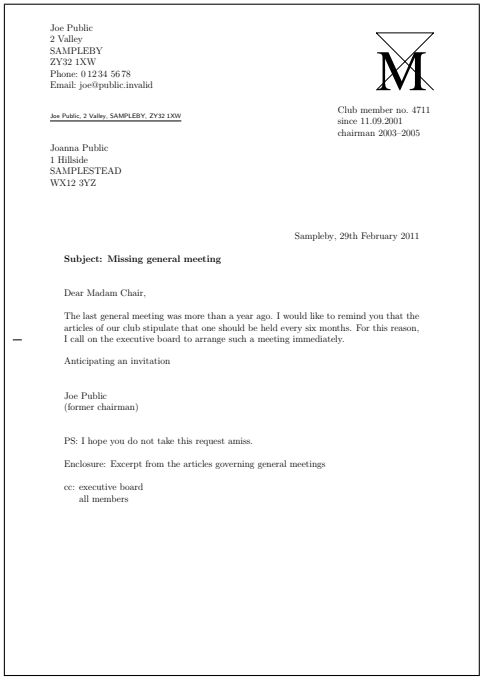


Figure 4.21.: result of a short letter with extended sender, logo, recipient, extra sender information, place, date, subject opening, text, closing, modified signature, postscript, distribution list, enclosure and hole-punch mark using an `lco` file

```
\cc{executive board\\all members}
\end{letter}
\end{document}
```

Nevertheless, the result does not change, as shown in figure 4.21.

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 non-ASCII characters. For example, use "`\ss`" to produce a German "ß".

In table 4.19, page 245 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 stipulates that the address field should be placed 8 mm from the right edge of the paper, the headline and the sender attributes are also placed at a correspondingly small distance from the paper edge. This also applies to the reference line when using the `refline=wide` option (see section 4.10, page 211). If you have this kind of problem, create your own `lco` file that loads `SN` first and then changes `toaddrhpos` (see section 4.10.3, page 205) to a smaller value. In addition, you should also reduce `toaddrwidth` accordingly.

By the way, the `DIN` `lco` file is always loaded automatically as the first `lco` file. This ensures that all pseudo-lengths will have more or less reasonable default values. Therefore you do not

need to load this default file on your own.

Table 4.19.: Predefined `lco` files

`DIN`
: parameters for letters on A4 paper, complying with German standard DIN 676; suitable for window envelopes in the sizes C4, C5, C6, and C6/5 (C6 long).

`DINmtext`
: parameters for letters on A4 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`
: parameters for Japanese letters on A4 paper; suitable for Japanese window envelopes of type Kaku A4, in which the window is approximately 90 mm wide by 45 mm high, and positioned 25 mm from the left and 24 mm from the top edge (see appendix A).

`KOMAold`
: parameters for letters on A4 paper using a layout close to that of the 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 differently with this `lco` file than with the other `lco` files.

`NF`
: parameters for French letters, complying with NF Z 11-001; suitable for window envelopes of type DL (110 mm by 220 mm) with a window 45 mm wide by 100 mm high placed about 20 mm from the lower right edge; this file was originally developed by Jean-Marie Pacquet, who also provides LyX integration in addition to extensions at [Pac].

`NipponEH`
: parameters for Japanese letters on A4 paper; suitable for Japanese window envelopes of types Chou or You 3 or 4, in which the window is approximately 90 mm wide by 55 mm high, and positioned 22 mm from the left and 12 mm from the top edge (see appendix A).

`NipponEL`
: parameters for Japanese letters on A4 paper; suitable for Japanese window envelopes of types Chou or You 3 or 4, in which the window is approximately 90 mm wide by 45 mm high, and positioned 22 mm from the left and 12 mm from the top edge (see appendix A).

...

Table 4.19.: Predefined `lco` files (*continued*)

`NipponLH`
: parameters for Japanese letters on A4 paper; suitable for Japanese window envelopes of types Chou or You 3 or 4, in which the window is approximately 90 mm wide by 55 mm high, and positioned 25 mm from the left and 12 mm from the top edge (see appendix A).

`NipponLL`
: parameters for Japanese letters on A4 paper; suitable for Japanese window envelopes of types Chou or You 3 or 4, in which the window is approximately 90 mm wide by 45 mm high, and positioned 25 mm from the left and 12 mm from the top edge (see appendix A).

`NipponRL`
: parameters for Japanese letters on A4 paper; suitable for Japanese window envelopes of types Chou or You 3 or 4, in which the window is approximately 90 mm wide by 45 mm high, and positioned 25 mm from the left and 24 mm from the top edge (see appendix A).

`SN`
: parameters for Swiss letters with the 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`
: parameters for Swiss letters with the address field on the left side; suitable for Swiss window envelopes with the window on the left side in the sizes C4, C5, C6, and C6/5 (C6 long).

`UScommercial9`
: parameters for US letters on American letter paper; suitable for *commercial No. 9* US window envelopes with a window 4 1/2 in wide by 1 1/8 in high, positioned 7/8 in from the left and 1/2 in from the bottom, without the return address inside the window; when folded first at the middle mark then at the top fold mark, legal paper can also be used but results in a paper-size warning

...

Table 4.19.: Predefined `lco` files (*continued*)

`UScommercial9DW`	parameters for US letters on American letter paper; suitable for *commercial No. 9* US window envelopes with an recipient-address window 3 5/8 in wide by 1 1/8 in high, positioned 3/4 in from the left and 1/2 in from the bottom, and with a return-address window 3 1/2 in wide by 7/8 in high, positioned 5/16 in from the left and 2 1/2 in from the bottom; when folded first at the middle mark and then at the top fold mark, legal paper can also be used but results in a paper-size warning

4.21. Address Files and Form Letters

One of the most annoying things about creating form letters is typing up the different addresses. KOMA-Script provides basic support for this task.

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

scrlttr2 and scrletter can evaluate address files. This can be very useful for form letters. An address file must have the extension `.adr` and consists of a number `\adrentry` entries. An individual entry consists of eight parameters and may look, for example, like this:

```
\adrentry{McEnvy}
         {Flann}
         {1 High Street\\ Glasgow}
         {0141 123 4567}
         {builder}
         {}
         {buys everything}
         {FLANN}
```

You can use the fifth and sixth elements, `F1` and `F2`, for anything you want. Gender, academic grade, birth date, or the date the person joined a society are all possibilities. The last parameter, ***key***, should consist of more than one letter and be upper-case only so as not to interfere with existing TEX or LATEX commands.

Example: Mr McEnvy is one of your most important business partners. Since you maintain a frequent correspondence with him, it is too tedious to enter all his data again and again. KOMA-Script will do this work for you. For example, if you have saved your customer contacts in the `partners.adr` address file and you would like to write a letter to Mr McEnvy, you can save a great deal of effort by typing:

```
\input{partners.adr}
\begin{letter}{\FLANN}
```

```
    Your correspondence of today \dots
  \end{letter}
```

Please make sure that your TEX system can access your address file. Otherwise the \input command results in an error message. You can either put your address file in the same directory as your letter or configure your TEX system to look for a dedicated address directory.

\addrentry{*last-name*}{*first-name*}{*address*}{*phone*}{*F1*}{*F2*}{*F3*}{*F4*}{*key*}

Before you object that a total of two free parameters is too few, KOMA-Script alternatively offers the \addrentry command — note the additional "d" — which adds two more freely definable parameters but omits the comment parameter. Otherwise, you can use this command in exactly the same way as \adrentry.

Both \adrentry and \addrentry commands can be freely mixed in the adr files. I should note, however, that other packages, such as adrconv by Axel Kielhorn, may not be designed to use \addrentry. If necessary, you have to create the appropriate extensions yourself.

In addition to simplifying access to addresses, you can also use the adr files to create circulars or form letters. Thus you can create such mass mailings without a complicated connection to a database system.

Example: You want to sent a form letter to all members of your fishing club to invite them to the next general meeting.

```
\documentclass{scrlttr2}
\begin{document}
\renewcommand*{\adrentry}[8]{
  \begin{letter}{#2 #1\\#3}
    \opening{Dear members,}
    Our next general meeting will be on
    Monday, 12 August 2002.

    The following topics are \dots
    \closing{Regards,}
  \end{letter}
}
\input{members.adr}
\end{document}
```

If the address file also contains \addrentry commands, you must add a corresponding definition before loading the address file:

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

In this example, the extra freely-definable parameter is not used, and therefore \addrentry is defined using \adrentry.

Of course, the letter's contents can also be adapted to the characteristics of the address data. You can use the free parameters of the \adrentry and \addrentry commands for this.

Example: Suppose you use the fifth parameter of the \adrentry command to indicate the gender of a club member (m/f), and the sixth parameter to indicate the amount of membership dues that is unpaid. If you would like to write a reminder to each such member and address them personally, the next example will help you:

```
\renewcommand*{\adrentry}[8]{
  \ifdim #6pt>0pt\relax
  % #6 is an amount (floating-point number) greater than 0.
  % Thus, this selects all members owing dues.
    \begin{letter}{#2 #1\\#3}
      \if #5m \opening{Dear Mr #2,} \fi
      \if #5f \opening{Dear Ms #2,} \fi

      Unfortunately, we have noticed that you are in arrears
      with the payment of your membership fees.

      Please remit the outstanding balance of \pounds #6 to the club
      account.
     \closing{Regards,}
    \end{letter}
   \fi
}
```

It is therefore possible to tailor the text of the letter to the specific characteristics of the recipient and create the impression of a personal letter. The extent of the customisation is only limited by the maximum number of two free \adrentry parameters and four free \addrentry parameters.

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

It is possible to create address lists and telephone directories using adr files. You also need the adrconv package by Axel Kielhorn (see [Kie10]). This package contains interactive LaTeX documents which make it easy to create such lists.

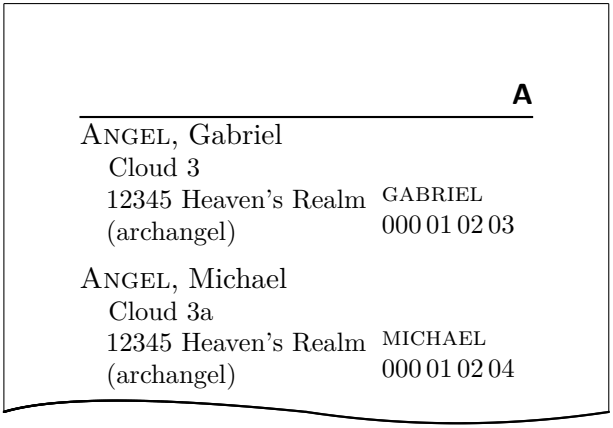
The address files have to be sorted already in order to obtain sorted lists. It is advisable to insert an \adrchar or \addrchar command containing the initial letter of the *last name* before the point in the list where this letter changes. scrlettr2 and scrletter will ignore these commands.

Example: Suppose you have the following, rather tiny address file from which you want to create an address book:

```
\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}
```

You can now process these entries using the `adrdir.tex` document from [Kie10]. One potential problem with this is that `adrdir.tex` up to and including Version 1.3 uses both the obsolete scrpage package and obsolete font commands which KOMA-Script has not supported for some time. If you receive an error message and cannot install a newer version, you can find a listing of `adrdir.tex` which indicates the changes necessary to avoid these errors at [Fel17] (in German).

The result looks something like this:



A

ANGEL, Gabriel
Cloud 3
12345 Heaven's Realm
(archangel)
GABRIEL
000 01 02 03

ANGEL, Michael
Cloud 3a
12345 Heaven's Realm
(archangel)
MICHAEL
000 01 02 04

The letter in the header is generated by answering "no" to the question "Names in the header?" See explanation in `adrdir.tex`.

You can find more about the adrconv package in its documentation. There you should also find information about whether the current version of adrconv supports the \addrentry and \addrchar commands. Previous versions only recognised the \adrentry and \adrchar commands.

v3.12

Headers and Footers with scrlayer-scrpage

Until version 3.11b of KOMA-Script, the scrpage2 package was the recommended way to customise headers and footers beyond the options provided by the `headings`, `myheadings`, `plain`, and `empty` page styles of the KOMA-Script classes. Since 2013, the scrlayer package has been included as a basic module of KOMA-Script. This package provides a layer model and a new page-style model based upon it. However, the package's interface is almost too flexible and consequently not easy for the average user to comprehend. For more information about this interface, see chapter 17 in part II. However, a few of the options that are actually part of scrlayer, and which are therefore taken up again in that chapter, are also documented here because they are required to use scrlayer-scrpage.

Many users are already familiar with the commands from scrpage2. For this reason, scrlayer-scrpage provides a method for manipulating headers and footers which is based on scrlayer, is largely compatible with scrpage2, and at the same time greatly expands the user interface. If you are already familiar with scrpage2 and refrain from direct calls to its internal commands, you can usually use scrlayer-scrpage as a drop-in replacement. This also applies to most examples using scrpage2 found in LaTeX books or on the Internet.

In addition to scrlayer-scrpage or scrpage2, you could also use fancyhdr (see [vO04]) to configure the headers and footers of pages. However, this package has no support for several KOMA-Script features, for example the element scheme (see `\setkomafont`, `\addtokomafont`, and `\usekomafont` in section 3.6, page 59) or the configurable numbering format for dynamic headers (see the `numbers` option and, for example, `\chaptermarkformat` in section 3.16, page 98 and page 111). Hence, if you are using a KOMA-Script class, you should use the new scrlayer-scrpage package. If you have problems, you can still use scrpage2. Of course, you can also use scrlayer-scrpage with other classes, such as the standard LaTeX ones.

Apart from the features described in this chapter, scrlayer-scrpage provides several more functions that are likely only of interest to a very small number of users and therefore are described in chapter 18 of part II, starting at page 446. Nevertheless, if the options described in part I are insufficient for your purposes, you should examine chapter 18.

5.1. Early or Late Selection of Options

The information in section 2.4 applies equally to this chapter. So if you have already read and understood section 2.4, you can skip ahead to section 5.2, page 254.

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

LaTeX allows users to pass class options as a comma-separated list of keywords in the optional argument to \documentclass. In addition to being passed to the class, these options are also passed on to all packages that can understand them. Users can also pass a similar comma-separated list of keywords in the optional argument of \usepackage. KOMA-Script extends the option mechanism for the KOMA-Script classes and some packages with further options. Thus most KOMA-Script options can also take a value, so an option does not necessarily take the form *option*, but can also take the form *option=value*. Except for this difference, \documentclass and \usepackage in KOMA-Script function as described in [Tea05b] or any introduction to LaTeX, for example [OPHS11].

v3.00

Setting the options with \documentclass has one major disadvantage: unlike the interface described below, the options in \documentclass are not robust. So commands, lengths, counters, and similar constructs may break inside the optional argument of this command. For example, with many non-KOMA-Script classes, using a LaTeX length in the value of an option results in an error before the value is passed to a KOMA-Script package and it can take control of the option execution. So if you want to use a LaTeX length, counter, or command as part of the value of an option, you should use \KOMAoptions or \KOMAoption. These commands will be described next.

\KOMAoptions{*option list*}
\KOMAoption{*option*}{*value list*}

v3.00

KOMA-Script also provides the ability to change the values of most class and package options even after loading the class or package. You can use the \KOMAoptions command to change the values of a list of options, as in \documentclass or \usepackage. Each option in the *option list* has the form *option=value*.

Some options also have a default value. If you do not specify a value, that is if you give the option simply as *option*, then this default value will be used.

Some options can have several values simultaneously. For such options, it is possible, with the help of \KOMAoption, to pass a list of values to a single *option*. The individual values are given as a comma-separated *value list*.

KOMA-Script uses the commands \FamilyOptions and \FamilyOption with the family "KOMA" to implement this ability. See part II, section 12.2, page 336.

Options set with \KOMAoptions or \KOMAoption will reach both the KOMA-Script class and any previously loaded KOMA-Script packages that recognise these options. If an option or a value is unknown, scrbase will report it as an error.

5.2. Header and Footer Height

The LaTeX standard classes do not use the footer much, and if they do use it, they put the contents into a \mbox which results in the footer being a single text line. This is probably the reason that LaTeX itself does not have a well-defined footer height. Although the distance between the last baseline of the text area and the baseline of the footer is defined with \footskip, if the footer consists of more than one text line, there is no definite statement whether this length should be the distance to the first or the last baseline of the footer.

Although the page header of the standard classes will also be put into a horizontal box, and therefore is also a single text line, LaTeX in fact provides a length to set the height of the header. The reason for this may be that this height is needed to determine the start of the text area.

\footheight
\headheight
autoenlargeheadfoot=*simple switch*

The scrlayer package introduces a new length, \footheight, analogous to \headheight. Additionally, scrlayer-scrpage interprets \footskip to be the distance from the last baseline of the text area to the first normal baseline of the footer. The typearea package interprets \footheight in the same way, so typearea's options for the footer height can also be used to set the values for the scrlayer package. See the footheight and footlines options in section 2.6, page 46) and option footinclude on page 43 of the same section.

If you do not use the typearea package, you should adjust the header and footer heights using appropriate values for the lengths where necessary. For the header, at least, the geometry package, for example, provides similar settings.

If you choose a header or footer height that is too small for the actual content, scrlayer-scrpage tries by default to adjust the lengths appropriately. At the same time, it will issue a warning containing suggestions for suitable settings. These automatic changes take effect immediately after the need for them has been detected and are not automatically reversed, for example, when the content of the header or footer becomes smaller afterwards. However,, this behaviour can be changed by using the autoenlargeheadfoot option. This option recognizes the values for simple switches in table 2.5, page 42. The option is activated by default. If it is deactivated, the header and footer are no longer enlarged automatically. Only a warning with hints for suitable settings is issued.

v3.25

5.3. Text Markup

The information in in section 3.6 largely applies to this chapter. So if you have already read and understood section 3.6, you can limit yourself to examining table 5.1, page 255 and then skip ahead to section 5.4, page 257.

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

With the help of the \setkomafont and \addtokomafont commands, you can attach particular font styling *commands* that change the appearance of a given *element*. Theoretically, all statements, including literal text, can be used as *commands*. You should, however, limit yourself to those statements that really change font attributes only. These are usually commands like \rmfamily, \sffamily, \ttfamily, \upshape, \itshape, \slshape, \scshape, \mdseries, \bfseries, \normalfont, as well as the font size commands \Huge, \huge, \LARGE, \Large, \large, \normalsize, \small, \footnotesize, \scriptsize, and \tiny. You can find these commands explained in [OPHS11], [Tea05b], or [Tea05a]. Colour switching commands like \normalcolor (see [Car17] and [Ker07]) are also acceptable. The use of other commands, in particular those that redefine things or or lead to output, is not supported. Strange behaviour is possible in these cases and does not represent a bug.

The command \setkomafont provides an element with a completely new definition of its font styling. In contrast, the \addtokomafont command merely extends an existing definition. You should not use either command inside the document body but only in the preamble. For examples of their use, refer to the sections for the respective element. The name and meaning of each element , as well as their defaults, are listed in table 5.1 . The specified defaults apply only if the corresponding element has not already been defined before loading scrlayer-scrpage. For example, the KOMA-Scriptclasses define pageheadfoot, and then scrlayer-scrpage uses the setting it finds.

With the \usekomafont command, the current font style can be changed to the one defined for the specified *element*.

Table 5.1.: Elements of scrlayer-scrpage whose font styles can be changed with the \setkomafont and \addtokomafont commands, and their defaults, if they have not been defined before loading scrlayer-scrpage

footbotline	horizontal line below the footer of a page style defined using scrlayer-scrpage. The font will be applied after \normalfont and the fonts of elements pageheadfoot and pagefoot. It is recommended to use this element for colour changes only. Default: *empty*

...

Table 5.1.: Elements whose font style can be changed (*continued*)

footsepline
: horizontal line above the footer of a page style defined using scrlayer-scrpage. The font will be applied after `\normalfont` and the fonts of elements pageheadfoot and pagefoot. It is recommended to use this element for colour changes only.
Default: *empty*

headsepline
: horizontal line below the header of a page style defined using scrlayer-scrpage. The font will be applied after `\normalfont` and the fonts of elements pageheadfoot and pagehead. It is recommended to use this element for colour changes only.
Default: *empty*

headtopline
: horizontal line above the header of a page style defined using scrlayer-scrpage. The font will be applied after `\normalfont` and the fonts of elements pageheadfoot and pagehead. It is recommended to use this element for colour changes only.
Default: *empty*

pagefoot
: contents of the page footer of a page style defined using scrlayer-scrpage. The font will be applied after `\normalfont` and the font of element pageheadfoot.
Default: *empty*

pagehead
: contents of the page header of a page style defined using scrlayer-scrpage. The font will be applied after `\normalfont` and the font of element pageheadfoot.
Default: *empty*

pageheadfoot
: contents of the page header or footer of a page style defined using scrlayer-scrpage. The font will be applied after `\normalfont`.
Default: `\normalcolor\slshape`

pagenumber
: pagination set with `\pagemark`. If you redefine `\pagemark`, you have to be sure that your redefinition also uses `\usekomafont{pagenumber}`!
Default: `\normalfont`

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

v3.12

Sometimes, although this is not recommended, the font setting of an element is used for settings that are not actually related to the font. If you want to apply only the font setting of an element but not those other settings, you can use `\usefontofkomafont` instead of `\usekomafont`. This will activate the font size and baseline skip, the font encoding, the font family, the font series, and the font shape of an element, but no further settings as long as those further settings are local.

You can also switch to a single one of those attributes using one of the other commands. Note that `\usesizeofkomafont` uses both the font size and the baseline skip.

However, you should not take these commands as legitimizing the insertion of arbitrary commands in an element's font setting. To do so can lead quickly to errors (see section 21.5, page 476).

5.4. Using Predefined Page Styles

The easiest way to create custom headers and footers with scrlayer-scrpage is to use one of the predefined page styles.

```
\pagestyle{scrheadings}
\pagestyle{plain.scrheadings}
```

The scrlayer-scrpage package provides two page styles that you can reconfigure to your liking. The first page style is `scrheadings`, which is intended as a page style with running heads. Its defaults are similar to the page style `headings` of the standard LaTeX or KOMA-Script classes. You can configure exactly what appears in the header or footer with the commands and options described below.

The second page style is `plain.scrheadings`, which is intended to be a style with no running head. Its defaults resemble those of the `plain` page style of the standard or KOMA-Script classes. You can configure exactly what appears in the header or footer with the commands and options described below.

You could, of course, configure `scrheadings` to be a page style without a running head and `plain.scrheadings` to be a page style with a running head. It is, however, advisable to adhere to the conventions mentioned above. The two page styles mutually influence one another. Once you apply one of these page styles, `scrheadings` will become accessible as `headings` and the page style `plain.scrheadings` will become accessible as `plain`. Thus, if

you use a class or package that automatically switches between `headings` and `plain`, you only need to select `scrheadings` or `plain.scrheadings` once. Direct patches to the corresponding classes or packages are not necessary. This pair of page styles can thus serve as a drop-in replacement for `headings` and `plain`. If you need more such pairs, please refer to section 18.2 in part II.

```
\lehead[plain.scrheadings content]{scrheadings content}
\cehead[plain.scrheadings content]{scrheadings content}
\rehead[plain.scrheadings content]{scrheadings content}
\lohead[plain.scrheadings content]{scrheadings content}
\cohead[plain.scrheadings content]{scrheadings content}
\rohead[plain.scrheadings content]{scrheadings content}
```

You can set the contents of the header for the `plain.scrheadings` and `scrheadings` page styles with these commands. The optional argument sets the content of an element of the `plain.scrheadings` page style, while the mandatory argument sets the content of the corresponding element of the `scrheadings` page style.

The contents of even — or left-hand — pages can be set with `\lehead`, `\cehead`, and `\rehead`. The "`e`" appearing as the second letter of the commands' names stands for "*even*".

The contents of odd — or right-hand — pages can be set with `\lohead`, `\cohead`, and `\rohead`. The "`o`" appearing as the second letter of the commands' names stands for "*odd*".

Note that in one-sided printing, only right-hand pages exist, and LaTeX designates these as odd pages regardless of their page number.

Each header consists of a left-aligned element that can be set with `\lehead` or `\lohead`. The "`l`" appearing as the first letter of the commands' names stands for "*left aligned*".

Similarly, each header has a centred element that can be set with `\cehead` or `\cohead`. The "`c`" appearing as the first letter of the command' names stands for "*centred*".

Likewise, each header has a right-aligned element that can be set with `\rehead` or `\rohead`. The "`r`" appearing as the first letter of the commands' names stands for "*right aligned*".

These elements do not have individual font attributes that you can change using the commands `\setkomafont` and `\addtokomafont` (see section 3.6, page 59). Instead, they use an element named `pagehead`. Before this element is applied, the `pageheadfoot` element will also be applied. See table 5.1 for the defaults of these elements.

The meaning of each command for headers in two-sided printing is illustrated in figure 5.1.

Example: Suppose you're writing a short article and you want the author's name to appear on the left side of the page and the article's title to appear right. You can write, for example:

```
\documentclass{scrartcl}
\usepackage{scrlayer-scrpage}
\lohead{John Doe}
```

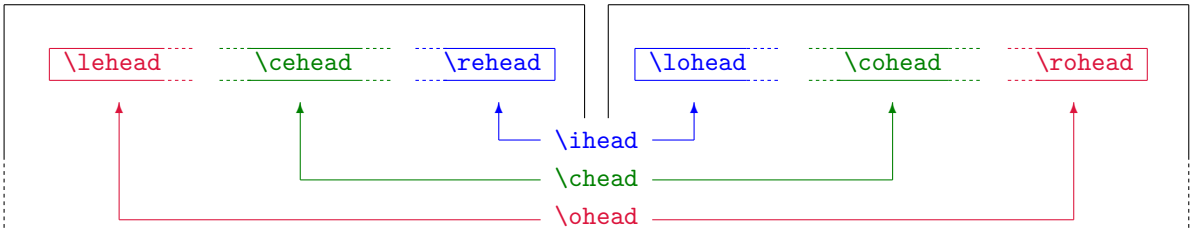


Figure 5.1.: The meaning of the commands for setting the contents of page headers shown on a two-page schematic

```
\rohead{Page style with \KOMAScript}
\begin{document}
\title{Page styles with \KOMAScript}
\author{John Doe}
\maketitle
\end{document}
```

But what happens? On the first page there's only a page number in the footer, while the header remains empty!

The explanation is simple: The scrartcl class, like the default article class, switches to the `plain` page style for the page which contains the title. After the command `\pagestyle{scrheadings}` in the preamble of our example, this actually refers to the `plain.scrheadings` page style. The default for this page style when using a KOMA-Script class is an empty page header and a page number in the footer. In the example, the optional arguments of `\lohead` and `\rohead` are omitted, so the `plain.scrheadings` page style remains unchanged and the result for the first page is actually correct.

Now add enough text to the example after `\maketitle` so that a second page is printed. You can simply add `\usepackage{lipsum}` to the document preamble and `\lipsum` below `\maketitle`. You will see that the header of the second page now contains the author and the document title as we wanted.

For comparison, you should also add the optional argument to `\lohead` and `\rohead`. Change the example as follows:

```
\documentclass{scrartcl}
\usepackage{scrlayer-scrpage}
\lohead[John Doe]
       {John Doe}
\rohead[Page style with \KOMAScript]
       {Page style with \KOMAScript}
\begin{document}
\title{Page styles with \KOMAScript}
```

```
\author{John Doe}
\maketitle
\end{document}
```

Now you have a header on the first page just above the title itself. That is because you have reconfigured page style plain.scrheadings with the two optional arguments. As you probably appreciate, it would be better to leave this page style unchanged, as a running head above the document title is rather annoying.

By the way, as an alternative to configuring plain.scrheadings you could, if you were using a KOMA-Script class, have changed the page style for pages that contain title headers. See \titlepagestyle in section 3.12, page 83.

Note that you should never put a section heading or section number directly into the header using one of these commands. Because of the asynchronous way that TEX lays out and outputs pages, doing so can easily result in the wrong number or heading text in the running head. Instead you should use the mark mechanism, ideally in conjunction with the procedures explained in the next section.

```
\lehead*[plain.scrheadings content]{scrheadings content}
\cehead*[plain.scrheadings content]{scrheadings content}
\rehead*[plain.scrheadings content]{scrheadings content}
\lohead*[plain.scrheadings content]{scrheadings content}
\cohead*[plain.scrheadings content]{scrheadings content}
\rohead*[plain.scrheadings content]{scrheadings content}
```

v3.14

The starred versions of the previously described commands differ from the ordinary versions only if you omit the optional argument *plain.scrheadings content*. In this case, the version without the star does not change the contents of plain.scrheadings. The starred version, on the other hand, uses the mandatory argument *scrheading content* for plain.scrheadings as well. So if both arguments should be the same, you can simply use the starred version with only the mandatory argument.

Example: You can shorten the previous example using the starred versions of \lohead and \rohead:

```
\documentclass{scrartcl}
\usepackage{scrlayer-scrpage}
\lohead*{John Doe}
\rohead*{Page style with \KOMAScript}
\begin{document}
\title{Page styles with \KOMAScript}
\author{John Doe}
\maketitle
\end{document}
```

```
\lefoot[plain.scrheadings content]{scrheadings content}
\cefoot[plain.scrheadings content]{scrheadings content}
\refoot[plain.scrheadings content]{scrheadings content}
\lofoot[plain.scrheadings content]{scrheadings content}
\cofoot[plain.scrheadings content]{scrheadings content}
\rofoot[plain.scrheadings content]{scrheadings content}
```

You can define the contents of the footer for scrheadings and plain.scrheadings with these commands. The optional argument defines the content of an element of plain.scrheadings, while the mandatory argument sets the content of the corresponding element of scrheadings.

The contents of even — or left-hand — pages are set with \lefoot, \cefoot, and \refoot. The "e" appearing as the second letter of the commands' names stands for "*even*".

The contents of odd — or right-hand — pages are set with \lofoot, \cofoot, and \rofoot. The "o" appearing as the second letter of the commands' names stands for "*odd*".

Note that in one-sided printing, only right-hand pages exist, and LaTeX designates these as odd pages regardless of their page number.

Each footer consists of a left-aligned element that can be set with \lefoot or \lofoot. The "l" appearing as the first letter of the commands' names stands for "*left aligned*".

Similarly, each footer has a centred element that can be set with \cefoot or \cofoot. The "c" in the first letter of the command' names stands for "*centred*".

Likewise, each footer has a right-aligned element that can be set with \refoot or \rofoot. The "r" in the first letter of the commands' names stands for "*right aligned*".

However, these elements do not have individual font attributes that can be changed with the \setkomafont and \addtokomafont commands (see section 3.6, page 59). Instead, they use an element named pagefoot. Before this element is applied, the font element pageheadfoot is also applied. See table 5.1 for the defaults of the fonts of these elements.

The meaning of each command for footers in two-sided printing is illustrated in figure 5.2.

Example: Let's return to the example of the short article. Let's say you want to specify the publisher in the left side of the footer. You would change the example above to:

```
\documentclass{scrartcl}
```

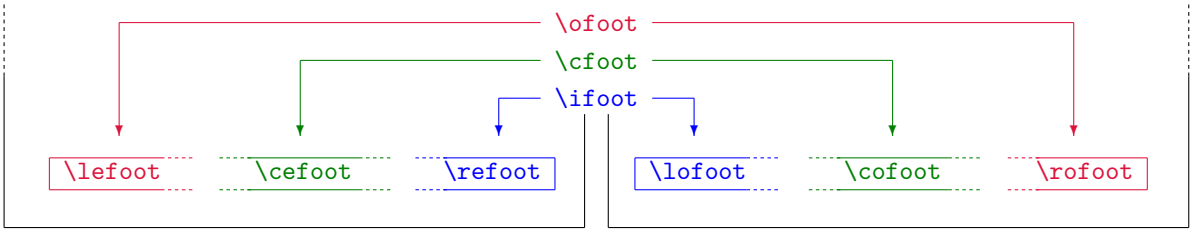


Figure 5.2.: The meaning of the commands for setting the contents of page footers shown on a two-page schematic

```
\usepackage{scrlayer-scrpage}
\lohead{John Doe}
\rohead{Page style with \KOMAScript}
\lofoot{Smart Alec Publishing}
\usepackage{lipsum}
\begin{document}
\title{Page styles with \KOMAScript}
\author{John Doe}
\maketitle
\lipsum
\end{document}
```

Once again the publisher is not printed on the first page with the title. The reason is the same as in the example with `\lohead` above. And the solution for getting the publisher on the first page is similar:

```
\lofoot[Smart Alec Publishing]
       {Smart Alec Publishing}
```

Now you decide that the header and footer should use an upright but smaller font in place of the default slanted font:

```
\setkomafont{pageheadfoot}{\small}
```

In addition, the header, but not the footer, should be bold:

```
\setkomafont{pagehead}{\bfseries}
```

It is important that this command does not occur until after scrpage-scrlayer has been loaded because the KOMA-Script class defines `pagehead` as an alias for `pageheadfoot`. Only by loading scrpage-scrlayer will `pagehead` become an element independent of `pageheadfoot`.

Now add one more `\lipsum` and the `twoside` option when loading scrartcl. First of all, you will see the page number moves from the centre to the outer margin of the page footer, due to the changed defaults of `scrheadings` and `plain.scrheadings` for two-sided printing with a KOMA-Script class.

Simultaneously, the author, document title, and publisher will vanish from page 2. They only appear on page 3. That's because we've only used commands for odd pages. You can recognise this by the "`o`" in the second position of the command names.

Now, we could simply copy those commands and replace the "`o`" with an "`e`" to define the contents of *even* pages. But with two-sided printing, it makes more sense to use mirror-inverted elements, i. e. the left element of an even page should become the right element of the odd page and visa versa. To achieve this, we also replace the first letter "`l`" with "`r`":

```
\documentclass[twoside]{scrartcl}
\usepackage{scrlayer-scrpage}
\lohead{John Doe}
\rohead{Page style with \KOMAScript}
\lofoot[Smart Alec Publishing]
       {Smart Alec Publishing}
\rehead{John Doe}
\lohead{Page style with \KOMAScript}
\refoot[Smart Alec Publishing]
       {Smart Alec Publishing}
\usepackage{lipsum}
\begin{document}
\title{Page styles with \KOMAScript}
\author{John Doe}
\maketitle
\lipsum\lipsum
\end{document}
```

Since it is a bit cumbersome to define left and right pages separately in cases such as the previous example, a simpler solution for this common case is introduced below.

Allow me once again an important note: you should never put a section heading or section number directly into the footer using one of these commands. Because of the asynchronous way that TEX lays out and outputs pages, doing so can easily result in the wrong number or heading text in the running head. Instead you should use the mark mechanism, ideally in conjunction with the procedures explained in the next section.

```
\lefoot*[plain.scrheadings content]{scrheadings content}
\cefoot*[plain.scrheadings content]{scrheadings content}
\refoot*[plain.scrheadings content]{scrheadings content}
\lofoot*[plain.scrheadings content]{scrheadings content}
\cofoot*[plain.scrheadings content]{scrheadings content}
\rofoot*[plain.scrheadings content]{scrheadings content}
```

v3.14

The starred versions of the previously described commands differ only if you omit the optional argument *[plain.scrheadings content]*. In this case, the version without the star does not change the contents of plain.scrheadings. The starred version, on the other hand, uses the mandatory argument *scrheading content* for plain.scrheadings as well. So if both arguments should be the same, you can simply use the starred version with just the mandatory argument.

Example: You can shorten the previous example using the star versions of \lofoot and \refoot:

```
\documentclass[twoside]{scrartcl}
```

```
\usepackage{scrlayer-scrpage}
\lohead{John Doe}
\rohead{Page style with \KOMAScript}
\lofoot*{Smart Alec Publishing}
\rehead{John Doe}
\lohead{Page style with \KOMAScript}
\refoot*{Smart Alec Publishing}
\usepackage{lipsum}
\begin{document}
\title{Page styles with \KOMAScript}
\author{John Doe}
\maketitle
\lipsum\lipsum
\end{document}
```

```
\ohead[plain.scrheadings content]{scrheadings content}
\chead[plain.scrheadings content]{scrheadings content}
\ihead[plain.scrheadings content]{scrheadings content}
\ofoot[plain.scrheadings content]{scrheadings content}
\cfoot[plain.scrheadings content]{scrheadings content}
\ifoot[plain.scrheadings content]{scrheadings content}
```

To configure the headers and footers for two-sided printing with the previously described commands, you would have to configure the left and right sides separately from one another. In most cases, however, the left and right sides are more or less symmetrical. An item that appears on the left of an even page should appear on the right of an odd page and vice versa. Centred elements are usually centred on both sides.

To simplify the definition of such symmetric page styles, scrlayer-scrpage has shortcuts. The `\ohead` command corresponds to a call to both `\lehead` and `\rohead`. The `\chead` command corresponds to a call to both `\cehead` and `\cohead`. And the `\ihead` command corresponds to a call to both `\rehead` and `\lohead`. The same applies to the equivalent commands for the page footer. An outline of these relationships can also be found in figure 5.1 on page 259 and figure 5.2 on page 261.

Example: You can simplify the previous example using the new commands:

```
\documentclass[twoside]{scrartcl}
\usepackage{scrlayer-scrpage}
\ihead{John Doe}
\ohead{Page style with \KOMAScript}
\ifoot[Smart Alec Publishing]
      {Smart Alec Publishing}
\usepackage{lipsum}
\begin{document}
```

```
\title{Page styles with \KOMAScript}
\author{John Doe}
\maketitle
\lipsum\lipsum
\end{document}
```

Because one-sided printing treats all pages as odd pages, these commands are synonymous with the corresponding right-side commands when in one-sided mode. Therefore in most cases you will only need these six commands instead of the twelve described before.

Allow me once again an important note: you should never put a section heading or section number directly into the footer using one of these commands. Because of the asynchronous way that TEX lays out and outputs pages, doing so can easily result in the wrong number or heading text in the running head. Instead you should use the mark mechanism, ideally in conjunction with the procedures explained in the next section.

```
\ohead*[plain.scrheadings content]{scrheadings content}
\chead*[plain.scrheadings content]{scrheadings content}
\ihead*[plain.scrheadings content]{scrheadings content}
\ofoot*[plain.scrheadings content]{scrheadings content}
\cfoot*[plain.scrheadings content]{scrheadings content}
\ifoot*[plain.scrheadings content]{scrheadings content}
```

v3.14

The previously described commands also have starred versions that differ only if you omit the optional argument *[plain.scrheadings content]*. In this case, the version without a star does not change the content of plain.scrheadings. The version with the star, on the other hand, also uses the mandatory argument *scrheadings content* for plain.scrheadings. So if both arguments should be the same, you can simply use the starred version with only the mandatory argument.

Example: You can shorten the previous example using the star version of \ifoot:

```
\documentclass[twoside]{scrartcl}
\usepackage{scrlayer-scrpage}
\ihead{John Doe}
\ohead{Page style with \KOMAScript}
\ifoot*{Smart Alec Publishing}
\usepackage{lipsum}
\begin{document}
\title{Page styles with \KOMAScript}
\author{John Doe}
\maketitle
\lipsum\lipsum
\end{document}
```

pagestyleset=*setting*

The examples above refer several times to the default settings of the page styles scrheadings and plain.scrheadings. In fact, scrlayer-scrpage currently provides two different defaults for these page styles. You can select them manually with the pagestyleset option.

The KOMA-Script *setting* selects the defaults, which are also set automatically if the option is not specified and a KOMA-Script class is detected. In two-sided printing, scrheadings uses outer-aligned running heads in the header and outer-aligned page numbers in the footer. In one-sided printing, the running head will be printed in the middle of the header and the page number in the middle of the footer. Upper- and lower-case letters are used in the automatic running heads as they actually appear in the sectioning headings. This corresponds to the markcase=used option. The plain.scrheadings page style has no running heads, but the page numbers are printed in the same manner.

However, if the scrlttr2 class is detected, the default settings are based on the page styles of that class. See section 4.13, page 226.

The standard *setting* selects defaults that match the page styles of the standard classes. This is also activated automatically if the option has not been specified and no KOMA-Script class is detected. In this case, for two-sided printing scrheadings uses running heads inner-aligned in the header, and the page numbers will be printed — also in the header — outer-aligned. One-sided printing uses the same settings, but since only right-hand pages exist in this mode, the running head will always be left-aligned and the page number right-aligned. The automatic running heads — despite considerable typographic objections — are converted to capital letters, as they would be with markcase=upper. In one-sided printing, the plain.scrheadings page style differs considerably from scrheadings because the page number is printed in the middle of the footer. Unlike the plain page style in the standard classes, plain.scrheadings omits the page number in two-sided printing. The standard classes print the page number in the middle of the footer, which does not match the rest of the page styles in two-sided printing. The running head is omitted in plain.scrheadings.

Note that using this option activates the scrheadings page style.

5.5. Manipulating Page Styles

section 5.4 explains how the page styles scrheadings and plain.scrheadings are defined and how these defaults can be changed. But topics such as creating running headers, changing the widths of the header and footer, and putting horizontal lines above or below the header or footer have yet to be described. Although these capabilities are actually part of the scrlayer package, they will be explained below because these basic features of scrlayer make up an important part of scrlayer-scrpage.

```
\automark[section level of the right mark]{section level of the left mark}
\automark*[section level of the right mark]{section level of the left mark}
\manualmark
```

In both the standard LaTeX classes and the KOMA-Script classes, the decision of whether to use automatic or static running heads is made by using the appropriate page style. Running heads repeat some descriptive text, such as a title, that is appropriate to the page or column, usually in the header, more rarely in the footer. As already explained in section 3.12, you get automatic running heads with `headings`

article, scrartcl

In the article classes article or scrartcl, the `headings` page style uses the section heading, which is either the mandatory or the optional argument of `\section`, for the running head of one-sided documents. Two-sided documents use this section heading as the *left mark* and the subsection heading as the *right mark*. The left mark is printed, as the name indicates, on left-hand (verso) pages. The right mark is printed on right-hand (recto) — in one-sided printing this means on all — pages. The classes by default also delete the right mark whenever they put the section heading into the left mark.

report, scrreprt, book, scrbook

The report and book classes start one level higher. Thus they use the chapter heading as the right mark in one-sided printing. In two-sided printing, the chapter heading is the left mark and the section heading is the right mark.

If you use `myheadings`, the marks in the page header still exist, and the page numbers are placed in the same way, but section commands no longer set the marks automatically. You can set them manually using the commands `\markright` and `\markboth`, which are described later in this section.

This distinction has been eliminated by scrlayer. Instead of distinguishing between automatic and manual running heads by which page style is selected, there are two new commands: `\automark` and `\manualmark`.

The `\manualmark` command switches to manual marks and deactivates the automatic filling of the marks. In contrast, `\automark` and `\automark*` define which section levels should be used to set the mark automatically. The optional argument is the *section level of the right mark*, the mandatory argument the *section level of the left mark*. The arguments should always be the name of a section level like `part`, `chapter`, `section`, `subsection`, `subsubsection`, `paragraph`, or `subparagraph`.

Normally, the higher level should be used for the left mark and the lower level for the right mark. This is only a convention and not a requirement, but it makes sense.

Please note that not every class provides running heads for every section level. For example, the standard classes never use `\part` in the heading. The KOMA-Script classes, on the other hand, support all levels.

The difference between `\automark` and `\automark*` is that `\automark` overrides all previous commands to automatically set the mark, while `\automark*` changes only the behaviour of the section levels specified in its arguments.

Example: Suppose you want chapter headings to be used as the running head of even pages and the section heading to be the running head of odd pages, as usual. But on odd pages you also want the chapter headings to be used as the running head until the first section appears. To do so, you first have to load scrlayer-scrpage and select the scrheadings page style, so the document starts with:

```
\documentclass{scrbook}
\usepackage{scrlayer-scrpage}
\pagestyle{scrheadings}
```

Next, ensure that the chapter headings set both the left and the right marks:

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

Then the section heading should also set right marks:

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

Here the starred version is used, since the previous \automark command should remain in effect. Additionally, the mandatory argument for the *section level of the left mark* is empty because this mark should remain unchanged.

All that's missing now is a bit of document content to show the result:

```
\usepackage{lipsum}
\begin{document}
\chapter{Chapter Heading}
\lipsum[1-20]
\section{Section Heading}
\lipsum[21-40]
\end{document}
```

We use the extremely useful lipsum package to generate some dummy text with command \lipsum.

If you test the example, you will see that the first page of the chapter appears, as usual, without a running head, since this page automatically uses the plain page style plain.scrheadings (see the \chapterpagestyle on page 83). Pages 2–4 have the chapter headings in the running head. After the section heading on page 4, the running head of page 5 changes to this section heading. From this page to the end, the running head alternates from page to page between the chapter and section headings.

automark
autooneside=*simple switch*
manualmark

Instead of the commands described previously, you can also use the `manualmark` and `automark` options to switch between automatic and manual running heads. `automark` always uses the default `\automark[section]{chapter}` for classes with `\chapter` and `\automark[subsection]{section}` for other classes.

In one-sided printing, you normally want only the higher section levels to provide the running title. The default option `autooneside` corresponds to this behaviour. The option accepts the values for simple switches listed in table 2.5, page 42. If you deactivate this option, the optional and mandatory arguments of `\automark` and `\automark*` will again control the running head in one-sided printing.

Example: Suppose you have a one-sided report but want running heads similar to those in the previous book example. Specifically, the chapter headings should be used as the running head until the first section appears. From the then on, the section heading should be used. So we modify the previous example a little bit:

```
\documentclass{scrreprt}
\usepackage[autooneside=false]{scrlayer-scrpage}
\pagestyle{scrheadings}
\automark[section]{chapter}
\usepackage{lipsum}
\begin{document}
\chapter{Chapter Heading}
\lipsum[1-20]
\section{Section Heading}
\lipsum[21-40]
\end{document}
```

As you can see an `\automark*` command is not required in this case. You should try the example with `autooneside` set to `true`, or remove the option, for comparison. You will notice a difference in the running head from page 4 on.

Note that merely loading the package does not have any effect on whether automatic or manual running heads are used, or what kind of sectioning headings fill the marks. Only by explicitly using the option `automark` or `manualmark`, or the command `\automark` or `\manualmark`, will the conditions here be initialized.

draft=*simple switch*

This KOMA-Script option accepts the values for simple switches listed in table 2.5, page 42. If this option is active, all elements of the page styles will also show rulers. This can sometimes be useful during the drafting process. If this option has been set globally (see the optional argument of \documentclass) but you do not want the rulers, you can deactivate them for this package alone by using draft=false as an optional argument of \usepackage while loading the package.

\MakeMarkcase{*text*}
markcase=*value*

Automatic running heads, but not manual ones, use \MakeMarkcase for their output. If the command has not been defined, e. g., by the class while loading scrlayer, it is defined by default to output the argument *text* without changes. But the default can be changed either by redefining \MakeMarkcase. Using the markcase option with one of the values of table 5.2 also redefines \MakeMarkcase.

Unfortunately, the LaTeX command for converting text to upper case, \MakeUppercase, does not produce good results because it neither spaces characters nor balances lines appropriately. This is certainly in part due to the fact that a typographically correct upper-case conversion requires analysing the glyphs to account for the different letter shapes and their combinations while balancing the block. I therefore recommend that you avoid upper-case typesetting for running heads. This is usually possible with markcase=used. However, some classes insert \MarkUppercase, or even the TeX command \uppercase, into the running heads. For such cases, you can use the option markcase=noupper. This will also deactivate \MakeUppercase and \uppercase inside the running heads.

You can find all valid values for markcase in table 5.2.

\leftmark
\rightmark
\headmark
\pagemark

If you want to depart from the predefined page styles, you typically need to decide where to place the marks' contents. With \leftmark you can define what will appear in the left mark when the page is output.

Similarly, you can use \rightmark to define the contents of the right mark.

You can make life easier with \headmark. This extension of scrlayer is a shorthand that resolves to either \leftmark or \rightmark depending on whether the current page is even or odd.

The \pagemark command has nothing to do with TeX's mark mechanism. It is serves to output a formatted page number. The font of element pagenumber will be used for the

Table 5.2.: Available values for the `markcase` option to select upper/lower case typesetting in automatic running heads

`lower`	redefines `\MakeMarkcase` to convert the automatic running heads into lower-case letters using `\MakeLowercase` (lower case typesetting).
`upper`	redefines `\MakeMarkcase` to convert the automatic running heads into upper-case letters using `\MakeUppercase` (upper case typesetting).
`used`	redefines `\MakeMarkcase` to use automatic running heads without any case changes.
`ignoreuppercase`, `nouppercase`, `ignoreupper`, `noupper`	redefines not only `\MakeMarkcase` but also `\MakeUppercase` and `\uppercase` locally to the running heads to leave the automatic running heads unchanged.

output. This can be changed using the `\setkomafont` or `\addtokomafont` commands (see also section 3.6, page 59).

Example: Suppose you want the running head to be aligned to the left margin and the page number to the right margin in one-sided printing. The following minimal working example does just this:

```
\documentclass{scrreprt}
\usepackage{blindtext}
\usepackage[automark]{scrlayer-scrpage}
\pagestyle{scrheadings}
\ihead{\headmark}
\ohead*{\pagemark}
\chead{}
\cfoot[]{}
\begin{document}
\blinddocument
\end{document}
```

The blindtext package and its `\blinddocument` command have been used here to quickly generate sample document content for the example.

The `\ihead` and `\ohead*` commands configure the desired marks. The starred variant `\ohead*` also configures the page number with the `plain.scrheadings` page style used on the first page of a chapter.

Because these page styles have predefined marks in the centre of the header and footer, those elements are cleared by using `\chead` and `\cfoot` with empty arguments. Alternatively you could use `\clearpairofpagestyles` *before* `\ihead`. You

will find this command described in section 18.2.

Please note that the empty optional argument of \cfoot in the example above is not the same as omitting the optional argument. You should try it yourself and have a look at the difference in the footer of the first page.

Advanced users can find more mark-setting commands starting on page 446.

```
\partmarkformat
\chaptermarkformat
\sectionmarkformat
\subsectionmarkformat
\subsubsectionmarkformat
\paragraphmarkformat
\subparagraphmarkformat
```

KOMA-Script classes and the scrlayer package typically use these commands internally to format the section numbers. They also support the \autodot mechanism of the KOMA-Script classes. If desired, these commands can be redefined to achieve a different formatting of section numbers.

Example: For example, if you want to have running heads without a section number, this is how you do it:

```
\renewcommand*{\sectionmarkformat}{}
```

```
\partmark{Text}
\chaptermark{Text}
\sectionmark{Text}
\subsectionmark{Text}
\subsubsectionmark{Text}
\paragraphmark{Text}
\subparagraphmark{Text}
```

Most classes use these commands internally to set the marks according to the sectioning commands. The argument should contain the text without the number of the sectioning unit. The number is automatically determined using the current section level if you use numbered headings.

However, not all classes use such a command for every section level. The standard classes, for example, do not call \partmark upon a \part command.

If you redefine these commands, be sure to check whether the numbers will be output via the secnumdepth before setting the number even if you do not change the secnumdepth counter yourself, because packages and classes may do so locally and rely on correct handling of secnumdepth.

The scrlayer package also redefines these commands whenever you use \automark or \manualmark or the corresponding options, to activate or deactivate the desired running heads.

\markleft{*left mark*}
\markright{*right mark*}
\markboth{*left mark*}{*right mark*}
\markdouble{*mark*}

Regardless of whether you are working with manual or automatic running heads, you can always change the contents of the *left mark* or the *right mark* using these commands. Note that the left-hand mark resulting from \leftmark will be the last mark placed on the corresponding page, while the right-hand mark resulting from \rightmark is the first mark placed on the corresponding page. For more details, see to \rightfirstmark in section 17.6, page 441.

If you are using manual running heads, the marks remain valid until they are explicitly replaced by reusing the corresponding commands. However, if you are using automatic running heads, the marks can become invalid with the next section heading, depending on the automatic configuration.

You can also use these commands in conjunction with the starred versions of the sectioning commands.

Example: Suppose you write a preface of several pages placed just before the table of contents but not appearing in it. However, since you use dividing lines in your header, you want a running head for this preface:

```
\documentclass[headsepline]{book}
\usepackage{scrlayer-scrpage}
\pagestyle{scrheadings}
\usepackage{blindtext}
\begin{document}
\chapter*{Preface}
\markboth{Preface}{Preface}
\blindtext[20]
\tableofcontents
\blinddocument
\end{document}
```

At first glance, this seems to produce the desired result. Taking a second look, however, you can see that the running title "Preface" does not appear in uppercase letters, unlike the other running heads. But that's easy to change:

```
\documentclass[headsepline]{book}
\usepackage{scrlayer-scrpage}
\pagestyle{scrheadings}
\usepackage{blindtext}
```

```
\begin{document}
\chapter*{Preface}
\markboth{\MakeMarkcase{Preface}}{\MakeMarkcase{Preface}}
\blindtext[20]
\tableofcontents
\blinddocument
\end{document}
```

Using command \MakeMarkcase results in getting the same letter case as for automatic running heads.

Now, let's move the \tableofcontents in front of the preface and remove the \markboth command. You'll discover that the preface now has the running head "CONTENTS". This is due to a quirk of \chapter* (see also section 3.16 on page 104). If you do not want a running head here, you can easily accomplish this by passing two empty arguments to \markboth:

```
\documentclass[headsepline]{book}
\usepackage{scrlayer-scrpage}
\pagestyle{scrheadings}
\usepackage{blindtext}
\begin{document}
\tableofcontents
\chapter*{Preface}
\markboth{}{}
\blindtext[20]
\blinddocument
\end{document}
```

v3.28 The command \markdouble does change the left mark and the right mark to the same contents. So \markdouble{*mark*} is a shorter form of \markboth{*mark*}{*mark*} with two identical arguments.

headwidth=*width*:*offset*:*offset*
footwidth=*width*:*offset*:*offset*

By default the header and footer are as wide as the type area. However, you can change this using these KOMA-Script options. The value *width* is the desired width of the header or footer. The *offset* defines how far the header or footer should be moved towards the outer — in one-sided printing to the right — margin. All three values are optional and can be omitted. If you omit a value, you can also omit the associated colon to the left of it. If only one *offset* is specified, it is used for both odd and even pages. Otherwise, the first *offset* is used for odd and the second *offset* for even pages in two-sided mode. If you only use one value without a colon, this will be the *width*.

v3.14

For both the *width* and the *offset* you can use any valid length value, LaTeX length, TeX dimension, or TeX skip. In addition, you can use an ε-TeX dimension expression with

Table 5.3.: Available symbolic values for the *width* value of options `headwidth` and `footwidth`

`foot`	the current width of the footer
`footbotline`	the current length of the horizontal line below the footer
`footsepline`	the current length of the horizontal line above the footer
`head`	the current width of the header
`headsepline`	the current length of the horizontal line below the header
`headtopline`	the current length of the horizontal line above the header
`marginpar`	the width of the marginal note column including the distance between the text area and the marginal note column
`page`	the width of the page taking into account any binding correction defined with the help of the typearea package (see the BCOR option in section 2.6, page 34)
`paper`	the width of the paper without considering any binding correction
`text`	the width of the text area
`textwithmarginpar`	the width of the text area including the marginal note column and the distance between the two (Note: only in this case is the default for *offset* zero)

the basic arithmetic operations `+`, `-`, `*`, `/`, and parentheses. See [Tea98, section 3.5] for more information on such expressions. See section 5.1 for more information on using a LaTeX length as an option value. The *width* can also be one of the symbolic values shown in table 5.3.

By default the header and the footer are the width of the text area. The default *offset* depends on the selected *width*. One-sided printing typically uses half the difference between *width* and the width of the text area. This centres the header horizontally above the text area. Two-sided printing, on the other hand, uses only a third of the difference between *width* and the width of the text area. However, if *width* is the width of the whole text area and the marginal note column, the default *offset* will be zero. If this is too complicated for you, you should simply specify the desired *offset* yourself.

headtopline=*thickness*:*length*
headsepline=*thickness*:*length*
footsepline=*thickness*:*length*
footbotline=*thickness*:*length*

The KOMA-Script classes provide only one separation line below the header and another above the footer, and you can only switch these lines on or off. But the scrlayer-scrpage package also lets you place lines above the header and below the footer. And for all four lines, you can not only switch them on an off but also configure their *length* and *thickness*.

Both values are optional. If you omit the *thickness*, a default value of 0.4 pt is used, producing a so-called *hairline*. If you omit the *length*, the width of the header or footer will be used. If you omit both, you can also omit the colon. If you use only one value without colon, this is the *thickness*.

Of course, the *length* can be not just shorter than the current width of the header or footer but also longer. See also the options ilines, clines and olines later in this section.

In addition to the length and thickness, you can also change the colour of the lines. Initially the colour depends on the colour of the header or footer. In addition to this, however, the settings of the corresponding elements headtopline, headsepline, footsepline and footbotline are applied. You can change these using the \setkomafont or \addtokomafont commands (see section 3.6, page 59). By default these elements are empty, so they do not change the current font or colour. Font changes at this point, unlike colour changes, make little sense anyway and are therefore not recommended for these elements.

plainheadtopline=*simple switch*
plainheadsepline=*simple switch*
plainfootsepline=*simple switch*
plainfootbotline=*simple switch*

You can use these options to apply the settings for the lines to the plain page style. You can find the available values for *simple switch* in table 2.5 on page 42. If one of these options is activated, the plain page style will use the line settings given by the options and commands described above. If the option is deactivated, the plain will not show the corresponding line.

ilines
clines
olines

As previously explained, dividing lines for the header or footer can be longer or shorter than the width of the header or footer respectively. But the question remains how these lines are aligned. By default, all lines are aligned to the left margin in one-sided printing and to the inner margin in two-sided printing. This corresponds to using the ilines option. Alternatively, you can use the clines option to centre the lines with respect to the width of the header or footer, or the olines option to align them to the outer (or right) margin.

The Day of the Week with scrdate

Originally, the scrdate package could only give the day of the week for the current date. Nowadays, it offers this and more for any date in the Gregorian calendar.

`\CenturyPart{`*`year`*`}`
`\DecadePart{`*`year`*`}`

v3.05a

The `\CenturyPart` command returns the value of the century digits — thousands and hundreds — of a *`year`*. The `\DecadePart` command, on the other hand, gives the value of the remaining digits, i. e. the tens and the units. The *`year`* can have any number of digits. You can assign the value directly to a counter or use it for calculations with `\numexpr`. To output it as an Arabic number, you should prefix it with `\the`.

Example: You want to calculate and print the century of the current year.

```
The year \the\year\ is year \the\DecadePart{\year}
of the \engord{\numexpr\CenturyPart{\year}+1\relax} century.
```

The result would be:

The year 2020 is year 20 of the 21st century.

This example uses the engord package. See [Obe10] for more information.

Note that the counting method used here treats the year 2000 as year 0 — and therefore the first year — of the 21st century. If necessary, however, you can make a correction with `\numexpr`, as shown for the ordinal number in the example.

`\DayNumber{`*`year`*`}{`*`month`*`}{`*`day`*`}`
`\ISODayNumber{`*`ISO-date`*`}`

v3.05a

These two commands return the value of the day-of-the-week number for any date. They differ only in the method of specifying the date. While the `\DayNumber` command requires the *`year`*, *`month`*, and *`day`* as separate parameters, the `\ISODayNumber` command expects an *`ISO-date`* as a single argument, *`ISO-date`*, using the ISO notation *`year-month-day`*. It does not matter if the *`month`* or *`day`* have one or two digits. You can use the result of both commands to assign directly to a counter or for calculations using `\numexpr`. To print it as an Arabic number, you should 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.

It is particularly worth noting that you can even step a specified number of days into the future or or the past from a given date.

Example: You want to know the number of the day of the week 12 days from now and 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 could be, for example:

> In 12 days, the number of the day of the week will be 5, and 24 days before the 24th December 2027 it will be 2.

The days of the week are numbered as follows: 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 are less interested in the number of the day of the week than in its name. Therefore, the `\DayNameByNumber` command returns the name of the day of the week corresponding to a day-of-the-week number. This number can be the result, for example, of `\DayNumber` or `\ISODayNumber`. The two commands `\DayName` and `\ISODayName` directly return 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 by \ISODayName{2027-12-24},
24th~December~2027 the amount of \dots.
```

The result will be:

> Please pay by Friday, 24th December 2027 the amount of ....

Once again, it is particularly worth noting that you can perform calculations, to a certain extent:

Example: You want to know the names of the day of the week 12 days from now and 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}, while two weeks
and three days after a Wednesday will be a
\DayNameByNumber{3+2*7+3}.
```

The result could be, for example:

> In 12 days, the name of the day of the week will be Friday, and 24 days before the 24th December 2027 it will be Tuesday, while two weeks and three days after a Wednesday will be a Saturday.

```
\ISOToday
\IsoToday
\todaysname
\todaysnumber
```

v3.05a

In the previous examples, the current date was always specified cumbersomely using the TEX registers `\year`, `\month`, and `\day`. The `\ISOToday` and `\IsoToday` commands directly return the current date in ISO-notation. These commands differ only in the fact that `\ISOToday` always outputs a two-digit month and day, while `\IsoToday` outputs single-digit numbers for values less than 10. The `\todaysname` command directly returns the name of the current day of the week, while `\todaysnumber` returns the number of the current day of the week. You can find more information about using this value in the explanations of `\DayNumber` and `\ISODayNumber`.

Example: I want to show you on what day of the week this document was typeset:

```
This document was created on a \todaysname.
```

This will result, for example, in:

> This document was created on a Sunday.

For languages that have a case system for nouns, note that the package cannot decline words. The terms are given in the form appropriate for displaying a date in a letter, which is the nominative singular for the currently supported languages. Given this limitation, the example above will not work correctly if translated directly into some other languages.

The names of the weekdays in scrdate all have initial capital letters. If you need the names completely in lower case, for example because that is the convention in the relevant language, simply wrap the command with the LATEX `\MakeLowercase` command:

```
\MakeLowercase{\todaysname}
```

This converts the whole argument into lower-case letters. Of course, you can also do this for \DayNameByNumber, \DayName, and \ISODayName commands described above.

\nameday{*name*}

Just as you can directly modify the output of \today with \date, so you can change the output of \todaysname to *name* with \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 write:

```
\nameday{workday}
```

After this, the previous example will result in:

This document was created on a workday.

There's no corresponding command to change the result of \ISOToday or \IsoToday.

\newdaylanguage{*language*}{*Monday*}{*Tuesday*}{*Wednesday*}{*Thursday*}{*Friday*}{*Saturday*}{*Sunday*}

Currently the scrdate package recognizes the following languages:

- Croatian (`croatian`),
- v3.13 Czech (`czech`),
- Danish (`danish`),
- Dutch (`dutch`),
- v3.13 English (`american`, `australian`, `british`, `canadian`, `english`, `UKenglish`, and USenglish),
- Finnish (`finnish`),
- v3.13 French (`acadian`, `canadien`, `francais`, and `french`),
- v3.13 German (`austrian`, `german`, `naustrian`, `ngerman`, `nswissgerman`, and `swissgerman`),
- Italian (`italian`),
- Norwegian (`norsk`),
- v3.13 Polish (`polish`),
- Slovak (`slovak`),

- Spanish (`spanish`),
- Swedish (`swedish`).

You can also configure it for additional languages. To do so, the first argument of `\newdaylanguage` is the name of the language, and the other arguments are the names of the corresponding days of the week.

In the current implementation, it does not matter whether you load scrdate before or after ngerman, babel, or similar packages. In each case the correct language will be used provided it is supported.

To be more precise, as long as you select a language in a way that is compatible with babel, scrdate will use the correct language. If this is not the case, you will get (US) English names.

Of course, if you create definitions for a language that was previously unsupported, please mail them to the author of KOMA-Script. There is a good chance that future versions of KOMA-Script will add support for that language.

The Current Time with scrtime

This package lets you find the current time. Starting with version 3.05, the package also supports the option interface already familiar from the KOMA-Script classes and various other KOMA-Script packages. See, for example, section 2.4 for more information.

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

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

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

Example: The line

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

results, for example, in:

Your train departs at 11:58.

or:

Your train departs at 23:09.

In contrast to the previous example a line like:

```
This day is already \thistime*[\ hours and\ ] minutes old.
```

results in:

This day is already 11 hours and 58 minutes old.

or:

This day is already 12 hours and 25 minutes old.

\settime{*time*}

\settime sets the output of \thistime and \thistime* to a fixed value. In this case, the optional parameter of \thistime or \thistime* is ignored, since the complete string returned by \thistime or \thistime* has been explicitly defined. \settime.

12h=*simple switch*

v3.05a

With the 12h option, you can select whether to print the time given by \thistime and \thistime* in 12- or 24-hour format. The option accepts the values for simple switches listed in table 2.5, page 42. Using the option without a value is equivalent to 12h=true, and therefore activates the 12-hour-format. The default, however, is 24h.

You can set this option globally in the optional argument of \documentclass, as a package option in the optional argument of \usepackage, or even after loading the package using \KOMAoptions or \KOMAoption (see, e. g. section 2.4, page 33). However the option no longer has any effect on the if you call \settime. After invoking this command, the time is output only with the value and in the format specified there.

For the sake of compatibility with earlier versions of scrtime, the option 24h will switch to 24-hour format if used in the optional argument of \documentclass or \usepackage. However, you should not use this option any longer.

Accessing Address Files with scraddr

The scraddr package is a small extension to KOMA-Script's letter class and letter package. Its goal is to make access to the data in address files easier and more flexible.

8.1. Overview

Basically, the package provides a new loading mechanism for address files consisting of \adrentry and the newer \addrentry format entries, as described in chapter 4 starting on page 247.

\InputAddressFile{*file name*}

The \InputAddressFile command is the main command of scraddr. It reads the content of the address file given as its parameter. If the file is not found, an error message is issued.

For each entry in this address file, the command generates a set of macros to access the data. For large address files, this will require 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 4.21, starting on page 247. The subdivision of the address file with the help of \adrchar or \addrchar, also discussed there, has no meaning for scraddr and is simply ignored by the package.

```
\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, that is, 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-empty. To guarantee safe functioning, you should use only ASCII letters in the ***key***.

Furthermore, if the file contains more than one entry with the same ***key*** name, the last occurrence will be used.

8.2. Usage

To use the package, we need a valid address file. For example, the file `lotr.adr` 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}%
          {3 Bagshot Row\\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 fourth parameter, the telephone number, has been left blank, since there are no phones in Middle Earth. And as you can see, blank fields are possible. On the other hand, you cannot simply omit an argument altogether.

With the `\InputAddressFile` command described above, we read the address file into our letter document:

```
\InputAddressFile{lotr}
```

With the help of the commands introduced in this chapter we can now write a letter to old Tom Bombadil, in which we ask him if he can remember two companions from olden times.

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

   Or \FreeIII{TOM}, as your beloved \FreeI{TOM} calls you. Do
   you still remember Mr \LastName{FRODO}, or more precisely
   \Name{FRODO}, since there was also Mr \FreeI{FRODO}. He was
   \Comment{FRODO} in the Third Age and \FreeIV{FRODO}. \Name{SAM},
   \Comment{SAM}, accompanied him.

    Their passions were very worldly. \FirstName{FRODO} enjoyed
    smoking \FreeII{FRODO}. His companion 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}
```

You can also produce the combination of `\FirstName{`*key*`}` and `\LastName{`*key*`}` used in the `\opening` of this letter with `\Name{key}`.

You can use the fifth and sixth parameters of the `\adrentry` or `\adrentry` for any purpose you wish. You can access them with the `\FreeI` and `\FreeII` commands. In this example, the fifth parameter contains the name of the most important person in the life of the person in the entry. The sixth contains the person's favourite thing. The seventh parameter is a comment or in general also a free parameter. You can access it with the `\Comment` or `\FreeIII` commands. `\FreeIV` is only valid for `\addrentry` entries. For `\adrentry` entries, it results in an error. You can find more details in the next section.

8.3. Package Warning Options

As mentioned above, you cannot use the `\FreeIV` command with `\adrentry` entries. However, you can configure how scraddr reacts in such a situation by package options. Note that this package does not support the extended options interface with `\KOMAoptions` and `\KOMAoption`. You should therefore specify the options either as global options in `\documentclass` or as local options in `\usepackage`

adrFreeIVempty
adrFreeIVshow
adrFreeIVwarn
adrFreeIVstop

These four options let you choose from four different reactions, ranging from *ignore* to *abort*, if \FreeIV is used within an \adrentry entry.

adrFreeIVempty – the command \FreeIV will be ignored

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

adrFreeIVwarn – a warning is written in the logfile

adrFreeIVstop – the LaTeX run will abort with an error message

The default setting is adrFreeIVshow.

Creating Address Files from an Address Database

In previous versions of KOMA-Script, the addrconv package was an integral part of the KOMA-Script system. The main connection to KOMA-Script was that with the help of this package, it was possible to create address files compatible with the KOMA-Script letter class or with the scraddr package from an address database in BibTeX format.

```
@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 the one above, you can use BibTeX and various BibTeX styles to create address files. There are also some special LaTeX files that make it possible to create various telephone and address lists from the address files.

However, the addrconv package was actually an independent package, including features beyond what is required for KOMA-Script. That is why addrconv has not been included in KOMA-Script for some time. The adrconv package, with a single *d*, entirely replaces addrconv. If it is not included in your TeX distribution, you can download it from [Kie10] and install it separately.

Using Basic Features of the KOMA-Script Classes in Other Classes with the scrextend Package

There are some features that are common to all KOMA-Script classes. This applies not only to the scrbook, scrreprt, and scrartcl classes, which are intended to replace the standard classes book, report, and article, but also to a large extend the KOMA-Script class scrlttr2, the successor to scrlettr, which is intended for letters. These basic features, which can be found in the classes mentioned above, are also provided by package scrextend from KOMA-Script version 3.00 onward. This package should not be used with KOMA-Script classes. It is intended for use with other classes only. If you attempt to load the package with a KOMA-Script class, scrextend will detect this and reject loading it with a warning message.

The fact that scrletter can be used not only with KOMA-Script classes but also with the standard classes is partly due to scrextend. If scrletter detects that it is not being used with a KOMA-Scriptclass, it automatically loads scrextend. Doing so makes all required KOMA-Script classes available.

Of course, there is no guarantee that scrextend will work with all classes. The package has been designed primarily to extend the standard classes and those derived from them. In any case, before you use scrextend, you should first make sure that the class you are using does not already provide the feature you need.

In addition to the features described in this chapter, there are a few more that are primarily intended for authors of classes and packages. These can be found in chapter 12, starting on page 331. The scrbase package documented there is used by all KOMA-Script classes and the scrextend package.

All KOMA-Script classes and the scrextend package also load the scrlfile package described in chapter 13 starting on page 359. Therefore the features of this package are also available when using scrextend.

In contrast, only the KOMA-Script classes scrbook, scrreprt, and scrartcl load the tocbasic package (see chapter 15 starting on page 374), which is designed for authors of classes and packages. For this reason, the features defined there are found only in those classes and not in scrextend. Of course you can still use tocbasic together with scrextend.

10.1. Early or Late Selection of Options

The information in section 2.4 applies equally to this chapter. So if you have already read and understood section 2.4, you can skip ahead to section 10.2, page 291.

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

LaTeX allows users to pass class options as a comma-separated list of keywords in the optional argument to \documentclass. In addition to being passed to the class, these options are also passed on to all packages that can understand them. Users can also pass a similar comma-separated list of keywords in the optional argument of \usepackage. KOMA-Script

v3.00 extends the option mechanism for some packages with further options. Thus most KOMA-Script options can also take a value, so an option does not necessarily take the form *option*, but can also take the form *option=value*. Except for this difference, \documentclass and \usepackage in KOMA-Script function as described in [Tea05b] or any introduction to LaTeX, for example [OPHS11].

Setting the options with \documentclass has one major disadvantage: unlike the interface described below, the options in \documentclass are not robust. So commands, lengths, counters, and similar constructs may break inside the optional argument of this command. For example, with many non-KOMA-Script classes, using a LaTeX length in the value of an option results in an error before the value is passed to a KOMA-Script package and it can take control of the option execution. So if you want to use a LaTeX length, counter, or command as part of the value of an option, you should use \KOMAoptions or \KOMAoption. These commands will be described next.

\KOMAoptions{*option list*}
\KOMAoption{*option*}{*value list*}

v3.00 KOMA-Script also provides the ability to change the values of most package options even after loading the package. You can use the \KOMAoptions command to change the values of a list of options, as in \documentclass or \usepackage. Each option in the *option list* has the form *option=value*.

Some options also have a default value. If you do not specify a value, that is if you give the option simply as *option*, then this default value will be used.

Some options can have several values simultaneously. For such options, it is possible, with the help of \KOMAoption, to pass a list of values to a single *option*. The individual values are given as a comma-separated *value list*.

KOMA-Script uses the commands \FamilyOptions and \FamilyOption with the family "KOMA" to implement this ability. See part II, section 12.2, page 336.

Options set with \KOMAoptions or \KOMAoption will reach any previously loaded KOMA-Script packages that recognise these options. If an option or a value is unknown, scrbase will report it as an error.

10.2. Compatibility with Earlier Versions of KOMA-Script

The information in section 2.5 applies equally to this chapter. So if you have already read and understood section 2.5 you can skip ahead to page 291, page 291.

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

With scrextend, you can choose whether the source file should, as much as possible, continue to produce exactly the same result within a LaTeX run or should be formatted according to the modifications of the latest version. You can specify the version with which you want your file to be compatible by using the `version` option. Compatibility with the oldest supported KOMA-Script version can be achieved with `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`, you can select the latest version. In this case, you give up backwards compatibility. If the option is used without a value, `last` is assumed. This also corresponds to the default setting, as long as you do not use any deprecated options.

v3.01a

Compatibility is primarily a question of line and page breaks (wrapping). If you choose compatibility with an older version, new options that do not affect wrapping are still available. The `version` option does not affect any wrapping changes that are the result of fixing unambiguous errors. If you need unconditional wrapping compatibility even in the case of bugs, you should physically save the old KOMA-Script version you need together with your document.

Note that you cannot change the `version` option after loading the scrextend package. Setting this option with `\KOMAoptions` or `\KOMAoption` will therefore cause an error.

10.3. Optional, Extended Features

The scrextend package provides some optional, extended features. These features are not available by default but can be activated. These features are optional because, for example, they may conflict with features of the standard classes of other commonly used packages.

`extendedfeature=`*feature*

With this option, you can activate an extended *feature* of scrextend. This option is available only while loading scrextend. You must therefore specify this option in the optional argument of `\usepackage{scrextend}`. An overview of all available features is shown in table 10.1.

Table 10.1.: Overview of the optional extended features of scrextend

title	title pages have the additional features of the KOMA-Script classes; this applies not only to the commands for the title page but also to the `titlepage` option (see section 10.7, from page 294)

10.4. Draft Mode

The information in section 3.3 applies equally to this chapter. So if you have already read and understood section 3.3, you can skip ahead to section 10.5 on page 292.

`draft=`*simple switch*
`overfullrule=`*simple switch*

The `draft` option distinguishes between documents being drafted and finished documents. The *simple switch* can be one of the standard values for simple switches from table 2.5, page 42. If you activate this option, small black boxes will be output at the end of overly long lines. These boxes make it easier for the untrained eye to locate the paragraphs that require manual post-processing. By contrast, the default, `draft=false`, shows no such boxes. Incidentally, such lines often disappear when you use the microtype package [Sch13].

v3.25 Since the `draft` option can lead to all sorts of unwanted effects with various packages, KOMA-Script allows you to control this marking of overly long lines separately with the `overfullrule` option. If this option is enabled, the marker is again displayed.

10.5. Choosing the Document Font Size

The information in section 3.5 applies equally to this chapter. So if you have already read and understood section 3.5, you can skip directly to section 10.6, page 293.

`fontsize=`*size*

While the standard classes support only a very limited number of font sizes, KOMA-Script provides the ability to specify any *size* for the main font. You can also use any known TEXunit as a unit for the *size*. If the *size* is specified without a unit, it is assumed to be `pt`.

If you set the option within the document, the main font size and the dependent font sizes of the commands `\tiny`, `\scriptsize`, `\footnotesize`, `\small`, `\normalsize`, `\large`, `\Large`, `\LARGE`, `\huge` and `\Huge` are changed. This can be useful, for example, if you want the appendix to be set in a smaller font size.

Note that using this option after potentially loading typearea does not automatically recalculate the type area and margins (see \recalctypearea, section 2.6, page 40). However, if this recalculation is performed, it will be based on the current main font size. The effects of changing the main font size upon other loaded packages or the class used depends on these packages and on the class. This means that you can encounter errors which are not the fault of KOMA-Script.

This option should by no means be misinterpreted as a substitute for \fontsize (see [Tea05a]). Also, you should not use it in place of one of the font size commands that are relative to the main font, from \tiny to \Huge.

10.6. Text Markup

The information in in section 3.6 largely applies to this chapter. So if you have already read and understood section 3.6, you can skip ahead to section 10.7, page 294. In this case, however, note that scrextend supports only the elements for the document title, the dictum, the footnotes, and the labeling environment. from table 3.2, page 60. Although the disposition element exists, scrextend uses it only for the document title.

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

With the help of the \setkomafont and \addtokomafont commands, you can attach particular font styling *commands* that change the appearance of a given *element*. Theoretically, all statements, including literal text, can be used as *commands*. You should, however, limit yourself to those statements that really change font attributes only. These are usually commands like \rmfamily, \sffamily, \ttfamily, \upshape, \itshape, \slshape, \scshape, \mdseries, \bfseries, \normalfont, as well as the font size commands \Huge, \huge, \LARGE, \Large, \large, \normalsize, \small, \footnotesize, \scriptsize, and \tiny. You can find these commands explained in [OPHS11], [Tea05b], or [Tea05a]. Colour switching commands like \normalcolor (see [Car17] and [Ker07]) are also acceptable. The use of other commands, in particular those that redefine things or or lead to output, is not supported. Strange behaviour is possible in these cases and does not represent a bug.

The command \setkomafont provides an element with a completely new definition of its font styling. In contrast, the \addtokomafont command merely extends an existing definition. You should not use either command inside the document body but only in the preamble. For examples of their use, refer to the sections for the respective element. The name and meaning of each element are listed in table 3.2, page 60 . However, in scrextend only the listed elements for the document title, dictum, footnotes, and the labeling environment are supported. Although the disposition element exists, scrextend uses it only for the document title.

The \usekomafont command can be used to switch the current font style to the specified *Element*.

Example: Suppose you want to print the document title in a red serif font. You can do this using:

```
\setkomafont{title}{\color{red}}
```

You will need the color or the xcolor package for the \color{red} command. Using \normalfont is unnecessary in this case because it is already part of the definition of the title itself. This example also needs the extendedfeature=title option (see section 10.3, page 291).

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

v3.12 Sometimes, although this is not recommended, the font setting of an element is used for settings that are not actually related to the font. If you want to apply only the font setting of an element but not those other settings, you can use \usefontofkomafont instead of \usekomafont. This will activate the font size and baseline skip, the font encoding, the font family, the font series, and the font shape of an element, but no further settings as long as those further settings are local.

You can also switch to a single one of those attributes using one of the other commands. Note that \usesizeofkomafont uses both the font size and the baseline skip.

10.7. Document Titles

This information in section 3.7 largely applies to this chapter. So if you have already read and understood section 3.7, you can skip to section 10.8, page 299. However, the capabilities of scrextend for handling the document title are part of the optional, advanced features. Therfore they are only available, if extendedfeature=title is selected while loading the package (see section 10.3, extendedfeature).

Furthermore, scrextend cannot be used with a KOMA-Script class. Because of this, you should replace

```
\documentclass{scrbook}
```

with

```
\documentclass{book}
\usepackage[extendedfeature=title]{scrextend}
```

for all examples from section 3.7, if you want to try them with scrextend.

titlepage=*simple switch*
titlepage=firstiscover
\coverpagetopmargin
\coverpageleftmargin
\coverpagerightmargin
\coverpagebottommargin

This option determines whether to use document title pages or in-page titles when using \maketitle (see page 296). Any value from table 2.5, page 42 can be used for *simple switch*.

With the titlepage=true option, invoking \maketitle creates titles on separate pages. These pages are set inside a titlepage environment, and they normally have neither header nor footer. Compared to standard LaTeX, KOMA-Script significantly expands the handling of the titles. These additional elements can be found on the following pages.

In contrast, with the titlepage=false option, invoking \maketitle creates an *in-page* title. This means that the title is specially emphasized, but it may be followed by more material on the same page, for instance an abstract or a section.

v3.12 The third choice, titlepage=firstiscover not only activates title pages but also prints the first title page of \maketitle, i. e. either the half-title or the main title, as a cover page. Any other setting of the titlepage option will cancel this setting. The margins of the cover page are given by \coverpagetopmargin, \coverpageleftmargin, \coverpagerightmargin, and \coverpagebottommargin. The defaults of these depend on the lengths of \topmargin and \evensidemargin and can be changed with \renewcommand.

The default depends on the class used and scrextend recognizes it in a way that compatible with the standard class. If a class does not set up a comparable default, it will be an in-page title.

\begin{titlepage}...\end{titlepage}

The standard classes and KOMA-Script set all title pages in a special environment: the titlepage environment. This environment always starts a new page — in two-sided printing a new right-hand page — and in single-column mode. For this page, the style is changed to \thispagestyle{empty}, so that neither page number nor running head is 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, described next, you should design a new one with the help of this environment.

A simple example for a title page with titlepage is shown in section 3.7 on page 65

\maketitle[*page number*]

While the standard classes produce at most one title page that can have three items (title, author, and date), with KOMA-Script \maketitle can produce up to six pages. In contrast to the standard classes, \maketitle in KOMA-Script accepts an optional numeric argument. If it is used, this number is the page number of the first title page. This page number is not output, but it affects the subsequent numbering. You should definitely choose an odd number, because otherwise the whole count gets mixed up. In my opinion, there are only two useful applications for the optional argument. On the one hand, you could give the the logical page number -1 to the half-title in order to give the full title page the number 1. On the other hand, you could use it to start at a higher page number, for example, 3, 5, or 7, to accommodate other title pages added by the publishing house. The optional argument is ignored for *in-page* titles. You can change the page style of such a title page by redefining the \titlepagestyle macro (see section 3.12, page 83).

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. Note also that \maketitle should not be used inside a titlepage environment. As shown in the examples, you should use either \maketitle or titlepage, but not both.

The following commands only define the contents of the title. Therefore they must be used before \maketitle. It is, however, not necessary and, when using the babel package not recommended, to include these in the preamble before \begin{document} (see [BB13]). You can find examples in section 3.7, starting on page 66.

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

In earlier times the inner book was often not protected from dirt by a cover. This function was then assumed by the first page of the book, which usually had just a short title, known as the *half-title*. Nowadays the extra page often appears before the real main 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 verso of the half-title is the frontispiece. The half-title is set on its own page even when in-page titles are used. The output of the half-title defined with \extratitle takes place as part of the title produced by \maketitle.

v3.25

An example of a simple title page with half-title and main title is shown in section 3.7 on page 66

```
\titlehead{title head}
\subject{subject}
\title{title}
\subtitle{subtitle}
\author{author}
\date{date}
\publishers{publisher}
\and
\thanks{footnote}
```

There are seven elements available for the content of the main title page. The main title page is output as part of the title pages created by \maketitle, while the definitions given here only apply to the respective elements.

The *title head* is defined with the command \titlehead. It occupies the entire text width, at the top of the page, in normal justification, and it can be freely designed by the user. It uses the font element with same name (see table 3.4, page 68).

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

The *title* is set in a very large font size. Along with the font size, the font element title is applied (see table 3.4, page 68).

The *subtitle* is set just below the title using the font element of the same name (see table 3.4, page 68).

Below the *subtitle* appears the *author*. Several authors can be specified in the argument of \author. They should be separated by \and. The output uses the font element of the same name. (see table 3.4, page 68).

Below the author or authors appears the date in the font of the element of the same name. The default value is the current date, as produced by \today. The \date command accepts arbitrary information — even an empty argument. The output uses the font element of the same name (see table 3.4, page 68).

Finally comes the *publisher*. Of course this command can also be used for any other information of minor importance. If necessary, the \parbox command can be used to typeset this information over the full page width like a regular paragraph instead of centring it. It should then be considered equivalent to the title head. Note, however, that this field is placed above any existing footnotes. The output uses the font element of the same name (see table 3.4, page 68).

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, such as in the *author* argument of the command \author. However, in order for the footnote element to be aware of the scrextend package, not only does the title extension need to be enabled, it must also be set to use footnotes with this package (see the introduction to section 10.11,

page 301). Otherwise, the font specified by the class or other packages used for the footnotes will be used.

v3.12 For the output of the title elements, the font can be set using the \setkomafont and \addtokomafont command (see section 10.6, page 293). The defaults are listed in table 3.3, page 68.

With the exception of *title head* and any footnotes, all output is centred horizontally. The formatting of each element is briefly summarized in table 3.4, page 68.

Note that for the main title, \huge will be used after the font switching element title. So you cannot change the size of the main title using \setkomafont or \addtokomafont.

An example for a title page using all the elements offered by KOMA-Script is shown in section 3.7 on page 69.

A common misconception concerns the function of the full title page. It is often erroneously assumed to be the cover or dust jacket. Therefore, it is frequently expected that the title page will not follow the normal layout for two-sided typesetting but will have equally large left and right margins.

But if you pick up a book and open it, you will quickly find at least one title page inside the cover, within the so-called book block. Precisely these title pages are produced by \maketitle.

As is the case with the half-title, the full title page belongs to the book block, and therefore should have the same page layout as the rest of the document. A cover is actually something that you should create in a separate document. After all, it often has a very distinct 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, such as cardboard, and possibly with another printer.

Nevertheless, since KOMA-Script 3.12 the first title page issued by \maketitle can be formatted as a cover page with different margins. Changes to the margins on this page do not affect the other margins. For more information about this option, see titlepage=firstiscover on page 295.

\uppertitleback{*titlebackhead*}
\lowertitleback{*titlebackfoot*}

In two-sided printing, the standard classes leave the back (verso) of the title page empty. However, with KOMA-Script the back of the full title page can be used for other information. There are exactly two elements which the user can freely format: *titlebackhead* and *titlebackfoot*. The header can extend to the footer and vice versa. Using this guide as an example, the legal disclaimer was set with the help of the \uppertitleback command.

\dedication{*dedication*}

v3.12 KOMA-Script offers its own dedication page. This dedication is centred and set by default with a slightly larger font. The exact font setting for the `dedication` element, which is taken from table 3.3, page 68, can be changed with the `\setkomafont` and `\addtokomafont` commands (see section 10.6, page 293).

An example with all title pages provided by KOMA-Script is shown in section 3.7 on page 70.

10.8. Detecting Odd and Even Pages

The information in section 3.11 applies equally to this chapter. So if you have already read and understood section 3.11, you can skip ahead to page 299, page 299.

In two-sided documents we distinguish left and right pages. Left pages always have an even page number, and right pages always have an odd page number.

\Ifthispageodd{*true part*}{*false part*}

v3.28 If you want to determine whether text appears on an even or odd page, KOMA-Script provides the `\Ifthispageodd` command. The *true part* argument is executed only if you are currently on an odd page. Otherwise the *false part* argument is executed.

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 each change to the text. Only then will the decision be correct. In the first run, a heuristic is used to make the initial choice.

In section 21.1, page 473, advanced users can find more information about the problems of detecting left and right pages, or even and odd page numbers.An example for `\Ifthispageodd` is shown on page 79 in section 3.11.

10.9. Choosing a Predefined Page Style

One of the basic features of a document is the page style. In LaTeX, the page style primarily determines the content of headers and footers. The scrextend package does not define any page styles itself. Instead it uses the page styles of the LaTeX kernel.

\titlepagestyle

On some pages `\thispagestyle` automatically selects a different page style. Currently, scrextend only does this for title pages, and only if `extendedfeature=title` has been used (see section 10.3, page 291). In this case the page style specified in `\thispagestyle` will be used. The default for `\thispagestyle` is `plain`. This page style is defined by the LaTeX kernel, so it should always be available.

10.10. Interleaf Pages

The information in section 3.13 applies equally to this chapter. So if you have already read and understood section 3.13, you can skip ahead to section 10.11, page 301.

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

With this option, you can define the page style of the interleaf pages created by the commands `\cleardoublepage`, `\cleardoubleoddpage`, or `\cleardoubleevenpage` to advance to the desired page. You can use any previously defined *page style* (see section 10.9 from page 299 and chapter 5 from page 252). In addition, `cleardoublepage=current` is also possible. This case corresponds to the default prior to KOMA-Script 2.98c and creates an interleaf page without changing the page style. Starting with KOMA-Script 3.00, the default follows the recommendation of most typographers and creates interleaf pages with the `empty` page style unless you switch compatibility to earlier KOMA-Script versions (see option `version`, section 10.2, page 291). You can find an example for setting the page style of interleaf pages in section 3.13, page 86.

```
\clearpage
\cleardoublepage
\cleardoublepageusingstyle{page style}
\cleardoubleemptypage
\cleardoubleplainpage
\cleardoublestandardpage
\cleardoubleoddpage
\cleardoubleoddpageusingstyle{page style}
\cleardoubleoddemptypage
\cleardoubleoddplainpage
\cleardoubleoddstandardpage
\cleardoubleevenpage
\cleardoubleevenpageusingstyle{page style}
\cleardoubleevenemptypage
\cleardoubleevenplainpage
\cleardoubleevenstandardpage
```

The LaTeX kernel provides the `\clearpage` command, which ensures that all pending floats are output and then starts a new page. There is also the `\cleardoublepage` command, which works like `\clearpage` but which starts a new right-hand page in two-sided printing (see the `twoside` layout option in section 2.4, page 41). An empty left-hand page in the current page style is output if necessary.

With \cleardoubleoddstandardpage, KOMA-Script works as exactly in the way just described for the standard classess. The \cleardoubleoddplainpage command, on the other hand, additionally changes the page style of the empty left page to plain in order to suppress the running title. Likewise, the \cleardoubleoddemptypage command uses the empty page style to suppress both running title and page number on the empty left-hand side. The page is thus completely empty. If you want to specify your own *page style* for the interleaf page, this should be given as an argument of \cleardoubleoddusingpagestyle. You can use any previously defined *page style* (see chapter 5).

Sometimes you want chapters to start not on the right-hand but on the left-hand page. Although this layout contradicts classic typography, it can be appropriate if the double-page spread at the beginning of the chapter very specific contents. For this reason, KOMA-Script provides the \cleardoubleevenstandardpage command, which is equivalent to the \cleardoubleoddstandardpage command except that the next page is a left page. The same applies to the \cleardoubleevenplainpage, \cleardoubleevenemptypage, and \cleardoubleevenpageusingstyle commands.

The \cleardoublestandardpage, \cleardoubleemptypage, and \cleardoubleplainpage commands, and the single-argument \cleardoublepageusingstyle command, as well as the standard \cleardoublepage command, correspond to the commands previously explained for the scrextend package to transition to the next odd page.

In two-sided printing, \cleardoubleoddpage always moves to the next left-hand page and \cleardoubleevenpage to the next right-hand page. The style of the interleaf page to be inserted if necessary is defined with the cleardoublepage option.

For an example that uses \cleardoubleevenemptypage, see section 3.13, page 88.

10.11. Footnotes

The information in section 3.14 applies equally to this chapter. So if you have already read and understood section 3.14, you can skip ahead to page 304, page 304.

The footnote capabilities of the KOMA-Script classes are also provided by scrextend. By default, the formatting of footnotes is left to the class used. This changes as soon as you issue the \deffootnote command, which is explained in detail on page 303.

The options for adjusting the dividing line above footnotes, however, are not provided by scrextend.

footnotes=*setting*
\multfootsep

Many classes mark footnotes by default in the text with a small superscript number. If several footnotes appear in succession at the same point, it gives the impression that there is one footnote with a large number rather than multiple footnotes (e. g. footnote 12 instead of footnotes 1 and 2). With footnotes=multiple, footnotes that follow each other directly are

separated with a delimiter instead. The default delimiter in \multfootsep is defined as a comma without a space:

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

This can be redefined.

The whole mechanism is compatible with the footmisc package, version 5.3d to 5.5b (see [Fai11]). It affects footnote markers placed using \footnote, as well as those placed directly with \footnotemark.

You can switch back to the default footnotes=nomultiple at any time using the \KOMAoptions or \KOMAoption command. However, if you encounter any problems using another package that alters the footnotes, you should not use this option, nor should you change the *setting* anywhere inside the document.

A summary of the available *setting* values of footnotes can be found in table 3.11, page 89.

\footnote[*number*]{*text*}
\footnotemark[*number*]
\footnotetext[*number*]{*text*}
\multiplefootnoteseparator

Footnotes in KOMA-Script are produced, as they are in the standard classes, with the \footnote command, or alternatively the pair of commands \footnotemark and \footnotetext. As in the standard classes, it is possible for a page break to occur 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 move the footnote to the next page. Unlike the standard classes, KOMA-Script can recognize and separate consecutive footnotes automatically. See the previously documented option footnotes.

If instead you want to place this delimiter manually, you can do so by calling \multiplefootnoteseparator. However, users should not redefine this command, as it contains not only the delimiter but also the delimiter's formatting, for example the font size selection and the superscript. The delimiter itself is stored in the previously described \multfootsep command.

You can find examples and additional hints in section 3.14 from page 90.

\footref{*reference*}

Sometimes you have a footnote in a document to which there are several references in the text. An inconvenient way to typeset this would be to use \footnotemark to set the number directly. The disadvantage of this method is that you need to know the number and manually set every \footnotemark command. And if the number changes because you add or remove an earlier footnote, you will have to change each \footnotemark. KOMA-Script thefore offers the \label mechanism to handle such cases. After placing a \label inside the footnote, you can

use \footref to set all the other marks for this footnote in the text. When setting footnote marks with the \label mechanism, any changes to the footnote numbers will require at least two LaTeX runs to ensure correct numbers for all \footref marks.

You can find an example of how to use \footref in section 3.14 on page 90.

Note that statements like \ref or \pageref are fragile and therefore you should put \protect in front of them if they appear in moving arguments such as headings.

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

KOMA-Script sets footnotes slightly differently than the standard classes do. As in the standard classes, the footnote mark in the text is rendered with small, superscript numbers. The same formatting is used in the footnote itself. The mark in the footnote is typeset right-justified in a box with a width of *mark width*. The first line of the footnote follows directly.

All subsequent 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, the first line of each paragraph is indented by the value of *parindent*.

figure 3.1 on page 91 shows the different parameters. The default configuration of the KOMA-Script classes is as follows:

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

\textsuperscript controls both the superscript and the smaller font size. The command \thefootnotemark contains the current footnote mark without any formatting. The scrextend package, by contrast, does not change the default footnote settings of the class you are using. Simply loading the package, therefore, should not lead to any changes in the formatting of footnote marks or footnote text. To use the default settings of the KOMA-Script classes with scrextend, you must change the settings above yourself. For example, you can insert the line of code above immediately after loading the scrextend package.

The footnote, including the footnote mark, uses the font specified in the footnote element. You can change the font of the footnote mark separately using the \setkomafont and \addtokomafont commands (see section 10.6, page 293) for the footnotelabel element. See also table 3.2, page 60. The default setting is no change to the font. However, with scrextend these elements will only change the fonts if footnotes are handled by the package, that is, after using \deffootnote. Please don't misuse this element for other purposes, for example to set the footnotes ragged right (see also \raggedfootnote, page 304).

The footnote mark in the text is defined separately from the mark in front of the actual footnote. This is done with \deffootnotemark. The default setting is:

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

With this default, the font for the `footnotereference` element is used (see table 3.2, page 60). Thus, the footnote marks in the text and in the footnote itself are identical. You can change the font with the commands `\setkomafont` and `\addtokomafont` (see section 10.6, page 293).

For examples, see section 3.14, page 92.

`\raggedfootnote`

v3.23

By default KOMA-Script justifies footnotes just as in the standard classes. But if you use `\deffootnote` you can also change the justification separately from the rest of the document by redefining `\raggedfootnote`. Valid definitions are `\raggedright`, `\raggedleft`, `\centering`, `\relax` or an empty definition, which is the default. The alignment commands of the ragged2e package are also valid (see [Sch09]). You can find a suitable example in section 3.14, page 93.

10.12. Dicta

The information in section 3.17 applies equally to this chapter. However, scrextend does not support the commands `\setchapterpreamble` and `\setpartpreamble`. Whether the class you are using offers an equivalent instruction can be found in the documentation for the respective class .So if you have already read and understood section 3.17, you can skip ahead to section 10.13, page 305.

A common element in a document is an epigraph or quotation that is set above or below a chapter or section heading, typically right-justified. The epigraph and its source are usually specially formatted. KOMA-Script refers to such an epigraph as a *dictum*.

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

You can set such a saying with the help of the `\dictum` command. The dictum, along with an optional *`author`*, is inserted in a `\parbox` (see [Tea05b]) of width `\dictumwidth`. However, `\dictumwidth` is not a length which can be set with `\setlength`. It is a macro that can be redefined using `\renewcommand`. The default is `0.3333\textwidth`, which is one third of the text width. The box itself is aligned with the command `\raggeddictum`. The default is `\raggedleft`, that is, right justified. `\raggeddictum` can be redefined with `\renewcommand`.

You can align the *`dictum`* within the box using `\raggeddictumtext`. The default is `\raggedright`, that is, left justified. You can also redefine this macro with `\renewcommand`.The output uses the default font setting for the element `dictum`, which can be

changed with the commands \setkomafont and \addtokomafont (see section 10.6, page 293). Default settings are listed in table 3.16, page 115.

If an *author* is defined, it is separated from the *dictum* by a horizontal rule spanning the full width of the \parbox. This rule is defined in \dictumrule as a vertical object with

v3.10

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

The \raggeddictumauthor command defines the alignment for the rule and the *author*. The default is \raggedleft. This command can also be redefined using \renewcommand. The format is defined with \dictumauthorformat. This macro expects the *author* text as its argument. By default \dictumauthorformat is defined as

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

Thus the *author* is set enclosed in rounded parentheses. For the dictumauthor element, you can define a different font than that used for the dictum element. The default settings are listed in table 3.16. Changes can be made using the \setkomafont and \addtokomafont commands (see section 10.6, page 293).

10.13. Lists

The information in section 3.18 applies equally to this chapter. So if you have already read and understood section 3.18, you can skip ahead to section 10.14, page 306. However, the scrextend package only defines the labeling, addmargin, and addmargin* environments. All other list environments are left to the responsibility of the class used.

Because lists are standard elements of LaTeX, examples have been omitted in this section. Nevertheless, you can find examples either in section 3.18, page 116 or in any LaTeX tutorial.

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

Another form of description list is only available in the KOMA-Script classes and scrextend : the labeling environment. Unlike description, you can specify a pattern for labeling whose length determines the indentation of all items. Furthermore, you can put an optional *delimiter* between the item and its description. The font used to format the item and the separator can be changed with the \setkomafont and \addtokomafont commands (see section 10.6, page 293) for the element labelinglabel and labelingseparator (see table 3.2, page 60).

v3.02

Originally, this environment was implemented for things like “Premise, Evidence, Proof”, or “Given, Find, Solution” that are often used in lecture handouts. These days, however, the environment has very different applications. For example, the environment for examples in this guide was defined with the labeling environment.

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

Like quote and quotation, which are available in the standard and the KOMA-Script classes, the addmargin environment changes the margin. However, unlike the first two environments, addmargin lets the user change the width of the indentation. Apart from this change, 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 the value *left indentation* is used for the left margin instead of *indentation*.

The starred variant addmargin* differs from the normal version only in the two-sided mode. Furthermore, the difference only occurs if the optional argument *inner indentation* is used. In this case, the value of *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 allow negative values for all parameters. The environment then protrudes into the margin accordingly.

Whether a page is going to be on the left or right side of the book cannot be determined reliably on the first LaTeX run. For details please refer to the explanation of the commands \Ifthispageodd (section 10.8, page 299) and \ifthispagewasodd (section 21.1).

The interplay of environments such as lists and paragraphs gives rise to frequent questions. Therefore, you can find further explanation in the description of the parskip option in section 21.1.

10.14. Marginal Notes

The information in section 3.21 applies equally to this chapter. So if you have already read and understood section 3.21, you can skip ahead to chapter 11, page 308.

In addition to the text area, which normally fills the type area, documents often contain a column for marginalia. You can set marginal notes in this area. This guide makes frequent use of them.

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

Marginal notes in LaTeX are usually inserted with the \marginpar command. They are placed in the outer margin. One-sided documents use the right border. Although you can specify a different marginal note for \marginpar in case it winds up in the left margin, marginal notes are always fully justified. However, experience has shown that many users prefer left- or right-justified marginal notes instead. For this purpose, KOMA-Script offers the \marginline command.

For a detailed example, see section 3.21 at page 145.

Advanced users will find notes about difficulties that can arise using `\marginpar` in section 21.1. These remarks also apply to `\marginline`. In addition, chapter 19 introduces a package that you can use to create note columns with their own page breaks.

Support for the Law Office with scrjura

If you want to write a contract, the articles of association for a company or an association, a law, or a legal commentary, the package scrjura will provide typographical support. Although scrjura is intended to provide general help for legal documents, the contract is the central element of the package. Particular attention is paid to clauses, titles, and numbered provisions — if there are several of them in a clause —, numbered sentences, entries in the table of contents, and cross references according to German standards.

The package has been developed in cooperation with Dr Alexander Willand of Karlsruhe. Many of its features go back to constructive inquiries from Prof Heiner Richter of the Hochschule Stralsund University of Applied Sciences.

Note that the package works with hyperref. Nevertheless, hyperref has to be loaded after scrjura as usual.

11.1. Early or Late Selection of Options

The information in section 2.4 applies equally to this chapter. So if you have already read and understood section 2.4, you can skip ahead to section 11.2, page 309.

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

LaTeX allows users to pass class options as a comma-separated list of keywords in the optional argument to \documentclass. In addition to being passed to the class, these options are also passed on to all packages that can understand them. Users can also pass a similar comma-separated list of keywords in the optional argument of \usepackage. KOMA-Script extends the option mechanism for some packages with further options. Thus most KOMA-Script options can also take a value, so an option does not necessarily take the form *option*, but can also take the form *option=value*. Except for this difference, \documentclass and \usepackage in KOMA-Script function as described in [Tea05b] or any introduction to LaTeX, for example [OPHS11].

v3.00

Setting the options with \documentclass has one major disadvantage: unlike the interface described below, the options in \documentclass are not robust. So commands, lengths, counters, and similar constructs may break inside the optional argument of this command. For example, with many non-KOMA-Script classes, using a LaTeX length in the value of an option results in an error before the value is passed to a KOMA-Script package and it can take control of the option execution. So if you want to use a LaTeX length, counter, or command as part of the value of an option, you should use \KOMAoptions or \KOMAoption. These commands will be described next.

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

v3.00

KOMA-Script also provides the ability to change the values of most package options even after loading the package. You can use the \KOMAoptions command to change the values of a list of options, as in \documentclass or \usepackage. Each option in the *option list* has the form *option=value*.

Some options also have a default value. If you do not specify a value, that is if you give the option simply as *option*, then this default value will be used.

Some options can have several values simultaneously. For such options, it is possible, with the help of \KOMAoption, to pass a list of values to a single *option*. The individual values are given as a comma-separated *value list*.

KOMA-Script uses the commands \FamilyOptions and \FamilyOption with the family "KOMA" to implement this ability. See part II, section 12.2, page 336.

Options set with \KOMAoptions or \KOMAoption will reach both the KOMA-Script class and any previously loaded KOMA-Script packages that recognise these options. If an option or a value is unknown, scrbase will report it as an error.

11.2. Text Markup

The information in in section 3.6 largely applies to this chapter. So if you have already read and understood section 3.6, you can skip ahead to section 11.3, page 311.

LaTeX offers different possibilities for logical and direct markup of text. In addition to the choice of the font, this includes commands for choosing the font size and orientation. For more information about the standard font facilities, see [OPHS11], [Tea05b], and [Tea05a].

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

With the help of the \setkomafont and \addtokomafont commands, you can attach particular font styling *commands* that change the appearance of a given *element*. Theoretically, all statements, including literal text, can be used as *commands*. You should, however, limit yourself to those statements that really change font attributes only. These are usually commands like \rmfamily, \sffamily, \ttfamily, \upshape, \itshape, \slshape, \scshape, \mdseries, \bfseries, \normalfont, as well as the font size commands \Huge, \huge, \LARGE, \Large, \large, \normalsize, \small, \footnotesize, \scriptsize, and \tiny. You can find these commands explained in [OPHS11], [Tea05b], or [Tea05a]. Colour switching commands like \normalcolor (see [Car17] and [Ker07]) are also acceptable. The use of other commands, in particular those that redefine things or or lead to output, is not supported. Strange behaviour is possible in these cases and does not represent a bug.

Table 11.1.: Elements whose scrjura font styles can be changed with \setkomafont and \addtokomafont, including their default settings

Clause alias for *environment name*.Clause within a contract environment, for example contract.Clause within contract; if no corresponding element is defined, contract.Clause is used
contract.Clause heading of a paragraph within contract (see section 11.4, page 313); Default: \sffamily\bfseries\large
Name.Clause heading of a paragraph within a *name* environment defined with \DeclareNewJuraEnvironment as long as the setting was made with ClauseFont or the item was subsequently defined (see section 11.6, page 321); Default: *none*
parnumber paragraph numbers within a contract environment (see section 11.4.2, page 316); Default: *empty*
sentencenumber v3.26 sentence number of \Sentence (see section 11.4.3, page 317); Default: *empty*

The command \setkomafont provides an element with a completely new definition of its font styling. In contrast, the \addtokomafont command merely extends an existing definition. You should not use either command inside the document body but only in the preamble. For examples of their use, refer to the sections for the respective element. The name and meaning of each element , as well as their defaults, are listed in table 11.1 .

With the \usekomafont command, the current font style can be changed to the one defined for the specified *element*.

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

v3.12 Sometimes, although this is not recommended, the font setting of an element is used for settings that are not actually related to the font. If you want to apply only the font setting of an element but not those other settings, you can use \usefontofkomafont instead of

\usekomafont. This will activate the font size and baseline skip, the font encoding, the font family, the font series, and the font shape of an element, but no further settings as long as those further settings are local.

You can also switch to a single one of those attributes using one of the other commands. Note that \usesizeofkomafont uses both the font size and the baseline skip.

However, the misuse of the font settings is strongly discouraged (see section 21.5, page 476)!

11.3. Table of Contents

The headings of clauses can also be added automatically to the table of contents, if desired. Therefore the package uses \DeclareTOCStyleEntry (see section 15.3, page 386) to define an *entry level* named cpar.

v3.27

juratotoc=*simple switch*
juratotoc=*level number*

Clauses are shown in the table of contents only if their *level number* is less than or equal to the tocdepth counter (see section 3.9, page 76). By default, the *level number* is \maxdimen, which is also used if the option is switched off with the *simple switch* (see table 2.5, page 42). Because the tocdepth counter usually has a one-digit value, clause entries are therefore not normally displayed in the table of contents.

If you switch on the option using the *simple switch*, the *level number* 2 is used so that clauses are shown in the table of contents on the same level as subsections. For the default setting of tocdepth, clauses are then shown in all KOMA-Script classes.

v3.27

Internally usage of this option results in a call of \DeclareTOCStyleEntry with the *style* default and a corresponding value of level in the *option list*.

juratocindent=*indent*
juratocnumberwidth=*number width*

These two options can be used to determine the indentation for the clause entries in the table of contents as well as the space reserved for the numbers there. The defaults are the same as for subsection entries in scrartcl.

v3.27

Internally usage of these options results in calls of \DeclareTOCStyleEntry with the *style* default and indent=*indent* respectively numwidth=*number width* in the *option list*.

11.4. Environment for Contracts

The essential mechanisms of scrjura are available only inside the contract environment.

`\begin{contract}...\end{contract}`

Currently, this is the one and only environment for scrjura. Using it activates automatic numbering of paragraphs and the `\Clause` and `\SubClause` commands, which will be documented below, are given concrete form.

The `contract` environment must not be nested within itself. Within a document, however, you can use the environment several times. The clauses within these environments are treated as if they were within a single environment. As a result, ending the environment really only temporarily interrupts it, and the old environment is continued by the beginning of a new environment. However, you cannot end the environment within a clause.

`contract`

The whole document becomes a contract if you use this option while loading the package with `\usepackage` or as a global option with `\documentclass`. The document then behaves exactly as if it contained one `contract` environment.

Note that you cannot set the `contract` option with `\KOMAoption` or `\KOMAoptions`. Thus you cannot switch the option off again. Instead, you should use a `contract` environment directly.

11.4.1. Clauses

Clauses[1] in a legal sense are defined in scrjura only within contracts, that is inside the `contract` environment or other environments declared with `\DeclareNewJuraEnvironment` (see section 11.6, page 319).

`\Clause[`*`options`*`]`
`\SubClause[`*`options`*`]`

These are the most important commands inside of a contract. Without using any additional *`options`*, `\Clause` creates the heading of a clause, which consists of the sign "§", followed by its number. In contrast, `\SubClause` creates the heading of a clause with the last number used by `\Clause` and adds a lower-case letter. `\SubClause` is mainly intended for cases where an act or a contract is amended and not only are clauses changed or deleted but new clauses are inserted between existing ones without completely changing the numbering.

Both commands accept a a comma-separated list of *`options`*. An overview of the available properties is shown in table 11.2. The most important of them will be discussed in more detail.

By default, a skip of two lines is inserted before the heading and a skip of one line afterwards. You can change the size of these skips with the `preskip` and `postskip` options. The new values

[1] In English, a "clause" in a legal document is a section, paragraph, or phrase that relates to a particular point. Although it is common in English to also use the terms "article" or "section" for what we here call a "clause", we use the latter term throughout to avoid confusion with the article class and the `\section` and `\paragraph` sectioning divisions of most document classes.

Table 11.2.: Available properties for the optional argument of `\Clause` and `\SubClause`

`dummy` The heading will not be printed but is counted in the automatic numbering.
`head=`*`running head`* If running heads are active, this *`running head`* is used instead of the clause *`title`*.
`nohead` The running head stays unchanged.
`notocentry` Does not make an entry into the table of contents.
`number=`*`number`* Uses *`number`* for the output of the clause number.
`preskip=`*`skip`* Changes the vertical *`skip`* before the clause heading.
`postskip=`*`skip`* Changes the vertical *`skip`* after the clause heading.
`title=`*`title`* The clause *`title`* will be printed in addition to the clause number. This is also used as the default for the *`running head`* and the *`entry`* in the table of contents.
`tocentry=`*`entry`* Regardless of the clause *`title`*, an *`entry`* into the table of contents will be made, if such entries are activated.

apply not only to the current clause but from the current clause until the end of the current contract environment. You can also make the appropriate settings in advance with

```
\setkeys{contract}{preskip=skip,
                   postskip=skip}
```

regardless of the specific clause and outside of a contract environment. You can also set these options inside the preamble after loading scrjura, but you cannot set them while loading the package or by using \KOMAoptions or \KOMAoption.

By default, clause headings use the font style `\sffamily\bfseries\large`. You can change this font style at any time using the `contract.Clause` element with \setkomafont and \addtokomafont (see section 3.6, page 309). Inside the `contract` environment, you can also use `Clause` instead of `contract.Clause`.

With the `title`, `head`, and `tocentry` options, you can title a clause in addition to the number. You should put the value of each options inside curly brackets. Otherwise, for example, commas which are meant to be part of the value will be confused with the delimiters between different options. Empty values for `head` and `tocentry` cause empty entries. If you want to avoid an entry, use the `nohead` and `notocentry` options.

Instead of consecutive numbers, you can also set a clause number manually with the `number` option. However, this does not affect the numbers of the subsequent clauses. Empty numbers are not possible. Fragile commands inside *number* have to be protected with `\protect`. You should use only numbers and letters as a `number`.

With the `dummy` option, you can suppress the output of the whole heading of a clause. The automatic numbering, however, will still count this clause. In this way, you can skip an automatically numbered clause with

```
\Clause{dummy}
```

in case the clause corresponding clause has been deleted in a later version of a contract.

Note that the `dummy` option only accepts the values `true` and `false`. All other values are usually ignored, but can lead to an error message in the worst case scenario.

`\Clauseformat{`*number*`}`

As already mentioned, clauses and subclauses are normally numbered. The number is formatted with the help of the `\Clauseformat` command, which expects the number as the only argument. The default is the following:

```
\newcommand*{\Clauseformat}[1]{\S~#1}
```

This produces the section mark, `\S` (§), followed by a non-breaking space and the number. If you redefine this command, be sure it remains expandable.

`juratitlepagebreak=`*simple switch*

Usually, page breaks are prohibited within heading of all kinds. However, some lawyers require page breaks within clause headings. You can allow such a break by using `juratitlepagebreak`. You can find the available values for *simple switch* in table 2.5, page 42.

`clausemark=`*value*

Since clauses are a subordinate structure with independent numbering, they do not produce running heads by default. You can, however, create running heads with various settings. You can find the available *values* and their meanings in table 11.3.

11.4.2. Paragraphs

Within clauses, scrjura usually numbers paragraphs automatically. With this, the paragraphs provide a powerful structuring element, similar to `\paragraph` or `\subparagraph` in normal documents. For this reason, contracts usually use a vertical skip between paragraphs. The scrjura package does not provide its own mechanism for this. Instead, it uses the `parskip` option of the KOMA-Script classes (see section 3.10, page 77).

Table 11.3.: Available values for the `clausemark` option to activate running heads

`both`	Clauses generate left and right marks for running heads, if the document provides automatic running heads.
`false`, `off`, `no`	Clauses do not generate marks for running heads and therefore do not change running heads.
`forceboth`	Clauses use `\markboth` to generate left and right marks for running heads even if the document does not provide automatic running heads for the current page style.
`forceright`	Clauses use `\markright` to generate right marks for running heads even if the document does not provide automatic running heads for the current page style.
`right`	Clauses generate right marks for running heads, if the document provides automatic running heads.

`parnumber=`*value*

The default numbering of paragraphs is `parnumber=auto` and `parnumber=true`. Sometimes you may need to disable the automatic numbering. You can do this with `parnumber=false`. In this case, only the sentence numbering is reset.

To implement this option, it has been necessary to hook into the paragraph-building mechanism of LaTeX. In some rare cases, this can have a negative effect. If so, you can undo the change with `parnumber=manual`. On the other hand, LaTeX itself sometimes undoes the change. In those cases you can activate it again with `parnumber=auto`.

Clauses that consist of a single paragraph do not automatically receive a paragraph number. For this to work, there must not be two clauses with an identical number in a document. However should you ever need such numbering, you should switch to another contract environment (see `\DeclareNewJuraEnvironment`, section 11.6, page 319). Note that the number of paragraphs in a clause is not available before the end of the clause. Therefore you need a least two LaTeX runs before the automatic paragraph numbering is correct.

par
\thepar
\parformat
\parformatseparation

For numbering the paragraphs inside a clause we use the par counter. The output of \thepar will display an Arabic number, because the default is \arabic{par}. \parformat provides the format, which is \thepar in rounded brackets. When numbering a paragraph manually, you should also use \parformat. It makes sense to call \parformat with a subsequent \parformatseparation, or at least a \nobreakspace or tilde.

v0.7 With automatic numbering, \parformat is followed by \parformatseparation, which currently consists of \nonbreakspace, the non-breakable space.

The paragraph number is usually printed using the currently active font. However, you can change this default for the parnumber element at any time with \setkomafont and \addtokomafont (see section 11.2, page 309).

Note that scrjura assumes internally that \thepar is an Arabic number. Therefore you should definitely not redefine it!

\withoutparnumber

If the paragraph number is not printed, scrjura executes the \withoutparnumber command at the beginning of the new paragraph. The initial definition of this command is empty. This means it is a kind of dummy command that does nothing. It has been implemented because of a user request. Most users can ignore this command.

\ellipsispar[*number*]
\parellipsis

v0.7 Sometimes — particularly in comparative commentaries — it is desirable to omit paragraphs but to mark the omission. Those omitted paragraphs should be taken into account by the paragraph counter. The package scrjura provides the command \ellipsispar to do this.

By default, \ellipsispar omits precisely one paragraph. Using the optional argument, you can omit multiple paragraphs. In any case, the output shows just one unnumbered paragraph, which consists only of the ellipsis defined by \parellipsis. The automatic numbering of paragraphs takes the *number* of omitted paragraphs into account.

Example: Suppose you are writing a comment on the German[2] penal code, but only on paragraph 3 of § 2. Nevertheless, you'd like to indicate the omission indirectly. You can do this with:

[2] Please remember, this translation does not refer to an existing law but is only an example of how you might realise such a commentary with scrjura.

```
\documentclass[parskip=half]{scrartcl}
\usepackage{scrjura}
\begin{document}
\begin{contract}
  \Clause{title={Temporal application},number=2}
  \ellipsispar[2]

  If the law that applies at the time the criminal act is
  committed is changed before the verdict, then the most
  lenient law shall be applicable.

  \ellipsispar[3]
\end{contract}
\end{document}
```

To see the result, just give it a try.

The ellipsis is by default `\textellipsis`, if such a command is defined. If not, `\dots` is used. You can redefine `\parellipsis` at any time with `\renewcommand`.

11.4.3. Sentences

Paragraphs in contracts consist of one or more sentences, some of which may also be numbered. However, as automatic numbering is cumbersome and error-prone, it has not yet been implemented in scrjura. Semi-automatic numbering, however, is supported.

```
sentence
\thesentence
\sentencenumberformat
\Sentence
```

Manual numbering of sentences is done with the `\Sentence` command. It adds one to the `sentence` counter. By default, `\sentencenumberformat` prints `\thesentence` as an Arabic number in superscript.

v3.26

v3.26

The sentence number is usually printed using the currently active font. However, you can change this default for the `parnumber` element at any time with `\setkomafont` and `\addtokomafont` (see section 11.2, page 309).

Using babel offers an easy way to define a shorthand for `\Sentence`:

```
\useshorthands{'}
\defineshorthand{'S}{\Sentence\ignorespaces}
```

With this definition, any space after `'S` will be ignored. You can even use the dot as an abbreviation for a dot and a new sentence number:

```
\defineshorthand{'.}{. \Sentence\ignorespaces}
```

For details regarding \useshorthands and \defineshorthands, please consult the manual of the babel package (see [BB13]). You can find an example of their application in section 11.8, page 323.

11.5. Cross-References

The conventional mechanism to set cross-references using \label, \ref, and \pageref does not suffice for clauses, paragraphs, and sentences. Therefore scrjura provides additional commands.

```
\ref{label}
\refL{label}
\refS{label}
\refN{label}
```

The commands \refL, \refS, and \refN give a full reference to clause, paragraph and sentence. \refL is a long text, \refS a short text, and \refN an abbreviated, numeric form. \ref defaults to \refL.

```
\refClause{label}
\refClauseN{label}
```

These commands reference a clause without displaying the paragraph or sentences. \refClause puts a section mark (§) in front of the reference, while \refClauseN does not.

```
\refPar{label}
\refParL{label}
\refParS{label}
\refParN[number format]{label}
```

You can reference just a paragraph with \refParL, \refParS and \refParN. The differences between the forms correspond to the differences between \refL, \refN and \refS. A feature worth noting is the optional argument of \refParN. Usually the numeric reference to a paragraph uses a Roman number. You can, however, specify a different *number format* in the optional argument. This option primarily makes sense to use Arabic numbers. By default, \refPar is \refParL.

\refSentence{*label*}
\refSentenceL{*label*}
\refSentenceS{*label*}
\refSentenceN{*label*}

You can reference a sentence with \refSentenceL, \refSentenceS, or \refSentenceN. Again, there is a long text form, a short text form, and a numerical form. By default, \refSentence is \refSentenceL.

ref=*value*

The results of \ref, \refPar, and \refSentence depend on the *value* of the ref option. The defaults are \refL, \refParL and \refSentenceL. You can find the available values and their meaning in table 11.4.

Example: Suppose you always want to reference paragraphs in the form "paragraph 1 in clause 1". As there is no predefined command for this, you have to create your own definition from the available options. You can achieve this easily with:

```
\newcommand*{\refParM}[1]{%
  paragraph~\refParN[arabic]{#1}
  in clause~\refClauseN{#1}%
}
```

This new command can be used in the same way as \refParL.

You can find examples of results of the basic commands in table 11.5.

11.6. Additional Environments

Some users do not use scrjura to draft contracts or commentaries on individual laws but to examine different types of laws, which may not necessarily use the section prefix (§) before the title of each clause but perhaps something like "Art." or "IAS", and so forth. An independent counter is also required for each of these different clause types.

\DeclareNewJuraEnvironment{*name*}[*options*]{*start commands*}{*end commands*}

v0.9

You can use this command to define new and independent environments for contracts or other legal texts. The argument *name* is the name of the new environment, of course. The *start commands* are commands which will be executed at the beginning of the environment, as if they were added directly after \begin{*name*}. Correspondingly *end commands* will be executed at the end of the environment, as if added directly before \end{*name*}. Without any *options* the new environment behaves like the contract environment, but with its own counters. It is possible to set *options* in a comma-separated list. See table 11.6 for the currently supported *options*.

Table 11.4.: Available values for the `ref` option to configure the cross-reference format of `\ref`, `\refPar`, and `\refSentence`

`long` A combination of `parlong` and `sentencelong`.
`numeric` A combination of `parnumeric` and `sentencenumeric`.
`clauseonly`, `onlyclause`, `ClauseOnly`, `OnlyClause` A combination of `paroff` and `sentenceoff`. Note that `\refPar` and `\refSentence` produce empty results!
`parlong`, `longpar`, `ParL` Paragraphs are referenced in long textual form.
`parnumeric`, `numericpar`, `ParN` Paragraphs are referenced in simple numerical form.
`paroff`, `nopar` Paragraphs have no reference. Note that `\refPar` produces an empty result!
`parshort`, `shortpar`, `ParS` Paragraphs are referenced in short textual form.
`sentencelong`, `longsentence`, `SentenceL` Sentences are referenced in long textual form.
`sentencenumeric`, `numericsentence`, `SentenceN` Sentences are referenced in simple numeric form.
`sentenceoff`, `nosentence` Sentences have no reference. Note that `\refSentence` produces an empty result!
`sentenceshort`, `shortsentence`, `SentenceS` Sentences are referenced in short textual form.
`short` A combination of `parshort` and `sentenceshort`.

Table 11.6.: Options provided by `\DeclareNewJuraEnvironment` for new contract environments

`Clause=`*`command`* Defines the *`command`* to which the `\Clause` command is mapped within the environment. This *`command`*, like the one documented for `contract`, expects exactly one argument. To use it correctly requires advanced knowledge of the scrjura's internal functioning. Furthermore, the requirements for the *`command`* may change in future versions. Therefore it is recommended not to use this option!

...

Table 11.5.: Example outputs of the `ref`-independent cross-reference commands

Command	Example output
`\refL{`*label*`}`	§ 1 paragraph 1 sentence 1
`\refS{`*label*`}`	§ 1 par. 1 sent. 1
`\refN{`*label*`}`	§ 1 I 1.
`\refClause{`*label*`}`	§ 1
`\refClauseN{`*label*`}`	1
`\refParL{`*label*`}`	paragraph 1
`\refParS{`*label*`}`	par. 1
`\refParN{`*label*`}`	I
`\refParN[arabic]{`*label*`}`	1
`\refParN[roman]{`*label*`}`	i
`\refSentenceL{`*label*`}`	sentence 1
`\refSentenceS{`*label*`}`	sent. 1
`\refSentenceN{`*label*`}`	1.

Table 11.6.: Options of `\DeclareNewJuraEnvironment` (*continued*)

`ClauseFont=`*commands*

If this option is used, a new *Name*`.Clause` element is defined using `\newkomafont`, with the *commands* used as its default setting. If the element was previously defined as an alias (see `\aliaskomafont` in section 21.5, page 476), it will become an independent element instead. If it has already been defined as an independent element, `\setkomafont` is used to set the *commands* a new font settings. Please note the limitations for font settings in section 11.2, page 309.

`SubClause=`*command*

Defines the *command* to which the `\SubClause` command is mapped within the environment. This *command*, like the one documented for `contract`, expects exactly one argument. To use it correctly requires advanced knowledge of the scrjura's internal functioning. Furthermore, the requirements for the *command* may change in future versions. Therefore it is recommended not to use this option!

`Sentence=`*command*

Defines the *command* to which the `\Sentence` is mapped within the environment. This *command* must not have an argument. Typically it should add one to the `sentence` (using `\refstepcounter`) counter and display it appropriately. It is particularly important to avoid adding unwanted spaces.

...

Table 11.6.: Options of `\DeclareNewJuraEnvironment` (*continued*)

`ClauseNumberFormat=`*command*
: Formats the numbers of clauses within the environment. The *command* should expect exactly one argument: the number of the clause. If the *command* implements a series of commands and the number is the last argument of a that series, you can directly use the series of commands as *command*.

Example: To define the environment for articles we mentioned in the preface of this section, it is sufficient to write:

```
\DeclareNewJuraEnvironment{Article}[ClauseNumberFormat=Art.~]{}{}
```

If we are using a KOMA-Script class and want to separate the paragraphs in this environment with space instead of using paragraph indentation, we can use:

```
\DeclareNewJuraEnvironment{Article}[ClauseNumberFormat=Art.~]
                           {\KOMAoptions{parskip}}{}
```

In cross-references, "Art." will of course be used instead of "§".

The new environment is used like `contract`:

```
\begin{Article}
  \Clause{}
  Human dignity is inviolable. To respect and protect people is a
  duty of all state authority.
\end{Article}
```

11.7. Support for Different Languages

The scrjura package has been developed in cooperation with a German lawyer. Therefore it initially supported only the languages `german`, `ngerman`, `austrian`, and `naustrian`. Nevertheless, it has been designed to support common language packages like babel. Users can easily make changes by using `\providecaptionname` (see section 12.4, page 350). If you have definitive information about the correct legal terms and conventions of a language, please contact the KOMA-Script author. Support for English has been added in this way, and so scrjura now also provides terms for the languages `english`, `american`, `british`, `canadian`, `USenglish`, and `UKenglish`.

Table 11.7.: Meanings and English defaults of language-dependent terms, if not already defined

Command	Meaning	Default
\parname	long form “paragraph”	paragraph
\parshortname	short form “paragraph”	par.
\sentencename	long form “sentence”	sentence
\sentenceshortname	short form “sentence”	sent.

```
\parname
\parshortname
\sentencename
\sentenceshortname
```

These are the language-dependent terms used by scrjura. The meaning of the terms and their English defaults are shown in table 11.7. The package itself defines them by using \providecaptionname inside \begin{document} only if other requirements have not already been met. If you use scrjura with an unsupported language, the package will throw an error.

11.8. A Detailed Example

You may remember the letter from chapter 4, in which a club member wanted to remind the board about an overdue meeting that was prescribed by the club’s by-laws. Such club by-laws are a kind of contract, and you can create them using scrjura.

```
\documentclass[fontsize=12pt,parskip=half]
              {scrartcl}
```

We use class scrartcl. Because paragraph distance instead of paragraph indentation is usual in club by-laws, we load the class with option parskip=half (see section 3.10, page 77).

```
\usepackage[british]{babel}
```

The club rules are in British English. Therefore we load the babel package with the british option too.

```
\usepackage[T1]{fontenc}
\usepackage{lmodern}
\usepackage{textcomp}
```

We make some default font settings and load the textcomp package because it provides an improved section mark (§) and — something that may be useful in other circumstances — a usable Euro symbol for some fonts.

Although it is not relevant for English, if we were writing an another language, we would want to input non-ASCII characters directly. To do so, we could let LaTeX detect the input encoding with the selinput or set it with the inputenc packages.

```
\usepackage{enumerate}
```

Later in the document, we want lists numbered not with Arabic numbers but with lower-case letters. We can do this easily with the enumerate package.

```
\usepackage[clausemark=forceboth,
            juratotoc,
            juratocnumberwidth=2.5em]
           {scrjura}
\useshorthands{'}
\defineshorthand{'S}{\Sentence\ignorespaces}
\defineshorthand{'.}{. \Sentence\ignorespaces}

\pagestyle{myheadings}
```

Now it is time for scrjura. The `clausemark=forceboth` option forces clauses to create left and right marks for the running head. On the other hand, we do not want `\section` to change the marks for the running head. Therefore we use the `myheadings` page style. This page style generally does not provide automatic running heads.

Later, we also want a table of contents with the clauses. This can be achieved with the `juratotoc` option. Doing so we will see that the default width for these numbers is insufficient for the clause numbers in the table of contents. With `juratocnumberwidth=2.5em`, we reserve more space.

The definition of shorthands has already been explained in section 11.4.3. In this example we do the same thing to simplify the input.

```
\begin{document}
```

It is time to begin the actual document.

```
\subject{By-Laws}
\title{CfCH}
\subtitle{Club for Club Hoppers}
\date{11.\,11.\,2011}
\maketitle
```

Like other documents, the by-laws have a title. We created it with the usual KOMA-Script commands (see section 3.7, starting on page 64).

```
\tableofcontents
```

As already mentioned, we want to create a table of contents.

```
\addsec{Preamble}

The club landscape in England is diverse. But we have
unfortunately been forced to conclude that it often
suffers seriously when dealing with seriousness.
```

Preambles are not unusual in club by-laws. Here we use `\addsec` to create one because we want it to have an entry in the table of contents.

```
\appendix
```

Here we use a small trick. The articles of the club by-laws should be numbered with upper-case letters instead of Arabic numbers, just as the appendix sections of an article using scrartcl are.

```
\section{Overview}

\begin{contract}
```

We begin the contract with the first article.

```
\Clause{title={Name, Legal Form, Headquarters}}

The name of this club shall be the ‘‘Club for Club
Hoppers’’ and is not registered in any club register.

’S The club is a non-economic, useless club’. It has no
headquarters because its members heads are in their
hindquarters.

The fiscal year is from March 31st through April 1st.
```

The first clause has a number and a title. We will do the same with all following clauses.

The first paragraph of the clause contains nothing unusual. Because it is not the only paragraph, every paragraph will be automatically preceded by a paragraph number. Note that the numbering the first paragraph requires at least two LaTeX runs. Since this is the case for the table of contents as well, this does not create any additional problems.

In the second paragraph we have two sentences. Here we can see the shorthands `’S` and `’.` in action. The first one only generates the sentence number. The second one generates not only the full stop but also the sentence number. With this, both sentences are numbered.

```
\Clause{title={Purpose of the Club}}

’S The club is pointless but not useless’. Rather,
it should put the serious handling of seriousness on a
sound footing.

For this purpose, the club members can
\begin{enumerate}[\qquad a)]
\item pick their noses,
\item crack nuts,
\item such their thumbs.
\end{enumerate}

The club is selfish and stands by it.

The club has no financial means.\label{a:mittel}
```

The second clause: again this contains several paragraphs, some of which include several sentences. The second paragraph also has a numbered list. In the last paragraph, we set a label, because we want to reference it later.

```
\Clause{title={Club Officers}}

The club officers hold honorary positions.

'S If the club had resources (see \ref{a:mittel}), it
could afford a full-time manager'. Without the necessary
funds, this is not possible.
```

The third clause contains something special: a cross-reference. Here we use the long form with clause, paragraph, and sentence. If we decided later that sentences should not be included in the reference, we could use the `ref=nosentence` option to set this globally.

```
\Clause{title={Club Hopper},dummy}
\label{p.maier}
```

Here we have a special kind of clause. In earlier versions of the club by-laws, this was a real clause, but it was later removed. However, the numbering of the following clauses should not be changed by removing this one. Therefore the `\Clause` statement has not been removed but supplemented by option `dummy`. With this, we also can set a label even though the clause will not be printed.

```
\end{contract}

\section{Membership}

\begin{contract}
```

Another article begins. To avoid problems with the paragraph numbering, we interrupt the `contract` environment.

```
\Clause{title={Types of Members},dummy}
```

The first clause of the next article also has been deleted.

```
\Clause{title={Becoming a Member}}

Everyone can purchase a membership from one of the
associations listed in \refClause{p.maier}.\label{a.preis}

'S To become a member, an informal application is
required'. This application should be submitted in green
ink on pink paper.

Membership applications cannot be rejected.
```

Here we have a real clause again. We cross-reference one of the deleted clauses and also set a label.

```
\SubClause{title={Amendment to the Previous Clause}}

'S With the repeal of \refClause{p.maier},
\ref{a.preis} has become impractical'. In its place,
memberships can be inherited.
```

Once more, this is a special kind of clause. This time we have not removed a clause but added one without renumbering the following clauses. To do so, we use \SubClause. Therefore the clause number is the same like the previous one but with an appended "a".

```
\Clause{title={Termination of Membership}}

'S Membership ends with one's life'. For non-living
members, membership does not end.

\Clause{title={General Meeting}}

A general meeting shall take place twice per year.

The interval between two general meetings shall be
no more than 6~months, 1~week, and 2~days.

The invitation to the next general meeting shall be sent
no earlier than 6~months from the previous general
meeting.

\SubClause{title={Amendment to the General Meeting}}

The general meeting may be held at the earliest 2~weeks after
the invitation is received.
\end{contract}
```

The other clauses of this article are very usual. You already know all the features used for them.

```
\section{Validity}

\begin{contract}
\Clause{title={Effective Date}}

These articles will enter into force on 11.\,11.\,2011 at
11:11~am.

'S If any provision of these by-laws is in conflict with
any other, the by-laws will be repealed on
```

```
    11.\,11.\,2011 at 11:11~am and 11~seconds'. The club is
    considered to be dissolved in this case.

    \end{contract}
```

There follows another article no special features.

```
    \end{document}
```

Then the LaTeX document ends. You can see first three pages in figure 11.1.

11.9. State of Development

Since KOMA-Script 3.24, the scrjura package has shared the version number of the classes and other important packages of KOMA-Script. Nevertheless, you should note that so far, the interaction of the `contract` environment with the many different settings possible with other LaTeX environments, packages, or classes has not been tested. The main reason for this is that scrjura is very specialised and far beyond the author's ordinary practice. So the author mostly relies on detailed user feedback.

By-Laws

CfCH

Club for Club Hoppers

11. 11. 2011

Contents

Preamble

The club landscape in England is diverse. But we have unfortunately been forced to conclude that it often suffers seriously when dealing with seriousness.

A. Overview

§ 1 Name, Legal Form, Headquarters

(1) The name of this club shall be the "Club for Club Hoppers"and is not registered in any club register.

(2) [1]The club is a non-economic, useless club. [2]It has no headquarters because its members heads are in their hindquarters.

(3) The fiscal year is from March 31st through April 1st.

§ 2 Purpose of the Club

(1) [1]The club is pointless but not useless. [2]Rather, it should put the serious handling of seriousness on a sound footing.

(2) For this purpose, the club members can

a) pick their noses,

b) crack nuts,

c) such their thumbs.

(3) The club is selfish and stands by it.

(4) The club has no financial means.

§ 3 Club Officers

(1) The club officers hold honorary positions.

(2) [1]If the club had resources (see § 2 paragraph 4 sentence 1), it could afford a full-time manager. [2]Without the necessary funds, this is not possible.

B. Membership

§ 6 Becoming a Member

(1) Everyone can purchase a membership from one of the associations listed in § 4.

(2) [1]To become a member, an informal application is required. [2]This application should be submitted in green ink on pink paper.

(3) Membership applications cannot be rejected.

§ 6a Amendment to the Previous Clause

[1]With the repeal of § 4, § 6 paragraph 1 sentence 1 has become impractical. [2]In its place, memberships can be inherited.

§ 7 Termination of Membership

[1]Membership ends with one's life. [2]For non-living members, membership does not end.

§ 8 General Meeting

(1) A general meeting shall take place twice per year.

(2) The interval between two general meetings shall be no more than 6 months, 1 week, and 2 days.

(3) The invitation to the next general meeting shall be sent no earlier than 6 months from the previous general meeting.

Figure 11.1.: First three pages of the example club by-laws of section 11.8

Part II.

KOMA-Script for Advanced Users and Experts

In this part, you can find information for the authors of LaTeX packages and classes. This applies not only to commands that are useful for implementing new packages and classes, but also to interfaces that allow further alteration of KOMA-Script. Moreover, this part provides information on obsolete options and instructions, as well as background information on the implementation of KOMA-Script.

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 in the scrbase Package

The scrbase package provides basic features intended for use by authors of packages and classes. Its use is not limited merely to wrapper classes which in turn load a KOMA-Script class. Authors of classes that otherwise have nothing to do with KOMA-Script can also benefit from the functionality of scrbase.

12.1. Loading the Package

Whereas users load packages with `\usepackage`, authors of packages or classes should use `\RequirePackage`. Authors of wrapper packages can also use `\RequirePackageWithOptions`. The `\RequirePackage` command has the same optional argument for package options as `\usepackage`. In contrast, `\RequirePackageWithOptions` does not have an optional argument but passes to this package all the options that were previously given when loading the wrapper package or class. See [Tea06] for more information about these commands.

The scrbase package needs the functionality of the keyval package internally. This can also be provided by the xkeyval package. If needed, scrbase itself loads keyval.

The keyval package lets you define keys and assign values to them. The options provided by scrbase also use keyval syntax: *key=value*.

12.2. Keys as Attributes of Families and Their Members

As already mentioned in section 12.1, scrbase uses the keyval package to set keys and their values. However, scrbase extends this functionality. Whereas with keyval a key belongs to only one family, scrbase also recognizes family members. A key can therefore belong to both a family and one or more family members. Additionally, you can assign a value to the key of a family member, to the key of a family, or to the keys of all family members.

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

scrbase needs to know the members of a family for various reasons. Therefore, you must first define a new family using `\DefineFamily`, which produces an empty member list. If the family has already been defined, nothing happens. Therefore an existing member list will not be overwritten.

Next, you can add a new member to the family using `\DefineFamilyMember`. If the family does not exist, this will result in an error message. If the member already exists, nothing happens. If the optional *member* is omitted, the default value "`.\@currname.\@currext`" is used. While the class or package is being loaded, `\@currname` and `\@currext` together represent the file name of the class or package.

Theoretically, it is possible, to define a member without a name using an empty optional *member* argument. But this is the same as the family itself. You should use only letters and digits for the *family* name, and the first character of *member* should be something else, preferably a point. Otherwise, it could happen that the members of one family will clash with the members of other families.

scrbase assigns the family "KOMA" to itself and adds the member ".scrbase.sty". The values "KOMA" and "KOMAarg" are reserved for KOMA-Script. If you are creating a bundle of packages, you should use the name of the bundle as *family* and the name of each package as *member* of that *family*.

Example: Suppose you are writing a bundle called "master butcher". Within that bundle, you have the packages salami.sty, mettwurst.sty, and kielbasa.sty. Therefore, you decide to use family name "butcher" and you add the lines

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

to each of the package files. When loading the three packages, this will add the members ".salami.sty", ".mettwurst.sty", and ".kielbasa.sty" to the family "butcher". After loading all three packages, all three members will be defined.

```
\DefineFamilyKey[member]{family}{key}[default]{action}
\FamilyKeyState
\FamilyKeyStateUnknown
\FamilyKeyStateProcessed
\FamilyKeyStateUnknownValue
\FamilyKeyStateNeedValue
```

The \DefineFamilyKey command defines a *key*. If you specify a *member*, the *key* becomes an attribute of that member in the given *family*. If you do not specify a *member*, the member ".\@currname.\@currext" is assumed. If you later assign a value to the *key*, the *action* will be executed and the value made an argument of *action*. Within *action*, "#1" stands for that value. If you omit the value, the *default* is used instead. If you do not specify a *default*, the *key* then always requires you to pass an explicit value.

Ultimately,

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

results in a call to

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

where \define@key is provided by the keyval package (see [Car99a]). However, there are some additional precautions taken with the call to \define@key, and the *action* will be extended to include these precautions.

v3.12 Success or failure in executing the *action* should be reported back to scrbase through \FamilyKeyState so that the package itself can take further action as needed. This could be, for example, an error message or merely warning of an unknown option. You should not report errors directly!

The default state of *action* before execution is \FamilyKeyStateUnknown. This indicates that it is not known whether or not the key could be processed successfully. If this state remains unchanged after the *action* has been executed, scrbase writes a message to the log file and assumes the that state is \FamilyKeyStateProcessed.

The \FamilyKeyStateProcessed state indicates that the key and the value assigned to it have been successfully processed and everything is OK. You can switch to this state by calling \FamilyKeyStateProcessed itself.

The \FamilyKeyStateUnknownValue state indicates that the key has been processed but that the value passed to it was either unknown or not allowed. For example, typearea reports this condition if you try to set the twoside option to unknown. You can set this state by simply calling \FamilyKeyStateUnknownValue.

The \FamilyKeyStateNeedValue state indicates that the key could not be processed because it expects a value, but it was called without such a value. This state is set automatically if you use a key that does not have a *default* value without assigning a value. Theoretically, you could set this state explicitly with \FamilyKeyStateNeedValue, but you should not need to set it yourself.

In addition, you can define additional error conditions by redefining \FamilyKeyState with a short text message. Generally, however, the four predefined states should be sufficient.

Example: Suppose each of the three packages from the previous example should have a key named coldcuts. When used, a switch should be set in each of the packages. For the salami package, for example, this could look like this:

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

When called, the value will therefore be either true or false. This example does not test for illegal values. Instead, it is always reported that the key was processed completely and correctly. If the key is used later, one of the permitted values, or no value, must be used. In the second case, the default *true* value will be used.

The definitions for the other packages are nearly identical. Only the string "`salami`" has to be replaced.

`\RelaxFamilyKey[`*member*`]{`*family*`}{`*key*`}`

v3.15

If a *key* has been previously defined as a *member* of a *family*, that definition will be cancelled. Afterwards the *key* will no longer be defined for this *member* of the *family*. You can use `\RelaxFamilyKey` for a *key* that is not defined for this *member* of the *family*.

If you do not specify a *member*, then the member "`.\@currname.\@currext`" is assumed, just as with `\DefineFamilyKey`. However, `\RelaxFamilyKey` is only rarely used while loading a package rather than at runtime. Therefore the *member* should usually be specified explicitly as well.

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

In essence, extending keys from families to both families and family members means that either keys or key-value pairs can be used as normal class or package options. The `\FamilyProcessOptions` command is an extension of `\ProcessOption*` from the LaTeX kernel (see [Tea06], which processes not only options declared with `\DeclareOption`, but also all keys of the given member. If you omit the optional argument *member*, the member "`.\@currname.\@currext`" is used.

One feature worth noting is that keys which are attached not to a family member but to a family have an empty family member. Such keys are set before the keys of the members.

Example: If, in the packages from the previous example, you add the line

```
\FamilyProcessOptions{butcher}
```

after you define the key, you can specify the `coldcuts` option when loading the package. If you specify the option globally in `\documentclass`, the option will be passed automatically to all three packages if they are loaded later.

Note that packages always process global options before local options. While unknown global options result in an entry in the `log` file and the option being otherwise ignored, unknown local options result in an error message.

You can think of `\FamilyProcessOptions` as an extension of either `\ProcessOption*` or the *key=value* mechanism of keyval. Ultimately, with the help of `\FamilyProcessOptions`, *key=value* pairs become options.

As with `\ProcessOptions`, `\FamilyProcessOptions` must not be used while executing options code. In particular, you cannot load packages while processing options.

\BeforeFamilyProcessOptions[*member*]{*family*}{*code*}

v3.18

Authors of wrapper classes in particular sometimes need a hook to execute *code* before \FamilyProcessOptions. The scrbase package provides such a hook, and you can add *code* to it with \BeforeFamilyProcessOptions. The *member* and *family* parameters are same as those of \FamilyProcessOptions. However, you can also add *code* to the hook for families or members that have not yet been defined.

Note that the hook of a family member is automatically deleted after it is executed. But if you use an empty *member*, this hook will be executed for every member of the *family* and will not be deleted.

Example: You are writing a smokedsausage package that loads mettwurst. But you do not want to be able to set the coldcut option with this package. So you use \BeforeFamilyProcessOptions to deactivate that option before you load the package:

```
\RequirePackage{scrbase}
\BeforeFamilyProcessOptions[.mettwurst.sty]{butcher}{%
  \RelaxFamilyKey[.mettwurst.sty]{butcher}{coldcut}%
}
\RequirePackageWithOptions{mettwurst}
```

If a user tries to load your package with the coldcut option, the mettwurst package will throw an undefined option error. If coldcut is used as a global option, the mettwurst package will ignore it. But default settings inside mettwurst, for example using \FamilyExecuteOptions before \FamilyProcessOptions are not affected. Of course, you can also insert your own default for smokedsausage via \BeforeFamilyProcessOptions in mettwurst.

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

This command is an extension of \ExecuteOptions from the LaTeX kernel (see [Tea06]). The command processes not only options defined with \DeclareOption but also all keys of the given *family*. If you omit the optional argument \member, "\.\@currname\.\@currext" is used.

One feature worth noting is that keys which are attached not to a family member but to a family have an empty family member. Such keys are set before the keys of the members.

Example: Suppose the coldcuts option should be set by default in the previous examples. In this case only line

```
\FamilyExecuteOptions{butcher}{coldcuts}
```

has to be added.

v3.20 If you call \FamilyExecuteOptions with an unknown option inside the *options list*, you will get an error. An exception to this rule occurs when the *member* has an option called @else@. In this case, the @else@ option will be used instead of the unknown one. The value passed to the @else@ option is the unknown option with the value specified in the call. KOMA-Script itself uses this feature, for example inside the definition of sectioning commands, to evaluate the style option before all other attributes.

You can also use this command inside the code executed when processing the option.

\FamilyOptions{*family*}{*options list*}

Unlike normal options defined with \DeclareOption, the *key*s can also be set after loading a class or package. To do this, you call \FamilyOptions. The *options list* has the form

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

after which the value assignment can be omitted for *key*s that have a defined default.

The statement sets the keys of all members of the specified family. If a *key* also exists as an attribute of the family itself, then the family key is set first, followed by the member keys in the order in which they were defined. If a given *key* does not exist, either for the family or for any member of the family, then \FamilyOptions will throw an error. An error also occurs if a *key* exists for some members but each of those members returns an error through \FamilyKeyState.

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

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

If no sausage package has yet been loaded, the undefined option coldcuts leads to an error message. You can avoid this by defining a corresponding key for the package before the last line of the code above:

```
\DefineFamilyKey{butcher}%
                {coldcuts}[true]{}%
```

However, sausage packages loaded after sausagesalad still do not produce cold cuts. You can correct this by replacing the previous command with:

```
\AtBeginDocument{%
  \DefineFamiyKey[.sausagesalad.sty]%
                 {butcher}%
```

```
                    {coldcuts}[true]{}%
}
\DefineFamilyKey{butcher}%
                  {coldcuts}[true]{%
  \AtBeginDocument{\FamilyOptions{butcher}%
                                  {coldcuts=#1}}%
}%
```

Thus, the option is defined during `\begin{document}` so that it no longer functions for the sausagesalad package. Because `\@currname` and `\@currext` no longer contain the file name of the package, you must use the optional argument of `\DefineFamilyKey`.

But until this redefinition is performed, it uses a definition that executes the option again for the family and all its members during `\begin{document}`, thus setting it for other sausage packages. The delay in executing `\FamilyOptions` is crucial here. For one thing, it includes only the packages loaded afterwards. For another, it ensures that its own `coldcuts` option has already been redefined. This avoids endless recursion.

v3.27

Like `\FamilyExecuteOptions`, `\FamilyOptions` offers special handling for an option named `@else@`. If a family member has such an option, it will be executed whenever an option in the *option list* is unrecognized by the member. If the family itself defines an option `@else@`, it will only be called if neither the family nor any of its members were able to fully process the given option by executing `\FamilyKeyStateProcessed`.

`\FamilyOption{`*family*`}{`*option*`}{`*value list*`}`

In addition to options that have mutually exclusive values, there may be options that can take multiple values at the same time. To use `\FamilyOptions` for that type of option, it would be necessary to invoke the same option several times with different value assignments. Instead, you can easily assign a whole list of values to a single *option* using `\FamilyOption`. The *value list* is a comma separated list of values, also known as *csv*: *value*`,`*value*... Note in this context that you can use a comma in a value by putting the value inside braces. This command's other functionality is the same previously described for `\FamilyOptions`.

Example: The sausagesalad package should have one more option to add additional ingredients. Each of the ingredients sets a switch, as was done previously for the cold cuts.

```
\newif\if@saladwith@onions
\newif\if@saladwith@gherkins
\newif\if@saladwith@chillies
\DefineFamilyKey{butcher}{ingredient}{%
  \csname @saladwith@#1true\endcsname
}
```

Here the three ingredients “onions”, “gherkins”, and “chillies” have been defined. There is no error message for unknown ingredients.

For a salad with onions and gherkins, you can use

```
\FamilyOptions{butcher}{%
  ingredient=onions,ingredient=gherkins}
```

or simply

```
\FamilyOption{butcher}
             {ingredient}{onions,gherkins}
```

v3.27 An `@else@` option is processed in the same manner as with \FamilyOptions.

`\AtEndOfFamilyOptions{`*action*`}`
`\AtEndOfFamilyOptions*{`*action*`}`

v3.12 Sometimes it is useful to delay the execution of an *action* that is part of a value assignment to a key until all assignments inside one \FamilyProcessOptions, \FamilyExecuteOptions, \FamilyOptions, or \FamilyOption is finished. You can do this v3.23 using \AtEndOfFamilyOptions or its starred variant inside an option definition. However, reporting failure states of *action* is not possible with this command, nor should it be used outside an option definition.

The two variants differ in case there are nested option definitions, when executing an option requires further option calls. In this case all actions specified by \AtEndOfFamilyOptions will be executed when the innermost option call returns. In contrast, the actions of \AtEndOfFamilyOptions* are not executed until the outermost option call returns. However, the order of the actions of both commands is emphatically undefined! It is not guaranteed that the action requested first will be executed first, nor the converse.

`\FamilyBoolKey[`*member*`]{`*family*`}{`*key*`}{`*switch name*`}`
`\FamilySetBool{`*family*`}{`*key*`}{`*switch name*`}{`*value*`}`

Boolean switches have been used several times in the previous examples. In the example with `coldcuts` option, the user had to assign either `true` or `false` as a value. There was no error message if the user provided the wrong value. Because boolean switches are a common use case, you can easily define them with scrbase using \FamilyBoolKey. The *member*, *family*, and *key* arguments are the same as those used by \DefineFamilyKey (see page 332). The *switch name* is the name of the switch without the prefix `\if`. If a switch with this name does not exist already, \FamilyBoolKey will define it and initialize it to *false*. Internally, \FamilyBooKey uses \FamilySetBool as the *action* of \DefineFamilyKey. The *default* for such an option is always `true`.

\FamilySetBool, on the other hand, accepts `on` and `yes`, in addition to *true*, to turn the switch on, and *off* and *no*, in addition to *false*, to turn it off. Unknown values will result

in a call to \FamilyUnknownKeyValue with the arguments *family*, *key*, and *value*, which sets \FamilyKeyState accordingly. As a result, an error message about an unknown value assignment is printed if necessary (see also page 343 and page 332).

Example: The key coldcuts should be defined more robustly in the sausage packages. Additionally, all sausage packages should use the same key, so that either all sausage packages will produce cold cuts or none will.

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

A test of whether to produce cold cuts would look like this:

```
\if@coldcuts
   ...
\else
   ...
\fi
```

This would be the same in all three sausage packages, thereby defining the attribute "coldcuts" as a family option:

```
\@ifundefined{if@coldcuts}{%
  \expandafter\newif\csname if@coldcuts\endcsname
}{}%
\DefineFamilyKey[]{butcher}{coldcuts}[true]{%
  \FamilySetBool{butcher}{coldcuts}%
                         {@coldcuts}%
                         {#1}%
}
```

or shorter:

```
\FamilyBoolKey[]{butcher}{coldcuts}%
                         {@coldcuts}
```

taking advantage of the treatment of empty family members as opposed to omitting the optional argument explained on page 332, which applies not only to \DefineFamilyKey but also for \FamilyBoolKey.

Since \FamilyKeyState is already set by \FamilySetBool, you can check its status with the help of \DefineFamilyKey. For example, in the first case you could add an equivalence test to \FamilySetBool to perform additional actions depending on whether \FamilySetBool succeeded or not:

```
\ifx\FamilyKeyState\FamilyKeyStateProcessed
    ...
\else
    ...
\fi
```

Note that it is essential to perform the test with \ifx at this point. Fully expanding tests like \ifstr should be avoided here. Depending on the current status and the comparison status, they can lead to different error messages as well as incorrect results.

\FamilyInverseBoolKey[*member*]{*family*}{*key*}{*switch name*}
\FamilySetInverseBool{*family*}{*key*}{*switch name*}{*value*}

v3.27

These two commands differ from \FamilyBoolKey and \FamilySetBool, respectively, only in that the logic is inverted. This means that the values true, yes, and on set the boolean switch given by *switch name* to \iffalse and therefore deactivate it. The values false, no, and off set the boolean switch to \iftrue and therefore activate it.

\FamilyNumericalKey[*member*]{*family*}{*key*}[*default*]{*command*}{*value list*}
\FamilySetNumerical{*family*}{*key*}{*command*}{*value list*}{*value*}

While switches can accept only two values, there are also keys that recognize several values. For example an alignment can be not just left or not-left, but left, centred, or right. Internally, such differentiation is often made using \ifcase. This TEX command expects a numerical value. Therefore in scrbase the command to define a macro by a ***key*** has been named \FamilyNumericalKey. The ***value list*** has the form: {*value*}{*definition*},{*value*}{*definition*}, ... The ***value list*** defines not just the values permitted for the ***key***. For each ***value***, the ***definition*** of the macro *command* also is given. Usually, the ***definition*** is just a numerical value. Although other content is possible, there is currently a restriction that the ***definition*** must be fully expandable, and it will be expanded during the assignment.

Example: The sausage for the sausage salad can be cut differently. For example, the cold cuts could simply remain uncut or be cut into thick or thin slices. This information should be stored in the command \cuthow.

```
\FamilyNumericalKey{butcher}%
                   {saladcut}{cuthow}{%
                     {none}{none},{no}{none},{not}{none}%
                     {thick}{thick},%
                     {thin}{thin}%
                   }
```

Not cutting anything can be selected by

```
\FamilyOptions{butcher}{saladcut=none}
```

or

```
\FamilyOptions{butcher}{saladcut=no}
```

or

```
\FamilyOptions{butcher}{saladcut=not}
```

In all three cases \cuthow would be defined with the content none. It can be useful to provide several values for the same result, as shown in this example.

Now it's very likely that the manner of cutting will not be printed but should be evaluated later. In this case, a textual definition would be rather impractical. If the key is defined like this:

```
\FamilyNumericalKey{butcher}%
                   {saladcut}{cuthow}{%
                     {none}{0},{no}{0},{not}{0}%
                     {thick}{1},%
                     {thin}{2}%
                   }
```

then you can use a condition like the following:

```
\ifcase\cuthow
  % uncut
\or
  % thickly cut
\else
  % thinly cut
\fi
```

Internally, \FamilyNumericalKey uses \FamilySetNumerical as the *action* of \DefineFamilyKey. If an unknown value is assigned to such a key, \FamilySetNumerical will call \FamilyUnknownKeyValue with the *family*, *key* and *value* arguments. This leads to an error signalled with the \FamilyKeyStateUnknownValue status in \FamilyKeyState. Similarly, when calling \FamilySetNumerical the success is signalled via \FamilyKeyStateProcessed in \FamilyKeyState.

\FamilyCounterKey[*member*]{*family*}{*key*}[*default*]{*LaTeX counter*}
\FamilySetCounter{*family*}{*key*}{*LaTeX counter*}{*value*}

v3.12

While \FamilyNumericalKey defines a macro in which a numeric value corresponds to a symbolic value, there are, of course, circumstances when a *key* directly represents a *LaTeX counter* to which a numeric value should be assigned immediately. For this case, you can use \FamilyCounterKey, which calls \FamilySetCounter internally. There are some basic tests of the *value* argument to determine if the *value* argument appears to be suitable for assignment to a counter. The assignment will only be made if these tests succeed. However, not all errors can be detected here, so an incorrect assignment can also lead to an error message from TeX itself. Errors that are detected, however, are signalled by \FamilyKeyStateUnknownValue.

v3.15 If the value is omitted, the *default* is used instead. If there is no *default*, the *key* can be used only with an explicit value.

```
\FamilyCounterMacroKey[member]{family}{key}[default]{macro}
\FamilySetCounterMacro{family}{key}{macro}{value}
```

v3.12 These two commands differ from the previously described \FamilyCounterKey and \FamilySetCounter only by the fact that they do not assign a *value* to a LaTeX counter, but define a \macro with the *value*. This *value* is locally assigned to a counter and then its expanded value is used. Therefore, its value at the time the option is called applies.

```
\FamilyLengthKey[member]{family}{key}[default]{length}
\FamilySetLength{family}{key}{length}{value}
\FamilyLengthMacroKey[member]{family}{key}[default]{macro}
\FamilySetLengthMacro{family}{key}{macro}{value}
\FamilyUseLengthMacroKey[member]{family}{key}[default]{macro}
\FamilySetUseLengthMacro{family}{key}{macro}{value}
```

v3.12 With \FamilyLengthKey, you can define a *key* that represents a *length*. It does not matter whether the *length* is a LaTeX length, a TeX skip, or a TeX dimension. Internally the *length* will be set to the *value* using \FamilySetLength. There are some basic tests to decide whether this *value* is valid to assign to *length*. The assignment will only take place if these tests succeed. However, not all errors can be detected, so an inaccurate *value* can still result in a TeX error. Errors that are detected, however, are signalled by \FamilyKeyStateUnknownValue.

v3.15 If the value is omitted, the *default* is used instead. If there is no *default*, the *key* can be used only with an explicit value.

When you use \FamilyLengthMacroKey and \FamilySetLengthMacroKey, or \FamilySetLengthMacro and \FamilySetUseLengthMacro, the *value* is stored not in a *length* but in a *macro*. \FamilyLengthMacroKey and \FamilySetLengthMacroKey define the *macro* to be the *value* as evaluated at the point of definition, similar to \setlength. In

v3.20 contrast, \FamilyUseLengthMacroKey and \FamilySetUseLengthMacroKey store the *value* directly, and so the *value* is reevaluated each time the *macro* is used.

```
\FamilyStringKey[member]{family}{key}[default]{command}
\FamilyCSKey[member]{family}{key}[default]{command name}
```

v3.08 This defines a *key* that accepts any value. The value will be stored in the specified \command. If there is no optional argument for the *default*, \FamilyStringKey is the same as:

```
\DefineFamilyKey[member]{family}{key}
                {\defcommand{#1}}
```

If you use the optional argument *default*, \FamilyStringKey corresponds to:

```
\DefineFamilyKey[member]{family}{key}
                [default]
                {\defcommand{#1}\FamilyKeyStateProcessed}
```

Example: By default 250 g of sausage salad should be produced. However, the amount should be configurable by an option. To do so, the quantity to be created is stored in the macro `\saladweight`. The option to change the weight should also be called `saladweight`:

```
\newcommand*{\saladweight}{250g}
\FamilyStringKey{butcher}%
                {saladweight}[250g]{\saladweight}
```

To switch back to the default weight after changing it, you can simply call the option without the weight:

```
\FamilyOptions{butcher}{saladweight}
```

This is possible because the default quantity was also set as the default value in the definition.

In this case, there are no unknown values because all values are simply used for a macro definition. Note, however, that paragraph breaks are not allowed when assigning a value to the key.

v3.25 In contrast to `\FamilyStringKey`, the `\FamilyCSKey` command expects for the final argument not a macro but only the name of a command, for example `{saladweight}` instead of `{\saladweight}`.

`\FamilyUnknownKeyValue{family}{key}{value}{value list}`

The command `\FamilyUnknownKeyValue` throws an error message due to an unknown or illegal values by means of `\FamilyKeyState`. The *value list* is a comma separated list of v3.12 permissible values in the form '*value*', '*value*' ... However, the *value list* is currently not evaluated by scrbase.

Example: The user should now also be able to select whether the cold cuts should be cut thick or thin. Thick should be the default setting, which should be used even if the user does not specify how to cut the coldcuts.

```
\@ifundefined{if@thincut}{%
  \expandafter
  \newif\csname if@thincut\endcsname}{}%
\@ifundefined{if@coldcuts}{%
  \expandafter
  \newif\csname if@coldcuts\endcsname}{}
\DefineFamilyKey{butcher}%
```

```
                {coldcuts}[true]{%
  \FamilySetBool{butcher}{coldcuts}%
                                  {@coldcuts}%
                                  {#1}%
  \ifx\FamilyKeyState\FamilyKeyStateProcessed
    \@thincutfalse
  \else
    \Ifstr{#1}{thin}{%
      \@coltcutstrue
      \@finecuttrue
      \FamilyKeyStateProcessed
    }{%
      \FamilyUnknownKeyValue{butcher}%
                            {coldcuts}%
                            {#1}{%
                              'true', 'on', 'yes',
                              'false', 'off', 'no',
                              'thin'%
                            }%
    }%
  \fi
}%
```

First we try to set the boolean coldcuts switch using \FamilySetBool. If this succeeds, i. e. if \FamilyKeyState corresponds to \FamilyKeyStateProcessed, thincut will be deactivated. Otherwise, we check if the value is equal to `thin` rather than one of the valid values for a boolean switch. In this case, both coldcuts and thincut are activated and the state will be switched to \FamilyKeyStateProcessed. If the previous test fails, the error state signalled by \FamilySetBool is reset to \FamilyKeyStateUnknownValue. The list of the permissible values will be added to it. However, since this list is no longer used, it would have been easy to skip the call to \FamilyUnknownKeyValue in the example and thus assume the error status of \FamilySetBool.

The \ifstr command used in the test is explained on page 346 in section 12.3.

12.3. Conditional Execution

The scrbase package provides several commands for conditional execution. Mostly it does not rely on the TEX syntax of conditionals such as

```
\iftrue
  ...
\else
```

```
  ...
\fi
```

but uses the LaTeX syntax with arguments similar to those used by LaTeX commands like `\IfFileExists`, `\@ifundefined`, `\@ifpackageloaded`, and many others.

From KOMA-Script 3.28, following a recommendation from LaTeX team members, scrbase does not longer use `\if...` but `\If...` for commands that expect an argument instead of using the TeX syntax. Some commands from prior versions of KOMA-Script have been removed. You may use package iftex (see [The19]) to replace the functionality. Others have been renamed.

`\Ifundefinedorrelax{`*name*`}{`*then code*`}{`*else code*`}`

v3.28 This command works like `\@ifundefined` from the LaTeX kernel (see [BCJ+05]). So the *then code* will be executed if *name* is the name of a command that is currently either not defined or `\relax`. Otherwise, the *else code* will be executed. Unlike `\@ifundefined`, no hash memory is allocated nor is `\`*Name* set to `\relax` if `\`*name* was previously undefined.

`\Ifnotundefined{`*name*`}{`*then code*`}{`*else code*`}`

v3.28 If the command with the given name has already been defined, the *then code* will be executed. Otherwise, the *else code* will be executed. Since ε-TeX already has a primitive `\ifdefined`, this somewhat unwieldy name, unfortunately, had to be chosen. There is no corresponding internal command.

`\Ifpdfoutput{`*then code*`}{`*else code*`}`

v3.28 If a PDF file is generated, the *then code* will be executed. Otherwise, the *else code* will be executed. It does not matter whether PDF file is created using luaTeX, pdfTeX, or VTeX, or XeTeX.

`\Ifpsoutput{`*then code*`}{`*else code*`}`

v3.28 If a PostScript file is generated, the *then code* will be executed. Otherwise, the *else code* will be executed. VTeX can generate PostScript directly, which is recognized here. However, if VTeX is not used but the switch `\if@dvips` has been defined, the decision depends on that switch. KOMA-Script provides `\if@dvips` in typearea.

\Ifdvioutput{*then code*}{*else code*}

v3.28 If a DVI-file is generated, the *then code* will be executed. Otherwise, the *else code* will be executed. A DVI file is always assumed to be generated if no direct output of a PDF file or a PostScript file can be detected.

\if@atdocument *then code* \else *else code* \fi

This conditional command intentionally exists only as an internal command. In the document preamble, \if@atdocument corresponds to \iffalse. After \begin{document}, it corresponds to \iftrue. Authors of classes and packages may find this command useful if a command should behave differently depending on whether it is used in the preamble or inside document body. Note that this command is a condition using TEX syntax and not LATEX syntax!

\Ifstr{*string 1*}{*string 2*}{*then code*}{*else code*}

v3.28 Both *string* arguments are expanded and then compared. If the expansions are the same, the *then code* will be executed. Otherwise the *else code* will be executed. There is no corresponding internal command.

\Ifstrstart{*string 1*}{*string 2*}{*then code*}{*else code*}

v3.28 Both *string* arguments are expanded and then compared. If *string 1*, apart from white space, begins with *string 2*, the *then code* will be executed. Otherwise, the *else code* will be executed. There is no corresponding internal command.

\Ifisdimen{*expression*}{*then code*}{*else code*}

v3.28 If *expression* expands to a \dimen, i. e. a TEX length register, the *then code* will be executed. Otherwise the *else code* will be executed. The command is not completely expandable, and there is no corresponding internal command.

\Ifisdimension{*expression*}{*then code*}{*else code*}

v3.28 If *expression* expands to something syntactically equivalent to a length, the *then code* will be executed. Otherwise the *else code* will be executed. Note that unknown units will cause an error message. The command is not completely expandable, and there is no corresponding internal command.

\Ifdimen{*string*}{*then code*}{*else code*}

v3.28 The *then code* will be executed if the first-order expansion of *string* consists of digits and a valid length unit. Otherwise, the *else code* will be used. There is no corresponding internal command.

\Ifisdimexpr{*expression*}{*then code*}{*else code*}

v3.28 If *expression* expands to an ε-TEX \dimexpr, the *then code* will be executed. Otherwise, the *else code* will be executed. Note that illegal expressions will result in error messages. The command is not completely expandable, and there is no corresponding internal command.

\Ifisskip{*expression*}{*then code*}{*else code*}

v3.28 If *expression* expands to a \skip, i. e. a TEX distance, the *then code* will be executed. Otherwise, the *else code* will be executed. The command is not completely expandable, and there is no corresponding internal command.

\Ifisglue{*expression*}{*then code*}{*else code*}

v3.28 If *expression* expands to something syntactically equivalent of the value of a skip, the *then code* will be executed. Otherwise, the *else code* will be executed. Note that invalid units will result in an error message. The command is not completely expandable, and there is no corresponding internal command.

\Ifisglueexpr{*expression*}{*then code*}{*else code*}

v3.28 If *expression* expands to a \glueexpr, i. e. an ε-TEX distance expression, the *then code* will be executed. Otherwise, the *else code* will be executed. Note, that illegal expressions will result in error messages. The command is not completely expandable, and there is no corresponding internal command.

\Ifiscounter{*counter*}{*then code*}{*else code*}

v3.28 If *counter* is defined as a LATEX counter, the *then code* will be executed. Otherwise, the *else code* will be executed. The command is not completely expandable, and there is no corresponding internal command.

\Ifiscount{*count*}{*then code*}{*else code*}

v3.28 If *count* expands to a \count, i. e. a TEX counter, the *then code* will be executed. Otherwise, the *else code* will be executed. The command is not completely expandable, and there is no corresponding internal command. For tests of LATEX counters, see \Ifiscounter.

\Ifisinteger{*expression*}{*then code*}{*else code*}

v3.28 If *expression* expands to something syntactically equivalent to of the value of a counter, i. e. a negative or positive integer, the *then code* will be executed. Otherwise, the *else code* will be executed. The command is not completely expandable, and there is no corresponding internal command.

\Ifnumber{*string*}{*then code*}{*else code*}

v3.28 The *then code* will be executed if the first-order expansion of *string* consists only of digits. Otherwise, the *else code* will be used. There is no corresponding internal command.

\Ifisnumexpr{*expression*}{*then code*}{*else code*}

v3.28 If *expression* expands to a \numexpr, i. e. an ε-TEX number expression, the *then code* will be executed. Otherwise, the *else code* will be executed. Note that illegal expressions will result in error messages. The command is not completely expandable, and there is no corresponding internal command.

\IfActiveMkBoth{*then code*}{*else code*}

v3.27 The LATEX kernel uses \@mkboth to distinguish between automatic and manual running heads. Usually, it sets both marks with automatic running heads. With manual running heads, it does not set any mark. To determine whether automatic running heads are active, many packages compare \@mkboth with either \markboth or \@gobbletwo. But this does not account for all cases in which \@mkboth may be redefined. Therefore \IfActiveMkBoth tests if \@mkboth would actually set a mark, even when \marks appears in the definition of \@mkboth. If such an active \@mkboth is detected, the *then code* is executed. In all other cases, the *else code* is used.

Example: For example, suppose you want to set the right mark if and only if automatic running heads are used, e. g., the pagestyle **headings**. For a first attempt, you might use:

```
\ifx\@mkboth\markboth \markright{running head}\fi
```

Later you discover that some package does not use the usual

```
\let\@mkboth\markboth
```

but

```
\renewcommand{\@mkboth}{\markboth}
```

to activate automatic running heads. Because of this, your comparison always fails and never calls \markright. To solve this, you try the following change:

```
\ifx\@mkboth\@gobbletwo\else \markright{running head}\fi
```

Unfortunately, now \markright is also called for manual running heads, because someone has defined

```
\renewcommand{\@mkboth}[2]{%
  \typeout{DEBUG: ignoring running head setting}%
}
```

for this case.

Fortunately, both problems are easy to solve with scrbase:

```
\IfActiveMkBoth{\markright{Kolumnentitel}}{}
```

By the way, an even simpler solution for the problem in the example would be to use \@mkright from package scrlayer (see section 17.6, page 445).

12.4. Defining Language-Dependent Terms

Beginners often find it difficult to change language-dependent terms \listfigurename, by default usually "List of Figures." For example, if these are simply redefined with \renewcommand in the document preamble, they will not survive a later change of language. If you use babel, the redefinition in the preamble is overwritten with \begin{document}.

To define or change language-dependent terms, you normally have to redefine commands like \captionsenglish so that the new or redefined terms are defined in addition to the previous terms. This is made more difficult by the fact that some packages like german or ngerman redefine those settings when they are loaded. These definitions, unfortunately, occur in a way that undoes all the previous changes. For this reason, it makes sense to delay changes until \begin{document} by using \AtBeginDocument, that is, after all packages have been loaded. A user can also use \AtBeginDocument or redefine the language-dependent terms not in the preamble but after \begin{document}.

Adding further to the difficulty, some packages define additional language-dependent terms in \captions*language*, while others use \extras*language*. So the user must understand the commands very well in order to use the correct one in the right way.

The scrbase package therefore provides additional commands to define or modify language-dependent terms, relieving the user of many of these problems. These commands also let you simultaneously define or change the language-dependent terms of several dialects or forms of a language.

```
\defcaptionname{language list}{term}{definition}
\providecaptionname{language list}{term}{definition}
\newcaptionname{language list}{term}{definition}
\renewcaptionname{language list}{term}{definition}
\defcaptionname*{language list}{term}{definition}
\providecaptionname*{language list}{term}{definition}
\newcaptionname*{language list}{term}{definition}
\renewcaptionname*{language list}{term}{definition}
```

With these four commands and their starred variants, you can assign a *definition* for a particular language to a *term*. Several languages can be concatenated with comma in the *language list*.

The *term* is always a macro. The commands differ depending on whether a given language or a *term* within a given language is already defined at the time the command is called.

If a language is not defined, `\providecaptionname` does nothing other than write a message to the log file. This happens only once for each language. If a language is defined but does not yet contain a corresponding *term*, it will be defined using *definition*. However, the *term* will not be redefined if the language already has such a definition; instead, an appropriate message is written to the log file.

On the other hand, if a language has not yet been defined, `\newcaptionname` defines a new language command will be created. For the language `USenglish`, for example, this would be the language command `\captionsUSenglish`. This definition will also be noted in the log file. If *term* is not yet defined in this language, it will be defined using *definition*. If the *term* already exists in a language, an error message is issued.

The `\renewcaptionname` command behaves still differently. If a language is undefined, an error message is issued. If the *term* is not defined in this language, an error message is also issued. If the *term* is defined in the language, it will be redefined to *definition*.

v3.12 The `\defcaptionname` command always defines the *term*, thus overwriting any previous definition. As with `\providecaptionname`, the language specified need not be previously defined.

KOMA-Script itself uses `\providecaptionname` to define the commands in section 22.4. page 509.

Example: If you prefer “fig.” instead of “figure” in `USenglish`, you can achieve this using:

```
\renewcaptionname{USenglish}{\figurename}{fig.}
```

If you want the same change not only in `USenglish` but also in `UKenglish`, you do not need an additional:

```
\renewcaptionname{UKenglish}{\figurename}{fig.}
```

but can simply extend the *language list*:

```
\renewcaptionname{USenglish,UKenglish}{\figurename}{fig.}
```

You can extend the *language list* in the same manner with `american`, `australian`, `british`, `canadian`, and `newzealand`.

v3.12 Since KOMA-Script 3.12, you no longer need to delay the definition or redefinition until `\begin{document}` using `\AtBeginDocument` because scrbase does this itself if the commands are called in the document's preamble. Additionally, scrbase now checks if a term should be redefined in `\extras`*language* instead of `\captions`*language*. The new starred variants of the commands always use `\extras`*language*. So redefining language-dependent terms for packages like hyperref that use `\extras`*language* should work as expected.

Language-dependent terms that are commonly defined by classes and language packages are described in table 12.1.

Table 12.1.: Overview of language-dependent terms of typical 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 enclosures of a letter

`\figurename`
: prefix heading of figure captions

...

Table 12.1.: Overview of common language dependent terms (*continued*)

`\glossaryname`
: heading of the glossary

`\headtoname`
: “to” in header of letter pages

`\indexname`
: heading of the index

`\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

12.5. Identifying KOMA-Script

Although — or especially because — scrbase is generally designed as a package for authors of classes and packages, it is of course used by the KOMA-Script classes and most KOMA-Script packages. It therefore contains two commands that are present in all KOMA-Script classes and all basic KOMA-Script packages.

`\KOMAScript`

This command simply sets the word "KOMA-Script" in sans-serif font and with a slight tracking for the capitals. By the way, all KOMA-Script classes and packages define this command as required. The definition is robust using `\DeclareRobustCommand`. Since packages that do not belong to KOMA-Script can also define this command, its existence should not be interpreted as an indication that a KOMA-Script package is in use.

`\KOMAScriptVersion`

KOMA-Script defines the major version of KOMA-Script in this command. It has the form "*date version* KOMA-Script". This major version is same for all KOMA-Script classes and all KOMA-Script packages used by the classes. For this reason, it can be queried after you load scrbase, too. For example, this guide was made using KOMA-Script version "2020/04/19 v3.30 KOMA-Script".

12.6. Extensions to the LaTeX Kernel

Sometimes the LaTeX kernel itself provides commands but lacks other very similar commands that would often be useful. A few such commands are provided by scrbase for authors of packages and classes.

`\ClassInfoNoLine{`*class name*`}{`*information*`}`
`\PackageInfoNoLine{`*package name*`}{`*information*`}`

The LaTeX kernel already provides authors of classes and packages commands like `\ClassInfo` and `\PackageInfo` to write information, along with the current line number, to the log file. In addition to `\PackageWarning` and `\ClassWarning`, which throw warning messages with line numbers, it also provides `\PackageWarningNoLine` and `\ClassWarningNoLine` for warning messages without line numbers. However the obvious commands `\ClassInfoNoLine` and `\PackageInfoNoLine`, for writing information without line numbers into the log file, are missing. The scrbase package provides them.

`\l@addto@macro{`*command*`}{`*extension*`}`

The LaTeX kernel provides an internal command `\g@addto@macro` to extend the definition of macro `\command` globally with *extension*. This works in this form only for macros that have no arguments. However, sometimes you may need a command like this that works locally within the current group. The scrbase package provides such a command with `\l@addto@macro`. An alternative is to use the etoolbox or xpatch package, which offers a whole range of such commands for different purposes (see [Leh11] or [Gre12]).

12.7. Extensions to the Mathematical Features of ε-TEX

ε-TEX, which is used by LATEX and loaded by KOMA-Script, has extended capabilities for calculating simple arithmetic with TEX counters and integers using `\numexpr`. The four basic arithmetic operations and parentheses are supported. Division is rounded correctly. Sometimes additional operators would be useful.

`\XdivY{`*dividend*`}{`*divisor*`}`
`\XmodY{`*dividend*`}{`*divisor*`}`

v3.05a The `\XdivY` command returns the value of the integer quotient, with the `\XmodY` command giving the value of the remainder. This type of of division is defined by the equation

$$dividend = divisor \cdot integer\ quotient + remainder$$

where *dividend*, *divisor*, and *remainder* are integers, *remainder* is greater or equal to 0 and less than *divisor*, and *divisor* is a natural number greater than 0.

You can assign the value to a counter or use it directly within an expression using `\numexpr`. To output the value as an Arabic number, you must prefix it with `\the`.

v3.27

12.8. General Mechanism for Multi-Level Hooks

The LATEX kernel provides a few *hooks* in the processing of a document where additional code can be inserted. Class and package authors should be very familiar with `\AtBeginDocument` and `\AtEndDocument`. KOMA-Script offers similar features in some places, for example to hook code into the execution of sectioning commands. Over the years, there have been two problems:

- There are never enough hooks.
- There is some code that should only be executed once, as if it almost drops off the hook, as well as code that must be executed each time the hook is encountered, thus remaining permanently on the hook.

Usually, to define a single hook you must define a command that collects the code for each hook. This code is then stored in another internal macro, which must be inserted at the point where the collected code is executed. The more hooks you insert, the more such commands there are. To allow for both single-use code and permanent code may even require two hooks, and therefore twice the number of commands to be defined.

The example of sectioning commands shows that the first problem can sometimes be exacerbated by having only a single point for code execution. One package author may need only one way to execute the same code for all sectioning commands. Another package author would prefer to execute different code for certain sectioning commands only. Meeting both

these demands would require a general hook as well as a hook for each sectioning command. These requirements are doubled again because of the second problem.

KOMA-Script therefore offers a generalized hook mechanism in scrbase that provides multi-level hooks for both single-use and persistent code. These hooks are named *do-hooks* because of their implementation. From this also comes the names of the instructions by which they are controlled.

`\ExecuteDoHook{`*`specifier`*`}`

v3.27

Hooks are implemented with this command. The *specifier* determines the name or names of the hook. The *specifier* is always completely expanded for analysis.

The *specifier* generally consists of strings separated by forward slashes ("/"). Initially, `\ExecuteDoHook` divides the string at the first slash. The first part is the name. The remainder (without the slash) is the argument. Then the code for the hook with this name is executed. Next the first part of the remaining string is divided again from the remainder, combined with a slash after the name to form a new name, and the code with this name is executed. This process continues until the code for the hook with the name *specifier* has been executed and the argument is empty.

In the simplest case, the *specifier* consists of a single name. In this case, the persistent code is executed first, with an empty argument, for exactly one hook specified by this name.

At each stage of execution, single-use code is also executed after the hook's persistent code, and then the single-use code is globally removed from the hook.

Example: By inserting `\ExecuteDoHook{heading/begingroup/`*`name`*`}` into the execution of each sectioning command defined with `\DeclareSectionCommand`, the KOMA-Script classes ultimately have six hooks inserted at that code-point, executed in this order:

1. `heading` with the argument `begingroup/`*`name`* for persistent code,
2. `heading` with the argument `begingroup/`*`name`* for single-use code,
3. `heading/begingroup` with the argument *name* for persistent code,
4. `heading/begingroup` with the argument *name* for single-use code,
5. `heading/begingroup/`*`name`* with an empty argument for persistent code,
6. `heading/begingroup/`*`name`* with an empty argument for single-use code.

The *name* is the name of section level specified in `\DeclareSectionCommand`, `\DeclareNewSectionCommand` or `\ProvideSectionCommand`, or the sectioning command specified by it, for example `chapter` or `subparagraph`. Looking at the list above, and considering that there are various sectioning commands, it becomes clear that the `heading` and `heading/begingroup` hooks will be called multiple times for each sectioning command.

\AddtoDoHook{*name*}{*command*}
\AddtoOneTimeDoHook{*name*}{*command*}

v3.27

\AddtoDoHook appends persistent code to the hook named *name*. The code to be added is the *command*, to which the argument mentioned in the description of \ExecuteDoHook is appended as a parameter.

Example: Suppose you want to count how many times the \section command is executed. This would be simple, continuing the previous example:

```
\newcounter{sectionCounter}
\AddtoDoHook{heading/begingroup/section}
            {\stepcounter{sectionCounter}}
```

However, in reality, \stepcounter{sectionCounter}{} would be executed. Remember that an argument is always appended as a parameter. In the case of the hook named heading/begingroup/section, this argument is empty. Since an empty parameter becomes an empty group here, it is better to consume this empty parameter:

```
\newcommand*{\stepcountergobble}[2]{%
  \stepcounter{#1}%
}
\AddtoDoHook{heading/begingroup/section}
            {\stepcountergobble{sectionCounter}}
```

Here, the appended, empty parameter of \stepcountergobble is read but not used.

If instead of \section, you want to count all sectioning commands, you only need to change the hook name:

```
\AddtoDoHook{heading/begingroup}
            {\stepcountergobble{sectionCounter}}
```

By the way, the appended parameter is not empty in this case but contains the name of the sectioning level or sectioning command. If you wanted to count the sectioning commands individually, you could just use this:

```
\newcommand*{\stepCounter}[1]{%
  \stepcounter{#1Counter}%
}
\AddtoDoHook{heading/begingroup}
            {\stepCounter}
```

Of course, you must also define the counters partCounter, chapterCounter down to subparagraphCounter.

The \AddtoOneTimeDoHook command works similarly but adds the *command* to the single-use code. This code will be globally removed after the first execution.

\ForDoHook{*specification*}{*command*}

v3.27 While \ExecuteDoHook executes the commands previously stored with \AddtoDoHook or \AddtoOneTimeDoHook for the hook given by the *specifier*, this macro executes the *command* immediately. There are two parameters added to the *command*. The first is the hook's name; the second, the hook's argument.

This command is a byproduct of the implementation of \ExecuteDoHook. Normally, neither end users nor package authors should need this command.

\SplitDoHook{*specifier*}{*head macro*}{*remainder macro*}

v3.27 As can be seen from the preceding explanations, the parameter of a *command* added with \AddtoDoHook or \AddtoOneTimeDoHook can also be a multi-part *specifier*. You can use \SplitDoHook to divide a *specifier* into the front element and the remainder. The *head macro* will be set to the front element. The *remainder macro* will be set to the rest. If there is no remainder left, the *remainder macro* is set to empty. If the *specifier* was already empty, a warning is issued and both the *head macro* and the *remainder macro* are set to empty.

Example: If you want to increase a counter at the start of a group in which a heading is issued but decrease it at the end, you can do this with two hooks:

```
\AddtoDoHook{heading/begingroup}
            {\stepCounter}
\newcommand*{\restepCounter}[1]{%
  \addtocounter{#1Counter}{-1}%
}
\AddtoDoHook{heading/endgroup}
            {\restepCounter}
```

But you could also use a single hook and split its parameters:

```
\newcommand*{\changeCounter}[1]{%
  \SplitDoHook{#1}{\Group}{\Level}%
  \Ifstr{\Group}{begingroup}{%
    \stepcounter{\Level Counter}%
  }{%
    \Ifstr{\Group}{endgroup}{%
      \addtocounter{\Level Counter}{-1}%
    }{}%
  }%
}
\AddtoDoHook{heading}
            {\changeCounter}
```

As you can see, the first solution is much simpler. In addition, it would be easy to overlook including the empty group for the second false-case in the second example.

That would be fatal, however, as there make be other hooks with named `heading` but with different arguments.

Strictly speaking, this command is a byproduct of the implementation of \ForDoHook.

12.9. Obsolete Options and Commands

Prior release of scrbase provided some options and commands that has been removed or replaced meanwhile. Those are documented in this section only for completeness but shouldn't be used any longer.

`\FamilyElseValues`

Since version 3.12 the command is deprecated. Nevertheless, scrbase detects its use and issues a message requesting that you update the code accordingly.

`internalonly=`*value*

Since version 3.28 this option is deprecated. For compatibility it is processed (see \FamilyKeyStateProcessed) but ignored.

Controlling Package Dependencies with scrlfile

The introduction of LaTeX 2$_\varepsilon$ in 1994 brought many changes in the handling of LaTeX extensions. The package author today has a whole series of macros available to determine if another package or class has been loaded and whether specific options are being used. The package author can even load other packages or specify certain options in case the package is loaded later. This has led to the expectation that the order in which package are loaded would not be important. Sadly, this hope has not been fulfilled.

13.1. About Package Dependencies

More and more often, different packages either newly define or redefine the same macro. In such a case, the order in which a package is loaded becomes very important. Sometimes, users find it very difficult to understand the resulting behaviour. Sometimes it is necessary to react in a specific way when another package is loaded.

As a simple example, consider loading the longtable package with a KOMA-Script document class. The longtable package defines its own table captions. These are perfectly suited to the standard classes, but they do not match the default settings for KOMA-Script captions, nor do they react to the relevant configuration options. 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.

Previously, the only way to solve this problem was to delay the redefinition until the beginning of the document using `\AtBeginDocument`. However, if users want to change the relevant commands themselves, they should do so in the preamble of the document. But this is impossible because KOMA-Script will overwrite the users' definitions at `\begin{document}`. They would also need to perform the redefinition with `\AtBeginDocument`.

But KOMA-Script does not actually need to wait for `\begin{document}` to redefine the macros. It would be sufficient to postpone the redefinition until after the longtable package has been loaded. Unfortunately, the LaTeX kernel does not define necessary commands. The scrlfile package provides a remedy for this problem.

It is also conceivable that you would like to save the definition of a macro in a temporary macro before a package is loaded and restore it after the package has been loaded. The scrlfile package allows this, too.

The use of scrlfile is not limited to package dependencies. Dependencies can also be considered for any other file. For example, you can ensure that loading the not unimportant file `french.ldf` automatically leads to a warning.

Although the package is particularly of interest for package authors, there are also applications for normal LaTeX users. Therefore, this chapter gives examples for both groups.

13.2. Actions Before and After Loading

The scrlfile package can execute actions both before and after loading files. The commands used to do so distinguish between ordinary files, classes, and packages.

```
\BeforeFile{file}{commands}
\AfterFile{file}{commands}
```

`\BeforeFile` ensures that the *commands* are executed before the next time *file* is loaded. `\AfterFile` works in a similar fashion, and the *commands* will be executed after the *file* has been loaded. Of course, if *file* is never loaded, the *commands* will never be executed. For *file*, you should specify any extensions as part of the file name, as you would with `\input`.

To implement those features, scrlfile redefines the well-known LaTeX command `\InputIfFileExists`. If this command does not have the expected definition, scrlfile issues a warning. This occurs in case the command is changed in future LaTeX versions or has already been redefined by another package.

LaTeX uses the `\InputIfFileExists` command every time it loads a file. This occurs regardless of whether the file is loaded with `\include`, `\LoadClass`, `\documentclass`, `\usepackage`, `\RequirePackage`, or similar commands. Only

```
\input foo
```

loads the file `foo` without using `\InputIfFileExists`. You should therefore always use

```
\input{foo}
```

instead. Notice the parentheses surrounding the file name!

```
\BeforeClass{class}{commands}
\BeforePackage{package}{commands}
```

These two commands work the same way as `\BeforeFile`. The only difference is that the *class* or *package* is specified with its class or package name and not with its file name. That means you should omit the file extensions `.cls` or `.sty`.

```
\AfterClass{class}{commands}
\AfterClass*{class}{commands}
\AfterClass+{class}{commands}
\AfterClass!{class}{commands}
\AfterAtEndOfClass{class}{commands}
\AfterPackage{package}{commands}
\AfterPackage*{package}{commands}
\AfterPackage+{package}{commands}
\AfterPackage!{package}{commands}
\AfterAtEndOfPackage{package}{commands}
```

The \AfterClass and \AfterPackage commands work much like \AfterFile. The only difference is that the *class* or *package* is specified with its class or package name and not with its file name. That means you should omit the file extensions .cls or .sty.

The starred versions function somewhat differently. If the class or package has already been loaded, they execute the *commands* immediately rather than waiting until the next time the class or package is loaded.

v3.09 The plus version executes the *commands* after the class or package has been completely loaded. This behaviour differs from that of the starred version only if you use the command when the class or package has begun loading but has not yet finished. If the class or package has not finished loading, the *commands* will always be executed before the commands in \AtEndOfClass or \AtEndOfPackage.

If a class uses \AtEndOfClass or a package uses \AtEndOfPackage to execute commands after the class of package file has been loaded completely, and if you want to execute *commands* after these deferred commands have been executed, you can use the exclamation-mark versions

v3.09 \AfterClass! or \AfterPackage!.

If you want to execute the *commands* only when the class or package is loaded later and

v3.09 outside the context of that class or package, you can use \AfterAtEndOfClass for classes and \AfterAtEndOfPackage for packages.

Example: The following example for class and package authors shows how KOMA-Script itself makes use of the new commands. The class scrbook contains the following:

```
\AfterPackage{hyperref}{%
  \@ifpackagelater{hyperref}{2001/02/19}{}{%
    \ClassWarningNoLine{scrbook}{%
      You are using an old version of the hyperref
      package!\MessageBreak%
      This version has a buggy hack in many
      drivers,\MessageBreak%
      causing \string\addchap\space to behave
      strangely.\MessageBreak%
      Please update hyperref to at least version
```

```
    6.71b}}}
```

Old versions of the hyperref package redefine a macro of the scrbook class in a way that is incompatible with newer KOMA-Script versions. Newer versions of hyperref refrain from making this change if a newer version of KOMA-Script is detected. In case hyperref is loaded at a later stage, scrbook ensures that a check for an acceptable version of hyperref is performed immediately after the package is loaded. If this is not the case, a warning is issued.

Elsewhere in three of the KOMA-Script classes, you can find the following:

```
\AfterPackage{caption2}{%
  \renewcommand*{\setcapindent}{%
```

After loading caption2, and only if it has been loaded, KOMA-Script redefines its own \setcapindent command. The exact code of the redefinition is irrelevant. The important thing to note is that caption2 takes control of the \caption macro and that therefore the normal definition of the \setcapindent command would have no effect. The redefinition thus improves interoperability with caption2.

There are also, however, instances where these commands are useful to normal LaTeX users. For example, suppose you create a document from which you want to generate both a PostScript file, using LaTeX and dvips, and a PDF file, using pdfLaTeX. The document should also contain hyperlinks. In the table of contents, you have entries that span several lines. This is not a problem for the pdfLaTeX method, since here hyperlinks can be broken across multiple lines. However, this is not possible with the hyperref driver for dvips or hyperTeX. In this case, you would like hyperref to use the `linktocpage` option. The decision as to which driver is loaded is made automatically by `hyperref`.

Everything else 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 of a fairly long figure caption, but
    one that does not use the optional caption argument that would
    allow you to write a short caption in the list of figures.}
\end{figure}
\end{document}
```

If either of the hyperref drivers `hypertex` or `dvips` is used, the useful hyperref option `linktocpage` will be set. However, if you create a PDF file with pdfLaTeX, the option will not be set because then the hyperref driver `hpdftex.def` will be used. This means that neither `hdvips.def` nor `hypertex.def` will be loaded.

Incidentally, you can also load scrlfile before `\documentclass`. In this case, however, you should use `\RequirePackage` instead of `\usepackage` (see [Tea06]).

`\BeforeClosingMainAux{`*commands*`}`
`\AfterReadingMainAux{`*commands*`}`

These commands differ in one detail from the commands explained previously. Those commands enable actions before or after loading files. That is not the case here. Package authors often want to write something to the `aux` file after the last document page has been shipped out. To do so, ignoring the resulting problems they create, they often use code such as the following:

```
\AtEndDocument{%
  \if@filesw
    \write\@auxout{%
      \protect\writethistoaux%
    }%
  \fi
}
```

However, this does not really solve the problem. If the last page of the document has already been shipped out before `\end{document}`, the code above will not result in any output to the `aux` file. If you try to fix this new problem using `\immediate` just before `\write`, you would have the opposite problem: if the last page has not yet been shipped out before `\end{document}`, `\writethistoaux` would be written to the `aux` file too early. Therefore you often see solutions like:

```
\AtEndDocument{%
  \if@filesw
    \clearpage
    \immediate\write\@auxout{%
      \protect\writethistoaux%
    }%
  \fi
}
```

However, this solution has the disadvantage that it forces the last page to be shipped out. A command such as

```
\AtEndDocument{%
  \par\vspace*{\fill}%
  Note at the end of the document.\par
```

```
}
```

will no longer cause the note to appear beneath the text of the last real page of the document but at the end of one additional page. Furthermore, `\writethistoaux` will again be written to the `aux` file one page too early.

The best solution for this problem would be if you could write directly to the `aux` file immediately after the final `\clearpage` that is part of `\end{document}` but before closing the `aux` file. This is the purpose of `\BeforeClosingMainAux`:

```
\BeforeClosingMainAux{%
  \if@filesw
    \immediate\write\@auxout{%
      \protect\writethistoaux%
    }%
  \fi
}
```

This will be successful even if the final `\clearpage` inside of `\end{document}` does not actually ship out any page or if `\clearpage` is used within an `\AtEndDocument` command.

However, there one important limitation using `\BeforeClosingMainAux`: you should not use any typesetting commands inside the *commands* of `\BeforeClosingMainAux`! If you ignore this restriction, the result is just as unpredictable as the results of the problematic suggestions above that use `\AtEndDocument`.

v3.03 The `\AfterReadingMainAux` command actually executes the *commands* after closing and reading the `aux` file inside `\end{document}`. This is only useful in a few very rare cases, for example to write statistical information to the `log` file which is valid only after reading the `aux` file, or to implement additional *rerun* requests. Typesetting commands are even more dangerous inside these *commands* than inside the argument of `\BeforeClosingMainAux`.

13.3. Replacing Files at Input

The previous sections in this chapter have explained commands to perform actions before or after loading a particular file, package, or class. You can also use scrlfile to input a completely different file than the one that was requested.

`\ReplaceInput{`*original file*`}{`*replacement file*`}`

v2.96 This command defines a replacement for the file specified in the first argument, *original file*. If LaTeX is instructed to load this file, the *replacement file* will be loaded instead. The replacement-file definition affects all files loaded using `\InputIfFileExists`, whether they are loaded by the user or internally by LaTeX. To do so, scrlfile redefines `\InputIfFileExists`.

Example: You want to input the `\jobname.xua` file instead of the `\jobname.aux` file. To do this, you use

```
\ReplaceInput{\jobname.aux}{\jobname.xua}
```

If additionally you replace `\jobname.xua` by `\jobname.uxa` using

```
\ReplaceInput{\jobname.xua}{\jobname.uxa}
```

then `\jobname.aux` will also be replaced by `\jobname.uxa`. In this way, you can process the whole replacement chain.

However, a replacement that results in a loop such as

```
\ReplaceInput{\jobname.aux}{\jobname.xua}
\ReplaceInput{\jobname.xua}{\jobname.aux}
```

will cause a *stack size error*. So it is not possible to replace a file that has already been replaced once by itself again.

In theory, it would also be possible to use this command to replace one package or class with another. But LaTeX would recognize that the requested file name does not match the name of the package or class. You can find a solution for this problem below.

```
\ReplaceClass{original class}{replacement class}
\ReplacePackage{original package}{replacement package}
```

v2.96 You should never replace a class or package using the \ReplaceInput command described above. Doing so would result in a LaTeX warning about mismatched class or package names. Real errors are also possible if a class or package is loaded under an incorrect file name.

Example: You replace the scrpage2 package with its official successor, scrlayer-scrpage, by using

```
\ReplaceInput{scrpage2.sty}{scrlayer-scrpage.sty}
```

Loading scrpage2 will then lead to the following warning:

```
LaTeX warning: You have requested 'scrpage2',
               but the package provides 'scrlayer-scrpage'.
```

Users may be greatly confused by such a warning because they requested not scrlayer-scrpage but scrpage2, which was replaced by scrlayer-scrpage.

One solution to this problem is to use `\ReplaceClass` or `\ReplacePackage` instead of \ReplaceInput. Note that in this case, as with `\documentclass` and `\usepackage`, you should give the name of the class or package and not the complete file name.

The replacement class works for all classes loaded with `\documentclass`, `\LoadClassWithOptions`, or `\LoadClass`. The replacement package works for all packages loaded with `\usepackage`, `\RequirePackageWithOptions`, or `\RequirePackage`.

Please note that the *replacement class* or the *replacement package* will be loaded with the same options that would have been passed to the *original class* or *original package*.

If you replace a class or package with one that does not support a requested option, you will receive the usual warnings and errors. However, you can declare such missing options using \BeforeClass or \BeforePackage.

Example: Suppose you want to replace the oldfoo package with the newfoo package when the former package is loaded. You can do this with

```
\ReplacePackage{oldfoo}{newfoo}
```

Suppose the old package provides an `oldopt` option, but the new package does not. With

```
\BeforePackage{newfoo}{%
  \DeclareOption{oldopt}{%
    \PackageInfo{newfoo}%
                {option `oldopt' not supported}%
  }}%
```

you can declare this missing option for the newfoo package. This avoids an error message when the oldfoo package is loaded with the option that is unsupported by newfoo.

If, on the other hand, the newfoo package supports a `newopt` option that should be used instead of the `oldopt` option, you can also achieved this:

```
\BeforePackage{newfoo}{%
  \DeclareOption{oldopt}{%
    \ExecuteOptions{newopt}%
  }}%
```

You can even specify different default options that apply when loading the new package:

```
\BeforePackage{newfoo}{%
  \DeclareOption{oldopt}{%
    \ExecuteOptions{newopt}%
  }%
  \PassOptionsToPackage{newdefoptA,newdefoptB}%
                       {newfoo}%
}
```

or directly:

```
\BeforePackage{newfoo}{%
  \DeclareOption{oldopt}{%
    \ExecuteOptions{newopt}%
  }%
}%
\PassOptionsToPackage{newdefoptA,newdefoptB}%
                     {newfoo}%
```

Note that in the last example, the call to \PassOptionsToPackage occurs not within but after \BeforePackage

Of course, to replace classes, you must load scrlfile before the class using \RequirePackage instead of \usepackage.

\UnReplaceInput{*file name*}
\UnReplacePackage{*package*}
\UnReplaceClass{*class*}

v3.12 You can also remove a replacement. You should remove the replacement definition for an input file using \UnReplaceInput, for a package using \UnReplacePackage, and for a class using \UnReplaceClass.

13.4. Preventing File Loading

v3.08 Classes and packages written for use within companies or academic institutions often load many packages only because users need them frequently, not because they are required by the class or package itself. If a problem occurs with one of these automatically loaded packages, you somehow have to keep the problematic package from being loaded. Once again, scrlfile provides a simple solution.

\PreventPackageFromLoading[*alternate code*]{*package list*}
\PreventPackageFromLoading*[*alternate code*]{*package list*}

v3.08 Calling this command before loading a package with \usepackage, \RequirePackage, or \RequirePackageWithOptions effectively prevents the package from being loaded if it is found in the the *package list*.

Example: Suppose you work at a company where all documents are created using Latin Modern. The company class, compycls, therefore contains these lines:

```
\RequirePackage[T1]{fontenc}
\RequirePackage{lmodern}
```

But now, for the first time, you want to use XeLaTeX or LuaLaTeX. Since the recommended fontspec package uses Latin Modern as the default font anyway, and loading fontenc would not be a good idea, you want to prevent both packages from being loaded. Therefore, you load the class in your own document as follows:

```
\RequirePackage{scrlfile}
\PreventPackageFromLoading{fontenc,lmodern}
\documentclass{firmenci}
```

The example above also shows that you can load scrlfile before the class. In this case, you must use \RequirePackage because \usepackage before \documentclass is not permitted.

If you specify an empty *package list* or if it contains a package that has already been loaded, \PreventPackageFromLoading issues a warning, while \PreventPackageFromLoading* merely writes a note to the log file only.

v3.12

You can use the optional argument to execute code instead of loading the package. But you cannot load any other packages or files inside *alternate code*. To load another package, see \ReplacePackage in section 13.2 on page 365. Note also that the *alternate code* will be executed several times if you try to load the package more than once!

v3.12

\StorePreventPackageFromLoading{*command*}
\ResetPreventPackageFromLoading

v3.08

\StorePreventPackageFromLoad defines \command to be the current list of packages that should not be loaded. In contrast, \ResetPreventPackageFromLoading resets the list of packages that should not be loaded. After \ResetPreventPackageFromLoading, you can load all packages again.

v3.08

Example: Suppose you need to load a package inside another package and you do not want the user to be able to prevent that package from being loaded with \PreventPackageFromLoading. So you reset the do-not-load list before you load this package:

```
\ResetPreventPackageFromLoading
\RequirePackage{foo}
```

Unfortunately, from this point on the user's entire do-not-load list would be lost. To avoid this, you first store the list and then restore it later:

```
\newcommand*{\Users@PreventList}{}%
\StorePreventPackageFromLoading\Users@PreventList
\ResetPreventPackageFromLoading
\RequirePackage{foo}
\PreventPackageFromLoading{\Users@PreventList}
```

Note that \StorePreventPackageFromLoading defines the \Users@PreventList macro even if it has already been defined. In other words, calling \StorePreventPackageFromLoading overwrites existing \command definitions without checking. Therefore this example uses \newcommand* to get an error message if \Users@PreventList has already been defined.

Note that when you manipulate the list stored by \StorePreventPackageFromLoading, you are responsible for making sure it can be restored. For example, the list elements must be separated by comma, must not contain white space or group braces, and must be fully expandable.

Also note, that \ResetPreventPackageFromLoading does not clear the *alternate code* for a package. But this code will not be executed so long as the package is not added again to the do-not-load list.

\UnPreventPackageFromLoading{*package list*}
\UnPreventPackageFromLoading*{*package list*}

v3.12

Instead of completely resetting the list of packages that should not be loaded, you can also specify individual packages to remove from the list. The starred version of the command also deletes the *alternate code*. For example, restoring packages to the do-not-load list from a stored list will not reactivate their *alternate code* in this case.

Example: Suppose you want to prevent a foo package from being loaded, but you do not want to execute any outdated *alternate code* that may exist. Instead, only your new *alternate code* should be executed. You can do this as follows:

```
\UnPreventPackageFromLoading*{foo}
\PreventPackageFromLoading[\typeout{alternate code}]{foo}
```

For the \UnPreventPackageFromLoading command, it does not matter whether or not the package has been prevented from being loaded before.

Of course you can also use the command to indirectly delete the *alternate code* of all packages:

```
\StorePreventPackageFromLoading\TheWholePreventList
\UnPreventPackageFromLoading*{\TheWholePreventList}
\PreventPackageFromLoading{\TheWholePreventList}
```

In this case the packages that have been prevented from being loaded are still prevented from being loaded, but their *alternate code* has been deleted and will no longer be executed.

Economising and Replacing Files with scrwfile

One of the problems not solved by the introduction of ε-TEX is the fact that TEX can support only 18 open write handles. This number seems quite large at first, but many of these handles are already reserved. TEX itself uses handle 0 for the log file. LATEX needs handle 1 for `\@mainaux`, handle 2 for `\@partaux`, one handle for `\tableofcontents`, one handle for `\listoffigures`, one handle for `\listoftables`, and one handle for `\makeindex`. Every other such list generates another handle, and packages like hyperref or minitoc require write handles too.

The bottom line is that eventually you may get the following error message:

```
! No room for a new \write .
\ch@ck ...\else \errmessage {No room for a new #3}
                                                \fi
```

For some time, the simplest solution to this problem has been to use LuaLATEX instead of pdfLATEX or XƎLATEX. This eliminates the restriction, and the maximum number of files you can have open for writing is then limited only by the operating system. In reality, you usually so not need to worry about it any more.

The fact that LATEX always opens a new file for writing every table of contents, list of figures, list of tables, etc. has another disadvantage. Such lists are not only output by their respective commands, they also could not be output a second time because the associated auxiliary file[1] is empty after the respective command until the end of the document.

The scrwfile package makes a fundamental change to the LATEX kernel, which can solve both problems not only for LuaLATEX but also for pdfLATEX or XƎLATEX.

14.1. Fundamental Changes to the LATEX Kernel

LATEX classes use the LATEX kernel command `\@starttoc` to allocate a new file handle, such as for `\tableofcontents` or `\listoffigures`. This command not only loads the associated auxiliary file but also reopens it for writing. If entries to these lists are added using `\addcontentsline`, however, the system does not write directly to these auxiliary files. Instead, LATEX writes `\@writefile` commands to the `aux` file. Only while reading the `aux` file at the end of the document do those `\@writefile` commands become actual write operations in the auxiliary files. Additionally, LATEX does not close the auxiliary files explicitly. Instead it relies on TEX to close all open files at the end.

This procedure ensures that the auxiliary files are only written to within `\end{document}`, but they remain open throughout the entire LATEX run. scrwfile takes is starting point here, by redefining `\@starttoc` and `\@writefile`.

[1] The term *auxiliary file* here refers not to the main `aux` file but to the other internal files used indirectly via the `aux` file, e. g. the `toc` file, the `lof` file, or the `lot` file.

Of course changes to the LaTeX kernel always have the potential to cause incompatibilities with other packages. Those primarily affected may be those which also redefine `\@starttoc` or `\@writefile`. In some cases, it may help to change the order the packages are loaded. If you encounter such a problem, please contact the KOMA-Script author.

14.2. The Single-File Method

As soon as the package is loaded with

```
\usepackage{scrwfile}
```

scrwfile redefines `\@starttoc` so that it no longer allocates a write handle or opens a file for writing. `\@writefile` is redefined so that immediately before closing the `aux` file in `\end{document}`, it writes not to the usual auxiliary files but to a single new file with extension `wrt`. After reading the `aux` file, this `wrt` file will be processed once for each of the auxiliary files that `\@writefile` writes information to. However these auxiliary files do not all have to be open at the same time. Instead, only one is open at a time and is explicitly closed afterwards. Since LaTeX reuses an internal write file, scrwfile doesn't need its own write handle for this type of table of contents or list of floating environments.

Because of this behaviour, even if you have only one table of contents, once you load scrwfile you will have access to a write file handle for bibliographies, indexes, glossaries, and similar lists that do not use `\@starttoc`. Additionally, you can create any number of tables of contents and other lists that use `\@starttoc`.

14.3. The File Cloning Method

Since `\@writefile` has already been modified for the single-file method described in the previous system so that it no longer writes directly to the corresponding auxiliary file, a further possibility suggests itself. When copying the `\@writefile` statements into the `wrt` file, you can also copy them to other destinations.

`\TOCclone[`*list heading*`]{`*source extension*`}{`*destination extension*`}`

This cloning of file entries copies entire tables of contents or other lists. For this, you only need to specify the extension of the auxiliary file whose entries should be copied and the extension of a destination file. The entries are then copied there. Of course, you can also write additional entries to this cloned file.

You can manage the *destination extention* using tocbasic (see chapter 15). If such a file is already under the control of tocbasic, a warning will be issued. Otherwise, a new list for this extension will be created using tocbasic. You can set the heading for this list with the optional argument *list heading*.

You can then output this new content list, for example, with the command `\listof`*`destination extension`*. The content-list attributes `leveldown`, `numbered`, `onecolumn`, and `totoc` (see `\setuptoc` in section 15.2, page 383) are automatically copied to the destination list if they were already set in the source list. The `nobabel` attribute is always set for cloned content lists, because the language-selection commands in the source content list are already copied anyway.

Example: Suppose you want a short table of contents with only the chapter level in addition to the normal the table of contents:

```
\usepackage{scrwfile}
\TOCclone[Summary Contents]{toc}{stoc}
```

This creates a new table of contents with the heading "Summary Contents". The new table of contents uses an auxiliary file with the extension `stoc`. All entries to the auxiliary file with extension `toc` will also be copied to this new auxiliary file.

In order to have the new short table of contents display only the chapter entries, we use:

```
\addtocontents{stoc}{\protect\value{tocdepth}=0}
```

Although normally you cannot write to an auxiliary file before `\begin{document}`, the code above works in the preamble after loading scrwfile. Owing to the unconventional way of changing the `tocdepth` counter within the TOC file, this change only applies to this content list.

Later in the document, we then output the content list with the file extension `stoc` with:

```
\listofstoc
```

and this shows only the parts and chapters of the document.

Things become a bit more difficult if the summary contents are to be listed in the table of contents. This would seem to be possible with

```
\addtocontents{toc}{%
  \protect\addxcontentsline
    {stoc}{chapter}{\protect\contentsname}%
}
```

However, since all entries in `toc` are also copied to `stoc`, this entry would also be copied from the summary contents. So we cannot generate the entry from the content list. Because we use the tocbasic package, we can use the following:

```
\BeforeStartingTOC[toc]{%
  \addcontentslinedefault{stoc}{chapter}
                  {\protect\contentsname}%
}
```

Of course, this assumes that the toc file is under the control of the tocbasic package, which is indeed the case for all KOMA-Script classes. See section 15.2 on page 381 for more information about \BeforeStartingTOC.

Incidentally, the \addxcontentsline command used in the examples is also documented in chapter 15, page 378.

14.4. Note on the State of Development

Although this package has already been tested by many users and is often in production use, its development is still ongoing. Therefore, it is theoretically possible that there might be changes, especially to the internal functionality. It is likely that the package will be extended in the future. Some code for such extensions is already in the package. However, as there are no user commands that make use of these features, they are currently undocumented.

14.5. Known Package Incompatibilities

As mentioned in section 14.1, scrwfile redefines some commands of the LaTeX kernel. This happens not only while loading the package, but indeed at various times while the document is processed, for example just before reading the aux file. This results in incompatibility with packages that also redefine these commands at run time.

The titletoc package is an example for such an incompatibility. That package redefines \@writefile under some conditions at run time. If you use both scrwfile and titletoc, there is no warranty for the correct behaviour of either one. This is neither an error of titletoc nor of scrwfile.

Managing Content Lists with tocbasic

The main purpose of the tocbasic package is to give authors of packages and classes the ability to create their own tables or lists of content, content lists for short, similar to the list of figures and the list of tables, allowing classes and other packages some control over these lists. The tocbasic package also delegates control of the language-dependent parts of these content lists to the babel package (see [BB13]). Using tocbasic relieves package and class authors from the burden of implementing such features themselves.

As a minor side effect, the package can also be used to define new floating environments, or floating environments like non-floating environments for reference objects. For more details, after you read about the basic commands in the next four sections, see the example in section 15.5, which is compactly summarized in section 15.6.

KOMA-Script itself uses tocbasic not only for the table of contents but also for the already mentioned lists of figures and tables.

15.1. Basic Commands

The basic commands are primarily used to handle a list of all known file extensions that represent a table or list of contents. We call these auxiliary files[1] TOC files regardless of the file extension that is actually used. Entries to such files are typically written using `\addtocontents`, or `\addxcontentsline`. There are also commands to perform actions on all of these file extensions. Additionally, there are commands to set or unset features for the file associated with a given extension. Typically an file extension also has an owner. This owner may be a class or package, or an identifier of a category that the author of the class or package using tocbasic has chosen independently. For example, KOMA-Script uses the category `float` as owner for the `lof` and `lot` file extensions that are associated with the list of figures and list of tables, respectively. For the table of contents, KOMA-Script uses the file name of the class.

`\Ifattoclist{`*extension*`}{`*then code*`}{`*else code*`}`

v3.28

This command tests whether or not the *extension* already exists in the list of known file extensions. If the *extension* is already known, the *then code* will be executed. Otherwise, the *else code* will be executed.

Example: Suppose you want to know if the file name extension "`foo`" is already in use in order to report an error because it cannot be used:

```
\Ifattoclist{foo}{%
  \PackageError{bar}{%
```

[1] The term *auxiliary file* here refers not to the main `aux` file but to the other internal files used indirectly via the `aux` file, e. g. the `toc` file, the `lof` file, or the `lot` file.

```
    extension ‘foo’ already in use%
  }{%
    Each extension may be used only
    once.\MessageBreak
    The class or another package already
    uses extension ‘foo’.\MessageBreak
    This error is fatal!\MessageBreak
    You should not continue!}%
}{%
  \PackageInfo{bar}{using extension ‘foo’}%
}
```

`\addtotoclist[`*owner*`]{`*extension*`}`

This command adds the *extension* to the list of known extensions. But if the *extension* is known already, an error is reported to prevent duplicate use of the same *extension*.

If you specify the optional `[`*owner*`]` argument, the given *owner* for this file extension is also saved. If you omit the optional argument, tocbasic tries to determine the file name of the class or package currently being processed and saves it as the owner. This procedure only works if you call `\addtotoclist` while loading a class or package. It will fail if a user calls `\addtotoclist` afterwards. In this case, the owner is empty.

Note that passing an empty *owner* argument is not always the same as completely omitting the optional argument, including the square brackets. An empty argument would always result in an empty owner.

Example: Suppose you want to add the extension “`foo`” to the list of known file extensions while loading your package with the file name “`bar.sty`”:

```
\addtotoclist{foo}
```

This will add the extension “`foo`” with the owner “`bar.sty`” to the list of known extensions if this extension was not already in the list. If the class or another package has 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.
```

If you press the “`h`” key followed by return, you will get the following 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.
```

Perhaps your package also provides a command that dynamically generates a content list. In this case, you should use the optional argument of \addtotoclist to specify the owner.

```
\newcommand*{\createnewlistofsomething}[1]{%
  \addtotoclist[bar.sty]{#1}%
  % Do something more to make this content list available
}
```

Now if the user calls this command, for example with

```
\createnewlistofsomething{foo}
```

this will 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 can specify any *owner* you want, but it must be unique. For example, if you were the author of the float package, you could specify the category "float" instead of "float.sty" as the *owner*. In this case, the KOMA-Script options for the list of figures and the list of tables would also affect your content lists because KOMA-Script associates the file extensions "lof" for the list of figures and "lot" for the list of tables with the owner "float" and sets the options for this owner.

By the way, the scrhack package contains patches for several packages, such as float or listings, which provide their own content lists. If you use scrhack, among other things, the respective file extensions will be added to the list of known file extensions. Their *owner* is also "float". This is the basic building block, so to speak, allowing you to use the features of tocbasic and the KOMA-Script classes with these content lists as well.

\AtAddToTocList[*owner*]{*commands*}

This command adds the *commands* to an internal list of commands that will be processed whenever a file extension with the specified *owner* is added to the list of known extensions with \addtotoclist. The optional argument is handled in the same way as in the \addtotoclist command. If you leave the optional argument blank, the commands will always be executed, regardless of the owner, every time a new file extension is added to the list of known file extensions. Furthermore, while processing the commands, \@currext is set to the extension of the extension currently being added.

Example: tocbasic itself uses

```
\AtAddToTocList[]{%
  \expandafter\tocbasic@extend@babel
  \expandafter{\@currext}%
}
```

to add every file extension to the existing tocbasic integration with the babel package.

The two \expandafter commands in the example are needed because the argument of \tocbasic@extend@babel has to be expanded. See the description of \tocbasic@extend@babel in section 15.4, page 399 for more information.

\removefromtoclist[*owner*]{*extension*}

This command removes the *extension* from the list of known extensions. If the optional argument, [*owner*], is given, the *extension* will only be removed if it was added by this *owner*. This also applies to the empty *owner*. If, on the other hand, no [*owner*] is specified at all and the square brackets are also omitted, the owner is not tested and the *extension* is removed regardless of the owner.

\doforeachtocfile[*owner*]{*commands*}

To this point, we've introduced commands provide additional security for class and package authors, but also more overhead. With \doforeachtocfile, you can reap the first benefit for this. This command lets you execute the specified *commands* for each file extension associated with an *owner*. While processing the *commands*, \@currext is the extension of the current file. If you omit the optional [*owner*] argument, all file extensions are processed regardless of the owner. If the optional argument is empty, on the other hand, only extensions with an empty owner will be processed.

Example: If you want to output a list of all known file extensions to the terminal and to the log file, you can easily accomplish this:

```
\doforeachtocfile{\typeout{\@currext}}
```

If, on the other hand, you only want to output the extensions owned by "foo", this too is easy:

```
\doforeachtocfile[foo]{\typeout{\@currext}}
```

The KOMA-Script classes scrbook and scrreprt use this command to optionally put a vertical skip or the chapter heading in content lists using the chapteratlist feature. You can learn how to set this feature in section 15.2 on page 383.

\tocbasicautomode

This command redefines LaTeX kernel macro \@starttoc for class and package authors so that every time \@starttoc is called, the specified file extension is added to the list of known extensions, if it has not already been added. It also uses \tocbasic@starttoc instead of \@starttoc. See section 15.4, page 400 for more information about \tocbasic@starttoc and \@starttoc.

After you use \tocbasicautomode, every content list created with \@starttoc is automatically put under the control of tocbasic. Whether or not this leads to the desired result,

however, depends very much on the individual content list. The babel package extensions, at least, will work for all those content lists. Nevertheless, it is preferable for the class or package authors to use tocbasic explicitly. That way they can also take advantage of the other features of tocbasic, which are described in the following sections.

15.2. Creating a Content List

In the previous section, you learned how to maintain a list of known file extensions and how to automatically execute commands when adding new extensions to this list. You also saw a command that can be used to execute instructions for all known extensions or all extensions belonging to one owner. In this section, you will learn commands that apply to the files associated with these file extensions.

`\addtoeachtocfile[`*owner*`]{`*content*`}`

The `\addtoeachtocfile` command uses the LaTeX kernel command `\addtocontents` to write the *content* to every TOC file in the list of known file extensions that has the specified *owner*. If you omit the optional argument, the *content* is written to each file in the list of known file extensions. Incidentally, the actual file name is constructed from `\jobname` and the file extension. While writing the *content*, `\@currext` is the extension of the file currently being processed.

Example: You want to add a vertical space of one line to all TOC files.

```
\addtoeachtocfile{%
  \protect\addvspace{\protect\baselineskip}%
}
```

If, on the other hand, you want to do this only for the files that have the owner "foo", use:

```
\addtoeachtocfile[foo]{%
  \protect\addvspace{\protect\baselineskip}%
}
```

You should protect commands that should not be expanded when they are written by prefixing them with `\protect`, in the same way as you would in `\addtocontents`.

`\addxcontentsline{`*extension*`}{`*level*`}[`*section number*`]{`*text*`}`

This command is very similar to `\addcontentsline` from the LaTeX kernel. However, it has an additional optional argument for the *section number* of the entry, whereas for `\addcontentsline`, it is specified in the *text* argument. It is used to include numbered or unnumbered entries in the content list specified by the file *extension*, where *level* is

the sectioning level and *text* is the content of the corresponding entry. The page number is determined automatically.

In contrast to \addcontentsline, \addxcontentsline first tests whether the \add*level extension*entry command is defined. If so, it will be used for the entry, passing the *section number* as an optional argument and *text* as a mandatory argument. You can find an example of such a command provided by the KOMA-Script classes in \addparttocentry (see section 21.4, page 475). If the corresponding command is undefined, the internal command \tocbasic@addxcontentsline is used instead. This takes all four arguments as mandatory arguments and then uses \addcontentsline to create the desired entry. You can find more about \tocbasic@addxcontentsline in section 15.4, page 401.

One advantage of using \addxcontentsline rather than \addcontentsline is that the numberline feature is respected (see page 383). Furthermore, you can configure the form of the entries by defining the appropriate commands specific to the *level* and file *extension*.

\addxcontentslinetoeachtocfile[*owner*]{*level*}[*section number*]{*text*}
\addcontentslinetoeachtocfile[*owner*]{*level*}{*text*}

v3.12 These two commands are directly related to \addxcontentsline explained above, or to \addcontentsline from the LaTeXkernel. The difference is that these statements write *text* not just to a single file but to all the files of a given *owner* and, if the first optional argument is omitted, to all files in the list of known file extensions.

Example: Suppose you are a class author and you want to write the chapter entry not just in the table of contents but in all content-list files. Suppose further that #1 currently contains the text to be written.

```
\addxcontentslinetoeachtocfile{chapter}%
                              [\thechapter]{#1}
```

In this case, of course, the current chapter number should be expanded directly when writing to the TOC file, which is why it was not protected from expansion with \protect.

While writing the *content*, \@currext here is also the extension of the file being written to, as it is with \addtoeachtocfile.

v3.12 Whenever possible, the \addxcontentslinetoeachtocfile command is preferable to \addcontentslinetoeachtocfile because only the former offers the enhancements of \addxcontentsline. You can find more about these enhancements and benefits in the explanation of \addxcontentsline above.

```
\listoftoc[list-of title]{extension}
\listoftoc*{extension}
\listofeachtoc[owner]
\listofextensionname
```

You can use these commands to print the content lists. The stared version `\listoftoc*` needs only one argument, the *extension* of the file. The command first initializes the vertical and horizontal spacing of paragraphs, calls the hook to execute commands before reading the file, then reads the file, and finally executes the hook to execute commands after reading the file. Thus you can think of `\listoftoc*` as a direct replacement for the LaTeX kernel macro `\@starttoc`.

The version of `\listoftoc` without the star prints the complete content list and also creates an optional entry in the table of contents and the running heads. If you provide the optional `[list-of title]` argument, it is used both as the title for the list and as an optional entry in the table of contents and the running head. If the `[list-of title]` argument is empty, the title will be empty too. If, on the other hand, you completely omit the optional argument, including the square brackets, the `\listofextensionname` command will be used if it is defined. If it is undefined, a default replacement name is used and a warning is issued.

The `\listofeachtoc` command outputs all content lists associated with the given *owner*, or of all known file extensions if the optional argument is omitted. To output the correct titles, `\listofextensionname` should be defined.

You should define `\listofextensionname` appropriately for all file extensions because users may use `\listoftoc` without the optional argument, or `\listofeachtoc`, themselves.

Example: Suppose you have a new "list of algorithms" with the file extension `loa` and want to output it. The command

```
\listoftoc[List of Algorithms]{loa}
```

will do it for you. If, however, you want to output this list without a title, you could use

```
\listof*{loa}
```

In the second case, of course, there will be no entry in the table of contents. For more information about creating entries in the table of contents, see the `\setuptoc` command on page 383.

If you have defined

```
\newcommand*{\listofloaname}{%
  List of Algorithms%
}
```

already, then

```
\listoftoc{loa}
```

will suffice to print the content list with the desired heading. It may be easier for users to remember if you also define a simple list-of command:

```
\newcommand*{\listofalgorithms}{\listoftoc{loa}}
```

Because LaTeX normally opens a new file for each of these content lists, calling each of these 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 the scrwfile package (see chapter 14) can solve this problem.

Also, the scrhack package contains patches for several packages, such as float or listings, so that their content-list commands can use `\listoftoc`. As a result, many features of tocbasic and the KOMA-Script classes are also available for their content lists.

`\BeforeStartingTOC[`*extension*`]{`*commands*`}`
`\AfterStartingTOC[`*extension*`]{`*commands*`}`

Sometimes it's useful to be able to execute *commands* immediately before reading the auxiliary TOC file. With `\BeforeStartingTOC` you can do so either for a single file *extension* or for all files that are read using `\listoftoc*`, `\listoftoc`, or `\listofeachtoc`. You can also execute *commands* after reading the file if you define them with `\AfterStartingTOC`. If you omit the optional argument (or set an empty one) a general hook will be set and the commands will be applied to all content lists. The general before-hook is called before the individual one, and the general after-hook is called after the individual one. While executing the commands in these hooks, `\@currext` is the extension of the TOC file which is about to be or has just been read.

You can find an example using `\BeforeStartingTOC` in section 14.3 on page 372.

`\BeforeTOCHead[`*extension*`]{`*commands*`}`
`\AfterTOCHead[`*extension*`]{`*commands*`}`

You can also define *commands* that will be executed immediately before or after setting the title of a content list using `\listoftoc*` or `\listoftoc`. The treatment of the optional parameter and the meaning of `\@currext` is the same as described for `\BeforeStartingTOC` and `\AfterStartingTOC`.

`\MakeMarkcase{`*text*`}`

Whenever tocbasic sets a mark for a running head, The text of the mark will be an argument of `\MakeMarkcase`. You can use this command to change the case of the running head if necessary. The KOMA-Script classes use `\@firstofone` by default. This means the text of the running head will be set without changing the capitalisation. If you use a different class, `\MakeMarkcase` will be set to `\MakeUppercase`. However, tocbasic only defines this command if it is not already defined. It can therefore be predefined by another class or package and tocbasic will use that definition rather than overwriting it.

Example: For some strange reason, you want to set the running heads entirely in lower-case letters. To apply this change automatically for all running heads set by tocbasic, you define:

```
\let\MakeMarkcase\MakeLowercase
```

Let me give you some advice about `\MakeUppercase`. First of all, this command is not fully expandable. This means that it can cause problems interacting with other commands. Beyond that, typographers agree that whenever you set whole words or phrases in capital letters, additional spacing is absolutely necessary. However, adding a fixed spacing between all letters is not an adequate solution. Different pairs of letters require different spaces between them. Additionally, some letters already create gaps in the text that must be taken into account. Packages like ulem or soul can scarcely achieve this, nor can `\MakeUppercase`. The automatic letter spacing using the microtype package is in this respect only an approximate solution, because it does not take into account the concrete, font-dependent glyphs. Because typesetting all-capital text is expert work and almost always requires manual adjustment, ordinary users are recommended avoid using it, or to use it only sparingly and not in such an exposed place as the running head.

`\deftocheading{`*extension*`}{`*definition*`}`

The tocbasic package contains a default definition for typesetting the titles of content lists. You can configure this default definition through various features discussed in the `\setuptoc` comand below. If those features are not enough, you can use `\deftocheading` to define your own title for the content list with the given file *extension*. The *definition* of the title can use a single parameter, `#1`. When the command is called inside `\listoftoc` or `\listofeachtoc`, that `#1` will be replaced by the title of this content list.

Obviously, the *definition* is also responsible for the interpretation of additional features related to the title. This is especially true for the features `leveldown`, `numbered` and `totoc`, that are be explained next.

\setuptoc{*extension*}{*feature list*}
\unsettoc{*extension*}{*feature list*}

You can use these two commands to set and clear features for a TOC file *extension* or the content list associated with it. The *feature list* is a comma-separated list of features. The tocbasic package recognizes following features:

leveldown means that the content list's heading is created not with the highest sectioning level below \part — \chapter if available, \section otherwise — but with a heading of the next level below that. This feature is evaluated by the internal heading command. On the other hand, if a user-defined heading command has been created with \deftocheading, the person defining that command is responsible for evaluating the feature. The KOMA-Script classes set this feature using the listof=leveldownimportantlistof=leveldown option for all file extensions with the owner float.

nobabel prevents using the language-switching features of babel for the TOC file associated with the this file *extension*. This feature should be used only for content lists that are created in a fixed language, which means that changes of the language inside of the document will no longer affect the content list. The scrwfile package also uses this feature for cloned destinations, as because those files already inherit any language changes from the cloning source.

Note that you must set this feature before you add a file extension to the list of known extensions. Changing the feature afterwards will have no effect.

v3.27 **noindent** causes all content-list entry styles provided by KOMA-Script to ignore the indent attribute (see table 15.1, page 391) and instead to deactivate the indentation.

v3.17 **noparskipfake** prevents the insertion of a final \parskip before switched off paragraph spacing for content lists. In general, this will cause documents that use spacing between paragraphs to have less vertical space between the list heading and first entry than between normal headings and normal text. Normally, therefore, you will obtain a more uniform formatting without this feature.

v3.10 **noprotrusion** prevents character protrusion, which allows optical margin alignment, from being disabled in the content lists. By default, character protrusion is disabled when the microtype package, or another package that supports \microtypesetup, is loaded. So if you want optical margin alignment in the content lists, you must set this feature. Note, however, that character protrusion in content lists often results in incorrect results. This is a known issue with character protrusion.

numbered means that the heading for the content list should be numbered and included in the table of contents. This feature is evaluated by the internal heading command. However, if a user-defined heading command has been created with \deftocheading, the person

creating that definition is responsible evaluating the feature. The KOMA-Script classes set this feature using the `listof=numbered` option for all file extensions with the owner `float`.

v3.12 **`numberline`** means that any entries made using `\addxcontentsline` or `\addxcontentslinetoeachtocfile`, where the optional argument for the number is missing or empty, will be provided with an empty `\numberline` command. This usually results in these entries being left-aligned not with the number but with the text of the numbered entries of the same level. Using the `tocline` entry style can have additional side effects. See the style attribute `breakafternumber` and `entrynumberformat` in table 15.1 starting on page 390.

v3.20

KOMA-Script classes set this feature for the file extensions with the owner `float` if you use the `listof=numberline` option and for the file extension `toc` if you use the `toc=numberline` option. Similarly, this feature is reset if you use `listof=nonumberline` or `toc=nonumberline`.

v3.01 **`onecolumn`** means that this content list will automatically use LaTeX's internal one-column mode with `\onecolumn`. However, this applies only if this content list does not use the `leveldown` feature. The KOMA-Script classes scrbook and scrreprt activate this feature with `\AtAddToTocList` (see section 15.1, page 376) for all content lists with the owner `float` or with themselves as owner. Thus, for example, the table of contents, the list of figures, and the list of tables are automatically printed in a single column for both these classes. The multicolumn mode of the multicol package is expressly unaffected by this option.

`totoc` means that the title of content list should be included in the table of contents. This feature will be evaluated by the internal heading command. However, if an user-defined heading command has been created with `\deftocheading`, the person defining that command is responsible for evaluating this feature. The KOMA-Script classes set this feature using the `listof=totoc` option for all file extensions with the owner `float`.

The KOMA-Script classes recognize an additional feature:

`chapteratlist` ensures that an optional subdivision is added to the content list for each new chapter. By default, this subdivision is a vertical space. See `listof` in section 3.20, page 141 for more information about this option.

Example: Because KOMA-Script classes use tocbasic for the list of figures and list of tables, there is another way to prevent chapter subdivisions in these lists:

```
\unsettoc{lof}{chapteratlist}
\unsettoc{lot}{chapteratlist}
```

If you want the chapter subdivisions for your own list that you have defined with the file *extension* "loa" to use the same subdivision format used by the KOMA-Script classes, you can use

```
\setuptoc{loa}{chapteratlist}
```

And if you also want classes that use \chapter as the top-level structure to use the one-column mode automatically, you can use

```
\Ifundefinedorrelax{chapter}{}{%
  \setuptoc{loa}{onecolumn}%
}
```

Using \Ifundefinedorrelax requires the scrbase package (see section 12.3, page 345).

Even if your package will be used with another class, it does not hurt to set these features. To the contrary, if another class also evaluates these features, then your package would automatically use the features of that class.

As you can see, packages that use tocbasic already support a wide range of options for content lists that would otherwise require a great deal of effort to implement and which are therefore missing in many packages.

\Iftocfeature{*extension*}{*feature*}{*then code*}{*else code*}

v3.28 You can use this command to test if a *feature* was set for the given file *extension*. If so the *then code* will be executed, otherwise the *else code* will be. This can be useful, for example, if you define your own heading command using \deftocheading but want to support the features totoc, numbered or leveldown.

15.3. Configuring Content-List Entries

v3.20 Since version 3.20, the tocbasic package has been able not only to configure the tables or lists of contents and their auxiliary files but also to influence their entries. To do so, you can define new styles or you can use and configure one of the predefined styles. In the medium term, tocbasic will replace the experimental tocstyle package that never became an official part of the KOMA-Script bundle. The KOMA-Script classes themselves have relied completely on the tocbasic entry styles since version 3.20.

`tocdepth`

Entries to content lists are usually hierarchical. For this purpose, each entry level has a numerical value. The higher this value, the lower in the hierarchy is this level. In the standard classes, for example, parts have the level -1 and chapters have the value 0. The LaTeX counter `tocdepth` determines the deepest level that should be shown in the table of contents and other content lists.

For example, the book class sets `tocdepth` to 2, so entries of the levels `part`, `chapter`, `section`, and `subsection` are printed. Deeper levels like `subsubsection`, which has the numerical value 3, are not printed. Nevertheless the entries are part of the auxiliary file for the table of contents.

Note that most tocbasic entry styles, with the exception of `gobble` (see \DeclareTOCStyleEntry) observe `tocdepth`.

`\numberline{`*entry number*`}`
`\usetocbasicnumberline[`*code*`]`

v3.20

Although the LaTeX kernel already defines a `\numberline` command, the definition is not sufficient for tocbasic. Therefore tocbasic defines its own commands and sets `\numberline` as needed using `\usetocbasicnumberline` for each content-list entry. Redefining `\numberline`, therefore, is often ineffective and may result in warnings if you use tocbasic.

You can use the definition of tocbasic by putting `\usetocbasicnumberline` into your document's preamble. The command first checks if the current definition of `\numberline` uses certain important, internal commands of tocbasic. If this is not the case, `\usetocbasicnumberline` redefines `\numberline` and executes *code*. If you omit the optional argument, it issues a message about the redefinition with `\PackageInfo`. If you just do not want such a message, use an empty optional argument.

Note that `\usetocbasicnumberline` can change the internal switch `\@tempswa` globally!

`\DeclareTOCStyleEntry[`*option list*`]{`*style*`}{`*entry level*`}`
`\DeclareTOCStyleEntries[`*option list*`]{`*style*`}{`*entry level list*`}`

v3.20

These commands define or configure the content-list entries of a given *entry level*. The *entry level* argument is a symbolic name, e. g. `section`, for the entry to the table of contents of the section level with the same name or `figure` for an entry of a figure to the list of figures. A *style* is assigned to each *entry level*. The *style* has to be defined before using it as an argument of `\DeclareTOCStyleEntry` or `\DeclareTOCStyleEntries`. You can use the *option list* to configure the various, usually *style*-dependent, attributes of the entries.

Currently, tocbasic defines the following entry styles:

`default` defaults to a clone of the `dottedtocline` style. Class authors who use tocbasic are encouraged to change this style to the default content-list style of the class using

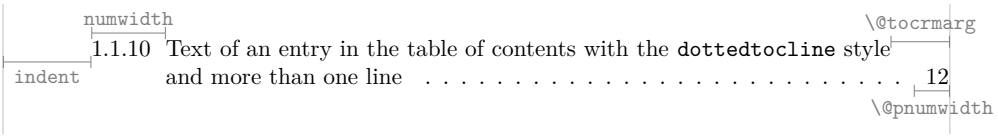


Figure 15.1.: Illustrations of some attributes of a TOC entry with the `dottedtocline` style

`\CloneTOCStyle`. For example the KOMA-Script classes change `default` into a clone of `tocline`.

`dottedtocline` is similar to the style used by the standard classes book and report for the `section` down to `subparagraph` entry levels of the table of contents and for the entries at the list of figures or list of tables. It supports the attributes `level`, `indent`, and `numwidth`. The entries are indented by the value of `indent` from the left. The width of the entry number is given by the value of `numwidth`. For multi-line entries, the indent will be increased by the value of `numwidth` for the second and following lines. The page number is printed using `\normalfont`. The entry text and the page number are connected by a dotted line. Figure 15.1 illustrates the attributes of this style.

`gobble` is the simplest possible style. Entries in this style, regardless of the setting of `tocdepth`, will never be printed. The style simply gobbles the entries, so to speak. It has the default `level` attribute, but it is never evaluated.

`largetocline` is similar to the style used by the standard classes for the `part` level. It supports the `level` and `indent` attributes only. The latter deviates from the standard classes, which do not support an indent of the `part` entries.

A penalty is set to permit page breaks before an entry of an appropriate level. The entries will be indented by the value of `indent` from the left and printed with the font style `\large\bfseries`. If `\numberline` is used, the number width is 3 em. `\numberline` is not redefined. The standard classes do not use `\numberline` for *`part`* entries. The value of *`indent`* also has no effect on the indentation from the second line and after in a multi-line entry.

Figure 15.2 illustrates the characteristics of this style. You will also notice that the style has adopted some inconsistencies present in the standard classes, e. g. the missing indent of the second and following lines of an entry or the different values of `\@pnumwidth` that results from the font-size dependency. This can result, in extreme cases, in the entry text coming too close. Note that the width of the entry number shown in the figure is only valid if `\numberline` has been used. The standard classes, however, use a distance of 1 em after the number.

`tocline` is a flexible style. The KOMA-Script classes use this style by default for all kinds

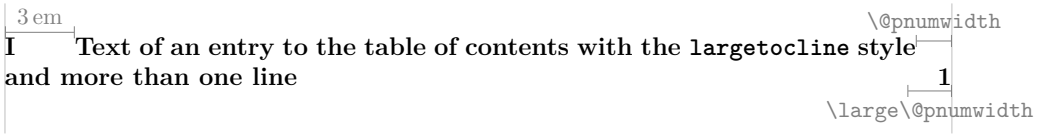


Figure 15.2.: Illustrations of some attributes of a TOC entry with style `largetocline`

of entries. Likewise, these classes define the clones `part`, `chapter`, and `section`, or `section` and `subsection` using this style, but add extra *`initial code`* to the clones to change their defaults.

The style supports 18 additional attributes in addition to the default `level` attribute. The defaults of all these attributes depend on the name of the *`entry level`* and correspond to the results of the standard classes. So after loading tocbasic, you can change the style of the entries in the table of contents of the standard classes into `tocline` using `\DeclareTOCEntryStyle` without this leading directly to major changes in their appearance. Thus you can precisely change only those attributes that are necessary for the desired changes. The same applies to the list of figures and the list of tables for the standard classes.

Because its great flexibility, this style can in principle replace the `dottedtocline`, `undottedtocline`, and `largetocline` styles, but this requires more effort to configure.

Figure 15.3 illustrates some of the length attributes of this style. The others are explained in table 15.1 starting on page 390.

v3.27 `toctext` is a special feature. While all other styles produce one paragraph per entry, this one produces one paragraph for all successive entries of this style. With 13 attributes in addition to the default `level` attribute, there are almost as many options as for `tocline`. However, this style depends on the fact that an unfinished paragraph will be

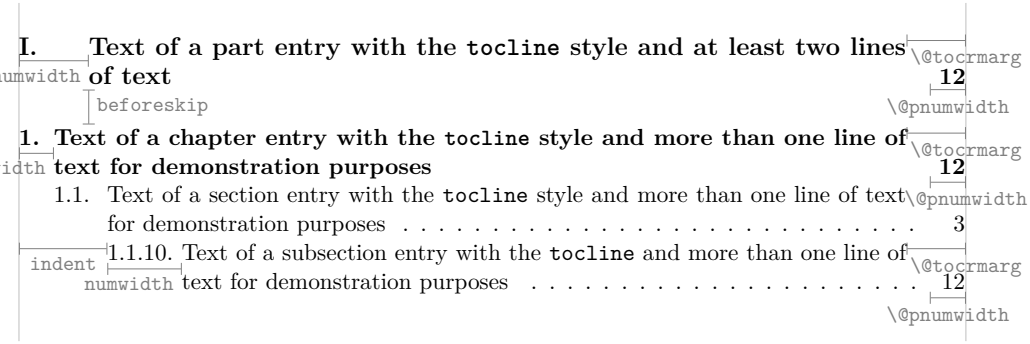


Figure 15.3.: Illustrations of some attributes of a TOC entry with the `tocline` style

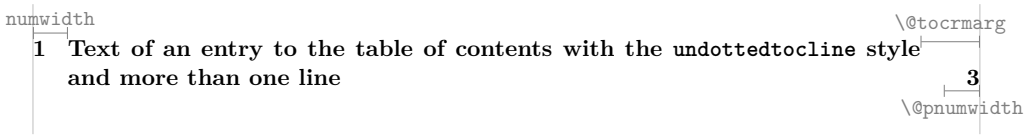


Figure 15.4.: Illustration of some attributes of the `undottedtocline` style with the example of a chapter title

concluded at the beginning of the entry for all other styles, as well as at the end of the current content list. So it should never be combined with entries or content lists that are generated outside of tocbasic.

`undottedtocline` is similar to the style used by the standard book and report classes for the `chapter` entry level, or by article for the `section` entry level in the table of contents. It supports only three attributes. A penalty is inserted permitting an appropriate page break before the entry, as is a vertical skip. The entries are printed with an indentation of `indent` from the left and in `\bfseries`. This is a departure from the standard classes, which do not support the indentation of these entry levels. `\numberline` is used unchanged. The width of the entry number is determined by `numwidth`. For multi-line entries the indent will be increased by the value of `numwidth` for the second and following lines. Figure 15.4 illustrates the attributes of this style.

v3.27 You can find an explanation of the attributes of all styles that tocbasic defines in table 15.1. In addition to the usual assignment with *`key=value`*, both commands understand an assignment in the form *`key:=entry level`* for all options of the KOMA-Script styles. In this case, the current setting of *`key`* for the *`entry level`* will be copied. For example, you can copy the current indent of the `figure` level using `indent:=figure`. For options that expect a length or integer value, you can also use *`key+=value`* to add *`value`* to the current setting of the *`key`*. To subtract simply, you can use a negative *`value`*. For example, with `indent+=1cm` you can increase the indent by 1 cm.

v3.21 If you use these attributes as options to `\DeclareNewTOC` (see page 404), you must prefix the names of the attributes with `tocentry`, e g., `level` becomes `tocentrylevel`. The copy operation with `:=` described above is also available here. However, addition with `+=` is not currently supported.

v3.20 If you use these attributes as options for `\DeclareSectionCommand` (see page 479) and similar commands, you must prefix the names of the attributes with `toc`, e g. `level` becomes `toclevel`. At this time, neither the copy operation with `:=` nor the addition operation with `+=` is supported.

Finally, using `\DeclareTOCStyleEntry` will define the internal command `\l@`*`entry level`*.

Table 15.1.: Attributes of the predefined TOC-entry styles of tocbasic

`afterpar=`*code*

v3.27

The specified *code* will be executed after the end of the paragraph in which an entry with the `toctext` style is printed. If several entries have such settings, their code will be executed in the order of the entries.

`beforeskip=`*length*

The vertical distance inserted before an entry of this level in the `tocline` style (see figure 15.3). The distance is made using either `\vskip` or `\addvspace` depending on the *entry level*, to maintain compatibility as far as possible with the standard classes and earlier versions of KOMA-Script.

For the `part` *entry level*, the attribute will be initialised with `2.25em plus 1pt`; for `chapter`, with `1em plus 1pt`. If the *chapter entry level* is undefined, `section` is initialised with `1em plus 1pt` instead. The initial value for all other levels is `0pt plus .2pt`.

`breakafternumber=`*switch*

switch is one of the values for simple switches from table 2.5, page 42. If the switch is active in the `tocline` style, there will be a line break after the number set with `\numberline`. The line after the entry number again starts flush left with the number. The default is false for the `tocline` style.

If the `numberline` feature has been activated for a content list (see `\setuptoc`, section 15.2, page 383), as is the case with the KOMA-Script classes when the `toc=numberline` option is used, then the unnumbered entries will nevertheless have a (by default empty) number line using the formatting of `entrynumberformat`.

`dynnumwidth=`*switch*

switch is one of the values for simple switches from table 2.5, page 42. If the switch is active with the `tocline` style, the `numwidth` attribute specifies a minimum value. If a previous LaTeX run has determined that the maximum width of the entry numbers of the same level plus the value of `numsep` is greater than this minimum, the calculated value will be used instead.

...

Table 15.1.: Attributes of the TOC-entry styles (*continued*)

`entryformat=`*command*

You can use this attributes to change the format of the entry. The value should be a *command* with exactly one argument. This argument is not necessarily fully expandable. You should not use commands like `\MakeUppercase`, which expects a fully expandable argument. Font changes are relative to `\normalfont\normalsize`. Note that the output of `linefill` and the page number are independent of `entryformat`. See also the `pagenumberformat` attribute .

The initial value of the attribute for the `part` *entry level* is `\large\bfseries`, and for `chapter`, it is `\bfseries`. If the `chapter` level is not defined, `section` uses `\bfseries`. All other levels print the argument unchanged.

`entrynumberformat=`*command*

You can use this attribute to format the entry number within `\numberline`. The value should be a *command* with exactly one argument. Font changes are relative to the one of attribute `entryformat`.

The initial *command* prints the argument unchanged. This means the entry number will be printed as it is.

If the `numberline` feature for a content list has been activated (see `\setuptoc`, section 15.2, page 383), as is the case with the KOMA-Script classes using the `toc=numberline` option, the unnumbered entries will execute the *command* as well.

`indent=`*length*

v3.27

For the `toctext` style, the *length* is the horizontal distance of the paragraph from the left margin. If different entries within the paragraph have different settings, the last one is used. For the remaining styles, the *length* is the horizontal distance of the entry from the left margin (see figure 15.1 and figure 15.3).

For the styles `tocline` and `toctext`, all entry levels whose names start with "`sub`" are initialised with the `indent+numwidth` of the entry level of the same name without this prefix. For the `dottedtocline`, `undottedtocline`, and `tocline` styles, the initial values of levels `part` down to `subparagraph` and the levels `figure` and `table` are compatible with the standard classes. All other levels do not have an initial value. Therefore you have to set an explicit value for such levels when they are defined first time.

If the `noindent` attribute is set for a content list via `\setuptoc`, the entries of all styles provided by KOMA-Script enforce the value 0 pt to deactivate the indent.

...

Table 15.1.: Attributes of the TOC-entry styles (*continued*)

`level=`*integer*

The numerical value of the *entry level*. Only those entries whose numerical value is less than or equal to the `tocdepth` counter are printed.
This attribute is mandatory for all styles and will be defined automatically when the style is declared.
For the `tocline` and `toctext` styles, all entry levels whose name starts with "`sub`" are initialised with the value of the entry level of the same name without this prefix plus one. For the `dottedtocline`, `largetocline`, `tocline`, `toctext`, and `undottedtocline` styles, the entry levels from `part` to `subparagraph`, as well as `figure` and `table`, are initialised to be compatible with the standard classes. For all other levels, the initialisation is done with the value of `\`*entry level*`numdepth`, if this is defined.

`linefill=`*code*

With the `tocline` style, you can change what is used to fill the space between the end of the entry text and the page number. The *linefill* attribute contains the *code* that prints this filler. For the `part` and `chapter` *entry level*s, the attribute is initialised with `\hfill`. If no `chapter` *entry level* has been defined, `section` also uses `\hfill`. All other entry levels are initialised with `\TOCLineLeaderFill` (see page 399).
Incidentally, if the *code* specified does not automatically fill the gap, you should also activate the `raggedpagenumber` attribute to avoid "`underfull \hbox`" messages.

`numsep=`*length*

The `tocline` style tries to ensure a minimum distance of *length* between the entry number and the entry text. If `dynnumwidth` is active, it will correct the number width to achieve this. Otherwise it simply throws a warning if the condition is not met.

v3.27 The `toctext` style, on the other hand, always adds a horizontal space of width *length* after the number of the entry.
The initial *length* is 0.4 em.

...

Table 15.1.: Attributes of the TOC-entry styles (*continued*)

`numwidth=`*`length`*

The width reserved for the entry number (see figure 15.1 to figure 15.4). With the `dottedtocline`, `tocline`, and `undottedtocline` styles, this *`length`* is added to the *`length`* of attribute `indent` for the second and following lines of the entry text.
With the `tocline` style, the initial *`length`* of all entries whose name starts with "`sub`" is the value of the level without this prefix plus 0.9 em, if such a level with corresponding attributes exists. With the `dottedtocline`, `undottedtocline`, and `tocline` styles, the initial *`length`*s of levels from `part` to `subparagraph`, as well as *`figure`* and *`table`*, are compatible with those of the standard classes. All other levels do not have an initial value. Therefore you must set `numwidth` explicitly when the entry level is first used.

`onendentry=`*`code`*

v3.27

The *`code`* is executed immediately after an entry with the `toctext` style, if this entry is not the last one of the paragraph. The user must ensure that the *`code`* does not result in the paragraph ending.
Note: In reality the *`code`* is not executed at the end of the entry but before the next entry with style `toctext`.

`onendlastentry=`*`code`*

v3.27

The *`code`* is executed immediately before the end of the paragraph with an entry in the `toctext` style, as long as this entry is the last one in the paragraph. The user must ensure that the *`code`* does not result in the paragraph ending.

`onstartentry=`*`code`*

v3.27

The *`code`* is executed immediately before an entry with the `toctext` style, unless it is the first one in the paragraph. The user must ensure that the *`code`* does not result in the paragraph ending.

`onstartfirstentry=`*`code`*

v3.27

The *`code`* is executed immediately before an entry with the `toctext` style if this entry is the first one of the paragraph. The user must ensure that the *`code`* does not result in the paragraph ending.

...

Table 15.1.: Attributes of the TOC-entry styles (*continued*)

`onstarthigherlevel=`*code*

The `tocline` style can execute *code* at the start of an entry, depending on whether the previous entry had numerical level greater than, the same as, or less than the current entry. The *code* specified by this attribute will be executed if the current entry has a greater numerical value, i. e. it is lower in the entry hierarchy, than the previous one.

Note that detecting the level of the previous entry only works so long as `\lastpenalty` has not changed since the previous entry.

The initial *code* is `\LastTOCLevelWasLower` (see page 399).

`onstartlowerlevel=`*code*

The `tocline` style can execute *code* at the start of an entry, depending on whether the previous entry had numerical level greater than, the same as, or less than the current entry. The *code* specified by this attribute will be executed if the current entry has a lower numerical value, i. e. it is higher in the entry hierarchy, than the previous one.

Note that detecting the level of the previous entry only works so long as `\lastpenalty` has not changed since the previous entry.

The initial *code* is `\LastTOCLevelWasHigher` (see page 399), which usually favours a page break before the entry.

`onstartsamelevel=`*code*

The `tocline` style can execute *code* at the start of an entry, depending on whether the previous entry had numerical level greater than, the same as, or less than the current entry. The *code* specified by this attribute will be executed if the current entry has the same numerical value, i. e. it is on the same level in the entry hierarchy, as the previous one.

Note that detecting the level of the previous entry only works so long as `\lastpenalty` has not changed since the previous entry.

The initial *code* is `\LastTOCLevelWasSame` (see page 399), which usually favours a page break before the entry.

`pagenumberbox=`*command*

By default the page number of an entry is printed flush right in a box of width `\@pnumwidth`. In the `tocline` style, you can change the *command* to print the number using this attribute. The *command* should expect exactly one argument, the page number.

This attribute is initialised with the box already mentioned.

. . .

Table 15.1.: Attributes of the TOC-entry styles (*continued*)

`pagenumberformat=`*command*

You can use this attribute to change the format of the page number of an entry. The *command* should expect exactly one argument, the page number. Font changes are relative to the font of `entryformat` followed by `\normalfont\normalsize`.
The initial *command* of entry level `part` prints the argument in `\large\bfseries` and of entry level `chapter` in `\bfseries`. For classes without `\l@chapter` `section` also uses `\bfseries`. The initial *command* of all other levels prints the argument in `\normalfont\normalcolor`.

`pagenumberwidth=`*length*

v3.27 You can use this attribute to locally change the width of the default box for the page number of an entry with the style `tocline` from `\@pnumwidth` to the specified *length*. Note that if you change the default page number box with the `pagenumberbox` attribute, the specified *length* will no longer be used automatically.

`prepagenumber=`*code*

v3.27 The `toctext` style executes the *code* between the text and the page number of the entry. Usually this is used to add a horizontal space or separator between text and page number.
The default is a non-breaking space using `\nonbreakspace`.

`raggedentrytext=`*switch*

v3.21 The *switch* is one of the values for simple switches from table 2.5, page 42. If the switch is active, the `tocline` style prints the text of an entry ragged right instead of fully justified, and only words that are longer than a text line are automatically hyphenated.
This *switch* is false by default.

`raggedpagenumber=`*switch*

The *switch* is one of the values for simple switches from table 2.5, page 42. If the switch is active, the `tocline` style does not force the page number to be right justified.
Depending on the value of `linefill`, setting this attribute could affect only whether a warning message appears, or the formatting of the page number as well. So it is important to set both attributes so that they correspond.
By default the *switch* is not activated and therefore corresponds with an initial value of `\hfill` or `\TOCLineLeaderFill` for the `linefill` attribute.

...

Table 15.1.: Attributes of the TOC-entry styles (*continued*)

`raggedright=`*switch*

v3.27 The *switch* is one of the values for simple switches from table 2.5, page 42. If the switch is active for any entry with the `toctext` style inside the same paragraph, the whole paragraph is printed ragged right.

`rightindent=`*length*

v3.27 You can use this attribute to locally change the right indent for the text of an entry with the `tocline` style from `\@tocrmarg` to the specified *length*.

v3.26 While `\DeclareTOCStyleEntry` defines only one *entry level*, `\DeclareTOCStyleEntries` can define an entire list of *entry level*s in one command. Each entry level in the comma-separated *entry-level list* is defined with the same *style* and settings of the given *option list*.

```
\DeclareTOCEntryStyle{style}[initial code]{command code}
\DefineTOCEntryOption{option}[default value]{code}
\DefineTOCEntryBooleanOption{option}[default value]{prefix}{postfix}{description}
\DefineTOCEntryCommandOption{option}[default value]{prefix}{postfix}{description}
\DefineTOCEntryIfOption{option}[default value]{prefix}{postfix}{description}
\DefineTOCEntryLengthOption{option}[default value]{prefix}{postfix}{description}
\DefineTOCEntryNumberOption{option}[default value]{prefix}{postfix}{description}
```

v3.20 `\DeclareTOCEntryStyle` is one of the most complex commands in KOMA-Script. It is therefore explicitly intended for LaTeX developers and not for ordinary LaTeX users. It lets you define new a *style* for content-list entries. Usually, entries to content lists are made using `\addcontentsline`, or preferably, if you use tocbasic, with `\addxcontentsline` (see section 15.1, page 378). In both cases LaTeX writes a corresponding `\contentsline` to the appropriate auxiliary file. When reading this auxiliary file, LaTeX then executes a `\l@`*entry level* command for each `\contentsline`.

If you later assign a *style* to an entry level using `\DeclareTOCStyleEntry`, the *initial code* is executed first, if provided, and then the *command code* for the definition of `\l@`*entry level*. The *command code* is the code that will be expanded and executed by `\l@`*entry level*. Inside *command code* `#1` is the name of the TOC entry level and `##1` and `##2` are the arguments of `\l@`*entry level*.

The *initial code* serves first to initialise all attributes of the *style*. Developers should make sure that all attributes are provided with values here. Only then does `\DeclareTOCStyleEntry` work without errors if an *option list* is not specified. The second task of the *initial code* is to define all the options that this *style* recognises. The `level` option is always defined automatically. The value of the `level` can be queried within the

command code with \@nameuse{#1tocdepth}, for example, to compare it with the tocdepth counter.

To define options for the attributes of the *style* inside the *initial code*, you can use the commands \DefineTOCEntryBooleanOption, \DefineTOCEntryCommandOption, \DefineTOCEntryIfOption, \DefineTOCEntryLengthOption, and \DefineTOCEntryNumberOption. These commands each define an *option* that, when called, defines a macro named *prefix entry level postfix* set to the given value or to the *default value* of the option. The \DefineTOCEntryIfOption command is a somewhat special case. It defines *prefix entry level postfix* as a command with two arguments. If the value passed to the option is one of the activation (true) values from table 2.5, page 42, the command expands to the first argument. If the value to the option is a deactivation (false) value, the command expands to the second argument.

v3.27 In addition to the usual options of the form *key=value*, the five \DefineTOCEntry...Option commands automatically define options of the form *key:=entry level*. These copy the value of another *entry level* if the value is stored in a macro with the same *prefix* and *postfix*. For the styles predefined by tocbasic, this is the case for all options of the same name independent of the name of the style. The commands \DefineTOCEntryLengthOption and \DefineTOCEntryNumberOption also define options of the form *key+=value*, which are used to add the new *value* to the value already stored in *prefix entry level postfix*.

The *description* should be a brief message that describes the sense of the option with some keywords. The tocbasic package uses this text in error messages, warnings, and information output on the terminal and to the log file.

The simplest style of tocbasic, gobble, is defined using:

```
\DeclareTOCEntryStyle{gobble}{}%
```

If you now define an entry level dummy in this style using:

```
\DeclareTOCStyleEntry[level=1]{gobble}{dummy}
```

this would correspond, among other things, to:

```
\def\dummytocdepth{1}
\def\l@dummy#1#2{}
```

For example, within the tocline style,

```
\DefineTOCEntryCommandOption{linefill}[\TOCLineLeaderFill]%
{scr@tso@}{@linefill}{filling between text and page number}%
```

is used to define the linefill option. By specifying \TOCLineLeaderFill as the *default value*, a call such as

```
\DeclareTOCStyleEntry[linefill]{tocline}{part}
```

would, among other things, create the definition

```
\def\scr@tso@part@linefill{\TOCLineLeaderFill}
```

If you want to define your own styles, you should first study the definition of the `dottedtocline` style. After you understand this definition, you can find many hints as to how to use the commands effectively in the much more complex definition of the `tocline` style.

However, in many cases it will be sufficient to clone an existing style using `\CloneTOCEntryStyle` and to change the initial code of the new style using `\TOCEntryStyleInitCode` or `\TOCEntryStyleStartInitCode`.

`\DefineTOCEntryOption` is merely used to define the other commands and you should not use it directly. Normally, there is no need for it. It is mentioned here only for the sake of completeness.

`\CloneTOCEntryStyle{`*style*`}{`*new style*`}`

v3.20

With this command you can clone an existing *style*. It defines a *new style* with the same attributes and settings as the existing *style*. The package itself uses `\CloneTOCEntryStyle` to declare the `default` style as a clone of `dottedtocline`. The KOMA-Script classes use the command to declare the styles `part`, `section`, and `chapter` or `subsection` as clones of `tocline` and then modify them with `\TOCEntryStyleInitCode` and `\TOCEntryStyleStartInitCode`. The scrbook and scrreprt classes newly declare the `default` style as a clone of `section`, and scrartcl declares it as a clone of `subsection`.

`\TOCEntryStyleInitCode{`*style*`}{`*initial code*`}`
`\TOCEntryStyleStartInitCode{`*style*`}{`*initial code*`}`

v3.20

Every TOC-entry style has an initialisation code. This is used whenever a *style* is assigned to an TOC entry using `\DeclareTOCEntryStyle`. This *initial code* should not have global side effects, because it is also used for local initialisation inside other commands like `\DeclareNewTOC`. The *initial code* not only defines all attributes of a *style*, but it also sets the defaults for those attributes.

You can use `\TOCEntryStyleStartInitCode` and `\TOCEntryStyleInitCode` to extend previously existing initialisation code with further *initial code*. `\TOCEntryStyleStartInitCode` adds *initial code* in front of the existing code. `\TOCEntryStyleInitCode` adds the *initial code* at the end of the existing initialisation code. The KOMA-Script classes, for example, use `\TOCEntryStyleStartInitCode` to properly initialise the fill, fonts, and vertical spacing of the `part` style cloned from `tocline`. For example, the scrbook and scrreprt classes use

```
\CloneTOCEntryStyle{tocline}{section}
\TOCEntryStyleStartInitCode{section}{%
  \expandafter\providecommand%
  \csname scr@tso@#1@linefill\endcsname
  {\TOCLineLeaderFill\relax}%
```

```
}
```

to define `section` as a modified clone of `tocline`.

`\LastTOCLevelWasHigher`
`\LastTOCLevelWasSame`
`\LastTOCLevelWasLower`

v3.20 At the beginning of entries using the `tocline` style, tocbasic executes one of these three commands depending on `\lastpenalty`. `\LastTOCLevelWasHigher` and `\LastTOCLevelWasSame` used in vertical mode add `\addpenalty{\@lowpenalty}` and therefore permit a page break before an entry with the same or higher hierarchical position. `\LastTOCLevelWasLower` is empty, so a page break between an entry and its first sub-entry is not permitted.

Users should not redefine these commands. Instead, you should change the behaviour of single entry levels using the `onstartlowerlevel`, `onstartsamelevel`, and `onstarthigherlevel` attributes.

`\TOCLineLeaderFill[`*`leader`*`]`

v3.20 This command is intended to be used as a value for the `linefill` option of the *`tocline`* TOC-entry style. It creates a connection between the end of the entry text and the entry's page number. You can specify the *`leader`*, which is repeated at regular intervals, as an optional argument. The default is a dot.

As the name suggests, the command uses `\leaders` to output the *`leader`*. The spacing used is defined analogously to the LaTeX kernel command `\@dottedtocline` by `\mkern\@dotsepmu`.

15.4. Internal Commands for Class and Package Authors

The tocbasic package provides some internal commands for the use of class and package authors. These commands all begin with the prefix `\tocbasic@`. But even class or package authors should not redefine them! Their inner functioning may be changed or extended at any time, so redefining these commands could significantly damage the tocbasic's operation.

`\tocbasic@extend@babel{`*`extension`*`}`

At every change of the current language, either at the beginning of the document or inside the document, the babel package (see [BB13]), or rather a LaTeX kernel enhanced by babel's language management, writes language-switching commands to the files with the `toc`, `lof`, and `lot` extensions. The tocbasic package extends this mechanism with `\tocbasic@extend@babel` so that it also works for other file extensions. The *`extension`* argument must be completely expanded! Otherwise the there is a risk that, for example, the meaning of the argument has already change at the time it is actually evaluated.

This command is typically invoked by default for every file *extension* added to the list of known extensions with \addtotoclist. You can suppress this with the nobabel feature (see \setuptoc, section 15.2, page 383). tocbasic does this automatically for the extensions toc, lof, and lot to avoid switching languages twice in the corresponding files.

There is usually no reason to call this command yourself. However, there could conceivably be content lists that are not under the control of tocbasic and so are not in tocbasic's list of known file extensions, but which nevertheless should use babel's language switching mechanism. You can call the command explicitly for those files. But please note that this should be done only once per file extension!

\tocbasic@starttoc{*extension*}

This command is the actual replacement for the \@starttoc command from the LaTeX kernel. It is the command behind \listoftoc* (see section 15.2, page 380). Class or package authors who want to take advantage of tocbasic should at least use this command, or even better, \listoftoc. The command uses \starttoc internally, but sets \parskip, \parindent to 0, and \parfillskip to 0 to infinity. Moreover, \@currext is set to the file extension of the current TOC file, so it can be evaluated during the subsequent execution of the hooks. You can find an explanation of these hooks below.

Because LaTeX opens a new content-list file for writing after reading that file, calling this command may result in an error message of the type

```
! No room for a new \write .
\ch@ck ...\else \errmessage {No room for a new #3}
                                                \fi
```

if no more write handles are available. You can solve this problem by loading the scrwfile package described in chapter 14, or by using LuaLaTeX.

\tocbasic@@before@hook
\tocbasic@@after@hook

The \tocbasic@@before@hook hook is executed immediately before reading an auxiliary file for a content list, before executing the commands defined with \BeforeStartingTOC command. You can extend this hook using \g@addto@macro.

Similarly, \tocbasic@@after@hook is executed immediately after reading such an auxiliary file and before executing the commands defined with \AfterStartingTOC. You can extend this hook using \g@addto@macro.

KOMA-Script uses these hooks to dynamically adjust content lists to the width of the heading numbers. Only classes and packages should use these hooks. Users should really use \BeforeStartingTOC and \AfterStartingTOC instead. Authors of packages should also prefer these commands. These hooks should not be used to generate any output!

If neither \listofeachtoc nor \listoftoc nor \listoftoc* are used to output the content list, the hooks should be executed explicitly.

`\tocbasic@`*extension*`@before@hook`
`\tocbasic@`*extension*`@after@hook`

These hooks are executed directly after `\tocbasic@@before@hook` or before `\tocbasic@@after@hook` for the TOC file with the corresponding file *extension*. Class and package authors should never change them under any circumstances! If neither `\listofeachtoc` nor `\listoftoc` nor `\listoftoc*` are used to output a content list, the hooks should nevertheless be called, if they are defined. These commands can be undefined.

`\tocbasic@listhead{`*title*`}`

This command is used by `\listoftoc` and `\listofeachtoc` to set the heading of the content list. This can be either the default heading of the tocbasic package or a custom definition. If you define your own command for the heading, you can also use `\tocbasic@listhead`. In this case, you should define `\@currext` to be the file extension of the corresponding TOC file before using `\tocbasic@listhead`.

`\tocbasic@listhead@`*extension*`{`*title*`}`

This command is used in `\tocbasic@listhead` to set the individual headings, optional table of contents entry, and running head, if it is defined. Otherwise, `\tocbasic@listhead` defines them before their use.

`\tocbasic@addxcontentsline{`*extension*`}{`*level*`}{`*number*`}{`*text*`}`
`\nonumberline`

v3.12

The `\tocbasic@addxcontentsline` command creates entry of the specified level in the TOC file with the given *extension*. Whether the entry is numbered or not depends on whether or not the *number* argument is empty. In this case the *text* will be prefixed by `\nonumberline` without any argument. Otherwise, `\numberline` with the *number* argument will used as usual.

The `\nonumberline` command is redefined inside `\listoftoc` (see section 15.2, page 380) depending on the `numberline` feature (see section 15.2, page 383). As a result, changing this feature results in changes of the corresponding TOC immediately at the next LaTeX run.

`\tocbasic@DependOnPenaltyAndTOCLevel{`*entry level*`}`
`\tocbasic@SetPenaltyByTOCLevel{`*entry level*`}`

v3.20

The `tocline` content-list style (see section 15.3) sets a `\penalty` at the end of each entry via `\tocbasic@SetPenaltyByTOCLevel` so that no page break can occur after an entry. The exact value chosen depends on the *entry level*.

At the beginning of an entry, `\tocbasic@DependOnPenaltyAndTOCLevel` is used to execute the value of the `onstartlowerlevel`, the `onstartsamelevel`, or the `onstarthigherlevel` style option, depending on `\lastpenalty` and the current *entry level*. By default, the first two permit a page break when executed in vertical mode.

Developers of `tocline`-compatible styles should copy this behaviour. To do so, they can fall back on these internal macros.

15.5. A Complete Example

This section provides a complete example of how to define your own floating environment including an associated content list and KOMA-Script integration using tocbasic. This example uses internal commands, that is, they have a "`@`" in their name. This means, that you must either put the code into a package or class or placed it between `\makeatletter` and `\makeatother`.

First, we need a new floating environment. That's easy with the following:

```
\newenvironment{remarkbox}{%
  \@float{remarkbox}%
}{%
  \end@float
}
```

The new environment is named `remarkbox`.

Each floating environment has a default placement. It consists of one or more of the well-known placement options: `b`, `h`, `p` and `t`.

```
\newcommand*{\fps@remarkbox}{tbp}
```

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 separate page.

Floating environments also have a numerical floating type between 1 and 31. Environments with the same active bit at the floating type cannot change their order. Figures and tables normally use type 1 and 2. So a figure that comes later in the source code than a table may be output earlier than the table and vice 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, a new counter is defined first, which is independent of the chapters or the counters of other structural levels. LaTeX itself also defines `\theremarkbox` with the default output as an Arabic number. This is then used to define the formatted output of the counter. The formatted output is again defined as a floating-point number for use in the `\caption` command.

Floating environments have their own content lists and those need an auxiliary file named \jobname and a file extension:

```
\newcommand*{\ext@remarkbox}{lor}
```

As the file extension, we use "lor".

With this, the floating environment works. But the content list of is still missing. So that we do not have to implement it ourselves, we use the tocbasic package. This is loaded with

```
\usepackage{tocbasic}
```

inside of document preambles. Class or package authors would use

```
\RequirePackage{tocbasic}
```

instead.

Now we register the file name extension with the tocbasic package:

```
\addtotoclist[float]{lor}
```

We use float as the owner so that all options of KOMA-Script classes that relate to lists of floating environments also apply to the new content list.

Next we define a title or heading for this content list:

```
\newcommand*{\listoflorname}{List of Remarks}
```

When working with multiple languages, the normal practice is to define an English title first and then, for example with the help of the scrbase package, to add titles for all the other languages you want to support. See section 12.4, starting on page 349.

Now all we have to do is define what a single entry in the content list should look like:

```
\newcommand*{\l@remarkbox}{\l@figure}
```

This specifies that entries in the list of remarks should look exactly like the entries in the list of figures. This would be the easiest solution. A more explicit definition would be something like:

```
\DeclareTOCStyleEntry[level=1,indent=1em,numwidth=1.5em]%
                     {tocline}{remarkbox}
```

You also want chapter entries to affect the content list.

```
\setuptoc{lor}{chapteratlist}
```

Setting this property allows this when you use a KOMA-Script class, and other class that supports this property. Unfortunately, the standard classes do not.

This should be enough. Users can now select different kinds of headings using the corresponding options of the KOMA-Script classes or \setuptoc, (e. g. with or without an entry in the table of contents, with or without numbering). But with a simple

```
\newcommand*{\listofremarkboxes}{\listoftoc{lor}}
```

you can simplify the usage even more.

As you've seen, just five one-line commands, of which only three or four are really necessary, refer to the content list. Nevertheless, the new list of remarks already provides the ability to place both numbered and unnumbered entries into the table of contents.You can use a lower sectioning level for the headings. Running heads are set for 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 each new \chapter. Even changes to the current language made with babel are included in the list of remarks.

Of course, package authors can add more features. For example, they could explicitly offer options to hide \setuptoc from users. Or they can refer to the tocbasic manual when explaining the appropriate features. The advantage of this is that users automatically benefit from any future extensions to tocbasic. However, if you do not want to burden the user with the fact that the file extension lor is used for the key terms, then

```
\newcommand*{\setupremarkboxes}{\setuptoc{lor}}
```

is sufficient to set a list of features passed as an argument to \setupremarkboxes as a list of features for the file extension lor.

15.6. Everything with Only One Command

The example in the previous section has shows that tocbasic makes it easy to define your own floating environments with their own content lists. This section shows how it can be even easier.

\DeclareNewTOC[*options*]{*extension*}

v3.06 This command declares a new content list, its heading, and the description of the entries controlled by tocbasic all in a single step. Optionally, you can also define floating and non-floating environments at the same time. Inside of both such environments, \caption creates entries for this new content list. You can also use the KOMA-Script extensions \captionabove, \captionbelow, and captionbeside (see section 3.20).

The *extension* argument is the file extension of the TOC file that represents the content list, as explained in section 15.1. This argument is mandatory and must not be empty!

The *options* argument is a comma-separated list, of the same type as, for example, \KOMAoptions (see section 2.4). However, those options cannot be set using \KOMAoptions! You can find an overview of all available options in table 15.2.

v3.20 If the tocentrystyle option is not used, the default style will be used if required. For information about this style, see section 15.3. If you do not want to define a command for entries to the content list, you can use an empty argument, i. e. tocentrystyle= or tocentrystyle={}.

v3.20 Depending on the style of the entries to the content list, you can set all valid attributes of the selected style as part of the *options*. To do so, you must add the prefix tocentry

to the names of the attributes given in table 15.1, starting on page 390. You can make later changes to the style of the entries at any time using \DeclareTOCStyleEntry. See section 15.3, page 386 for more information about the styles.

v3.06 Table 15.2.: Options for the \DeclareNewTOC command

v3.09 atbegin=*commands*
: The *commands* will be executed at the begin of the floating or non-floating environment.

v3.09 atend=*commands*
: The *commands* will be executed at the end of the floating or non-floating environment.

category=*string*

v3.27 : This option can be used as a synonym for owner=*string*.

counterwithin=*LaTeX counter*
: If you define a new floating or non-floating environment, a new counter *type* will be created as well (see option type). You can make this counter depenent another LaTeX counter in the same way, for example, that the figure counter in the book classes is dependent on the chapter counter.

float
: If set, defines a new content list and a floating environment, both named *type*, and an environment for double-column floats named *type**.

floatpos=*float positions*
: Each floating environment has default *float positions* that can be changed through the optional argument of the floating environment. The syntax and semantics are identical to those of the standard floating environments. If the option is not used, the default *float positions* are "tbp", that is *top*, *bottom*, *page*.

floattype=*number*
: Each floating environment has a *number*. Floating environments where only different bits are set can be moved past each other. The floating environments figure and table usually have the types 1 and 2, so they can move past each other. The numerical float type can be between 1 and 31. If common bits are set, the float types cannot be reordred. If no float type is given, the greatest possible one-bit type, 16, will be used.

...

Table 15.2.: Options for the \DeclareNewTOC command (*continued*)

`forcenames`
: If set, the names will be defined even if they were already defined before.

`hang=`*`length`*
v3.20
: This option has been deprecated since KOMA-Script 3.20. Instead, the amount of the hanging indent of entries to the content list depends on attributes of the TOC-entry style given by the `tocentrystyle` option. The KOMA-Script styles provide the `numwidth` attribute. If the style used has such an attribute, `\DeclareNewTOC` will initialise it with a default of 1.5 em. You can easily change the *`value`* using `tocentrynumwidth=`*`value`*. The KOMA-Script classes, for example, use `tocentrynumwidth=2.3em`.

`indent=`*`length`*
v3.20
: This option has been deprecated since KOMA-Script 3.20. Instead, the amount that entries to the content list are indented depends on attributes of the TOC-entry style given by the `tocentrystyle` option. The KOMA-Script styles provide the `indent` attribute. If the style used has such an attribute, `\DeclareNewTOC` will initialise it with a default of 1 em. You can easily change the *`value`* using `tocentryindent=`*`value`*. The KOMA-Script classes for example use `tocentrynumwidth=1.5em`.

`level=`*`number`*
v3.20
: This option has been deprecated since KOMA-Script 3.20. Instead, the level of the entries to the content list depends on attributes of the TOC-entry style given by the `tocentrystyle` option. Nevertheless, all styles have the `level` attrobite, and `\DeclareNewTOC` initialises it with a default value of 1. You can easily change the *`value`* using `tocentrylevel=`*`value`*.

`listname=`*`title`*
: Each content list has a heading, or title, that you can specify with this option. If the option is not specified, the title will be "List of *`entry type`*" (see the `types` option), with the first character of the *`entry type`* changed to upper case. It also defines the `\list`*`entry type`*`name` macro with this value, which you can change at any time. This macro, however, is only defined if it is not already defined or if the `forcenames` option is also set.

...

Table 15.2.: Options for the \DeclareNewTOC command (*continued*)

name=*entry name*

Both the optional prefix for entries in the content list and the labels in floating or non-floating environments (see the float and nonfloat options) require an *entry name* for an entry to the content list. If no *entry name* is given, the value of the type (see the type option) with the first character changed to upper case will be used. It also defines a *entry type*name macro with this value, which you can change at any time. This macro, however, is only defined if it is not already defined or if the forcenames option is also set.

nonfloat

If set, defines not only a content list but also a non-floating environment, *entry type*- (see the type option), which can be used similarly to a floating environment, but which does not move from the place where it is used.

owner=*string*

Every new content list has an owner in tocbasic (see section 15.1). You can specify this here. If no owner is specified, the owner "float" is used. The KOMA-Script classes use this owner for the list of figures and the list of tables.

setup=*list of attributes*

v3.25 The *list of attributes* is set with \setuptoc. Note that to specify multiple attributes in a comma-separated list, you must put this list between braces.

tocentrystyle=*TOC-entry style*

v3.20 *TOC-entry style* specifies the style that should be used for all entries to the content list corresponding to the *extension*. The name of the entry level is given by the type option. In addition to the options in this table, all attributes of the *TOC-entry style* can be used as options. To do so, you have to prefix the name of such an attribute with tocentry. For example, you can change the numerical level of the entries using the tocentrylevel option. For more information about the styles and their attributes see section 15.3, starting on page 385.

tocentry*style-option*=*value*

v3.20 Additional options depending on the *TOC-entry style* given by tocentrystyle. See section 15.3, page 385 for additional information about TOC-entry styles. See table 15.1, page 390 for information about the attributes of the predefined TOC-entry styles of package tocbasic that can be used as *style-option*.

...

Table 15.2.: Options for the \DeclareNewTOC command (*continued*)

type=*entry type*
: Sets the type of the newly declared content list. The *entry type* is also used as a base name for various macros and possibly environments and counters. It should therefore consist only of letters. If this option is not used, the file *extension* from the mandatory argument will be used as the *entry type*.

types=*string*
: In several places, the plural form of the *entry type* is required. If no plural is given, the value of the entry type with an "s" appended will be used.

unset=*list of attributes*
v3.25
: The *list of attributes* is unset with \unsettoc. Note that to specify a comma-separated list of attributes, you must put this list between braces.

Example: Using \DeclareNewTOC significantly shortens the example from section 15.5:

```
\DeclareNewTOC[%
  type=remarkbox,%
  types=remarkboxes,%
  float,% define a floating environment
  floattype=4,%
  name=Remark,%
  listname={List of Remarks}%
]{lor}
\setuptoc{lor}{chapteratlist}
```

In addition to the remarkbox and remarkbox* environments, this also defines the remarkbox counter; the commands \theremarkbox, \remarkboxname, and \remarkboxformat that are used for captions; the commands \listremarkboxnames and \listofremarkboxes that are used in the list of remarks; and some internal commands that depend on the file name extension lor. If the package should use a default for the floating type, the Optionfloattype option can be omitted. If the nonfloat option is specified, a non-floating environment, remarkbox-, will also be defined, inside which you can use \caption. Figure 15.3 compares the commands, counters, and environments of the example remarkbox environment to the commands, counters, and environments of figures.

And here is a possible use of the example environment:

```
\begin{remarkbox}
  \centering
  The same thing should always be typeset in the same way
  and with the same appearance.
```

Table 15.3.: Comparing the example `remarkbox` environment with the `figure` environment

`remarkbox`	`figure`	options of `\DeclareNewTOC`	short description
`remarkbox`	`figure`	`type`, `float`	floating environments of the respective types
`remarkbox*`	`figure*`	`type`, `float`	columns spanning floating environments of the respective types
`remarkbox`	`figure`	`type`, `float`	counter used by `\caption`
`\theremarkbox`	`\thefigure`	`type`, `float`	output command to the respective counters
`\remarkboxformat`	`\figureformat`	`type`, `float`	formatting command to the respective counters used by `\caption`
`\remarkboxname`	`\figurename`	`type`, `float`, `name`	names used in the label of `\caption`
`\listofremarkboxes`	`\listoffigures`	`types`, `float`	command to show the list of the respective environments
`\listremarboxname`	`\listfigurename`	`type`, `float`, `listname`	heading text of the respective list
`\fps@remarkbox`	`\fps@figure`	`type`, `float`, `floattype`	numeric float type for order perpetuation
`lor`	`lof`		file name extension of the TOC file of the respective list

```
    \caption{First Law of Typography}
    \label{rem:typo1}
  \end{remarkbox}
```

A snippet of a sample page with this environment might look like this:

> The same thing should always be typeset in the same way and with the same appearance.
>
> Remark 1: First Law of Typography

Users of hyperref should always use the `listname` option. Otherwise they may get an error message because hyperref usually has a problem with the `\MakeUppercase` command that is needed to convert the first letter of `types` to upper case.

15.7. Obsolete Befehle

Prior releases of tocbasic provide some commands that has been renamed, because of a statement of The LaTeX Project Team. Those deprecated commands should not be used any longer.

Currently, additional information on this topic can be found at the same point in the German KOMA-Script book [Koh20a] only.

Improving Third-Party Packages with scrhack

Some packages from other authors do not work well with KOMA-Script. It is often very tedious for the author of KOMA-Script to convince the authors of these packages to make specific improvements. This also applies to packages whose development has been discontinued. That's why the scrhack was created. This package alters the commands and definitions of other to work better with KOMA-Script. Some changes are also useful when using other classes.

16.1. Development Status

Although this package has been part of KOMA-Script for long time and is used by many users, there's one problem with it. Redefining macros from third-party packages depends on knowing the exact definition and use of those macros. This also means that it depends on specific versions of those packages. If you use a unknown version of such a package, scrhack may not be able to execute the required patch. In extreme cases, patching an unknown version can lead to an error.

Because scrhack must be continuously modified to accommodate new releases of third-party packages, it can never be considered complete. Therefore scrhack will remain permanently in a beta version. Although its use will generally be a benefit, its correct functioning cannot be permanently guaranteed.

16.2. Early or Late Selection of Options

The information in section 2.4 applies equally to this chapter. So if you have already read and understood section 2.4, you can skip ahead to section 16.3, page 412.

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

LaTeX allows users to pass class options as a comma-separated list of keywords in the optional argument to \documentclass. In addition to being passed to the class, these options are also passed on to all packages that can understand them. Users can also pass a similar comma-separated list of keywords in the optional argument of \usepackage. KOMA-Script extends the option mechanism for the KOMA-Script classes and some packages with further options. Thus most KOMA-Script options can also take a value, so an option does not necessarily take the form *option*, but can also take the form *option=value*. Except for this difference, \documentclass and \usepackage in KOMA-Script function as described in [Tea05b] or any introduction to LaTeX, for example [OPHS11].

v3.00

Setting the options with \documentclass has one major disadvantage: unlike the interface described below, the options in \documentclass are not robust. So commands, lengths,

counters, and similar constructs may break inside the optional argument of this command. For example, with many non-KOMA-Script classes, using a LaTeX length in the value of an option results in an error before the value is passed to a KOMA-Script package and it can take control of the option execution. So if you want to use a LaTeX length, counter, or command as part of the value of an option, you should use \KOMAoptions or \KOMAoption. These commands will be described next.

\KOMAoptions{*option list*}
\KOMAoption{*option*}{*value list*}

v3.00 KOMA-Script also provides the ability to change the values of most class and package options even after loading the class or package. You can use the \KOMAoptions command to change the values of a list of options, as in \documentclass or \usepackage. Each option in the *option list* has the form *option=value*.

Some options also have a default value. If you do not specify a value, that is if you give the option simply as *option*, then this default value will be used.

Some options can have several values simultaneously. For such options, it is possible, with the help of \KOMAoption, to pass a list of values to a single *option*. The individual values are given as a comma-separated *value list*.

KOMA-Script uses the commands \FamilyOptions and \FamilyOption with the family "KOMA" to implement this ability. See part II, section 12.2, page 336.

Options set with \KOMAoptions or \KOMAoption will reach both the KOMA-Script class and any previously loaded KOMA-Script packages that recognise these options. If an option or a value is unknown, scrbase will report it as an error.

16.3. Using tocbasic

In the early days of KOMA-Script, users wanted to handle lists of floating environments created with the float package in the same way as the list of figures and list of tables created by KOMA-Script itself. At that time the author of KOMA-Script contacted the author of float to propose an interface to support such an extension. A somewhat modified form of that interface was implemented with the \float@listhead and \float@addtolists commands.

Later it became apparent that these two commands were not flexible enough to fully support all of KOMA-Script's capabilities. Unfortunately, the author of float had already ceased development by that point, so further changes to this package cannot be expected.

Other package authors have also adopted these two commands, and it became apparent that the implementation in some packages, including float, means that all these packages can only be loaded in a specific order, even though they are otherwise unrelated to each other.

To overcome all these disadvantages and problems, KOMA-Script no longer officially supports this old interface. Instead, KOMA-Script warns if the old interface is used. At the same time, the tocbasic package (see chapter 15) has been designed and implemented as a central

interface for managing of table of contents and similar content lists. This package provides many more advantages and features than the two old commands.

Although the effort to use this package is very small, so far the authors of most of the packages that use the old interface have not made any adjustments. Therefore scrhack contains appropriate modifications of the float, floatrow, and listings packages. Merely loading scrhack is sufficient to make these packages recognize not only the setting of the KOMA-Script option `listof`, but also to react to the language switching features of the babel package. You can find more information about the features available by switching packages to tocbasic in section 15.2.

If you do not want this modification for any of the packages, or if it causes problems, you can deactivate it selectively with the `float=false`, `floatrow=false`, or `listings=false` option. Note that changing these options after loading the associated package has no effect!

16.4. Incorrect Assumptions about `\@ptsize`

Some packages assume that the class-internal macro `\@ptsize` both is defined and expands to an integer. For compatibility, KOMA-Script defines `\@ptsize` even if the basic font size is something other than 10 pt, 11 pt, or 12 pt. KOMA-Script also allows non-integer font sizes. So `\@ptsize` can, of course, also expand to a non-integer number.

v3.17 One of the packages that cannot cope with a non-integer `\@ptsize` is setspace. Additionally, the values set by this package are always dependent on the basic font size, even if the setting is made in the context of another font size. The scrhack package solves both problems by redefining `\onehalfspacing` and `\doublespacing` to set the spacing relative to the actual font size.

If you do not want this modification for the package, or if it causes problems, you can deactivate it selectively with the `setspace=false` option. Note that changing this option after loading setspace has no effect! Likewise, if you use setspace with either the `onehalfspacing` or `doublespacing` option, you must load scrhack first.

16.5. Older Versions of hyperref

Versions of hyperref before 6.79h set the link anchors after the heading of the starred versions of commands like `\part*`, `\chapter*`, etc. instead of before them. Since then, this problem has been resolved at the suggestion of KOMA-Script's author. But because the change took more than a year, a patch was added to scrhack. Although this can also be deactivated with `hyperref=false`, you should instead use the current hyperref release. In this case scrhack automatically deactivates this unnecessary patch.

16.6. Inconsistent Handling of \textwidth and \textheight

v3.18 The lscape package defines a landscape environment to set the page contents, but not the header or footer in landscape mode. Inside this environment, \textheight is set to the value of \textwidth, but \textwidth is not set to the former value of \textheight. This is inconsistent. As far as I know, \textwidth is left unchanged because setting it to \textheight could interfere with other packages or user commands. But changing \textheight also has this potential, and indeed it breaks, for example, showframe and scrlayer. Thus it would be best if \textheight too remained unchanged. scrhack uses the xpatch package (see [Gre12]) to modify the \landscape environment's start macro appropriately.

If you do not want this modification, or if it causes problems, you can deactivate it with the lscape=false option. Note that changing lscape subsequently with \KOMAoption or \KOMAoptions has an effect only if it was not false while loading lscape.

Incidentally, the pdflscape package also uses lscape, so scrhack affects the functioning of this package too.

16.7. Special Case for nomencl

v3.23 The hack for the nomencl represents a somewhat special case. First, it ensures that the optional table of contents entry for the nomenclature observes the toc=indentunnumbered option. Somewhat incidentally, it also reserves the file extensions nlo and nls for the owner nomencl (see \addtotoclist, section 15.1, page 375).

It also changes the thenomenclature environment to use \tocbasic@listhead for the heading (see section 15.4, page 401). In this way, the hack lets you set various attribute for the nls extension using \setuptoc. For example, not only can you enter the nomenclature in the table of contents with \setuptoc{nls}{numbered}, but you can also number it immediately. You can find more about \setuptoc and its available settings in section 15.2, starting on page 383. As a small but important side effect, this patch also gives the nomenclature also matching running head, if automatic running heads are activated, for example by using the headings page style.

This rather simple patch is an example of how packages that do not define floating environments can still benefit from tocbasic. However, you do not want this change, or if it causes problems, you can deactivate it with the nomencl=false option. The setting of the option when nomencl is loaded is crucial! Later changes to the option with \KOMAoption or \KOMAoptions have no influence and will lead to a corresponding warning.

16.8. Special Case for Section Headings

v3.27 Various packages assume that the sectioning commands are defined in a specific way, corresponding to the definitions in the standard classes. But for some classes this is not the

case. For example, the KOMA-Script classes use a completely different implementation to provide many additional features. But this can cause problems for packages that depend on the definition of the standard classes. As of version 3.27, scrhack offers the option to force the sectioning commands \part, \chapter, \section, \subsection, \subsubsection, \paragraph, and \subparagraph to be compatible with those in the standard classes. When \chapter is defined, the definitions are based on those in book. When \chapter is undefined, the definitions of article are used.

If you are using a KOMA-Script class, several features of these classes are also deactivated as side effect. For example, the commands to define or modify sectioning commands described in section 21.8 or option headings are no longer available, and commands like \partformat have different defaults.

Because this hack has the potential to do more harm than good, it issues several warnings. Also it is not activated simply by loading the scrhack package. If you want to use it, you must explicitly activate it with the standardsections option when you load the package. Late activation or deactivation is not supported.

Since there are often less invasive solutions to fix the problem of package incompatibilities, using this hack is not recommended. It is provided only as a last resort for emergencies.

v3.12

Defining Layers and Page Styles with scrlayer

Most users of graphics software are already familiar with the layer model for a page. Such a model is rather alien to LaTeX itself, but some packages, like eso-pic or textpos, provide a kind of background or foreground layer. scrlayer is another package that provides such background and foreground layers, but unlike the packages mentioned above, these layers are part of the page style. As a result, you can switch between different layers simply by switching the page style.

To do so, the package also supports a low-level interface for defining page styles that are based on a stack or list of layers, for adding layers to such a layer stack, either at the top or the bottom, or before or after a specific layer, and for removing individual layers and duplicates from a stack. In a nutshell, the page style interface of scrlayer provides commands to define page styles based on a stack of layers and to manage those stacks.

Nevertheless, using the layers directly is recommended for advanced users only. Interfaces for beginners and average users are provided by additional packages that in turn load scrlayer. See chapter 5 in part I of this manual.

17.1. Early or Late Selection of Options

The information in section 2.4 applies equally to this chapter. So if you have already read and understood section 2.4, you can skip ahead to section 17.2, page 417.

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

LaTeX allows users to pass class options as a comma-separated list of keywords in the optional argument to \documentclass. In addition to being passed to the class, these options are also passed on to all packages that can understand them. Users can also pass a similar comma-separated list of keywords in the optional argument of \usepackage. KOMA-Script extends the option mechanism for the KOMA-Script classes and some packages with further options. Thus most KOMA-Script options can also take a value, so an option does not necessarily take the form *option*, but can also take the form *option=value*. Except for this difference, \documentclass and \usepackage in KOMA-Script function as described in [Tea05b] or any introduction to LaTeX, for example [OPHS11].

v3.00

Setting the options with \documentclass has one major disadvantage: unlike the interface described below, the options in \documentclass are not robust. So commands, lengths, counters, and similar constructs may break inside the optional argument of this command. For example, with many non-KOMA-Script classes, using a LaTeX length in the value of an option results in an error before the value is passed to a KOMA-Script package and it can take control of the option execution. So if you want to use a LaTeX length, counter, or command

as part of the value of an option, you should use \KOMAoptions or \KOMAoption. These commands will be described next.

\KOMAoptions{*option list*}
\KOMAoption{*option*}{*value list*}

v3.00

KOMA-Script also provides the ability to change the values of most class and package options even after loading the class or package. You can use the \KOMAoptions command to change the values of a list of options, as in \documentclass or \usepackage. Each option in the *option list* has the form *option=value*.

Some options also have a default value. If you do not specify a value, that is if you give the option simply as *option*, then this default value will be used.

Some options can have several values simultaneously. For such options, it is possible, with the help of \KOMAoption, to pass a list of values to a single *option*. The individual values are given as a comma-separated *value list*.

KOMA-Script uses the commands \FamilyOptions and \FamilyOption with the family "KOMA" to implement this ability. See part II, section 12.2, page 336.

Options set with \KOMAoptions or \KOMAoption will reach both the KOMA-Script class and any previously loaded KOMA-Script packages that recognise these options. If an option or a value is unknown, scrbase will report it as an error.

17.2. Generic Information

The package needs some generic information about the class being used. Class authors can help scrlayer by providing the appropriate information. Otherwise the package will try to determine this information for itself. This works, for example, for the standard and the KOMA-Script classes. It may work with other classes, or it may fail in whole or in part.

This section describes some of the information that class authors can provide. Normally, users should not have to worry about it.

\if@chapter *then code* \else *else code* \fi

If \if@chapter is defined and corresponds to \iftrue, scrlayer takes the chapter level into account when, for example, processing the automark option. If it is defined but is not \iftrue, scrlayer handles only the \part, \section, \subsection, \sub...subsection, \paragraph, \subparagraph, \sub...subparagraph sectioning units. If the macro is undefined, scrlayer searches for \chapter. If \chapter is defined and does not correspond to \relax, scrlayer defines \if@chapter to \iftrue. Otherwise \if@chapter becomes \iffalse.

\if@mainmatter *then code* \else *else code* \fi

Classes like book or scrbook provide \frontmatter, \mainmatter, and \backmatter to switch between the front, main, and end parts of a book. Typically, these classes also use \if@mainmatter internally to decide whether the current text is part of the main body of the document or not. Classes like report and article have no \frontmatter, \mainmatter, or \backmatter and therefore also lack \if@mainmatter.

It is easier for scrlayer to avoid constantly testing for the existence of these commands to decide whether to work in the main matter or not, and instead to use \if@mainmatter with classes like report and article. So if \if@mainmatter is not defined, scrlayer defines it as a synonym for \iftrue.

Some classes, however, define \frontmatter, \mainmatter, or \backmatter but not \if@mainmatter. In this case, scrlayer also defines \if@mainmatter to be \iftrue and it extends definition of \frontmatter, \mainmatter, and \backmatter to set \if@mainmatter properly. However, if there are other, comparable commands for switching between different document parts. scrlayer will not recognize them, does not test for them, and therefore cannot extend them appropriately. In this case, scrlayer needs help of the class author to set \if@mainmatter correctly.

\DeclareSectionNumberDepth{*level name*}{*level depth*}

Generally each section level is assigned an integer number indicating its depth in the document structure. LaTeX needs this to manage the hierarchy of the section levels. But normally these values are known only to the particular class that defines the section commands. This class then uses the appropriate numbers inside the corresponding commands.

The scrlayer package also needs information about the section hierarchy. With the help of \DeclareSectionNumberDepth, you can map the name of a heading level to a corresponding *level depth*. For example, for the standard book class, the *level name*s would be part, chapter, section, subsection, subsubsection, paragraph, and subparagraph, and the corresponding *level depth*s would be -1, 0, 1, 2, 3, 4, and 5.

The scrlayer package tries to determine the *level depth*s on its own while loading and again during \begin{document}. But if this fails, for example because you use completely different sectioning commands, you can define the relationship explicitly with \DeclareSectionNumberDepth.

17.3. Declaring Layers

A layer is an abstract model of a page. Unlike a real, physical sheet of paper, this page is completely transparent. Typically, multiple layers are stacked atop one another and opaque material on one layer hides material on the layers below. The stack of layers is then merged to form the physical page. The scrlayer package provides two such stacks for each page: a background stack and a foreground stack. The background stack is printed beneath the

normal page content, while the foreground stack is printed above it. The normal page content is therefore a kind of a dividing layer between the two layer stacks.

A layer has several attributes that can be understood as answers to some basic questions:

Does the layer belong to the foreground or the background? Background layers are output before the normal content of the page. Therefore they appear behind or beneath the normal content of the page. Foreground layers are then output after the normal content. Therefore they appear on top of the normal content of the page. By default, a layer is both a background layer *and* a foreground layer and therefore is printed twice. Usually it makes sense to limit the layer explicitly to either the foreground or the background.

What is the position of the layer? To answer this question, there are attributes to define the layer's horizontal and vertical position.

How big is the layer? As with the position, there are also attributes to define the width and height of a layer. Thus a layer can be smaller or larger than the paper and it can be placed at different positions on the paper.

How are the horizontal and vertical positions measured? This question is answered by the alignment attribute. The horizontal position can be measured from the left edge of the paper to the left edge of the layer, to the centre of the layer, to the right edge of the layer. Similarly, the vertical position can be measured from the top edge of the paper to the top edge of the layer, the centre of the layer, or the bottom edge of the layer.

Is the layer intended for text or picture output? This question is closely related to the position. For example, users often expect the origin for a picture to be at the lower left corner of the layer. But this would not be suitable for text output. Therefore the origin of a text layer is the height of a standard text line below the top left corner of the layer. Picture layers, on the other hand, create a `picture` environment in which additional positioning commands are available.

Should the layer be printed on left or right pages? By default a layer will be printed on all pages. Note that LaTeX treats even pages as left-side pages and odd pages as right-side pages, but in one-sided printing, only right-hand pages exist regardless of the page number.

Should the layer be printed in one-sided or two-sided mode? By default a layer will be printed in both one-sided and two-sided printing. Nonetheless, a layer that is restricted to even pages will never be printed in one-sided printing and therefore is not a one-sided layer.

Should the layer be printed on float pages or normal pages? LaTeX produces float pages for objects from environments like tables or figures if they are allowed to appear on

a page without normal page contents (see option `p` for `figure` or `table`). In effect, the entire page is allowed to float within the document. Normal pages in this sense are all pages that are not float pages. Normal pages can also contain floats at the top, in the middle, or at the bottom of the page. Very large floats can give the impression of being page floats, while in reality they are floating environments placed at the top of a normal page.

What are the contents of the layer? The corresponding attribute simply contains what should be printed whenever the layer is output.

These eight questions immediately give rise to a number of attributes. Later in this guide we will describe additional attributes. However, they are only defined for convenience and can be expressed by a combination of these primary attributes.

```
\DeclareLayer[option list]{layer name}
\DeclareNewLayer[option list]{layer name}
\ProvideLayer[option list]{layer name}
\RedeclareLayer[option list]{layer name}
\ModifyLayer[option list]{layer name}
```

These commands can be used to define layers. The *layer name* must be fully expandable and should expand to ASCII letters only. Some additional characters are accepted, but their use is only recommended for advanced users.

The `\DeclareLayer` command does not care whether or not a layer with the given *layer name* already exists. It will under all circumstances define the layer with the attributes specified in the *option list*. An *option* can be either a key or a key followed by an equal sign and a value. Multiple options are separated by commas. To use a comma or a white space within the value of an option, you must put the value inside curly brackets. See table 17.1 for more information on keys, values, and the corresponding attributes.

Unlike `\DeclareLayer`, `\DeclareNewLayer` reports an error if a layer with the same *layer name* already exists. This prevents the user from accidentally using the same *layer name* more than once. This is especially useful when classes or packages also define layers internally.

In contrast, `\ProvideLayer` only defines a layer if there is no layer with the same name already. If the name is in use for another layer, the new definition is ignored. Therefore this command has the meaning, *define the layer only if it does not already exist.*

If you want to redefine an existing layer, you can use `\RedeclareLayer` or `\ModifyLayer`. With `\RedeclareLayer`, the layer is first reset to the default settings and then completely redefined via the specified options list. In contrast, `\ModifyLayer` does not reset the layer. Only those attributes which are explicitly set in the *option list* will be changed. Applying either command to a previously undefined *layer name* results in an error.

Table 17.1.: Options for defining page layers and the meaning of the corresponding layer attribute

v3.16 `addcontents=`*`Code`*

The specified *`code`* will be appended to the current value of the `contents` attribute, so the new content will appear at the end of the existing content. For more information about the handling of *`code`* see the `contents` option.

v3.16 `addheight=`*`additional height`*

The current value of the `height` attribute will be increased by the value of this option. You can use the same kind of values as for `height`.

v3.16 `addhoffset=`*`additional horizontal offset`*

The current value of the `hoffset` attribute will be increased by the value of this option. You can use the same kind of values as for `hoffset`.

v3.16 `addvoffset=`*`additional vertical offset`*

The current value of the `voffset` attribute will be increased by the value of this option. You can use the same kind of values as for `voffset`.

v3.16 `addwidth=`*`additional width`*

The current value of the `width` attribute will be increased by the value of this option. You can use the same kind of values as for `width`.

...

Table 17.1.: Options for defining layers (*continued*)

`align=`*`alignment characters`*

The *`alignment characters`* define the desired alignment of the layer. Each *`alignment character`* determines how the *`length`* of either the `hoffset` or the `voffset` option is interpreted. Multiple *`alignment character`*s can be used together (without spaces or commas) and are evaluated in the order they occur. Macros are not permitted in the value of the option. Valid *`alignment character`*s are:

`b` – align the layer at its bottom edge; the value of `voffset` is the distance from the top edge of the paper to the bottom of the layer.

`c` – align the layer at its centre; the values of `voffset` and `hoffset` are interpreted as the distance from the top left corner of the paper to the centre of the layer.

`l` – align the layer at its left edge; the value of `hoffset` is interpreted as the distance from the left edge of the paper to the left edge of the layer.

`r` – align the layer at its right edge; the value of `hoffset` is interpreted as the distance from the left edge of the paper to the right edge of the layer.

`t` – align the layer at its top edge; the value of `voffset` is interpreted as the distance from the top edge of the paper to the top edge of the layer.

`area={`*`hoffset`*`}{`*`voffset`*`}{`*`width`*`}{`*`height`*`}`

This composite option sets the primary attributes `hoffset=horizontal offset`, `voffset=vertical offset`, `width=width`, `height=height`.

v3.18 `backandforeground`

This option removes the restriction of a layer to the foreground or the background. In general, this option makes little sense, but it is provided for the sake of a complete user interface. This option does not expect or allow a value.

`background`

This option displays the layer in the background only. The default is to display layers in both the background and the foreground. This option does not expect or allow a value.

...

Table 17.1.: Options for defining layers (*continued*)

`bottommargin`
: This composite option sets the primary attributes `hoffset`, `voffset`, `width`, `height`, and `align` so that the layer spans the paper horizontally from the left edge to the right and vertically from immediately beneath the footer to the bottom edge of the paper.

`clone=`*`layer name`*
: This composite option sets all primary attributes of the layer to the same values as the primary attributes of the layer with the given *`layer name`*. See the notes on the *`layer name`* at the beginning of the explanation for `\DeclareLayer`. The layer to be cloned must also already exist.

`contents=`*`code`*
: The specified *`code`* will be expanded whenever the layer is printed. This *`code`* defines what you see for the layer. No tests are made to see if the code is valid. Errors in the *`code`* can therefore lead to multiple error messages on each page that prints the layer.

`evenpage`
: This option causes the layer to appear on even pages only, unlike the default, where a layer will appear on both even and odd pages. Since even pages only appear in two-sided printing, this option implies `twoside`. This option does not expect or allow a value.

v3.18 `everypage`
: This option is a combination of `oddorevenpage` and `floatornonfloatpage`. This option does not expect or allow a value.

`everyside`
: This option removes any restriction of the layer to one-sided or two-sided printing. This is the default setting. This option does not expect or allow a value.

v3.18 `floatornonfloatpage`
: This option removes any restriction of the layer to float or non-float pages and restores the default setting. This option does not expect or allow a value.

`floatpage`
: This option restricts the layer to appearing on float pages only. The default setting is for layers to appear on both float and non-float pages. This option does not expect or allow a value.

...

Table 17.1.: Options for defining layers (*continued*)

`foot`
: This composite option sets `hoffset`, `voffset`, `width`, `height`, and `align` so that the layer spans the page footer over the width of the text area. This option does not expect or allow a value.

`footskip`
: This composite option sets `hoffset`, `voffset`, `width`, `height`, and `align` so that the layer spans the vertical distance between the text area and the page footer over the width of the text area. Note, however, that although the height of this area depends on `\footskip`, it is not the same. This option does not expect or allow a value.

`foreground`
: This option displays the layer in the foreground only. The default is to display layers in both the background and the foreground. The option does not expect or allow a value.

`head`
: This composite option sets `hoffset`, `voffset`, `width`, `height`, and `align` so that the layer spans the vertical area of the page header for the width of the text area. The height corresponds to the length `\headheight`. This option does not expect or allow a value.

`headsep`
: This composite option sets `hoffset`, `voffset`, `width`, `height`, and `align` so that the layer spans the vertical distance between the page header and the text area for the width of the text area. The height corresponds to the length `\headsep`. This option does not expect or allow a value.

`height=`*`length`*
: Sets the height of the layer. Note that the *`length`* can be a LaTeX length declared with `\newlength`, a TeX length declared with `\newdimen` or `\newskip`, a length value like 10 pt, or a dimensional expression using +, -, /, *, (, and). For more information about valid dimensional expressions see [Tea98, section 3.5].

`hoffset=`*`length`*
: Sets the distance of the layer from the left edge of the paper. How the distance is measured depends on the `align` option. See the `height` option for more information about valid expressions for *`length`*.

...

Table 17.1.: Options for defining layers (*continued*)

`innermargin`
: This composite option sets `hoffset`, `voffset`, `width`, `height`, and `align` so that the layer spans the inner margin, from the edge of the page to the edge of the text area and from the top to the bottom of the page. In one-sided printing, the inner margin corresponds to the left margin. This option does not expect or allow a value.

`leftmargin`
: This composite option sets `hoffset`, `voffset`, `width`, `height`, and `align` so that the layer spans the left margin, from the left edge of the paper to the left edge of the text area and from the top of the paper to the bottom. This option does not expect or allow a value.

v3.19 `mode=`*`mode`*
: This primary option defines the *`mode`* in which the layer's content is output. The default is `mode=text`. The baseline of the first text line is placed the height of one standard text line below the top edge of the layer, so the text is usually neatly aligned with the top of the layer. In `picture` *`mode`*, on the other hand, the layer spans a `picture` environment with the origin at the bottom left corner of the layer. The `raw` *`mode`* is also also defined. By default it corresponds to `text` *`mode`*. Changing the *`mode`* of a layer usually results the contents shifting. Furthermore, the `picture` *`mode`* provides additional commands that result in errors with another *`mode`*. Therefore it usually makes no sense to change the *`mode`* of a layer after its initial declaration!

`nonfloatpage`
: This option restricts the layer to pages that are not float pages. The default is for the layer to appear on both float and non-float pages. This option does not expect or allow a value.

v3.18 `oddorevenpage`
: This option removes any restriction of the layer to odd or even pages, restoring the default behaviour that the layer should appear on both odd and even pages. The option does not expect or allow a value.

`oddpage`
: This option restricts the layer to odd pages only. The default is for layers to appear on both odd pages and on even pages. Note that in one-sided printing, all pages are odd, regardless of the page number. This option does not expect or allow a value.

...

Table 17.1.: Options for defining layers (*continued*)

`oneside`
: This option restricts the layer to one-sided printing only. The default is for layers to appear in both one-sided and two-sided printing. This option does not expect or allow a value.

`outermargin`
: The composite option sets `hoffset`, `voffset`, `width`, `height`, and `align` so that the layer spans the outer margin of the page, from the top to the bottom of the paper. The outer margin corresponds to the right margin in one-sided printing. This option does not expect or allow a value.

`page`
: This composite option sets `hoffset`, `voffset`, `width`, `height`, and `align` so that the layer spans the the whole paper. This option does not expect or allow a value.

v3.16 `pretocontents=`*`code`*
: The value of *`code`* is prefixed the current value of the `contents` attribute, so the new content will appear before the existing content. For more information about the handling of *`code`*, see the `contents` option.

`rightmargin`
: This composite option sets `hoffset`, `voffset`, `width`, `height`, and `align` so that the layer spans the right margin, from the right edge of the text area to the right edge of the paper and from the top to the bottom edge of the paper. This option does not expect or allow a value.

`textarea`
: This composite option sets `hoffset`, `voffset`, `width`, `height`, and `align` so that the layer spans the entire text area. This option does not expect or allow a value.

`topmargin`
: This composite option sets `hoffset`, `voffset`, `width`, `height`, and `align` so that the layer spans the vertical distance between the top edge of the paper to the header for the entire width of the paper. This option does not expect or allow a value.

`twoside`
: This option restricts the layer to two-sided printing. The default is for layers to appear in both one-sided and two-sided printing. This option does not expect or allow a value.

...

Table 17.1.: Options for defining layers (*continued*)

v3.18 `unrestricted`
: This option removes all output restrictions. It is a combination of `backandforeground`, `everyside`, and `floatornonfloatpage`. This option does not expect or allow a value.

`voffset=`*`length`*
: Sets the distance of the layer from the top of the paper. How the distance is measured depends on the `align` option. See the `height` option for more information about valid expressions for *`length`*.

`width=`*`length`*
: Sets the width of the layer. See the `height` option for more information about valid expressions for *`length`*.

`\ModifyLayers[`*`option list`*`]{`*`layer list`*`}`

v3.26 This command executes `\ModifyLayer` with the given *`option list`* for all layers of the comma-separated layer list. Therefore the option can be used to change attributes of several layers simultaneously.

```
\layerhalign
\layervalign
\layerxoffset
\layeryoffset
\layerwidth
\layerheight
```

These commands are only valid in the *`code`* specified with `contents`, `addcontents`, or `pretocontents`. In this case, they contain the layer's actual alignment, position, and dimensions that will be used for the output. However, this is not necessarily the actual dimensions of the layer's contents, e. g., if the contents are oversized or do not fill the layer completely.

v3.19 The primary layer attribute `align` is mapped to `\layerhalign` and `\layervalign`. The horizontal values `l` and `r` are copied to `\layerhalign`. The vertical values `t` and `b` are copied to `\layervalign`. The value `c`, which is both horizontal and vertical, is copied to both commands. If there are several conflicting values for `align`, only the last one is copied. Thus the resulting `\layerhalign` is either `l`, `c`, or `r`, and the resulting `\layervalign` is either `t`, `c`, or `b`.

Redefining these instructions to change the values stored in them is not permitted, as it would lead to unpredictable results.

\LenToUnit{*length*}

v3.19 This command originally came from eso-pic 2.0f. It converts lengths into multiples of \unit length and can therefore be used everywhere LaTeX expects picture coordinates or \unit length-dependent values. For more information, see [Nie15] and the descriptions of \putUR, \putLL, and \putLR below. If the command is already defined, e. g., by loading eso-pic before scrlayer, the package does not define it again.

\putUL{*content*}
\putUR{*content*}
\putLL{*content*}
\putLR{*content*}
\putC{*content*}

v3.19 You can use these commands inside the value of contents layer option if the layer is declared with mode=picture. In this case, \putUL places the *content* relative to the upper left corner of the layer and therefore is the same as \put(0,\LenToUnit{\layerheight}). \putUR places the *content* relative to the upper right corner of the layer and therefore is the same as \put(\LenToUnit{\layerwidth},\LenToUnit{\layerheight}). \putLL places the *content* relative to the lower left corner of the layer and therefore is the same as \put(0,0). \putLR places the *content* relative to the lower right corner and therefore is the same as \put(\LenToUnit{\layerwidth},0). Last but not least, \putC places the *content* relative to the centre of the layer.

Example: You want to determine exactly how accurately DIV=classic sets the height of the text area to the width of the page size. You declare a layer that both borders the text area and places a circle with a diameter of the paper width in the centre of the text area:

```
\documentclass[DIV=classic]{scrartcl}
\usepackage{pict2e}
\usepackage{scrlayer}
\DeclareNewLayer[%
  textarea,background,mode=picture,
  contents={%
    \putLL{\line(1,0){\LenToUnit{\layerwidth}}}%
    \putLR{\line(0,1){\LenToUnit{\layerheight}}}%
    \putUR{\line(-1,0){\LenToUnit{\layerwidth}}}%
    \putUL{\line(0,-1){\LenToUnit{\layerheight}}}%
    \putC{\circle{\LenToUnit{\paperwidth}}}%
  }
]{showtextarea}
\DeclareNewPageStyleByLayers{test}{showtextarea}
\pagestyle{test}
```

```
\begin{document}
\null
\end{document}
```

You will notice that typearea's mapping to an integer *DIV* value is very accurate in this example.

Incidentally, you will find more information about late medieval book-page canon sketched in the example in section 2.3, page 31.

The `\DeclareNewPageStyleByLayers` command just used in the example defines a new page style using the newly declared layer. It will be explained in section 17.4, page 432.

`\GetLayerContents{`*layer name*`}`

v3.16

This command returns the whole contents of a layer. It is important to note that using this command inside the *code* of the layer attributes `contents`, `addcontents`, or `pretocontents` can result in infinite recursion when referencing the contents of the current layer. You are responsible for avoiding such situations!

`\IfLayerExists{`*layer name*`}{`*then-code*`}{`*else-code*`}`

This command can be used to execute code depending on whether or not a layer with the specified *layer name* exists. If the layer exists, the *then-code* will be executed; otherwise, the *else-code* will be executed. Note that the command cannot really test whether a layer exists. Instead it uses a heuristic which will never yield a false negative but which, in extreme cases, could yield a false positive. False positives indicate a problem, such as the use of an incompatible package or a bad choice of internal macro names by the user.

`\DestroyLayer{`*layer name*`}`

If a layer with the given *layer name* exists, all macros belonging to it will be set to `\relax`. In page styles already defined with scrlayer, such destroyed layers are ignored. Destroyed layers can be redefined using `\DeclareNewLayer` or `\ProvideLayer`, but they can no longer be changed using `\RedeclareLayer` or `\ModifyLayer`.

`draft=`*simple switch*
`\layercontentsmeasure`

The KOMA-Script option `draft` can be used to active the draft mode of scrlayer. The draft mode uses the `\layercontentsmeasure` command to print a ruler at each layer edge, of which the top and left one is labelled in centimetres and the right and bottom one in inches. The rulers are drawn behind the content of each layer. This command can also be used as exclusive content of a layer.

17.4. Declaring and Managing Page Styles

We now understand how to define and manage layers, but so far we do not know how they are used. The possibly surprising answer is, with page styles. In LaTeX, page styles usually define headers and footers of the page.

The header and footer of odd or right-hand pages are printed on pages with odd page numbers in two-sided printing or on all pages in one-sided printing. This is directly comparable to the layer options `oddpage` and `evenpage`.

The page header is output before the main page contents. The page footer is output after the main page contents. This corresponds directly to the layer options `background` and `foreground`.

Therefore it makes sense to define page styles as lists of layers. But instead of just the four options mentioned above, you can use all the properties explained in section 17.3, table 17.1, on page 421.

As a result of these considerations, layer page styles are one type of page style that scrlayer provides. A layer page style consists of layers as well as several hooks. The layers have already been described in section 17.3. The hooks are points in the expansion or execution of page styles to which you can add code. Experienced users already know this concept from commands like `\AtBeginDocument` (see [Tea05b]) or `\BeforeClosingMainAux` (see page 363).

Page-style aliases are another type of page style which scrlayer provides. A page-style alias is actually a different page style. In other words, the name of a page-style alias is an alternative name for another page-style alias or for a page style. As a result, manipulating a page-style alias results in manipulating the original page style. If the original page style is itself an alias, its manipulation again results in the manipulation of the alias's original page style, and so on until a real page style is finally changed. The term *real page style* is used to distinguish it from a page-style alias. All page styles that are not page-style aliases are real page styles. Aliases can be defined for all page styles, not just those defined with scrlayer.

`\currentpagestyle`
`\toplevelpagestyle`

The scrlayer package patches the `\pagestyle` command so that it sets `\currentpagestyle` to the currently active page style. Note that `\thispagestyle` does not change `\currentpagestyle`. If you use `\thispagestyle`, however, `\currentpagestyle` can change within the LaTeX output routine. However, this change will only occur if `\currentpagestyle` has been actively protected from expansion until the execution of the output routine.

Note that the layer page styles described later in this section do not rely on this `\pagestyle` extension because they redefine `\currentpagestyle` themselves. This patch was made so that other, non-scrlayer page styles can use `\currentpagestyle`. Additionally, `\currentpagestyle` is empty after loading scrlayer and before using `\pagestyle` for the first time. Therefore, when

defining an end-user interface, you may find it useful to set the current page style to a default with an implicit \pagestyle statement.

If \pagestyle activates an alias page style, \currentpagestyle shows not the alternative name but the name of the real page style. You can get the alternative name using \toplevelpagestyle. However, it is not recommended to define page styles that produce different results depending on \toplevelpagestyle, e. g., using \Ifstr. If you were to activate such a page style using \thispagestyle you could get the wrong result.

v3.16

\BeforeSelectAnyPageStyle{*code*}
\AfterSelectAnyPageStyle{*code*}

The \BeforeSelectAnyPageStyle command adds *code* to the hook that will be executed inside \pagestyle, just before the page style is selected. Within the *code*, you can use #1 as a place-holder for the argument of \pagestyle.

The \AfterSelectAnyPageStyle command works similarly, but the *code* will be executed just after the page style has been selected and after \currentpagestyle has been set to the name of the real page style.

Note that the *code* of both commands is executed only if a page style is selected with \pagestyle and not if it is chosen in a different way, e. g., using \thispagestyle. Note also that you cannot remove *code* from the hook after adding it. But the *code* will be added locally, so you can use a group to limit the scope of the *code*.

\DeclarePageStyleAlias{*page style alias name*}{*original page style name*}
\DeclareNewPageStyleAlias{*page style alias name*}{*original page style name*}
\ProvidePageStyleAlias{*page style alias name*}{*original page style name*}
\RedeclarePageStyleAlias{*page style alias name*}{*original page style name*}

These commands can be used to define a page style with name *page style alias name* that is simply an alias for an existing page style with the name *original page style name*. If there is already a page style *page style alias name*, using \DeclarePageStyleAlias or \RedeclarePageStyleAlias will destroy the alias before recreating it.

\DeclareNewPageStyleAlias will throw an error message if a page style *page style alias name* has already been defined. It does not matter if the previously defined page style is a layer-page style, a page style alias, or another page style.

\ProvidePageStyleAlias will define the alias only if the *page style alias name* page style has not been defined before. If the *page style alias name* page style already exists, nothing will be done.

\RedeclarePageStyleAlias expects an existing *page style alias name* page style. It will destroy that page style and afterwards define the alias. If the *page style alias name* page style does not exist, you will get an error.

\DestroyPageStyleAlias{*page style name*}

This command renders the page style named *page style name* undefined for LaTeX if it is an alias for another page style. Afterwards, the page style can be newly defined with \DeclareNewPageStyleAlias or \ProvidePageStyleAlias.

\GetRealPageStyle{*page style name*}

This command will result in the (recursive) real page name of the page style if *page style name* is an alias of another page style. In all other cases, even if there's no alias and no page style named *page style name*, the result is simply *page style name*. The command is fully expandable and may be used, e.g., in the second argument of \edef.

\DeclarePageStyleByLayers[*option list*]{*page style name*}{*layer list*}
\DeclareNewPageStyleByLayers[*option list*]{*page style name*}{*layer list*}
\ProvidePageStyleByLayers[*option list*]{*page style name*}{*layer list*}
\RedeclarePageStyleByLayers[*option list*]{*page style name*}{*layer list*}

These commands declare a page style named *page style name*. The page style will consist of the layers given in *layer list*, a comma separated list of layer names. Note that the *page style name* and the layer names in the *layer list* must be fully expandable and should expand to letters. Several other characters are also accepted, but their use is recommended only for experienced users.

The *option list* is a comma-separated list of options in the form *key=value*. These options may be used to set additional features. Currently they are used to set the code that should be expanded or executed at several hooks. See the introduction to this section for more general information about hooks. See table 17.2 for detailed information on specific hooks.

Table 17.2.: Hook options for page styles (in order of execution)

onselect=*code*	The *code* of this hook is executed whenever the page style is selected, for example with \pagestyle. Note that \thispagestyle itself does not directly select a page style. In this case, the page style is only activated within LaTeX's output routine.
oninit=*code*	The *code* of this hook is executed whenever the output of the page style's layers is initialised. Note that this happens twice for each page: once for background layers and once for foreground layers.

...

Table 17.2.: Hook options for page styles (*continued*)

`ononeside=`*code*
: The *code* of this hook is executed whenever the output of the page style's layers in one-sided printing is initialised. Note that this happens twice for each page: once for background layers and once for foreground layers.

`ontwoside=`*code*
: The *code* of this hook is executed whenever the output of the page style's layers in two-sided printing is initialised. Note that this happens twice for each page: once for background layers and once for foreground layers.

`onoddpage=`*code*
: The *code* of this hook is executed whenever the output of the page style's layers on an odd page is initialised. Note that this happens twice for each page: once for background layers and once for foreground layers. Note also that in one-sided printing all pages are odd pages, not just those with odd page numbers.

`onevenpage=`*code*
: The *code* of this hook is executed whenever the output of the page style's layers on an even page is initialised. Note that this happens twice for each page: once for background layers and once for foreground layers. Note also that in one-sided printing there are no even pages. Instead, all pages are treated as odd pages, not just those with odd page numbers.

`onfloatpage=`*code*
: The *code* of this hook is executed whenever the output of the page style's layers on a float page is initialised. Note that this happens twice for each page: once for background layers and once for foreground layers. Note also that float pages are only those pages with one or more p-placed floating environments.

`onnonfloatpage=`*code*
: The *code* of this hook is executed whenever the output of the page style's layers on a non-float page is initialised. Note that this happens twice for each page: once for background layers and once for foreground layers. Note also that float pages are only the pages on which one or more p-placed floating environments are output. Other pages may well have t-, h-, or b-placed floating environments.

`onbackground=`*code*
: The *code* of this hook is executed whenever the output of the page style's background layers is initialised. Note that this happens once each page.

...

Table 17.2.: Hook options for page styles (*continued*)

onforeground=*code*	The *code* of this hook is executed whenever the output of the page style's foreground layers is initialised. Note that this happens once each page.

The \DeclarePageStyleByLayers command defines the page style regardless of whether a page style of *page style name* already exists. If necessary, the existing page style is completely redefined. However, if a page style alias *page style name* already exists, the associated real page style, not the page style alias itself, is redefined (see \GetRealPageStyle earlier in this section).

The \DeclareNewPageStyleByLayers command differs if a page style of the same name already exists. Regardless of whether it is a real or an alias page style, an error will be reported in this case.

In contrast, \ProvidePageStyleByLayers preserves the page style unchanged if a page style with name *page style name* already exists. If no such page style exists, it is defined as in \DeclarePageStyleByLayers.

The \RedeclarePageStyleByLayers in turn expects that a page style of of the name *page style name* already exists and then redefines its real page style. However, if no page style of the specified name exists, an error message results.

Also note the remarks below about the pseudo-page style @everystyle@.

```
\pagestyle{@everystyle@}
\pagestyle{empty}
```

The scrlayer package defines two specific page styles worth noting. The first of these is @everystyle@. This page style should never be used like a normal page style, for example with \pagestyle or \thispagestyle, or as the target of a page-style alias. Instead, the layers and hooks of this page style are used by all other layer page styles. The hooks of @everystyle@ are placed in front of their respective hooks, and the layers in front of the respective layers, of the active page style.

Thus adding a layer to the pseudo-page style @everystyle@ or code to a hook of this page style is like adding a layer or hook code to all layer page styles, even the empty one. There's one crucial difference: commands that reference the layers of a page style which include \ForEachLayerOfPageStyle, e. g., the commands \AddLayersToPageStyleBeforeLayer or \AddLayersToPageStyleAfterLayer, disregard the layers of the page style @everystyle@ when applied to a different layer page style.

The second, slightly different page style is empty. Normally the LaTeX kernel defines the empty page style to have an empty header and footer. The scrlayer package redefines this page style as a layerless page style. Nonetheless, you can use it like any other layer page style. The

main advantage of the layer page style over the original page style from the LaTeX kernel is that it also executes the hooks and layers of the psuedo-layer page style `@everystyle@`.

```
onpsselect=code
onpsinit=code
onpsoneside=code
onpstwoside=code
onpsoddpage=code
onpsevenpage=code
onpsfloatpage=code
onpsnonfloatpage=code
onpsbackground=code
onpsforeground=code
```

There is also a KOMA-Script option for each of the hooks in table 17.2. The names of the KOMA-Script options are similar to those for declaring page style layers, except that a "ps" is inserted after "on" at the beginning of the name . The values of these KOMA-Script options are used as the initial values for the corresponding hooks. These initial values are then extended by all the values assigned to the corresponding hooks in the *option list* of the declaration commands. You can remove the default with `\ModifyLayerPageStyleOptions`, which is explained later in this section.

```
singlespacing=simple switch
```

v3.24

If a document is printed with increased line spacing, e. g., using the setspace package, it is often undesirable for the header and footer to be printed with this increased spacing. This is particularly true if the header and footer consist of only a single line. In this case you can use KOMA-Script option `singlespacing`. However, the default is `singlespacing=false`! The option generally applies to all layer page styles. If you want single-spacing for only some page styles, use `oninit=\linespread{1}\selectfont` for those page styles.

```
deactivatepagestylelayers=simple switch
\ForEachLayerOfPageStyle{page style name}{code}
\ForEachLayerOfPageStyle*{page style name}{code}
```

As long as the KOMA-Script option `deactivatepagestylelayers` is not enabled, the `\ForEachLayerOfPageStyle` command can execute arbitrary *code* for each layer of the page style named *page style name*. Inside of *code*, the place holder `#1` serves as a reference to the name of the current layer.

Example: If you want to print the names of all layers of the `scrheadings` page style as a comma-separated list, you can use:

```
\let\commaatlist\empty
\ForEachLayerOfPageStyle{scrheadings}{%
  \commaatlist#1\gdef\commaatlist{, }}
```

In the example above, we had to use `\gdef` instead of `\def` because `\ForEachLayerOfPageStyle` executes the *code* inside a group to minimise side effects. However, `\gdef` redefines `\commaatlist` globally so the change will persist when the *code* for the next layer is executed.

v3.18 Alternatively, you can use `\def` with the starred variant `\ForEachLayerOfPageStyle*`. This form dispenses with the extra group when executing *code*. However, the user then has to take make sure that the *code* has no unwanted side effects. In particular, deactivating all layers using `deactivatepagestylelayers=true` within *code* would persist beyond the call to `\ForEachLayerOfPageStyle*`.

Various scrlayer commands also use `\ForEachLayerOfPageStyle` internally. Their behaviour can therefore also be changed using the KOMA-Script option `deactivatepagestylelayers`. With this option, you can temporarily disable and thus hide all layers of all layer page styles.

```
\AddLayersToPageStyle{page style name}{layer list}
\AddLayersAtBeginOfPageStyle{page style name}{layer list}
\AddLayersAtEndOfPageStyle{page style name}{layer list}
\RemoveLayersFromPageStyle{page style name}{layer list}
```

You can use these commands to add layers to a layer page style or to remove layers from a layer page style. The page style is referenced by its *page style name*. The layers are given by a comma-separated *layer list*.

Both the `\AddLayersToPageStyle` and the `\AddLayersAtEndOfPageStyle` commands insert the new layers at the bottom of the page style's layer list. Logically, the newly added layers lie above or in front of the old layers. Background layers, of course, still are logically behind the text layer and thus behind all foreground layers.

In contrast, the `\AddLayersAtBeginOfPageStyle` command inserts the new layers at the top of the page style's layer list. The layers are separately inserted to the top of the list in the same order they appear in the *layer list* parameter. This means that the last layer in the *layer list* parameter will become the first, and thus the lowest, layer of the page style.

The `\RemoveLayersFromPageStyle` command removes layers from the layer list of the page style named *page style name*. Attempting to remove layers which are not part of the page style's layer list will be ignored. But attempting to add layers to or remove layers from a page style that is neither a layer page style nor a page-style alias is an error and will be reported as such.

\AddLayersToPageStyleBeforeLayer{*page style name*}{*layer list*}{*reference layer name*}
\AddLayersToPageStyleAfterLayer{*page style name*}{*layer list*}{*reference layer name*}

These commands are similar to the previous ones. The existing layers of the page style, however, are searched for the *reference layer name*. The layers in the {*layer list*} parameter are then inserted before or after this reference layer. The order of the layers in the *layer list* is retained.

If the reference layer named *reference layer name* is not part of the given page style, nothing is inserted. On the other hand, if the page style is neither a layer page style nor a page-style alias, an error will be reported.

\UnifyLayersAtPageStyle{*page style name*}

The commands for defining page styles or adding layers to a page style do not care how many times a layer appears in a page style, so it is quite permissible for a layer to appear multiple times. In most cases, however, it does not make sense to use the same layer several times in one page style. Therefore, you can use \UnifyLayersAtPageStyle to remove all layer duplicates from the layer page style of the specified {*page style name*}.

Note that this function can change the order of layers. So if you need a specific order, you should remove all layers and add the layers in the order you want instead of using \UnifyLayersAtPageStyle.

\ModifyLayerPageStyleOptions{*page style name*}{*option list*}
\AddToLayerPageStyleOptions{*page style name*}{*option list*}

These two commands can be used to modify the options, and thus the hooks, of a layer page style. With \ModifyLayerPageStyleOptions, only the options in the comma-separated *option list* will be set to the new values given there. The previous values are lost. All options from table 17.2, page 432 are allowed. Options or hooks that are not specified in the *option list* will remain unchanged. If you want to set an option to do nothing you can set it to the value \relax. Setting an option to a new value using \ModifyLayerPageStyleOptions will remove the previous value, including the global default value. This statement is the only way to remove the global defaults of the KOMA-Script options from a page style.

\AddToLayerPageStyleOptions, on the other hand, does not overwrite the previous values. Instead it adds — or, more precisely, concatenates — the new values to old ones.

\IfLayerPageStyleExists{*page style name*}{*then code*}{*else code*}
\IfRealLayerPageStyleExists{*page style name*}{*then code*}{*else code*}

With these commands, you can execute code depending on whether a page style is a layer page style or not. The \IfLayerPageStyleExists command executes the *then code* if *page style name* is the name of a layer page style or a page-style alias that results in a layer page style. Otherwise, the command executes the *else code*. Internally, this command is often used to throw an error message if you use one of the layer page style commands with an *page style name* that does not correspond to a layer page style.

The \IfRealLayerPageStyleExists command goes one step further and executes the *then code* only if the page style specified by *page style name* is itself a layer page style. So if *page style name* is a page-style alias, this command executes the *else code* even if the alias resolves to a real layer page style.

\IfLayerAtPageStyle{*page style name*}{*layer name*}{*then code*}{*else code*}
\IfSomeLayerAtPageStyle{*page style name*}{*layer list*}{*then code*}{*else code*}
\IfLayersAtPageStyle{*page style name*}{*layer list*}{*then code*}{*else code*}

These commands can be used to test whether one or more layers are part of a page style. The \IfLayerAtPageStyle command expects exactly one *layer name* as the second argument. If this layer is part of the page style specified in *page style name*, the *then code* will be executed. Otherwise, the *else code* will be executed.

The \IfSomeLayerAtPageStyle and \IfLayersAtPageStyle commands allow a comma-separated *layer list*. \IfSomeLayerAtPageStyle will execute the *then code* if *at least one* of the layers in the *layer list* parameter is part of the layer list of *page style name*. In contrast, \IfLayersAtPageStyle executes the *then code* only if *all* the layers in the *layer list* parameter are part of the page style. If the condition is not met, the *else code* will be executed.

With suitable nesting of these commands, you can construct complex conditions. If you use only one *layer name* in the *layer list*, all three statements are synonymous.

\DestroyRealLayerPageStyle{*page style name*}

This statement makes a layer page style undefined. If a page-style alias or ordinary page style is specified, the command is ignored.

If *page style name* is the name of the current page style, the current page style will becomes a kind of empty page style. If the page style set with \thispagestyle is *page style name*, it will simply be reset. The previous \thispagestyle command loses its current effect.

Note that the layers of the page style are not destroyed automatically. If you want to destroy the layers too, you can do this with:

```
\ForEachLayerOfPageStyle{...}{\DestroyLayer{#1}}
```

before destroying the layer page style itself.

17.5. Header and Footer Height

The header and footer of a page are central elements not just of a page style. They can also serve to restrict layers using the appropriate options (see table 17.1, page 421). Therefore the heights of these elements must be defined.

The information in section 5.2 applies equally to this chapter. So if you have already read and understood section 5.2, you can skip ahead to section 17.6, page 439.

`\footheight`
`\headheight`

The scrlayer package introduces a new length, `\footheight`, analogous to `\headheight`of the LaTeX kernel. Additionally, scrlayer interprets `\footskip` to be the distance from the last baseline of the text area to the first normal baseline of the footer. The typearea package interprets `\footheight` in the same way, so typearea's options for the footer height can also be used to set the values for the scrlayer package. See the `footheight` and `footlines` options in section 2.6, page 46) and option `footinclude` on page 43 of the same section.

If you do not use the typearea package, you should adjust the header and footer heights using appropriate values for the lengths where necessary. For the header, at least, the geometry package, for example, provides similar settings.

If you choose a header or footer height that is too small for the actual content, scrlayer usually accepts this without issuing an error message or a warning. The header then expands according to its height, usually upwards, with the footer moved down accordingly. Information about this change is not obtained automatically. However, packages like scrlayer-scrpage that build upon scrlayer may contain their own tests which can lead to their own actions (see `\headheight` and `\footheight` on page 254).

17.6. Manipulating Page Styles

Although scrlayer itself does not define concrete page styles with content — the previously mentioned page styles `@everystyle@` and `empty` are initially defined without any layers, i. e., without content —, the package provides some options and commands to manipulate their contents.

```
automark
autooneside=simple switch
manualmark
\automark[section level of the right mark]{section level of the left mark}
\automark*[section level of the right mark]{section level of the left mark}
\manualmark
```

For most classes, the choice of a page style — generally `headings` or `myheading` — determines whether the running heads are created automatically or manually. With scrlayer the distinction is done with the two commands `\automark` and `\manualmark`.

The `\manualmark` command switches to manual marks and deactivates the automatic filling of the marks. In contrast, `\automark` and `\automark*` define which section levels should be used to set the mark automatically. The optional argument is the *section level of the right mark*, the mandatory argument the *section level of the left mark*. The arguments should always be the name of a section level like `part`, `chapter`, `section`, `subsection`, `subsubsection`, `paragraph`, or `subparagraph`.

Normally, the higher level should be used for the left mark and the lower level for the right mark. This is only a convention and not a requirement, but it makes sense.

Please note that not every class provides running heads for every section level. For example, the standard classes never use \part in the heading. The KOMA-Script classes, on the other hand, support all levels.

The difference between `\automark` and `\automark*` is that `\automark` overrides all previous commands to automatically set the mark, while `\automark*` changes only the behaviour of the section levels specified in its arguments. With this feature you can handle relatively complex cases.

Instead of the commands described previously, you can also use the `manualmark` and `automark` options to switch between automatic and manual running heads. `automark` always uses the default

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

for classes with \chapter and

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

for other classes.

In one-sided printing, you normally want only the higher section levels to provide the running title. The default option `autooneside` corresponds to this behaviour. The option accepts the values for simple switches listed in table 2.5, page 42. If you deactivate this option, the optional and mandatory arguments of `\automark` and `\automark*` will again control the running head in one-sided printing.

Note that merely loading the package does not have any effect on whether automatic or manual running heads are used, or what kind of sectioning headings fill the marks. Only by explicitly using the option `automark` or `manualmark`, or the command \automark or \manualmark,

will the conditions here be initialized.

You can find additional background information and examples of how to use these commands in the documentation for the scrlayer package scrlayer-scrpagein section 5.5, starting on page 267.

\MakeMarkcase{*text*}
markcase=*value*

Automatic running heads, but not manual ones, use \MakeMarkcase for their output. If the command has not been defined, e. g., by the class while loading scrlayer, it is defined by default to output the argument *text* without changes. But the default can be changed either by redefining \MakeMarkcase. Using the markcase option with one of the values of table 5.2 also redefines \MakeMarkcase.

Because of the poor typographical quality of the primitive capitalisation routine (see the explanation of markcase in section 5.5, page 270) the author of KOMA-Scriptrecommends that you avoid upper-case typesetting for running heads. This is usually possible with markcase=used. However, some classes insert \MarkUppercase, or even the TEX command \uppercase, into the running heads. For such cases, you can use the option markcase=noupper. This will also deactivate \MakeUppercase and \uppercase inside the running heads.

You can find all valid values for markcase in table 5.2, page 271.

\righttopmark
\rightbotmark
\rightfirstmark
\rightmark
\lefttopmark
\leftbotmark
\leftfirstmark
\leftmark

v3.16

LATEX typically uses a two-part TEX mark for page styles. Running heads can access the left part of that mark with \leftmark and the right part with \rightmark. In fact, it was probably intended to use \leftmark for the running head of left (even) pages and \rightmark for the running head of right (odd) pages of two-sided documents. In one-sided printing, however, the standard classes do not even set the left part of the mark.

TEX itself knows three ways to access a mark. The \botmark is the last valid mark of the most recent page that has been built. If no mark has been set on the page, it corresponds to the last mark set on the pages that have already been shipped out. The LATEX command \leftmark uses precisely this mark, so it returns the left part of the last mark of the page. This corresponds exactly to \leftbotmark. By comparison, \rightbotmark prints the right part of this mark.

\firstmark is the first mark of the last page that has been built. This is the first mark that has been set on the page. If no mark has been set on the page, it corresponds to the last mark of the pages that have already been shipped out. The LaTeX command \rightmark uses precisely this mark, so it returns the right part of the first mark of the page. This corresponds exactly to \rightfirstmark. By comparison, \leftfirstmark prints the left part of this mark.

\topmark is the content that \botmark had before building the current page. LaTeX itself does not use it. Nevertheless, scrlayer provides \lefttopmark to access the left part of this mark and \righttopmark to access the right part.

Note that the left and right portions of the mark can only be set together. Even if you use \markright to change only the right part, the left part will set again (unchanged). Accordingly, in two-sided printing, using the headings page style, the higher section levels always make both parts. For example, \chaptermark uses \markboth with an empty right argument in this case. This is the reason \rightmark or \rightfirstmark always shows an empty value on pages which start a chapter, even if there was a \sectionmark or \section on the same page to make the right part of the mark. Please note that using any of these commands to show the left or right part of the mark as part of the page content may lead to unexpected results. They are really meant for use in the header or footer of a page style only. Therefore, they should always be part of the contents of a layer when using scrlayer. But it does not matter whether the layer is restricted to the background or the foreground, since all layers are shipped out after building the current page.

If you need more information about the mark mechanism of TeX, please have a look at [Knu90, chapter 23]. The topic is flagged there as an issue for real experts. So if the explanation above confused you, please don't worry about it.

\headmark
\pagemark

You can make life easier with \headmark. This extension of scrlayer is a shorthand that resolves to either \leftmark or \rightmark depending on whether the current page is even or odd.

The \pagemark command has nothing to do with TeX's mark mechanism. It is serves to output a formatted page number. The font of element pagenumber will be used for the output. This can be changed using the \setkomafont or \addtokomafont commands (see also section 3.6, page 59).

If you are interested in an example showing the usage of the \headmark and \pagemark commands, see section 5.5, page 271. Internally, the scrlayer-scrpage package uses many such features of scrlayer.

If the options for the marks described above are not sufficient, additional commands for advanced users are documented starting on page 441.

```
\partmarkformat
\chaptermarkformat
\sectionmarkformat
\subsectionmarkformat
\subsubsectionmarkformat
\paragraphmarkformat
\subparagraphmarkformat
```

KOMA-Script classes and the scrlayer package typically use these commands internally to format the section numbers. They also support the \autodot mechanism of the KOMA-Script classes. If desired, these commands can be redefined to achieve a different formatting of section numbers. See the example in section 5.5, page 272 for more information.

```
\partmark{Text}
\chaptermark{Text}
\sectionmark{Text}
\subsectionmark{Text}
\subsubsectionmark{Text}
\paragraphmark{Text}
\subparagraphmark{Text}
```

Most classes use these commands internally to set the marks according to the sectioning commands. The argument should contain the text without the number of the sectioning unit. The number is automatically determined using the current section level if you use numbered headings.

However, not all classes use such a command for every section level. The standard classes, for example, do not call \partmark , whereas the KOMA-Script classes naturally support \partmark as well.

If you redefine these commands, be sure to check whether the numbers will be output via the secnumdepth before setting the number even if you do not change the secnumdepth counter yourself, because packages and classes may do so locally and rely on correct handling of secnumdepth.

The scrlayer package also redefines these commands whenever you use \automark or \manualmark or the corresponding options, to activate or deactivate the desired running heads.

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

Regardless of whether you are working with manual or automatic running heads, you can always change the contents of the *left mark* or the *right mark* using these commands. Note that the left-hand mark resulting from `\leftmark` will be the last mark placed on the corresponding page, while the right-hand mark resulting from `\rightmark` is the first mark placed on the corresponding page. For more details, see to `\rightfirstmark`, page 441.

If you are using manual running heads, the marks remain valid until they are explicitly replaced by reusing the corresponding commands. However, if you are using automatic running heads, the marks can become invalid with the next section heading, depending on the automatic configuration.

You can also use these commands in conjunction with the starred versions of the sectioning commands. You can find detailed examples illustrating usage of `\markboth` with the scrlayer-derived package scrlayer-scrpage in section 5.5, page 273.The command `\markdouble` does change the left mark and the right mark to the same contents. So `\markdouble{mark}` is a shorter form of `\markboth{mark}{mark}` with two identical arguments.

v3.28

```
\GenericMarkFormat{name of the section level}
```

By default, this command is used to format all section numbers in running heads below the subsection level, and for classes without `\chapter`, also for the section and subsection levels, unless the respective mark commands for those levels are defined before loading scrlayer. The command causes the package to use `\@seccntmarkformat` if this internal command is defined, as it is in the KOMA-Script classes. Otherwise `\@seccntformat` will be used, which is provided by the LaTeX kernel. The mandatory argument of the command contains the name of a sectioning command, such as `chapter` or `section` *without* the backslash prefix.

By redefining this command, you can change the default number format for all sectioning commands that use it. Classes can also change the default formatting also by defining this command.

A detailed example illustrating the interplay of the `\GenericMarkFormat` command with the page 443 command and `\sectionmarkformat` or `\subsectionmarkformat` when using the scrlayer-derived package scrlayer-scrpage is shown in section 18.1, page 447.

```
\@mkleft{left mark}
\@mkright{right mark}
\@mkdouble{mark}
\@mkboth{left mark}{right mark}
```

Within classes and packages, you may only want to use running heads if automatic running heads are active (see the automark option and the \automark command on page 267). In the standard LaTeX classes, this only works with \@mkboth. This command corresponds to either \@gobbletwo, which simply consumes both mandatory arguments, or \markboth, which sets both the left mark and the right mark. Packages like babel also change \mkboth, e. g., to enable language switching in the running head.

However, if you want to change only the *left mark* or the *right mark* without changing the other one, there is no corresponding command. The scrlayer package itself needs such commands to implement automatic running heads. So if \@mkleft, for setting the left mark only, or \@mkright, for setting the right mark only, or \@mkdouble, for setting both marks with the same content, is undefined when loading scrlayer, this package will define them. This definition uses the state of \@mkboth as an indication of whether to use automatic running heads. The commands will set the marks only in the case of automatic running heads.

Class and package authors can also fall back on these commands as appropriate if they want to set the left or right the marks only if automatic running heads are activated.

For more information about manipulating the contents of page styles, see also section 5.5 starting at page 266.

17.7. Defining and Managing Interfaces for End Users

v3.28

Until version 3.27 package scrlayer provided mechanism to manage concurrent end user interfaces. The mechanism has been marked as experimental and the usability limited. Only KOMA-Script itself has used this mechanism and from version 3.28 it is marked as deprecated. So the commands \scrlayerInitInterface, \scrlayerAddToInterface, \scrlayerAddCsToInterface, \scrlayerOnAutoRemoveInterface and the options forceoverwrite, autoremoveinterfaces should not be used any longer.

v3.12

Additional Features with the scrlayer-scrpage package

The scrlayer-scrpage package offers many features beyond what has been described in chapter 5 of part I of this guide. However, the average user will not normally need these extensions, and some of them are only provided for compatibility with scrpage2. The documentation here in part II serves to deepen and broaded your knowledge, and its mastery goes beyond basic skills.

18.1. Manipulating Page Styles

This section is a supplement to section 17.6. It describes features that may be too complicated for beginners.

```
\righttopmark
\rightbotmark
\rightfirstmark
\rightmark
\lefttopmark
\leftbotmark
\leftfirstmark
\leftmark
```

v3.16

LaTeX typically uses a two-part TeX mark for page styles. Running heads can access the left part of that mark with `\leftmark` and the right part with `\rightmark`. In fact, it was probably intended to use `\leftmark` for the running head of left (even) pages and `\rightmark` for the running head of right (odd) pages of two-sided documents. In one-sided printing, however, the standard classes do not even set the left part of the mark.

TeX itself knows three ways to access a mark. The `\botmark` is the last valid mark of the most recent page that has been built. If no mark has been set on the page, it corresponds to the last mark set on the pages that have already been shipped out. The LaTeX command `\leftmark` uses precisely this mark, so it returns the left part of the last mark of the page. This corresponds exactly to `\leftbotmark`. By comparison, `\rightbotmark` prints the right part of this mark.

`\firstmark` is the first mark of the last page that has been built. This is the first mark that has been set on the page. If no mark has been set on the page, it corresponds to the last mark of the pages that have already been shipped out. The LaTeX command `\rightmark` uses precisely this mark, so it returns the right part of the first mark of the page. This corresponds exactly to `\rightfirstmark`. By comparison, `\leftfirstmark` prints the left part of this mark.

\topmark is the content that \botmark had before building the current page. LaTeX itself does not use it. Nevertheless, scrlayer provides \lefttopmark to access the left part of this mark and \righttopmark to access the right part.

Note that the left and right portions of the mark can only be set together. Even if you use \markright to change only the right part, the left part will set again (unchanged). Accordingly, in two-sided printing, using the headings page style, the higher section levels always make both parts. For example, \chaptermark uses \markboth with an empty right argument in this case. This is the reason \rightmark or \rightfirstmark always shows an empty value on pages which start a chapter, even if there was a \sectionmark or \section on the same page to make the right part of the mark. Please note that using any of these commands to show the left or right part of the mark as part of the page content may lead to unexpected results. They are really meant for use in the header or footer of a page style only. Therefore, they should always be part of the contents of a layer when using scrlayer. But it does not matter whether the layer is restricted to the background or the foreground, since all layers are shipped out after building the current page.

If you need more information about the mark mechanism of TeX, please have a look at [Knu90, chapter 23]. The topic is flagged there as an issue for real experts.

\GenericMarkFormat{*name of the section level*}

By default, this command is used to format all section numbers in running heads below the subsection level, and for classes without \chapter, also for the section and subsection levels, unless the respective mark commands for those levels are defined before loading scrlayer. The command causes the package to use \@seccntmarkformat if this internal command is defined, as it is in the KOMA-Script classes. Otherwise \@seccntformat will be used, which is provided by the LaTeX kernel. The mandatory argument of the command contains the name of a sectioning command, such as chapter or section *without* the backslash prefix.

By redefining this command, you can change the default number format for all sectioning commands that use it. Classes can also change the default formatting also by defining this command.

Example: Suppose you want the section numbers of all levels in the running head of an article to be printed in white within a black box. Since the commands \sectionmarkformat and \subsectionmarkformat of the scrlayer package are defined with \GenericMarkFormat for articles using the standard LaTeX article class, you need to redefine only this one command:

```
\documentclass{article}
\usepackage{blindtext}
\usepackage[automark]{scrlayer-scrpage}
\pagestyle{scrheadings}
\usepackage{xcolor}
\newcommand*{\numberbox}[1]{%
```

```
  \colorbox{black}{\strut~\textcolor{white}{#1}~}}
\renewcommand*{\GenericMarkFormat}[1]{%
  \protect\numberbox{\csname the#1\endcsname}\enskip}
\begin{document}
\blinddocument
\end{document}
```

The colour has been changed using the xcolor package. More details can be found in that package's manual (see [Ker07]).

This example also uses an invisible strut. Any detailed LaTeX introduction should explain the related command `\strut`.

A helper macro, `\numberbox`, has been defined to format the number within a box. This command is prefixed in the redefinition of `\GenericMarkFormat` by `\protect` in order to protect it from expansion. This is necessary because otherwise the upper-case letter conversion of `\MakeUppercase` for the running head, would result in using the colours "`BLACK`" and "`WHITE`" instead of "`black`" and "`white`", and those colours are undefined. Alternatively, you could define `\numberbox` using `\DeclareRobustCommand*` instead of `\newcommand*` and omit `\protect` (see [Tea06]).

If you wanted to achieve the same effect with a KOMA-Script class or with the standard LaTeX classes book or report, you would also have to redefine, respectively, `\sectionmarkformat` and `\subsectionmarkformat`, or `\chaptermarkformat` and `\sectionmarkformat`, because these are not by default defined using `\GenericMarkFormat`:

```
\documentclass[headheight=19.6pt]{scrbook}
\usepackage{blindtext}
\usepackage[automark]{scrlayer-scrpage}
\pagestyle{scrheadings}
\usepackage{xcolor}
\newcommand*{\numberbox}[1]{%
  \colorbox{black}{\strut~\textcolor{white}{#1}~}}
\renewcommand*{\GenericMarkFormat}[1]{%
  \protect\numberbox{\csname the#1\endcsname}\enskip}
\renewcommand*{\chaptermarkformat}{\GenericMarkFormat{chapter}}
\renewcommand*{\sectionmarkformat}{\GenericMarkFormat{section}}
\begin{document}
\blinddocument
\end{document}
```

Here, option `headheight` is used to eliminate the warning, that also has been reported in the previous example.

```
\@mkleft{left mark}
\@mkright{right mark}
\@mkdouble{mark}
\@mkboth{left mark}{right mark}
```

Within classes and packages, you may only want to use running heads if automatic running heads are active (see the automark option and the \automark command on page 267). In the standard LaTeX classes, this only works with \@mkboth. This command corresponds to either \@gobbletwo, which simply consumes both mandatory arguments, or \markboth, which sets both the left mark and the right mark. Packages like babel also change \mkboth, e. g., to enable language switching in the running head.

However, if you want to change only the *left mark* or the *right mark* without changing the other one, there is no corresponding command. The scrlayer package itself needs such commands to implement automatic running heads. So if \@mkleft, for setting the left mark only, or \@mkright, for setting the right mark only, or \@mkdouble, for setting both marks with the same content, is undefined when loading scrlayer, this package will define them. This definition uses the state of \@mkboth as an indication of whether to use automatic running heads. The commands will set the marks only in the case of automatic running heads.

Class and package authors can also fall back on these commands as appropriate if they want to set the left or right the marks only if automatic running heads are activated.

18.2. Defining New Pairs of Page Styles

The two page styles scrheadings and plain.scrheadings were described in section 5.4. You can view them as a kind of pair, with scrheadings intended as the main page style for a running head and plain.scrheadings the corresponding plain page style without a running head but generally with pagination. In addition to configuring this predefined pair, scrlayer-scrpage also lets you define additional pairs of page styles. The name of the main page style, for example scrheadings, also serves as the name of the page-style pair.

The vast majority of users will not need more than the one predefined page-style pair, scrheadings. So the commands documented in this section are therefore extensions for special cases. Since I have not come across any suitable applications while writing this manual, there are no detailed examples. Should I ever run across a particularly nice application while providing support, I will happily include it in future versions. At the same time, however, I'm virtually certain that all such cases could also be solved using the predefined pair only.

```
\defpairofpagestyles[parent pair]{name}{definition}
\newpairofpagestyles[parent pair]{name}{definition}
\renewpairofpagestyles[parent pair]{name}{definition}
\providepairofpagestyles[parent pair]{name}{definition}
```

You can use these commands to define pairs of page styles similar to `scrheadings` and `plain.scrheadings`, where *name* is the name of the main page style corresponding to `scrheadings`. The name of the equivalent `plain` page style will be prefixed by `plain.` automatically. So *name* is not only the name of the pair of page styles, but also the name of the main page style of that pair, while `plain.`*name* is the name of the `plain` page style of this pair.

If you provide the optional *parent pair* argument, this is the name of a page-style pair whose settings are used to initialise the new page-style pair. So the new pair inherits the configuration of the *parent pair*.

Although section 5.4 might have created the impression that the commands described there apply only to `scrheadings` and `plain.scrheadings` only, this is true only so long as those two page styles are the only defined page-style pair. As soon as there are multiple page-style pairs, \lehead, \cehead, \rehead, \lohead, \cohead, \rohead, \lefoot, \cefoot, \refoot, \lofoot, \cofoot, \rofoot, \ihead, \chead, \ohead, \ifoot, \cfoot, and \ofoot all refer to page-style pair that was most recently activated.

In addition to those eighteen commands mentioned above, the three commands described below, \clearmainofpairofpagestyles, \clearplainofpairofpagestyles, and \clearpairofpagestyles, are designed to be used inside the *definition* argument. In this case, they represent a kind of default configuration of the page-style pair that is executed each time the pair is activated. You activate a page-style pair by activating either one two page styles in the pair. Typically, you do so with \pagestyle.

Note that the commands of section 5.5 on page 266 are general in nature and apply to all page styles defined with scrlayer-scrpage.

Although `\defpairofpagestyles` defines a page-style pair regardless of whether the corresponding page styles already exist, `\newpairofpagestyles` and `\providepairofpagestyles` do so only if the page styles are currently undefined. If at least one of the page styles is already defined, the new definition of `\providepairofpagestyles` will be ignored, whereas using `\newpairofpagestyles` results in an error message. To redefine existing page-style pairs, you can use `\renewpairofpagestyles`. With this an is thrown if either one of the two page styles of the pair does not already exist.

```
\clearmainofpairofpagestyles
\clearplainofpairofpagestyles
\clearpairofpagestyles
```

The \clearmainofpairofpagestyles command sets the main page style of the most recently activated page-style pair to be empty. In contrast, the \clearplainofpairofpagestyles command sets the plain page style of the active page-style pair to be empty. Finally, \clearpairofpagestyle sets both page styles of the activate pair to be empty.

But note that none of these commands removes the definitions of the *definition* argument that was specified when defining the page-style pair (see above). So if you activate the pair of page styles again, those definitions will be used again!

You can use these commands inside the *definition* of the page-style pair explained above. But you can also use them outside this definition. In this case, they refer to the most recently activated page-style pair.

18.3. Defining Complex Page Styles

In addition to the predefined page styles, scrlayer-scrpage also provides a more basic interface for defining new page styles. The page-style definitions discussed so far use this interface internally, as do the obsolete commands in section 18.4. scrlayer-scrpage. Because of its complexity, however, only advanced users should try to use it directly. Less experienced users can already achieve almost everything possible with this low-level interface by using the possibilities described previously.

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

You can use these commands to define a single page style with maximum flexibility, where *name* is the name of the page style that you want to define.

The parameters *header specification* and *footer specification* have identical structure:

```
(length of the line above,thickness of the line above)%
{specification for the left page in two-side layout}%
{specification for the right page in two-side layout}%
{specification for all pages in one-side laypout}%
(length of the line below,thickness of the line below)
```

The arguments in the round brackets are optional. That is, you can omit them together with the brackets. In that case, the length and thickness of the corresponding horizontal

Table 18.1.: The layers scrlayer-scrpage defines for a *name* page style

Name of the layer	Meaning of the layer
name`.head.above.line`	horizontal line above the header
name`.head.odd`	header of odd pages in two-sided printing
name`.head.even`	header of even pages in two-sided printing
name`.head.oneside`	header in one-sided printing
name`.head.below.line`	horizontal line below the header
name`.foot.above.line`	horizontal line above the footer
name`.foot.odd`	footer of odd pages in two-sided printing
name`.foot.even`	footer of even pages in two-sided printing
name`.foot.oneside`	footer in one-sided printing
name`.foot.below.line`	horizontal line below the footer

rules are based on the KOMA-Script options `headtopline`, `headsepline`, `footsepline`, and `footbotline` (see section 5.5, page 276).

All three arguments in curly brackets are mandatory and are used depending on the page and the layout settings. Their content can be anything you want. For page styles with running heads, however, you should use `\headmark`, `\leftmark`, or `\rightmark` inside the specification. Under no circumstances should you directly put the number or text of a sectioning command here. Because of the LaTeX's asynchronous page construction, the wrong numbers or text can appear in the header or footer if you do so.

The `\defpagestyle` command defines the page style regardless of whether it already exists or not. In contrast, `\newpagestyle` throws an error if a page style of the same *name* already exists. On the other hand, `\providepagestyle` simply ignores the definition if the *name* has already been used for a page style. Conversely, `\renewpagestyle` can only redefine an existing page style. For a new *name*, it throws an error.

All four commands are based on the `\DeclarePageStyleByLayers` command of the scrlayer package. You can find the layers that are defined for a page style *name* in table 18.1, and more information about layers and layer-page in chapter 17, starting on page 416.

Example: Suppose you want to set a background colour for the header of the `scrheadings` page style. From the introduction to this chapter and table 18.1, you know that `scrheadings` is a layer page style that includes the layers `scrheadings.head.even`, `scrheadings.head.odd`, and `scrheadings.head.oneside`. You now define three more layers for their backgrounds and add them at the beginning of the page style:

```
\documentclass{scrartcl}
\usepackage[automark]{scrlayer-scrpage}
\usepackage{xcolor}
```

```
\usepackage{blindtext}
\DeclareLayer[clone=scrheadings.head.oneside,
  contents={%
    \color{yellow}\rule[-\dp\strutbox]{\layerwidth}{\layerheight}%
  }%
]{scrheadings.head.oneside.background}
\DeclareLayer[clone=scrheadings.head.odd,
  contents={%
    \color{yellow}\rule[-\dp\strutbox]{\layerwidth}{\layerheight}%
  }%
]{scrheadings.head.odd.background}
\DeclareLayer[clone=scrheadings.head.even,
  contents={%
    \color{yellow}\rule[-\dp\strutbox]{\layerwidth}{\layerheight}%
  }%
]{scrheadings.head.even.background}
\AddLayersAtBeginOfPageStyle{scrheadings}{%
  scrheadings.head.oneside.background,%
  scrheadings.head.odd.background,%
  scrheadings.head.even.background%
}
\pagestyle{scrheadings}
\begin{document}
\blinddocument
\end{document}
```

As you can see, the example uses three layers so that the position and size of the background layers could simply be copied from the corresponding header layer using the `clone` option. This is easier than using only one background layer and dynamically calculating its position each time.

The coloured background itself was created using a `\rule` command. The size arguments of this `\rule` are given by `\layerwidth` and `\layerheight` which contain the current width and height of the layer itself. The optional argument of `\rule` is used to move the rule down by the height of a descender.

Instead of using new layers to colour the background in the example above, `\colorbox` and `\chead` could have been used. You can work out a solution using this method as an exercise. Likewise, you could have added the background layers individually just before the corresponding content layer. You can implement this as an exercise too.

18.4. Defining Simple Page Styles with a Tripartite Header and Footer

Currently, additional information on this topic can be found at the same point in the German KOMA-Script book [Koh20a] only.

18.5. Legacy Features of scrpage2

The scrlayer-scrpage package contains some legacy features that derive from scrpage2 and exist only to be as compatible as possible with that package. Users only need to understand these features if they want to edit an old document based on scrpage2. You should not use the items documented here in new documents!

`hmode=`*simple switch*

The scrpage2 package always outputs headers and footers in horizontal mode. In contrast, scrlayer-scrpage in the default setting only switches into horizontal mode when horizontal material is output. However, if you activate the `hmode` option, scrlayer-scrpage will behave like scrpage2 and switch to horizontal mode before any output. This can affect both the processing of white space at the beginning of the output and vertical alignment.

The options recognizes the standard values for simple switches listed in table 2.5 on page 42. The option is deactivated by default.

Currently, additional information on this topic can be found at the same point in the German KOMA-Script book [Koh20a] only.

Note Columns with scrlayer-notecolumn

Through version 3.11b, KOMA-Script supported note columns only in the form of marginal notes that get their contents from \marginpar and \marginline (see section 3.21, page 145). This kind of note column has several disadvantages:

- Marginal notes must be set completely on a single page. Page breaks inside marginal notes are not possible. This sometimes causes the marginal notes to protrude into the lower margin.
- Marginal notes near page breaks sometimes float to the next page and then, in the case of two-sided printing, cause alternate marginal columns to appear in the wrong margin.. This problem can be solved with the additional package mparhack or by using \marginnote from the marginnote package (see [Koh12]).
- Marginal notes inside floating environments or footnotes are not possible. This problem can also be solved with \marginnote of the marginnote package.
- There is only one marginal note column, or at most two if you use \reversemarginpar and \normalmarginpar. Note that \reversemarginpar is of less utility with two-sided documents.

Using marginnote leads to one more problem. Because the package does not have any collision detection, marginal notes that are set near to each other can partially or totally overlap. Moreover, depending on the settings used, \marginnote sometimes changes the baseline distance of the normal text.

The scrlayer-notecolumn package should solve all these problems. To do so, it relies on the basic functionality of scrlayer. However, using this package has a drawback: you can only output notes on pages that use a page style based on scrlayer. This disadvantage, however, can easily be resolved, or even turned into an advantage, with the help of scrlayer-scrpage.

19.1. Note about the State of Development

This package was originally developed as a so-called *proof of concept* to demonstrate the potential of scrlayer. Although it is still in its early stages of development, most of its stability is less a question of scrlayer-notecolumn than of scrlayer. Nevertheless, you can assume that there are still bugs in scrlayer-notecolumn. Please report such bugs whenever you find them. Some of the package's shortcomings are caused by the attempt to minimise complexity. For example, although note columns can break across several pages, there is no new paragraph break. TEX itself does not provide this.

Because the package is rather experimental, its instructions are found here in the second part of the KOMA-Script manual. Accordingly, it is primarily directed towards experienced

users. If you are a beginner or a user on the way to become an expert, some of the following explanations may be unclear or even incomprehensible. Please understand that I want to keep the effort spent on the manual to something halfway bearable when it comes to experimental packages.

19.2. Early or Late Selection of Options

The information in section 2.4 applies equally to this chapter. So if you have already read and understood section 2.4, you can skip ahead to section 19.3, page 457.

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

LaTeX allows users to pass class options as a comma-separated list of keywords in the optional argument to \documentclass. In addition to being passed to the class, these options are also passed on to all packages that can understand them. Users can also pass a similar comma-separated list of keywords in the optional argument of \usepackage. KOMA-Script extends the option mechanism for the KOMA-Script classes and some packages with further options. Thus most KOMA-Script options can also take a value, so an option does not necessarily take the form *option*, but can also take the form *option=value*. Except for this difference, \documentclass and \usepackage in KOMA-Script function as described in [Tea05b] or any introduction to LaTeX, for example [OPHS11].

v3.00

Setting the options with \documentclass has one major disadvantage: unlike the interface described below, the options in \documentclass are not robust. So commands, lengths, counters, and similar constructs may break inside the optional argument of this command. For example, with many non-KOMA-Script classes, using a LaTeX length in the value of an option results in an error before the value is passed to a KOMA-Script package and it can take control of the option execution. So if you want to use a LaTeX length, counter, or command as part of the value of an option, you should use \KOMAoptions or \KOMAoption. These commands will be described next.

\KOMAoptions{*option list*}
\KOMAoption{*option*}{*value list*}

v3.00

KOMA-Script also provides the ability to change the values of most class and package options even after loading the class or package. You can use the \KOMAoptions command to change the values of a list of options, as in \documentclass or \usepackage. Each option in the *option list* has the form *option=value*.

Some options also have a default value. If you do not specify a value, that is if you give the option simply as *option*, then this default value will be used.

Some options can have several values simultaneously. For such options, it is possible, with the help of \KOMAoption, to pass a list of values to a single *option*. The individual values are given as a comma-separated *value list*.

KOMA-Script uses the commands \FamilyOptions and \FamilyOption with the family "KOMA" to implement this ability. See part II, section 12.2, page 336.

Options set with \KOMAoptions or \KOMAoption will reach both the KOMA-Script class and any previously loaded KOMA-Script packages that recognise these options. If an option or a value is unknown, scrbase will report it as an error.

19.3. Text Markup

The information in in section 3.6 largely applies to this chapter. So if you have already read and understood section 3.6, you can skip ahead to section 19.4, page 458.

LaTeX offers different possibilities for logical and direct markup of text. In addition to the choice of the font, this includes commands for choosing the font size and orientation. For more information about the standard font facilities, see [OPHS11], [Tea05b], and [Tea05a].

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

With the help of the \setkomafont and \addtokomafont commands, you can attach particular font styling *commands* that change the appearance of a given *element*. Theoretically, all statements, including literal text, can be used as *commands*. You should, however, limit yourself to those statements that really change font attributes only. These are usually commands like \rmfamily, \sffamily, \ttfamily, \upshape, \itshape, \slshape, \scshape, \mdseries, \bfseries, \normalfont, as well as the font size commands \Huge, \huge, \LARGE, \Large, \large, \normalsize, \small, \footnotesize, \scriptsize, and \tiny. You can find these commands explained in [OPHS11], [Tea05b], or [Tea05a]. Colour switching commands like \normalcolor (see [Car17] and [Ker07]) are also acceptable. The use of other commands, in particular those that redefine things or or lead to output, is not supported. Strange behaviour is possible in these cases and does not represent a bug.

The command \setkomafont provides an element with a completely new definition of its font styling. In contrast, the \addtokomafont command merely extends an existing definition. You should not use either command inside the document body but only in the preamble. For examples of their use, refer to the sections for the respective element.

With the \usekomafont command, the current font style can be changed to the one defined for the specified *element*.

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

v3.12 Sometimes, although this is not recommended, the font setting of an element is used for settings that are not actually related to the font. If you want to apply only the font setting of an element but not those other settings, you can use `\usefontofkomafont` instead of `\usekomafont`. This will activate the font size and baseline skip, the font encoding, the font family, the font series, and the font shape of an element, but no further settings as long as those further settings are local.

You can also switch to a single one of those attributes using one of the other commands. Note that `\usesizeofkomafont` uses both the font size and the baseline skip.

However, you should not take these commands as legitimizing the insertion of arbitrary commands in an element's font setting. To do so can lead quickly to errors (see section 21.5, page 476).

19.4. Declaring New Note Columns

Loading the package automatically declares a note column named `marginpar`. As the name implies, this note column is placed in the area of the normal marginal column used by `\marginpar` and `\marginline`. The `\reversemarginpar` and `\normalmarginpar` settings are also taken into account, but only for all the notes on a page instead of note by note. The relevant setting is the one that applies at the end of the page, namely during the output of the note column. If you want to have notes in both the left and right margin of the same page, you should define a second note column.

The default settings for all newly declared note columns are the same as the defaults for `marginpar`. But you can easily change them during their initialisation.

Note that note columns can be output only on pages that use a page style based on the scrlayer package. The scrlayer-notecolumn package automatically loads scrlayer, which by default provides only `empty` page style. If you need additional page styles, scrlayer-scrpage is recommended.

```
\DeclareNoteColumn[option list]{note column name}
\DeclareNewNoteColumn[option list]{note column name}
\ProvideNoteColumn[option list]{note column name}
\RedeclareNoteColumn[option list]{note column name}
```

You can use these commands to create note columns. `\DeclareNoteColumn` creates the note column regardless of whether it already exists. `\DeclareNewNoteColumn` throws an error if the *note column name* has already been used for another note column. `\ProvideNoteColumn` simply does nothing in that case. You can use `\RedeclareNoteColumn` only to reconfigure an existing note column.

By the way, when reconfiguring existing note columns with `\DeclareNoteColumn` or `\RedeclareNoteColumn`, the notes that have already been generated for this column are retained.

Declaring a new note column always defines a new element for changing its font attributes with `\setkomafont` and `\addtokomafont`, if such an element does not yet exist. The name of the element is `notecolumn.`*note column name*. For this reason, the default note column `marginnote` has the element `notecolumn.marginpar`. You can directly specify the initial setting of the element's font when declaring a note column by using the `font` option within the *option list*.

The *option list* is a comma-separated list of keys with or without values, also known as options. The available options are listed in table 19.1, page 461. The `marginpar` option is set by default, but you can overwrite this default with your individual settings.

Because note columns are implemented using scrlayer, a layer is created for each note column. The layer name is the same as the name of the element, `notecolumn.`*note column name*. For more information about layers see section 17.3, starting on page 418.

Example: Suppose you are a professor of comedy law and want to write a treatise on the new "Statute Concerning the Riotous Airing of Common Humour", SCRACH for short. The better part of the work will consist of commentary on individual paragraphs of the statute. You decide on a two-column layout, with the comments in the main column and the paragraphs placed in a smaller note column on the right of the main column using a font that is smaller and in a different colour.

```
\documentclass{scrartcl}
\usepackage{lmodern}
\usepackage{xcolor}

\usepackage{scrjura}
\setkomafont{contract.Clause}{\bfseries}
\setkeys{contract}{preskip=-\dp\strutbox}

\usepackage{scrlayer-scrpage}
\usepackage{scrlayer-notecolumn}
```

```
\newlength{\paragraphscolwidth}
\AfterCalculatingTypearea{%
  \setlength{\paragraphscolwidth}{%
    .333\textwidth}%
  \addtolength{\paragraphscolwidth}{%
    -\marginparsep}%
}
\recalctypearea
\DeclareNewNoteColumn[%
  position=\oddsidemargin+1in
           +.667\textwidth
           +\marginparsep,
  width=\paragraphscolwidth,
  font=\raggedright\footnotesize
       \color{blue}
]{paragraphs}
```

The treatise should be a one-sided article. The font is Latin Modern, and the colour selection uses the xcolor package.

For formatting legal texts with the scrjura package, see chapter 11.

Since this document uses a page style with a page number, the scrlayer-scrpage package is loaded. Thus, note columns can be output on all pages.

Next, the scrlayer-notecolumn package is loaded. The required width of the note column is calculated with `\AfterCalculatingTypearea` after any recalculation of the type area. It should be one third of the type area minus the distance between the main text and the note column.

With this information, we define the new note column. For the positions, we use a simple dimension expression. Note that `\oddsidemargin` is not the total left margin but for historical reasons the left margin minus 1 inch. So we have to add this value.

This concludes the definition. Note that the note column would currently be placed inside the type area. This means that the note column would overwrite the text.

```
\begin{document}

\title{Commentary on the SCRACH}
\author{Professor R. O. Tenase}
\date{11/11/2011}
\maketitle
\tableofcontents

\section{Preamble}
```

```
The SCRACH is without doubt the most important law
on the manners of humour that has been passed in
the last thousand years. The first reading took
place on 11/11/1111 in the Supreme Manic Fun
Congress, but the law was rejected by the Vizier
of Fun. Only after the ludicrous, Manic Fun
monarchy was transformed into a representative,
witty monarchy by W. Itzbold, on 9/9/1999 was
the way finally clear for this law.
```

Because the text area was not reduced, the preamble is output extending over the whole width of the type area. To test this, you can temporarily add:

```
\end{document}
```

In the example, the question of how the text for the commentary can be set in a narrower column remains unresolved. You will discover how to do this by continuing the example below.

Table 19.1.: Available settings for declaring note columns

`font=`*font attribute*
: The font attributes of the note column set with `\setkomafont`. For possible values, refer to section 3.6, page 59.
Default: *empty*

`marginpar`
: Sets `position` and `width` to correspond to the marginal note column of `\marginpar`. Switching between `\reversemarginpar` and `\normalmarginpar` is only considered at the end of the page when the note column is output. Note that this option does not expect or allow any value.
Default: *yes*

`normalmarginpar`
: Sets `position` and `width` to use the normal marginal note column and ignore `\reversemarginpar` and `\normalmarginpar`. Note that this option does not expect or allow a value.
Default: *no*

...

Table 19.1.: Available settings for declaring note columns (*continued*)

`position=`*`offset`*
: Sets the horizontal offset of the note column from the left edge of the paper. The *`offset`* can be a complex expression as long as it is fully expandable and expands to a length or a dimensional expression at the time the note column is output. See [Tea98, section 3.5] for more information about dimensional expressions.
Default: *through the `marginpar` option*

`reversemarginpar`
: Sets `position` and `width` to use the reverse marginal note column of `\marginpar` with the `\reversemarginpar` setting. Note that this option does not expect or allow a value.
Default: *no*

`width=`*`length`*
: Sets the width of the note column. The *`length`* can be a complex expression as long as it is fully expandable and expands to a length or a dimensional expression at the time the note column is output. See [Tea98, section 3.5] for more information about dimensional expressions.
Default: *through the `marginpar` option*

19.5. Making a Note

After you declare a note column, you can create notes for this column. But these notes are not be output immediately. Initially, they are written to an auxiliary file with extension "`.slnc`". Specifically, they are first written to the `aux`-file and, when the `aux`-file is read inside `\end{document}`, they are copied to the `slnc`-file. If necessary, the `\nofiles` setting is also taken into account. At the next LaTeX run, this auxiliary file will be read piece by piece, according to the progress of the document, and at the end of the page the notes for that page will be output.

Note, however, that note columns are output only on pages whose page style is based on the scrlayer package. This package is loaded automatically by scrlayer-notecolumn and by default provides only the `empty` page style. If you need additional page styles, the scrlayer-scrpage package is recommended.

\makenote[*note-column name*]{*note*}
\makenote*[*note-column name*]{*note*}

You can use this command to make a new *note*. The current vertical position is used as the vertical position for the start of the *note*. The horizontal position for the note results from the defined position of the note column. To work correctly, the package relies on \pdfsavepos, \pdflastypos, and \pdfpageheight or their equivalent in newer LuaTEX versions. Without these commands, scrlayer-notecolumn will not work. The primitives should act exactly as they would using pdfTEX.

However, if the package detects a collision with a previous *note* in the same note column, the new *note* is moved below that earlier *note*. If the *note* does not fit on the page, it will be moved completely or partially to the next page.

The optional argument *note column name* determines which note column should be used for the *note*. If the optional argument is omitted, the default note column marginpar is used.

Example: Let's add a commented paragraph to the example of the previous section. The paragraph itself should be placed in the newly defined note column:

```
\section{Analysis}
\begin{addmargin}[0pt]{.333\textwidth}
  \makenote[paragraphs]{%
    \protect\begin{contract}
      \protect\Clause{%
        title={No Joke without an Audience}%
      }
      A joke can only be funny if is has an
      audience.
    \protect\end{contract}%
  }
  This is one of the most central statements of
  the law. It is so fundamental that it is quite
  appropriate to bow to the wisdom of the authors.
```

The addmargin environment, which is described in section 3.18, page 124, is used to reduce the width of the main text by the width of the column for the paragraphs.

Here you can see one of the few problems of using \makenote. Because the mandatory argument is written to an auxiliary file, commands inside this argument can, unfortunately, *break*. To avoid this, you should use \protect in front of all commands that should not expand when written to the auxiliary file. Otherwise, using a command inside this argument could result in error messages.

In principle you could now finish this example with

```
\end{addmargin}
\end{document}
```

to see a preliminary result.

If you test this example, you will see that the column for the legal text is longer than that of the commentary. If you add another section with another paragraph, you may encounter the problem that the commentary will not continue below the legal text but immediately after the previous comment. Next you will find a solution to this problem.

v0.1.2583

The problem with fragile commands mentioned in the example above does not occur with the starred variant. It uses `\detokenize` to prevent the expansion of commands. But this also means that you should not use commands in the *note* that change their definition within the document.

Unfortunately, both commands have two other known limitations. The first issue is related to colours using color or xcolor within the note columns. To make such colour changes possible each note column requires its own colour management using so-called *colour stacks*. Because the package was designed only as a *proof of concept* and because XeTeX does not support multiple colour stacks, XeTeX colour switching is restricted to the attributes of the font element `notecolumn.`*note column name*, a limitation which eliminates the time and effort required to implement custom colour management.

The second issue is of a more conceptual nature. The content of the auxiliary file that contains the note-column information is read while processing the page header. This has consequences in particular if the read occurs while an environment like `verbatim` is active. In this case, the `\catcode` settings of this environment would be active while reading the auxiliary file. This will inevitably lead to errors in processing and output. To attenuate this risk, the `\catcodes` of the characters from `\dospecials` are stored during `\begin{document}` and explicitly restored when reading from the auxiliary file.

`\syncwithnotecolumn[`*note column name*`]`

This command adds a synchronisation point in a note column and in the main text of the document. Whenever a synchronisation point is reached during the output of a note column or the main text, a mark will be generated that consists of the current page and the current vertical position.

In parallel with the generation of synchronisation points, scrlayer-notecolumn determines whether a mark has been set in the note column or the main text during the previous LaTeX run. If so, it compares their values. If the mark of the note column is lower on the current page or on a later page, the main text will be moved down to the position of the mark.

As a rule, you should not place synchronisation points within paragraphs of the main text but only between them. If you nonetheless use `\syncwithnotecolumn` inside a paragraph, the synchronisation point will be delayed until the current line has been output. This behaviour is similar to that of, e. g., `\vspace` in this respect.

Because synchronisation points are not recognized until the next LaTeX run, this mechanism requires at least three LaTeX runs. Any new synchronisation may also result in shifts of later synchronisation points, which in turn will require additional LaTeX runs. Such shifts are usually indicated by the message: “LaTeX Warning: Label(s) may have changed. Rerun to

get cross-references right." But reports about undefined labels may also indicate the need for another LaTeX run.

If you omit the optional argument, the default note column `marginpar` will be used. Note that an empty optional argument is not the same as omitting the optional argument!

You cannot use `\syncwithnotecolumn` inside a note itself, that is, inside the mandatory argument of `\makenote`! Currently the package cannot recognise such a mistake, and it causes new shifts of the synchronisation point with each LaTeX run, so the process will never terminate. To synchronise two or more note columns, you have to synchronise each of them with the main text. The recommended command for this is described below.

Example: Let's extend the example above by first adding a synchronisation point and then another paragraph with a comment:

```
\syncwithnotecolumn[paragraphs]\bigskip
\makenote[paragraphs]{%
  \protect\begin{contract}
    \protect\Clause{title={Humor of a Culture}}
    \setcounter{par}{0}%
    The humour of a joke can be determined by the
    cultural environment in which it is told.

   The humour of a joke can be determined by the
   cultural environment in which it acts.
  \protect\end{contract}
}
The cultural component of a joke is, in fact, not
negligible. Although the political correctness of
using the cultural environment can easily be
disputed, nonetheless the accuracy of such comedy
in the appropriate environment is striking. On
the other hand, a supposed joke in the wrong
cultural environment can also be a real danger
for the joke teller.
```

In addition to the synchronisation point, a vertical skip has been added with `\bigskip` to better distinguish each paragraph and the corresponding comments.

Further, this example illustrates another potential problem. Because the note columns uses boxes that are assembled and disassembled, counters inside note columns can sometimes jitter. In the example, therefore, the first paragraph would be numbered 2 instead of 1. This, however, can easily be fixed by a direct reset of the corresponding counter.

The example is almost complete. You just have to close the environments:

```
\end{addmargin}
\end{document}
```

> In reality, of course, all the remaining section of the law should also be commented. But let us focus on the main purpose.

But stop! What if, in this example, the *paragraphs* would no longer fit on the page? Would it be printed on the next page? We will answer this question in the next section.

`\syncwithnotecolumns[`*list of note column names*`]`

This command synchronises the main text with all note columns of the comma-separated *list of note column names*. The main text will be synchronised with the note column whose mark is closest to the end of the document. As a side effect, the note columns will be synchronised with each other.

If the optional argument is omitted or empty (or begins with `\relax`), synchronisation will be done with all currently declared note columns.

19.6. Forced Output of Note Columns

In addition to the normal output of note columns described in the previous section, you may sometimes need to output all collected notes that have not yet been output. This is especially useful when large notes cause more and more notes to be moved down to new pages. A good time to force the output is, for example, the end of a chapter or the end of the document.

`\clearnotecolumn[`*note column name*`]`

This command prints all notes of a particular note column that have not yet been output by the end of the current page but which were created on that or a previous page. Blank pages are generated as needed to output these pending notes. During the output of the pending notes of this note column, notes of other note columns may also be output, but only as necessary to output the pending notes of the specified note column.

During the output of the pending notes, notes created in the previous LaTeX run on the pages that are now replaced by the inserted blank pages may be output by mistake. This will be corrected automatically in one of the subsequent LaTeX runs. Such shifts are usually indicated by the message: "LaTeX Warning: Label(s) may have changed. Rerun to get cross-references right."

The note column whose pending notes are to be output is indicated by the optional argument *note column name*. If this argument is omitted, the notes of the default note column `marginpar` will be output.

The attentive reader will have noticed that the forced output of a note column is not unlike synchronisation. But if the forced output actually results in an output, it will be at the start of a new page and not just below the previous output. Nonetheless, a forced output usually results in fewer LaTeX runs.

`\clearnotecolumns[`*list of note column names*`]`

This command is similar to `\clearnotecolumn`, but the the optional argument here can be not only the name of one note column but a comma-separated *list of note column names*. The pending notes of all these note columns are then ouput.

If you omit the optional argument or leave it empty, all pending notes of all note columns will be output.

`autoclearnotecolumns=`*simple switch*

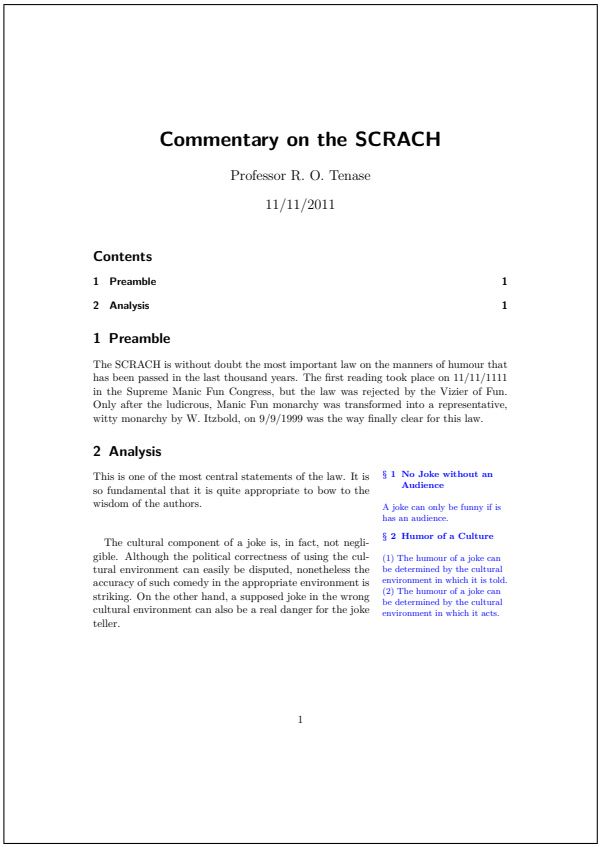
As a rule, pending notes will always be output if a document implicitly — e. g. because of a `\chapter` command — or explicitly executes `\clearpage`. This is also the case at the end of the document within `\end{document}`. The `autoclearnotecolumns` option can be used to control whether `\clearnotecolumns` should be executed automatically without arguments when running `\clearpage`.

Since this is generally the desired behaviour, the option is active by default. But you can change this with the appropriate value for a simple switch (see table 2.5, page 42) at any time.

Note that disabling the automatic output of pending notes can result in lost notes at the end of the document. So in this case you should insert `\clearnotecolumns` before `\end{document}` fore safety's sake.

The question from the end of the last section should now be answered. If the paragraph is to be completely output, it had to be wrapped to the next page. This is, of course, the default setting. However, since this would happen after the end of the `addmargin` environment, the forced output could still overlap with subsequent text. So in the example it would make sense to add another synchronisation point after the `addmargin` environment.

The result of the example is shown in figure 19.1.



Commentary on the SCRACH

Professor R. O. Tenase

11/11/2011

Contents

1 Preamble 1

2 Analysis 1

1 Preamble

The SCRACH is without doubt the most important law on the manners of humour that has been passed in the last thousand years. The first reading took place on 11/11/1111 in the Supreme Manic Fun Congress, but the law was rejected by the Vizier of Fun. Only after the ludicrous, Manic Fun monarchy was transformed into a representative, witty monarchy by W. Itzbold, on 9/9/1999 was the way finally clear for this law.

2 Analysis

This is one of the most central statements of the law. It is so fundamental that it is quite appropriate to bow to the wisdom of the authors.

§ 1 No Joke without an Audience

A joke can only be funny if is has an audience.

The cultural component of a joke is, in fact, not negligible. Although the political correctness of using the cultural environment can easily be disputed, nonetheless the accuracy of such comedy in the appropriate environment is striking. On the other hand, a supposed joke in the wrong cultural environment can also be a real danger for the joke teller.

§ 2 Humor of a Culture

(1) The humour of a joke can be determined by the cultural environment in which it is told. (2) The humour of a joke can be determined by the cultural environment in which it acts.

1

Figure 19.1.: A sample page for the example in this chapter

Additional Information about the typearea package

This chapter provides additional information about the typearea package. Some parts of the chapter are found only in the German KOMA-Script book [Koh20a]. This should not be a problem, because the average user, who only wants to use the package, will not normally need this information. Part of this material is intended for users who want to solve unusual problems or write their own packages using typearea. Another part covers typearea features that exist only for compatibility with earlier versions of KOMA-Script or with the standard classes. The features that exist only for compatibility with earlier versions of KOMA-Script are printed in a sans serif font. You should not use them any longer.

20.1. Experimental Features

This section describes experimental features. Experimental, in this context, means that correct functioning cannot be guaranteed. There can be two reasons for designating something experimental. First, the final function is not yet defined or its implementation is not yet finalised. Second, a feature may depend on internal functions of other packages and therefore the feature can not be guaranteed, if the other packages change.

`usegeometry=`*`simple switch`*

Usually typearea does not care much if you use it with the geometry package (see [Ume10]) in any configuration. In particular, this means that geometry does not recognise any changes to the page parameters done with typearea, for example when it changes the paper size — a feature not provided by geometry itself.

v3.17 Once you set option `usegeometry`, typearea tries to translate all of its options into options of geometry. If you activate new parameters inside the document, typearea even calls `\newgeometry` (see `\activateareas` in the following section). Since geometry does not support changes of paper size or page orientation with `\newgeometry`, typearea uses internal commands and geometry lengths to provide such changes as needed. This has been tested with geometry 5.3 through 5.6.

Note that using geometry and changing page size or orientation with typearea does not mean that geometry will automatically use the new paper size in an expected manner. For convenience, geometry provides far more options to adjust the page than are required to determine the type area, margins, header, footer, etc. — this is called *overdetermination* — and at the same time `\newgeometry` derives missing information from the known values — known as *value preservation* — so you often must explicitly specify all new values completely when you call `\newgeometry` on your own. Nevertheless, when typearea takes geometry into consideration, it opens up additional possibilities.

areasetadvanced=*simple switch*
\areaset[*BCOR*]{*width*}{*height*}

Usually, \areaset does not handle options to define the height of the header or footer, or whether margin elements should count as part of the type area in the same way as \typearea. v3.11 With the areasetadvanced option, you can make \areaset behave more like \typearea in this regard. Nevertheless, settings for the two commands that result in type areas of equal size still can differ because \typearea always adjusts the type area so that it contains an integer number of lines, potentially making the bottom margin up to one line smaller, whereas \areaset always sets the ratio between the top and bottom margins at 1:2. The type area can therefore be slightly shifted vertically depending on which command was used.

20.2. Expert Commands

This section describes commands that are of little or no interest to average users. These commands give experts additional possibilities. Because this information is addressed to experts, it appears in condensed form.

\activateareas

The typearea package uses this command convert the settings for the type area and margins to internal LaTeX lengths whenever the type area has been recalculated inside of the document, that is after \begin{document}. If the pagesize option has been used, it will be executed again with the same value. Thus, for example, the page size may actually vary within PDF documents.

Experts can also use this command if they have manually changed lengths like \textwidth or \textheight inside a document for any reason. If you do so, however, you are responsible for any necessary page breaks before or after you call \activateareas. Moreover, all changes made by \activateareas are local.

\storeareas{*command*}
\BeforeRestoreareas{*code*}
\BeforeRestoreareas*{*code*}
\AfterRestoreareas{*code*}
\AfterRestoreareas*{*code*}

\storeareas defines a *command* which you can use to restore all current type-area settings. So you can save the current settings, change them, and then restore the previous settings afterwards.

Example: You want a landscape page inside a document with portrait format. That's no problem using \storeareas:

```
\documentclass[pagesize]{scrartcl}

\begin{document}
\noindent\rule{\textwidth}{\textheight}

\storeareas\MySavedValues
\KOMAoptions{paper=landscape,DIV=current}
\noindent\rule{\textwidth}{\textheight}

\clearpage
\MySavedValues
\noindent\rule{\textwidth}{\textheight}
\end{document}
```

It's important to call \clearpage before \MySavedValues so that the saved values are restored on the next page. With two-sided documents, changes to the paper format should even use \cleardoubleoddpage or — if you are not using a KOMA-Script class — \cleardoublepage.

In addition, \noindent is used to avoid paragraph indents of the black boxes. Otherwise, you would not produce a correct image of the type area.

Note that neither \storeareas nor the defined *command* defined with it should be used inside a group. Internally, \newcommand is used to define the *command*. So reusing a *command* to store settings again would result in a corresponding error message.

v3.18 Often, it is useful to automatically execute commands like \cleardoubleoddpage before restoring the settings of a \command generated by \storeareas. You can do so using \BeforeRestoreareas or \BeforeRestoreareas*. Similarly, you can use \AfterRestoreareas or \AfterRestoreareas* to automatically execute *code* after restoring the settings. The variants with and without the star differ in that the starred variant only applies the *code* to future *command*s generated by \storeareas, whereas the regular variant also adds the *code* to previously defined *command*s.

```
\AfterCalculatingTypearea{code}
\AfterCalculatingTypearea*{code}
\AfterSettingArea{cod}
\AfterSettingArea*{code}
```

These commands serve to manage two hooks. The first two, \AfterCalculatingTypearea and its starred variant let experts execute *code* each time typearea recalculates the type area and margins, that is after every implicitly or explicit invocation of \typearea. Similarly,

v3.11 \AfterSettingArea and its starred variant allow for executing *code* every time \areaset has been used. The normal versions have a global scope, while changes made in the starred versions are only local. The *code* is executed immediately before \activateareas.

20.3. Local Settings with the typearea.cfg File

Currently, additional information on this topic can be found at the same point in the German KOMA-Script book [Koh20a] only.

20.4. More or Less Obsolete Options and Commands

Currently, additional information on this topic can be found at the same point in the German KOMA-Script book [Koh20a] only.

Additional Information about the Main Classes scrbook, scrreprt, and scrartcl as well as the Package scrextend

This chapter provides additional information about the KOMA-Script classes scrbook, scrreprt, and scrartcl and some commands that are also available in scrextend. Some parts of the chapter are found only in the German KOMA-Script book [Koh20a]. This should not be a problem because the average user, who only wants to use the classes, will seldom need this information. Some of this information is addressed to users with non-standard requirements or who want to write their own classes based on a KOMA-Script class. Because such descriptions are not addressed to beginners, they are condensed and require deeper knowledge of LaTeX. Other parts describe features that exist only for the sake of compatibility with the standard classes or earlier versions of KOMA-Script.

Currently, additional information on this topic can be found at the same point in the German KOMA-Script book [Koh20a] only.

21.1. Extensions to User Commands

Currently, additional information on this topic can be found at the same point in the German KOMA-Script book [Koh20a] only.

21.2. KOMA-Script's Interaction with Other Packages

Currently, additional information on this topic can be found at the same point in the German KOMA-Script book [Koh20a] only.

21.3. Detection of KOMA-Script Classes

Package authors sometimes need to detect a KOMA-Script class. Mostly this is of little or no interest to the average user. To identify the version of KOMA-Script that is being used, refer to \KOMAScriptVersion in section 12.5, page 353.

\KOMAClassName
\ClassName

\KOMAClassName stores the name of the KOMA-Script class currently in use. So if you want to know if a KOMA-Script class is used, you can easily test for this command. In contrast, \ClassName indicates which standard class has been replaced by this KOMA-Script class.

Note in this context, however, that the existence of \KOMAScript cannot guarantee that a KOMA-Script class is in use. For one thing, all KOMA-Script packages define this command.

For another, other packages may find it useful to define the KOMA-Script label using this name.

21.4. Entries to the Table of Contents

KOMA-Script classes provide advanced methods for generating and manipulating entries in the table of contents. Some of these are based on the tocbasic package (see section 15.3 starting on page 385). Other are implemented directly in the classes.

`\raggedchapterentry`

v3.21

In earlier versions of KOMA-Script, it was possible to define the `\raggedchapterentry` macro as `\raggedright` to print chapter entries in the table of contents left-aligned instead of fully justified. Officially, this feature has not existed in KOMA-Script since version 3.21.

In fact, the `raggedentrytext` attribute for the `tocline` TOC-entry style of the tocbasic package is implemented by setting the `\ragged`*`entry level`*`entry` macro to either `\relax` or `\raggedright`. This attribute is evaluated by checking whether the corresponding macro is defined as `\raggedright`. If so, the text is printed left-aligned. With any other definition, no ragged margins are used.

Since it was previously documented that `\raggedchapterentry` should not be defined as anything other than `\raggedright`, this behaviour is compatible with the documented behaviour of earlier versions. As noted in earlier releases, other definitions of `\raggedchapterentry` — and now also of `\raggedsectionentry` and similar macros for other entry levels — may lead to unexpected results.

You should select the desired justification for the all table-of-contents entries using the `raggedentrytext` attribute of `tocline` rather than attempting to set the attribute for specific entry levels.

`\addtocentrydefault{`*`level`*`}{`*`number`*`}{`*`heading`*`}`

v3.08

The KOMA-Script classes do not use `\addcontentsline` directly to make entries in the table of contents. Instead, they call `\addtocentrydefault` with similar arguments. The command can be used for both numbered and unnumbered entries. The *`level`* is the sectioning level, that is `part`, `chapter`, `section`, `subsection`, `subsubsection`, `paragraph`, or `subparagraph`. The formatted sectioning number is given the second argument, *`number`*. This argument can be empty. The text of the entry is given by the *`heading`* argument. You should protect fragile commands inside this argument with `\protect`.

One notable feature of the *`number`* argument is that an empty argument indicates that an unnumbered entry should be generated. By default, KOMA-Script uses

```
\addcontentsline{toc}{level}{heading}
```

for this. If the argument is not empty, however, an entry with number will be created and *number* is the formatted heading number. KOMA-Script uses

```
\addcontentsline{toc}{level}{%
  \protect\numberline{number}heading%
}
```

to create this entry.

Package authors and authors of wrapper classes can redefine this command to alter the entries. For example, you could use

```
\renewcommand{\addtocentrydefault}[3]{%
  \Ifstr{#3}{}{%
    \Ifstr{#2}{}{%
      \addcontentsline{toc}{#1}{#3}%
    }{%
      \addcontentsline{toc}{#1}{\protect\numberline{#2}#3}%
    }%
  }%
}%
```

to omit entries with an empty *heading*. In practice, such a change is not necessary because the KOMA-Script classes already use another method to suppress empty entries. See the description of the sectioning commands in section 3.16, starting on page 100 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. By default, all these commands simply pass their own *level* and arguments directly to \addtocentrydefault.

21.5. Font Settings

KOMA-Script classes not only provide an extended selection of basic font sizes, but they also let you define, manipulate, and apply elements with their own font settings.

\@fontsizefilebase
\changefontsizes{*font size*}

The scrsize prefix for file names of font-size files described in section 21.1 is just the default for the internal \@fontsizefilebase macro. This default is used only when the macro is not yet defined when loading a KOMA-Script class or the scrextend package. Authors of wrapper classes can redefine this macro to use completely different font-size files. Similarly authors of wrapper classes can change or deactivate the fallback solution for unknown font sizes by redefining the \changefontsizes macro. This macro has exactly one argument: the desired *font size*. However, the \changefontsizes macro is not designed as an end-user instruction.

\newkomafont[*warning*]{*element*}{*default*}
\aliaskomafont{*alias*}{*element*}

Experts can use \newkomafont to define a *default* for the font style of an *element*. Subsequently, that default can be changed with the \setkomafont and \addtokomafont commands (see section 3.6, page 59). Of course, merely defining the font style does not actually put it to use. You must make sure you include the \usekomafont command (see page 59) for this *element* in your code at the appropriate places. Calling \newkomafont for an existing element will result in error messages.

The optional *warning* argument defines a warning message. The KOMA-Script classes output it with \ClassWarning, and the scrextend package with \PackageWarning, whenever the default font style of that *element* is changed. The package scrkbase is listed as the generator of the warning.

The \aliaskomafont command defines an *alias* for a previously defined *element*. KOMA-Script informs the user in the log file about the actual name of the element if it uses an *alias*. You can use an *alias*, for example, if you think of a better name for an element and the old name should remain usable for the sake of compatibility. It can also increase user-friendliness by creating aliases for all the terms that different users may intuitively choose. KOMA-Script itself makes use of this possibility.

\addtokomafontrelaxlist{*macro*}
\addtokomafontonearglist{*macro*}
\addtokomafontgobblelist{*macro*}

As already mentioned in part I of this manual, font settings of elements should consist only of commands to select the size, family, encoding, series, shape, or colour. Colour changes are not always transparent in LaTeX and therefore can cause in unwanted side-effects if you use \usekomafont at an inappropriate place.

Users tend to put very different, sometimes critical, things into the font setting of an element, such as \MakeUppercase at the very end of the setting. As much as possible, the internal use of the font settings has been implemented so that many of these prohibited elements still do

no harm, and it usually works even if the last command in a font setting expects an argument, for example using \textbf instead of \bfseries. But there is no guarantee for that.

Internally, KOMA-Script must sometimes limit font changes to real font settings. This is accomplished, for example, by using \usefontofkomafont instead of \usekomafont (see section 3.6, page 64).

Nevertheless, the \usefontofkomafont command and its siblings have their limitations. Therefore you must not use a command that always needs a fully expandable argument inside the font setting of an element. But this is exactly what \MakeUppercase needs. Therefore KOMA-Script maintains an internal list of macros that should become \relax inside \usefontofkomafont and its siblings. Since KOMA-Script 3.24 only \normalcolor is added to this list by default.

v3.17 v3.24

Note that the given *macro* is actually just set to \relax. So any arguments within the font setting will be executed locally, if applicable. Therefore you should never add commands like \setlength to the list. You are responsible for all errors resulting caused by using \addtokomafontrelaxlist. Also, do not take this command as license to add all sorts of commands to the font settings!

v3.24

For commands whose first argument should be executed without an additional group, there is \addtokomatfontonearglist. The specified macro is set to \@firstofone. By default \MakeUppercase and \MakeLowercase are added to this list.

v3.19

If, on the other hand, a *macro* and its first argument should be ignored locally inside \usefontofkomafont and its siblings, you can use \addtokomafontgobblelist instead of \addtokomafontrelaxlist. An example of this is the command \color, which must be ignored along with the colour name and therefore is a member of this list by default.

Note the defaults of these three lists may change in future version. If you need a certain commands in one of the lists, you should should explicitly add them yourself.

\IfExistskomafont{*element*}{*then code*}{*else code*}
\IfIsAliaskomafont{*element*}{*then code*}{*else code*}

v3.15

Which elements are defined depends on the version of KOMA-Script. So it sometimes makes sense to test in advance whether a specific *element* even exists. The \IfExistskomafont command executes the *then code* if and only if an *element* has been defined using \newkomafont or \aliaskomafont and therefore can also be changed using \setkomafont or \addtokomafont and can be used by one of the \use...komafont commands. Otherwise it executes the *else code*.

v3.25

In contrast, \IfIsAliaskomafont executes *then code* only if *element* has been defined with \aliaskomafont as an alias of another element. For undefined elements as well as for elements defined with \newkomafont, however, it executes the *else code*.

21.6. Paragraph Indention or Gap

Because the KOMA-Script classes offer extended features for setting the paragraph indention or gap, direct changes of the standard lengths `\parskip`, `\parindent` and `\parfillskip` are not only unnecessary but mostly inadvisable.

`\setparsizes{`*indent*`}{`*distance*`}{`*last-line end space*`}`

KOMA-Script provides the option to set the indent of the first line of a new paragraph, the distance between paragraphs, and the white space required at the end of the last line of each paragraph. You should use this command whenever the `parskip=relative` option should recognize these changes. KOMA-Script itself uses this command, for example, in the form

```
\setparsizes{0pt}{0pt}{0pt plus 1fil}
```

to eliminate both the paragraph indentation and inter-paragraph spacing, as well as to allow any amount of white space at the end of the last line of the paragraph. Such values are useful if a paragraph consists of only a box that should be printed without vertical spacing and filling the whole column width. If, on the other hand, the box should only span the whole width but use the current settings for indentation and distance between paragraphs, then

```
\setlength{\parfillskip}{0pt plus 1fil}
```

is preferable.

v3.17 Starting with KOMA-Script 3.17, recalculating or reactivating the type area or the margins (see chapter 2) also readjusts the values of `\setparsizes` if they have not been changed in the meantime. This is one more reason not to change these values without using KOMA-Script. Setting compatibility to a KOMA-Script version prior to 3.17 (see section 3.2, page 56, option `version`) disables this recalculation.

21.7. Counters

Currently, additional information on this topic can be found at the same point in the German KOMA-Script book [Koh20a] only.

21.8. Sections

The KOMA-Script classes provide a wide range of options for modifying sectioning levels and their corresponding headings. You can even define new levels.

```
\DeclareSectionCommand[attributes]{name}
\DeclareNewSectionCommand[attributes]{name}
\RedeclareSectionCommand[attributes]{name}
\ProvideSectionCommand[attributes]{name}
```

v3.15 With these commands you can either define a new sectioning command, \name, or modify an existing sectioning command, \name. To do so, you use the optional argument to set several *attributes*. The *attributes* are a comma-separated list of *key*=*value* assignments. In addition to the style-independent attributes shown in table 21.1, page 480, there are also attributes that depend on the style. Currently the following styles are available:

v3.18 `chapter`: The style for chapter headings. This style is used by default for \chapter and indirectly for \addchap. You can define new sectioning commands using this style, but then they do not automatically have an \add... variant. To configure existing or new sectioning commands, you can also use the attributes of table 21.3, page 482. The \addchap command, like the starred variants, is configured automatically with \chapter and cannot be modified independently. Note that scrartcl does not provide this style.

scrbook, scrreprt

v3.18 `part`: The style for part headings. This style is used by default for \part and indirectly for \addpart. You can define new sectioning commands using this style, but then they do not automatically have an \add... variant. To configure existing or new sectioning commands, you can also use the attributes of table 21.4, page 483. Note that the \addpart command, like the starred variants, is configured automatically with \part and cannot be modified independently.

`section`: The style for section headings. This style is currently used for \section, \subsection, \subsubsection, \paragraph, and \subparagraph. You can define new sectioning commands using this style. To configure existing or new sectioning commands, you can also use the attributes of table 21.2, page 481. When redefining a command, the *key*s `style`, `afterskip`, `beforeskip`, and `level` are mandatory. The keys `afterindent`,`font` and `indent` are recommended. `tocindent` and `tocnumwidth` may also be required depending on the command's *name*. This also applies if a command that was not previously a sectioning command is redefined as a sectioning command using \RedeclareSectionCommand. Note that the \addsec command and its starred variants are configured automatically with \section and cannot be changed independently.

v3.24 v3.26

Defining a sectioning command also creates an element with the same *name*, if it does not already exist. For `chapter` and `part`, elements are also created for the prefix line. You can change its font settings using \setkomafont or \addtokomafont (see section 3.6, page 59).

\DeclareNewSectionCommand defines a new sectioning command. If the same *name* is already used by TEX for something else, the command will result in an error message and will not define anything.

\ProvideSectionSommand behaves similarly but does not issue an error message.

Table 21.1.: Available *key*s and *value*s for style-independent *attributes* when declaring sectioning command

key	*value*	Description
`counterwithin`	*counter name*	The value of the counter of the heading should depend on *counter name*. Whenever `\stepcounter` or `\refstepcounter` increases the value of *counter name*, the value of the counter of this heading is reset to 0. In addition, `\the`*counter name* is prefixed with a dot when this counter is output.
v3.19 `counterwithout`	*counter name*	Cancels a prior `counterwithin` setting. Therefore it makes sense only if you redefine an existing sectioning command.
`expandtopt`	*switch*	If the switch is on, all subsequent values for lengths will be completely expanded, evaluated, and stored as `pt` values when they are defined. This means lengths no longer depend on the current font size. If the switch is off, all subsequent values for lengths will be tentatively expanded and evaluated but stored for full evaluation at application time. You can use any values from table 2.5, page 42. The default is `false`.
`level`	*integer*	The numerical value of the sectioning level (see the `secnumdepth` counter, section 3.16, page 112); the value should be unique and is mandatory for new levels.
`style`	*name*	Defines the style of the heading.
v3.20 `tocstyle`	*name*	Defines the style of the entries in the table of contents. You can use every previously defined TOC-entry style (see section 15.3). An empty *name* prevents a new definition of the TOC-entry command `\l@...`.
v3.20 `toc`*option*	*value*	Additional options depending on the TOC-entry style selected with `tocstyle`. See section 15.3, page 385 for additional information about TOC-entry styles. You can find the attributes that can be used as *option*s for the predefined TOC-entry styles of the tocbasic package in table 15.1, page 390.

Table 21.2.: Additional *key*s and *value*s for attributes when declaring a sectioning command with the `section` style

key	*value*	Description
v3.26 `afterindent`	*switch*	The *switch* determines whether the current paragraph indent is applied to the first line following a freestanding heading (see `runin`). With the default, `bysign`, the sign of `beforeskip` determines the behaviour. If the value of `beforeskip` is negative, the indentation of the first paragraph following the heading is suppressed. You can use any value from table 2.5, page 42 to explicitly enable or disable this indentation.
`afterskip`	*length*	In the case of a run-in heading (see `runin`), the absolute value of the *length* is the horizontal space after the heading. That is, a positive distance is always inserted. In the case of a freestanding heading, the *length* is the vertical skip below the heading. With `runin=bysign`, a positive value results in a freestanding heading, while a negative value or zero results in a run-in heading.
`beforeskip`	*length*	Specifies the vertical space before the heading. With `afterindent=bysign`, if the value is negative, a positive distance — the amount of *length* — is still inserted, but in this case the indentation of the paragraph following the heading is suppressed.
`font`	*font commands*	The font settings that should be used for the heading in addition to `disposition`. You can use all settings that are allowed for `\setkomafont` and `\addtokomafont` for the element of the heading.
`indent`	*length*	The indentation of heading from the left margin.
v3.26 `runin`	*switch*	Defines whether to use a run-in or a freestanding heading. With the default, `bysign`, the sign of `afterskip` determines the behaviour. In this case, a positive value of `afterskip` results in a freestanding heading. You can use any values from table 2.5, page 42 to explicitly enable or disable a run-in heading.

Table 21.3.: Additional *keys* and *values* for attributes when declaring a sectioning command with the `chapter` style

	key	*value*	Description
v3.26	`afterindent`	*switch*	The *switch* determines whether to apply the indentation of the first paragraph following a heading. With the default, `bysign`, the sign of `beforeskip` defines the behaviour. If the value of `beforeskip` is negative, the indentation of the paragraph following the heading is suppressed. You can use any value from table 2.5, page 42 to explicitly enable or disable this indentation.
v3.26	`afterskip`	*length*	The vertical skip below the heading.
	`beforeskip`	*length*	The vertical skip before the heading. With `afterindent=bysign`, if the value is negative, a positive distance — the amount of *length* — is still inserted, but in this case the indentation of the first paragraph following the heading is suppressed.
	`font`	*font commands*	The font setting to use for the heading in addition to `disposition`. You can use all settings, that are allowed for `\setkomafont` and `\addtokomafont` for this element.
	`innerskip`	*length*	The vertical skip between the prefix line and the heading's text, if a prefix line is used.
	`pagestyle`	*page style name*	The name of the page style to use automatically for pages with the heading. There is no check to see if the *page style name* is valid. Therefore, incorrect names will result in error messages when the sectioning command is used.
	`prefixfont`	*font commands*	The font setting to use for the prefix line of the heading, in addition to the `disposition` and sectioning command elements. You can use all settings that are allowed for `\setkomafont` and `\addtokomafont` for the element of the prefix line.

Table 21.4.: Additional *keys* and *value*s for attributes when declaring a sectioning command with the `part` style

key	*value*	Description
v3.26 `afterindent`	*switch*	The *switch* determines whether to apply the indentation of the first paragraph following a heading. With the value `bysign`, the sign of `beforeskip` determines the behaviour. If the value of `beforeskip` is negative, the indentation of the first paragraph following the heading is suppressed. You can use any value from table 2.5, page 42 to explicitly enable or disable this indentation. For compatibility, the default is `false` for scrartcl and `true` for scrbook and scrreprt.
`afterskip`	*length*	The value is the vertical skip below the heading.
`beforeskip`	*length*	The vertical skip before the heading. With `afterindent=bysign`, if the value is negative, a positive distance — the amount of *length* — is still inserted, but in this case the indentation of the first paragraph following the heading is suppressed.
`font`	*font commands*	The font setting to use for the heading in addition to `disposition`. You can use all settings that are allowed for `\setkomafont` and `\addtokomafont` for the element of the heading.
`innerskip`	*length*	The vertical skip between the prefix line and the heading text in scrbook and scrreprt.
`pagestyle`	*page style name*	The name of the page style to use automatically on pages with the heading. There is no check to see if *page style name* is valid. Therefore, incorrect names will result in error messages when the sectioning command is used. This feature only exists in scrbook andscrreprt.
`prefixfont`	*font commands*	The font setting to use for the prefix line of the heading, in addition to the `disposition` and sectioning command elements. You can use all settings that are allowed for `\setkomafont` and `\addtokomafont` for the element of the prefix line.

\RedeclareSectionCommand, on the other hand, can only change an existing command to a sectioning command with the specified *attributes*. It does not check whether *name* is already a sectioning command. It only needs to be a *name* already a TEXcommand.

\DeclareSectionCommand does not check whether or not *name* is an existing TEX command name. It just defines a sectioning command *name* with the specified *attributes*.

Each sectioning command also has a corresponding counter with the same *name* that is allocated with \newcounter if necessary. The same naming rule applies to the corresponding output of the counter (\the*name*), the counter's format (*name*format), the command to generate a running head (*name*mark), the format of the counter in the running head (*name*markformat), the font element (*name*), and the section-depth number (*name*numdepth). The command for the running head, *name*mark, is defined by default not to generate a running head. The default output of the counter \the*name* is an Arabic number. If the counter is defined as depending on another counter with the counterwithin *key*, the output of this counter will be prefixed by a dot.

v3.20 In addition to the sectioning command itself, a command for corresponding entries to the table of contents is also defined. This is done using the tocbasic package. The *tocstyle* attribute defines the style of those entries. If you set an empty *name*, e. g. using tocstyle= or tocstyle={}, the command for the TOC entry will not be changed. This is important, for example, if you use another package to modify the table of contents. If you do not set the tocstyle attribute, the previous style will be used again during the redefinition.

v3.20 The different TOC-entry styles also have different additional attributes. You can set them directly if you prefix them with toc. For example, you can set the level of the TOC entries, level, using toclevel, the indention, indent, using tocindent, or the number width, numwidth, using tocnumwidth. For more TOC-entry style attributes see section 15.3, page 385.

Example: For some unknown reason, you want to redefine the \paragraph headings so that they are no longer run-in headings but are similar to \subsubsection. The vertical skip above the heading should be 10 pt and no additional vertical skip below the heading. To do so, you can use:

```
\RedeclareSectionCommand[%
  beforeskip=-10pt,%
  afterskip=1sp%
]{paragraph}
```

The negative value of beforeskip creates a positive vertical skip before the heading and simultaneously disables the paragraph indentation of the following text. Even though the specification did not want any vertical skip after the heading, a value of 1 sp has been given here. This is because LATEX doesn't recognize 0 pt as positive value. So 1 sp is the smallest possible positive value.

Generally, it is better to have some tolerance for adjusting the vertical spacing, the so-called *glue*:

```
\RedeclareSectionCommand[%
  beforeskip=-10pt plus -2pt minus -1pt,%
  afterskip=1sp plus -1sp minus 1sp%
]{paragraph}
```

Note that the glue also switches the algebraic sign before becoming a skip, if the value is negative. That is the reason for the negative glue values in the example. Additionally we used the occasion to minimize the vertical skip after the heading using glue too.

In the example above, we only needed to define the keys `beforeskip` and `afterskip` because since v3.15 KOMA-Script has defined `\paragraph` internally using `\DeclareSectionCommand`, and therefore the other settings can be adopted unchanged. The original definition of `\paragraph` in scrartcl reads:

```
\DeclareSectionCommand[%
  level=4,
  indent=0pt,
  beforeskip=3.25ex plus 1ex minus .2ex,
  afterskip=-1em,
  font={},
  tocindent=7em,
  tocnumwidth=4.1em,
  counterwithin=subsubsection
]{paragraph}
```

scrreprt and scrbook use slightly different values.

Some settings of `\chapter` depend on the `headings` option (see section 3.16, page 96). Table 21.5 shows the default values of these settings. An overview of all settings is shown in table 21.6. For more information about the default of the TOC-entry styles see section 15.3, page 385. Note that `1ex` and `\baselineskip` depend on the default font size of the heading or the table of contents entry.

v3.20

Table 21.5.: Defaults for the chapter headings of scrbook and scrreprt depending on the headings option

With headings=big:

Attribute	Default Value
afterskip	1.725\baselineskip plus .115\baselineskip minus .192\baselineskip
beforeskip	-3.3\baselineskip-\parskip
font	\huge

With headings=normal:

Attribute	Default Value
afterskip	1.5\baselineskip plus .1\baselineskip minus .167\baselineskip
beforeskip	-3\baselineskip-\parskip
font	\LARGE

With headings=small:

Attribute	Default Value
afterskip	1.35\baselineskip plus .09\baselineskip minus .15\baselineskip
beforeskip	-2.8\baselineskip-\parskip
font	\Large

Table 21.6.: Defaults for the headings of scrbook and scrreprt

\part:

Attribute	Default Value
afterskip	0pt plus 1fil
beforeskip	0pt plus 1fil + \baselineskip
font	see element part, table 3.15, page 103
innerskip	20pt
level	-1
prefixfont	see element partnumber, table 3.15, page 103
tocindent	0pt
toclevel	-1
tocnumwidth	2em
tocstyle	part

...

Table 21.6.: Default for the headings of scrbook and scrreprt *(continued)*

\chapter:

Attribute	Default Value
afterskip	see table 21.5
beforeskip	see table 21.5
font	see element chapter, table 3.15, page 103
innerskip	0.5\baselineskip
level	0
prefixfont	see element chapterprefix, table 3.15, page 103
tocindent	0pt
toclevel	0
tocnumwidth	1.5em
tocstyle	chapter

\section:

Attribute	Default Value
afterskip	2.3ex plus .2ex
beforeskip	-3.5ex plus -1ex minus -.2ex
font	see element section, table 3.15, page 103
indent	0pt
level	1
tocindent	1.5em
toclevel	1
tocnumwidth	2.3em
tocstyle	section

...

Table 21.6.: Default for the headings of scrbook and scrreprt *(continued)*

\subsection:

Attribute	Default Value
afterskip	1.5ex plus .2ex
beforeskip	-3.25ex plus -1ex minus -.2ex
font	see element subsection, table 3.15, page 103
indent	0pt
level	2
tocindent	3.8em
toclevel	2
tocnumwidth	3.2em
tocstyle	section

\subsubsection:

Attribute	Default Value
afterskip	1.5ex plus .2ex
beforeskip	-3.25ex plus -1ex minus -.2ex
font	see element subsubsection, table 3.15, page 103
indent	0pt
level	3
tocindent	7.0em
tocnumwidth	4.1em
toclevel	3
tocstyle	section

\paragraph:

Attribute	Default Value
afterskip	-1em
beforeskip	3.25ex plus 1ex minus .2ex
font	see element paragraph, table 3.15, page 103
indent	0pt
level	4
tocindent	10em
toclevel	4
tocnumwidth	5em
tocstyle	section

...

Table 21.6.: Default for the headings of scrbook and scrreprt *(continued)*

\subparagraph:

Attribute	Default Value
afterskip	-1em
beforeskip	3.25ex plus 1ex minus .2ex
font	see element subparagraph, table 3.15, page 103
indent	\scr@parindent
level	5
tocindent	12em
toclevel	5
tocnumwidth	6em
tocstyle	section

Incidentally, the internal macro \scr@parindent used in the settings for \subparagraph is the paragraph indent set by the parskip option or the \setparsizes command.

```
\DeclareSectionCommands[attributes]{list of names}
\DeclareNewSectionCommands[attributes]{list of names}
\RedeclareSectionCommands[attributes]{list of names}
\ProvideSectionCommands[attributes]{list of names}
```

v3.15

These commands can define or change a whole series of sectioning commands at once. The names of the sectioning commands are given by the comma-separated *list of names*.

These commands differ in two other ways from the previously described commands that only define or change a single sectioning command. First, in case of error — that is if a command already exists with \DeclareNewSectionCommands or is undefined with \RedeclareSectionCommands — the definition will be performed regardless. An appropriate an error message will, of course, be reported anyway.

Second, there is another attribute, increaselevel=*integer*. This attribute changes the meaning of the attributes level and toclevel (see table 21.1) so that their values become starting values for the first sectioning command of the *list of names*. For all other sectioning command in the *list of names*, the values of level and toclevel are successively increased by the value of increaselevel. If the increaselevel attribute is used without assigning a value, 1 is assumed.

\IfSectionCommandStyleIs{*name*}{*style*}{*then code*}{*else code*}

v3.27 In rare cases, it is useful to be able to test whether a sectioning command belongs to a specific *style*. If KOMA-Script has defined the sectioning command *name* using the given *style*, the *then code* will be used. Otherwise the *else code* will be executed. An error is reported if *name* is undefined or if it is not a KOMA-Script sectioning command.

\chapterheadstartvskip
\chapterheadmidvskip
\chapterheadendvskip
\partheadstartvskip
\partheadmidvskip
\partheadendvskip
\partheademptypage

These commands are used inside the headings of the previously described chapter and part styles and thus for the definitions of \chapter, \part, \addchap, and \addpart, as well as their starred variants \chapter*, \part*, \addchap*, and \addpart*. The \chapterheadstartvskip command is intended to insert a vertical skip before the chapter heading. Similarly, \chapterheadendvskip is a command intended to insert a vertical v3.15 skip after the chapter heading. If the chapter heading has a prefix number line (see option chapterprefix in section 3.16, page 95), \chapterheadmidvskip is also used between the number line and the heading text.

The \partheadstartvskip and \partheadendvskip commands insert vertical skips above and below part headings. A page break is interpreted as part of the vertical distance. Such scrbook, scrreprt a page break is part of the default definitions of \partheadendvskip in scrbook and scrreprt. v3.02 The \partheademptypage command produces an empty page after the part heading page of scrbook and scrreprt.

v3.15 Starting with KOMA-Script 3.15, the defaults of these seven commands are independent from the headings option (see section 3.16, page 96). The default definitions for the chapter v3.17 headings starting with KOMA-Script 3.17 correspond to:

```
\newcommand*{\chapterheadstartvskip}{\vspace{\@tempskipa}}
\newcommand*{\chapterheadmidvskip}{\par\nobreak\vskip\@tempskipa}
\newcommand*{\chapterheadendvskip}{\vskip\@tempskipa}
```

These defaults are reactivated every time you use headings=big, headings=normal, or headings=small. As a side effect, these options may affect not only chapter titles but all headings in the chapter style

The chapter style automatically sets the internal length \@tempskipa to the value that results from the \DeclareSectionCommand attribute beforeskip before calling \chapterheadstartvskip. Similarly, it sets this length to the value of the

afterskip attribute before calling \chapterheadendvskip, and to innerskip before calling \chapterheadmidvskip.

The default values of the distances of \part do not depend on the headings option. So the corresponding commands will not be redefined by this option. Their default definitions in scrbook and scrreprt correspond to:

v3.17

```
\newcommand*{\partheadstartvskip}{%
  \null\vskip-\baselineskip\vskip\@tempskipa
}
\newcommand*{\partheadmidvskip}{%
  \par\nobreak
  \vskip\@tempskipa
}
\newcommand*{\partheadendvskip}{%
  \vskip\@tempskipa\newpage
}
```

and of scrartcl:

```
\newcommand*{\partheadstartvskip}{%
  \addvspace{\@tempskipa}%
}
\newcommand*{\partheadmidvskip}{%
  \par\nobreak
}
\newcommand*{\partheadendvskip}{%
  \vskip\@tempskipa
}
```

The part style once again sets the internal length \@tempskipa according to the settings of \DeclareSectionCommand before using the commands.

If you redefine one of the commands used for the vertical skip in the original \@tempskipa but still want to be able to configure the lengths, for example with \RedeclareSectionCommand, you should also use \@tempskipa in the new definition. Since you can more easily configure the distances above, within, and below the headings using \RedeclareSectionCommand, you generally should not redefine the commands described here. Changing them should be reserved for more complex changes that cannot be accomplished with \RedeclareSectionCommand. An example that redefines \chapterheadstartvskip and \chapterheadendvskip to print extra rules above and below the chapter heading can be found at [KDP] (in German).

v3.17

\partlineswithprefixformat{*level*}{*number*}{*text*}

This command is used by headings with the part style to output the heading number and heading text. The *number* and *text* arguments are already formatted, including the font selections. Ultimately, this command controls the arrangement of the two parts of the heading.

v3.25

For unnumbered headings, *number* is a completely empty argument, so it does not contain any formatting commands.

The default definition is rather Spartan:

```
\newcommand{\partlineswithprefixformat}[3]{%
  #2#3%
}
```

Example: You want to have part headings in a light blue box with blue frame. The box should occupy only about three quarters of full width of the text area. So you try:

```
\documentclass{scrbook}
\usepackage{xcolor}
\renewcommand*{\partlineswithprefixformat}[3]{%
  \fcolorbox{blue}{blue!25}{%
    \parbox{.75\linewidth}{#2#3}%
  }%
}
\begin{document}
\part{Framed Part}
\end{document}
```

But surprisingly the heading is not longer centred — neither the box itself nor the text inside the box.

The reason for the missing centring of the box is that the end of the paragraph is hidden in the third argument of the command. So it still finishes the paragraph of the text inside the box but not the paragraph of the `\parbox` itself. To solve this you add a `\par` at the end of the definition.

The reason for the missing centring inside the box is that the alignment of `\raggedpart` is valid outside the box but not automatically inside a `\parbox`. To solve this you add `\raggedpart` inside the box.

With

```
\documentclass{scrbook}
\usepackage{xcolor}
\renewcommand*{\partlineswithprefixformat}[3]{%
  \fcolorbox{blue}{blue!25}{%
    \parbox{.75\linewidth}{\raggedpart #2#3}%
  }%
}
\par
\begin{document}
\part{Framed Part}
\end{document}
```

you get the expected result.

As the example shows, users who redefine this command must watch out for several side effects. In addition to preserving the text alignment, they also must prevent page breaks within the headings, for example if they insert extra paragraphs or space. The example above does not have this problem. Not only does the box prevent a page breaks anyway, but KOMA-Script itself also changes `\interlinepenalty` as part of *text* so to prevent page breaks there. It also finishes *text* with an internal paragraph break using `\@@par`.

The default definition of `\partlineswithprefixformat` does not use the first argument, *level*, nor is it needed in the example above. It is of interest only if you want to define several commands with the `part` style and need to distinguish the different levels. The predefined commands `\part`, `\part*`, `\addpart`, and `\addpart*` all share the same *level part*.

`\chapterlineswithprefixformat{`*level*`}{`*number*`}{`*text*`}`
`\chapterlinesformat{`*level*`}{`*number*`}{`*text*`}`

v3.19

These commands are used by headings with the `chapter` style to output the heading number and heading text, depending on the `chapterprefix` option (see section 3.16, page 95). If the option is `true`, `\chapterlineswithprefixformat` is used. Otherwise `\chapterlinesformat` determines the ouput.

The *number* and *text* arguments are already formatted, including the font selections. Ultimately, these commands thus control the arrangement of the two parts of the heading. For unnumbered headings, the *number* argument is completely empty, so it does not contain any formatting commands.

The defaults for these commands are:

```
\newcommand{\chapterlinesformat}[3]{%
  \@hangfrom{#2}{#3}%
}
\newcommand{\chapterlineswithprefixformat}[3]{%
  #2#3%
}
```

Example: You want to have chapter headings with yellow background. For the headings without a prefix line, you use the following definition in the document preamble:

```
\makeatletter
\renewcommand{\chapterlinesformat}[3]{%
  \colorbox{yellow}{%
    \parbox{\dimexpr\linewidth-2\fboxrule-2\fboxsep}{%
      \@hangfrom{#2}#3%
    }%
  }%
}
```

```
\makeatother
```

For chapter headings with prefix line, you use:

```
\renewcommand{\chapterlineswithprefixformat}[3]{%
  \colorbox{yellow}{%
    \parbox{\dimexpr\linewidth-2\fboxrule-2\fboxsep}{%
      #2#3%
    }%
  }%
}
```

Unfortunately, you discover that these redefinitions result in justified text for the headings. The reason is the `\parbox` command. To change this, you can use the `\raggedchapter` command (see section 3.16, page 107) inside the argument of `\parbox`. Otherwise `\raggedchapter` would be used only before `\chapterlineswithprefixformat` and `\chapterlinesformat`:

```
\makeatletter
\renewcommand{\chapterlinesformat}[3]{%
  \colorbox{yellow}{%
    \parbox{\dimexpr\linewidth-2\fboxrule-2\fboxsep}{%
      \raggedchapter
      \@hangfrom{#2}#3%
    }%
  }%
}
\makeatother
\renewcommand{\chapterlineswithprefixformat}[3]{%
  \colorbox{yellow}{%
    \parbox{\dimexpr\linewidth-2\fboxrule-2\fboxsep}{%
      \raggedchapter
      #2#3%
    }%
  }%
}
```

Remember to use `\makeatletter` and `\makeatother` only in the document preamble. Do not use it inside your own wrapper class or package. They are only needed here because of `\@hangfrom` in the definition of `\chapterlinesformat`.

As the example shows, users who redefine this command must watch out for several side effects. In addition to preserving the text alignment, they also must prevent page breaks within the headings, for example if they insert extra paragraphs or space. The example above does not have this problem. Not only does the box prevent a page breaks anyway, but KOMA-Script itself also changes `\interlinepenalty` as part of *text* so to prevent page breaks there. It also finishes *text* with an internal paragraph break using `\@@par`.

Incidentally, the \raggedchapter command is not part of *text*, as otherwise using \MakeUppercase inside redefined versions of these two commands would be much more difficult. Note, however, that typographic rules require individual adjustments of letter spacing in capitalised text. However the LaTeX \MakeUppercase command does not do this.

The default definitions do not use the first argument, *level*, nor is it needed in the example above. It is of interest only if you want to define several commands with the chapter style and need to distinguish the different levels. The predefined commands \chapter, \chapter*, \addchap, and \addchap* all share the same *level chapter*.

\sectionlinesformat{*level*}{*indent*}{*number*}{*text*}
\sectioncatchphraseformat{*level*}{*indent*}{*number*}{*text*}

v3.19

These commands are used by headings with the section style depending on whether the reading is run-in or on its own line. Free-standing headings use \sectionlinesformat for their output, while run-in headings use \sectioncatchphraseformat.

In both cases *indent* is the value of the horizontal indention of the heading relative to the text area. If the value is negative, the heading can protrude into the left margin.

The arguments *number* and *text* are already formatted, including the font settings. Ultimately, these commands thus control the arrangement of the two parts of the heading. For unnumbered headings, the *number* argument is completely empty, so it does not contain any formatting commands.

The default definitions are:

```
\newcommand{\sectionlinesformat}[4]{%
  \@hangfrom{\hskip #2#3}{#4}%
}
\newcommand{\sectioncatchphraseformat}[4]{%
  \hskip #2#3#4%
}
```

If you redefine one of these commands, you are responsible for preventing page breaks inside heading. KOMA-Script itself only changes \interlinepenalty to impede page breaks.

Example: As in the previous example with chapter headings, the free-standing headings of the section *level* should now be printed with a background colour. Headings of lower levels should not be changed:

```
\makeatletter
\renewcommand{\sectionlinesformat}[4]{%
  \@tempswafalse
  \Ifstr{#1}{section}{%
    \hspace*{#2}%
    \colorbox{yellow}{%
      \parbox{\dimexpr\linewidth-2\fboxrule-2\fboxsep-#2}{%
        \raggedsection
```

```
        \@hangfrom{#3}{#4}%
      }%
    }%
  }{%
    \@hangfrom{\hskip #2#3}{#4}%
  }%
}
\makeatother
```

With this code, area of the indention is not coloured if the heading is indented. If, however, the heading is placed in the left margin with negative indention, this area of the margin is also highlighted. You can move the `\hspace*` command into the `\colorbox` to change this behaviour.

Again, remember to use `\makeatletter` and `\makeatother` only in the document preamble. You should omit them inside your own wrapper class or package. They are only needed because of `\@hangfrom` in the definition of `\sectionlinesformat`.

The first argument, *level*, is not used by the default definition. The example shows how to use it to distinguish different heading levels of the same style `section`.

```
\ExecuteDoHook{heading/preinit/name}
\ExecuteDoHook{heading/postinit/name}
\ExecuteDoHook{heading/branch/star/name}
\ExecuteDoHook{heading/branch/nostar/name}
\ExecuteDoHook{heading/begingroup/name}
\ExecuteDoHook{heading/endgroup/name}
```

v3.27 In addition to all their attributes, \DeclareSectionCommand, \DeclareNewSectionCommand, \ProvideSectionCommand and \RedeclareSectionCommand provide several hooks in sectioning commands that can be manipulated using \AddtoDoHook. For information about the functionality of these *do-hooks* see section 12.8, page 354. The last element of the specifier is the *name* of the sectioning command, as it is for the declaration commands mentioned above.

It is important to avoid adding code to these hooks that would affect the page breaking or the position of the heading. Therefore only advanced users should use these hooks. In case of doubt, refer to the class's source code for information about exactly when a hook is executed. These hooks serve as a fallback before you indeed would need to resort to redefining a section command without using KOMA-Script features.

The hook `heading/preinit/`*name* is executed immediately before the sectioning command is initialised. At this point, no settings have been initialised. Even the paragraph that precedes the heading is not necessarily finished yet.

The hook `heading/postinit/`*name* is executed somewhat later. At this point, some settings have been initialised and the previous paragraph has already ended.

Only one of the hooks `heading/branch/nostar/`*name* or `heading/branch/star/`*name* is executed, after determining whether the starred or normal variant of the section command is used. At this point, the vertical gap above the heading has already been inserted.

The hook `heading/begingroup/`*name* is executed at the beginning of the group used to process the heading. This is also the last opportunity to intervene before the heading is printed.

Similarly, the hook `heading/begingroup/`*name* will be executed at the end of the group used to process the heading. Currently this is the last hook inside a section command.

Note that `\minisec` is not a real sectioning command and therefore these hooks are not available for it.

`\IfUseNumber{`*then code*`}{`*else code*`}`

v3.27

Strictly speaking, this is an internal command. It is defined inside sectioning commands only between the `.../begingroup/...` and `.../endroup/...` hooks. In this case, the `{`*then code*`}` is executed if the sectioning heading uses the non-star variant and should be numbered according to the value of `secnumdepth`. If the heading should not be numbered because the sectioning command uses the star variant or because of the value of `secnumdepth`, the *else code* is executed. Headings in the `chapter` style also take into account whether the headings appears in the main matter.

If you use the command outside a sectioning command, the result is unspecified. In this case, it will usually result in an error message and execute neither the *then code* nor the *else code*.

```
\SecDef{star command}{command}
\scr@startsection{name}{level}{indent}{beforeskip}{afterskip}{style commands}
                 [short version]{heading}
\scr@startsection{name}{level}{indent}{beforeskip}{afterskip}{style commands}*
                 {heading}
```

v3.15

As already explained in section 3.16 in the description of the sectioning commands beginning with page 100, KOMA-Script provides additional features for the optional argument of those commands. To achieve this, it was necessary to replace some LaTeX kernel commands:

- Instead of `\@startsection` KOMA-Script uses `\scr@startsection`. However, the definition of `\@startsection` is checked. If the implementation differs from the expected one, a warning is issued, several features of KOMA-Script are deactivated, and `\scr@startsection` uses a copy of `\@startsection` while `\@startsection` is redefined to use `\scr@startsection`.
- Instead of `\@dblarg` KOMA-Script uses an internal command to define sectioning commands.

- Instead of \secdef, KOMA-Script uses \SecDef to change \@dblarg as mentioned above. If the implementation of \secdef differs from the expected one, a warning is issued.
- \@sect is redefined to implement various extensions of KOMA-Script.
- \@ssect is redefined to implement various extensions of KOMA-Script.
- \@xsect is redefined to implement various extensions of KOMA-Script.

In future versions of KOMA-Script, consideration is being given to avoid changing the LaTeX kernel commands mentioned above and to replace them completely with custom, internal commands. Loading truly incompatible sectioning packages would automatically disable KOMA-Script extensions and transfer the responsibility for the sectioning commands entirely to these packages. On the other hand, extra effort would be required to maintain compatibility with other packages.

Package authors can use these commands as they would use the corresponding LaTeX kernel commands and therefore gain access to the additional features of KOMA-Script. However, these commands should not be redefined, as they may changed at any time, and then this redefinition could compromise KOMA-Script's functionality. The meaning of the parameters for these commands can be found in the LaTeX kernel manual [BCJ+05]. As an alternative to redefining such commands, KOMA-Script offers the hooks described previously.

v3.27

\At@startsection{*code*}
\Before@ssect{*code*}
\Before@sect{*code*}

v3.27

Until KOMA-Script v3.26b, these commands were available as an alternative to redefine \scr@startsection or \SecDef. Since KOMA-Script v3.27, these commands are deprecated.

Internally \At@startsection is now implemented with the heading/postinit hook. \Before@ssect is implemented with heading/branch/star, and \Before@sect uses heading/branch/nostar. The *code* is added using \AddtoDoHook. There are no provisions to remove this code once it has been added.

\appendixmore

The KOMA-Script classes have an unusual feature within the \appendix command. If the \appendixmore command is defined, \appendix will also execute this it. Internally, the KOMA-Script classes scrbook and scrreprt take advantage of this behaviour to implement the appendixprefix layout option (see section 3.16, page 95). You should take note of this in case you decide to define or redefine the \appendixmore macro. If this option has been used, you will receive an error message when using \newcommand{\appendixmore}{...}. This behaviour is intended to prevent you from overriding the options without realising 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 the chapterprefix layout option in section 3.16, page 95). For consistency, you also do not want such a line in the appendix either. Instead, you would like to see the word "Chapter" in the language of your choice written in front of the chapter letter and, simultaneously, in the page headings. Instead of using the appendixprefix layout option, you define the following 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 for the appendix of the class scrartcl. For example, you can write in the preamble of your document:

```
\newcommand*{\appendixmore}{%
  \renewcommand*{\sectionformat}{%
    \appendixname~\thesection\autodot\enskip}%
  \renewcommand*{\sectionmarkformat}{%
    \appendixname~\thesection\autodot\enskip}}
```

The redefined commands are explained in more detail in section 3.16, page 108 and page 111.

21.9. Bibliography

The information in this section is less important if you use packages like biblatex. In that case, the extensive features of such packages will supersede the extensions of the KOMA-Script classes described here.

\newbibstyle[*parent style*]{*name*}{*commands*}
\newblock
\@openbib@code
\bib@beginhook
\bib@endhook

The standard classes already provide the \newblock command to structure bibliography entries. The exact purpose of this command depends on the class options. Using option openbib redefines the commands \@openbib@code and \newblock itself at the end of the standard class. The standard classes execute the \@openbib@code command when starting the bibliography list, or more precisely, when defining the parameters for the list. You can assume that many packages which redefine the bibliography will execute this command.

The KOMA-Script classes do something similar. However, they do not redefine \@openbib@code at the end of the class. Instead, \newbibstyle defines the openstyle bibliography style. The *commands* given in the implementation contain the appropriate redefinition of \@openbib@code and \newblock. If you select this bibliography style using the bibliography=openstyle option, the *commands* will be executed immediately. This will redefine \@openbib@code and \newblock.

In addition to \@openbib@code and \newblock, \bib@beginhook and \bib@endhook can also be redefined by the *commands* of the style. The \bib@beginhook command is executed immediately after the heading and preamble of the bibliography but before the beginning of the list of bibliographic entries. The \bib@endhook command will be executed immediately after this list, at the end of the bibliography. If \BreakBibliography (see section 3.23, page 148) is used to interrupt the bibliography, these commands will also executed at the beginning and end of each part of the bibliography, immediately before and after \BreakBibliography.

The \newblock, \@openbib@code, \bib@beginhook, and \bib@endhook commands are initially defined to be empty when using a new bibliography style. After this, the *commands* of the *parent style* that was optionally specified when defining the style will be executed, followed by the *commands* for the bibliography style itself. As a result, these commands must be defined, if necessary, with \renewcommand, not \newcommand, inside the *instructions* argument.

If you use the \AtEndBibliography and \AfterBibliographyPreamble commands to declare additional *commands* to be executed after the preamble or at the end of the bibliography, the *commands* specified with \AfterBibliographyPreamble will only be executed once, at the beginning of the bibliography after the \bib@beginhook *commands*, and the *commands* of \AtEndBibliography will be executed only once at the end of the bibliography, before \bib@endhook.

For example, the multicol package (see [Mit11]) could be used to define a bibliography style for printing the bibliography with two columns:

```
\newbibstyle{twocolumstyle}{%
  \renewcommand*{\bib@beginhook}{\begin{multicols}{2}}%
```

```
    \renewcommand*{\bib@endhook}{\end{multicols}}}%
```

If you also want to define an open variation of this style, you can use the possibilities of inheritance here and specify a *parent style*:

```
\newbibstyle{twocolumopenstyle}[openstyle]{%
  \renewcommand*{\bib@beginhook}{\begin{multicols}{2}}%
  \renewcommand*{\bib@endhook}{\end{multicols}}}%
```

You can then select these newly defined styles with the `bibliography` option as usual.

Like `\BreakBibliography`, these commands lose all or part of their effect when `thebibliography` is redefined, for example by using biblatex.

21.10. More or Less Obsolete Options and Commands

Currently, additional information on this topic can be found at the same point in the German KOMA-Script book [Koh20a] only.

Additional Information about the scrlttr2 Class and the scrletter Package

This chapter provides additional information about the KOMA-Script class scrlttr2. Some parts of the chapter are found only in the German KOMA-Script book [Koh20a]. This should not be a problem, because the average user, who only wants to use the class or package, will not normally need this information. Part of this information is addressed to users for whom the default options are insufficient. Thus, for example, the first section describes in detail the pseudo-lengths that specify the letterhead page and which can be used be used to modify the its layout.

v3.15 Starting with KOMA-Script 3.15, you can use the scrletter package with one of the KOMA-Script classes scrartcl, scrreprt, or scrbook. It provides nearly all the features of scrlttr2 for those classes. There are, however, a few differences described later in this chapter.

22.1. Variables for Experienced Users

KOMA-Script provides commands not only to use predefined variables but also to define new variables or to change their automatic use within the reference line.

```
\newkomavar[description]{name}
\newkomavar*[description]{name}
\addtoreffields{name}
\removereffields
\defaultreffields
```

`\newkomavar` defines a new variable. This variable is referenced as *name*. Optionally, you can define a *description* for the *name* variable. Unlike the *name*, the *description* is not used to reference a variable. Instead, the *description* acts as a supplement to the content of a variable that can be printed as a label along with its content.

You can use the `\addtoreffields` command to add the *name* variable to the reference line (see section 4.10, page 211). The *description* and the content of the variable are added to the end of the reference line. The starred version `\newkomavar*` is similar to the unstarred version but also calls the `\addtoreffields` command. Thus, the starred version automatically adds the variable to the reference line.

Example: Suppose you need an additional field for a telephone extension in the reference line. You can define this field with

```
\newkomavar[Extension]{myphone}
\addtoreffields{myphone}
```

or more concisely with

```
\newkomavar*[Extension]{myphone}
```

When you define a variable for the reference line, you should always give it a description.

You can use the `\removereffields` command to remove all variables from the reference field. This includes the predefined variables of the class. The reference 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 `\defaultreffields` command resets the reference fields line to its predefined format. In doing so, all custom-defined variables are removed from the reference fields line.

You should not add the date to the reference line with the `\addtoreffields` command. Instead you should use the `refline` option to select whether the date should appear on the left or right side of the reference line, or not at all. These settings also affect the position of the date when no reference line is used.

`\usekomavar[`*command*`]{`*name*`}`
`\usekomavar*[`*command*`]{`*name*`}`

The `\usekomavar` and `\usekomavar*` commands are, like all commands where a starred version exists or which can take an optional argument, not fully expandable. Nevertheless, if you use them within `\markboth`, `\markright` or similar commands, you need not insert `\protect` beforehand. Of course this is also true for `\markleft` if you use the scrlayer-scrpage package. These commands cannot be used within commands that directly affect their argument, such as `\MakeUppercase`. To avoid this problem you can use commands like `\MakeUppercase` as an optional argument to `\usekomavar` or `\usekomavar*`. Then you will get the upper-case content of a variable with

```
\usekomavar[\MakeUppercase]{Name}
```

`\Ifkomavarempty{`*name*`}{`*true*`}{`*false*`}`
`\Ifkomavarempty*{`*name*`}{`*true*`}{`*false*`}`

It is important to know that the content 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.

Once again, this command cannot be used as the argument of `\MakeUppercase` or similar commands However, it is robust enough to be used as the argument of `\markboth` or `\footnote`, for example.

```
\foreachkomavar{list of variables}{command}
\foreachnonemptykomavar{list of variables}{command}
\foreachemptykomavar{list of variables}{command}
\foreachkomavarifempty{list of variables}{then-code}{else-code}
```

v3.27

The \foreachkomavar command executes the specified *command* for each variable in the comma-separated *list of variables*. The name of each variable is added as parameter to the *command*.

The \foreachnonemptykomavar command does the same but only for those variables that are not empty in sense of \Ifkomavarempty. Empty variables in the *list of variables* are ignored.

By contrast, the \foreachemptykomavar command executes the *command* only for variables that are empty in sense of \Ifkomavarempty. Accordingly, non-empty variables are ignored.

The \foreachkomavarifempty command is a kind of combination of the two previously described commands. It executes the *then-command* only for those variables in the *list of variables* that are empty, and the *else-command* for the non empty variables. As with *command*, the name of each variable is added as a parameter in both cases.

22.2. Additional Information about Page Styles

Currently, additional information on this topic can be found at the same point in the German KOMA-Script book [Koh20a] only.

22.3. lco Files for Experienced Users

Although you can use any paper size that the typearea package can configure, the output of the letterhead page may produce undesirable results with some formats. Unfortunately, there are no general rules to calculate the position of the address fields and the like for every available paper size. Instead, different parameter sets are needed for different paper sizes.

At present parameter sets and lco files exist only for A4-sized and letter-sized paper. Theoretically, however, the scrlttr2 class can support many more paper sizes. Therefore, it's necessary to verify that the correct paper size is used. This is even more true if you use scrletter, since the paper size depends on the class you use.

```
\LetterOptionNeedsPapersize{option name}{paper size}
```

To provide at least a warning when another *paper size* is used, you can 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 the following example shows, 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, that is upright or portrait, orientation for your letters. Let's 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 standard letter on A4-sized paper. The main difference is that A5 paper needs only one fold. So you want to disable the top and bottom fold marks. You can do this, for example, by placing the marks outside the paper area.

```
\ProvidesFile{a5.lco}
              [2002/05/02 letter class option]
\LetterOptionNeedsPapersize{a5}{a5}
\setplength{tfoldmarkvpos}{\paperheight}
\setplength{bfoldmarkvpos}{\paperheight}
```

Of course, it would be more elegant to deactivate the marks with the `foldmarks` option. In addition, you must adjust the position of the footer, that is, the `firstfootvpos` pseudo-length. I leave it to the reader to find an appropriate value. When using such an `lco` file, you must declare other `lco` file options like `SN` before you load "`a5.lco`".

visualize.lco

If you develop your own `lco` file, for example to modify the positions of various fields on the letterhead page because your own desires or requirements, it is helpful if you can make at least some elements directly visible. The `lco` file `visualize.lco` exists for this purpose. You can load this file as you would any other `lco` file. But this *letter class options* file must be loaded in the document preamble, and its effects cannot be deactivated. The `lco` file uses the eso-pic and graphicx packages, which are not part of KOMA-Script.

v3.04

\showfields{*field list*}

This command makes the space occupied by the fields on the letterhead page visible. The *field list* argument is a comma-separated list of fields to be shown. The following fields are supported:

`test` – is a 10 cm by 15 cm test field, 1 cm from the top and left edges of the paper. This field exists for debugging. You can use it as a benchmark to check whether the measurements have been distorted during the creation of the document.

`head` – is the header area of the letterhead page. This field is open at the bottom.

`foot` – is the footer area of the letterhead page. This field is open at the top.

`address` – is the address window area used by window envelopes.

`location` – is the field for the extra sender information.

`refline` – is the reference line. This field is open at the bottom.

You can change the colour of the visualisation with the`\setkomafont` and `\addtokomafont` (see section 4.9, page 179) commands using the `field` element. The default is `\normalcolor`.

`\setshowstyle{`*`style`*`}`
`\edgesize`

By default, `visualize.lco` indicates the individual areas with frames, which corresponds to the *`style`* `frame`. Areas open at top or bottom are not completely framed but have an open edge with with small arrows pointing up or down. Alternatively, you can use the *`style`* `rule`. In this case, the area is highlighted by a background colour. It isnot possible to distinguish open and closed areas. Instead a minimal height will be used for open areas. The third available *`style`* is `edges`, which shows 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 `\edgesize` macro with a default of 1 ex.

`\showenvelope(`*`width`*`,`*`height`*`)(`*`h-offset`*`,`*`v-offset`*`)[`*`instructions`*`]`
`\showISOenvelope{`*`format`*`}[`*`instructions`*`]`
`\showUScommercial{`*`format`*`}[`*`instructions`*`]`
`\showUScheck[`*`instructions`*`]`
`\unitfactor`

If you have loaded `visualize.lco`, you can use these commands to output a page with a drawing of an envelope. The envelope drawing is always rotated by 90° on a separate page and printed in 1:1 scale. The addressee window is generated automatically from the current data for the address position of the letterhead page: `toaddrvpos`, `toaddrheight`, `toaddrwidth`, and `toaddrhpos`. To do so requires knowing how much smaller the folded letter pages are than the width and height of the envelope. If you do not specify these two values, *`h-offset`* and *`v-offset`*, when calling `\showenvelope`, they are calculated from the fold marks and the paper size itself.

The `\showISOenvelope`, `\showUScommercial`, and `\showUScheck` commands are based on `\showenvelope`. With `\showISOenvelope`, you can create ISO-envelopes in C4, C5, C5/6, DL (also known as C5/6) or C6 *`format`*. With `\showUScommercial`, you can create a US commercial envelope in the 9 or 10 *`format`*. You can use `\showUScheck` for envelopes in US check format.

The position of the letterhead page inside the envelope is indicated with dashed lines. You can change the colour of these lines with the `\setkomafont` and `\addtokomafont` (see section 4.9, page 179) using the `letter` element. The default is `\normalcolor`.

The envelope drawing will be provided with dimensions automatically. You can change the colour of these dimension labels with the commands `\setkomafont` and `\addtokomafont`

(see section 4.9, page 179) using the `measure` element. The default is `\normalcolor`. The dimensions are given in multiples of `\unitlength`, with an accuracy of 1/`\unitfactor`, where the accuracy of TEX arithmetic is the actual limits. The default is 1. You can redefine `\unitfactor` using `\renewcommand`.

Example: You are generating a sample letter using the ISO-A4 format. The supported fields should be marked with yellow borders to check their position. Furthermore, the position of the window for a DL-size envelope should be checked with drawing. The dimension lines in this drawing should be red, and the numbers should use a smaller font, with the dimensions printed in cm with an accuracy of 1 mm. The dashed letterhead page in the envelope should be coloured 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[british]{babel}
\usepackage{lipsum}
\begin{document}
\setkomavar{fromname}{Joe Public}
\setkomavar{fromaddress}{2 Valley\\
                         SAMPLEBY\\
                         ZY32 1XW}
\begin{letter}{%
    1 Hillside\\
    SAMPLESTEAD\\
    WX12 3YZ%
  }
\opening{Hello,}
\lipsum[1]
\closing{Good bye}
\end{letter}
\setlength{\unitlength}{1cm}
\renewcommand*{\unitfactor}{10}
\showISOenvelope{DL}
\end{document}
```

This will show the letterhead page as the first page and the drawing of the envelope on the second page.

Note that poorly chosen combinations of `\unitlength` and `\unitfactor` can quickly lead to a TEX *arithmetic overflow* error. The dimension numbers shown may also differ slightly from

the actual values. Neither are errors in `visualize` but merely implementation limitations of TEX.

22.4. Language Support

The scrlttr2 class and the scrletter package support many languages. These include German (`german` for the old German orthography, `ngerman` for the new orthography; `austrian` for Austrian with the old German orthography, `naustrian` for Austrian with the new orthography; and `nswissgerman` for Swiss German with the new orthogrphy, `swissgerman` for Swiss German with the old orthography), English (among others, `english` without specification as to whether American or British should be used, `american` and `USenglish` for American English, and `british` and `UKenglish` for British English), French, Italian, Spanish, Dutch, Croatian, Finnish, Norwegian, Swedish, Polish, Czech, and Slovak.

v3.09 v3.13 v3.08

You can switch languages using the babel package (see [BB13]) with the `\selectlanguage{`*`language`*`}` command. Other packages like german (see [Rai98a]) and ngerman (see [Rai98b]) also define this command. As a rule though, the language selection occurs immediately as a direct consequence of loading such a package.

There is one more point to note about language-switching packages. The french package (see [Gau07]) makes changes well beyond redefining the terms in table 22.1. For instance, it redefines the `\opening` command, since the package simply assumes that `\opening` is always defined as it is in the standard letter class. This, however, is not the case with KOMA-Script. The french package thus overwrites the definition and does not work correctly with KOMA-Script. I regard this as a fault in the french package which, although reported decades ago, was unfortunately never eliminated.

If you use the babel package to switch to `french`, problems can occasionally occur. With babel, however, you can usually deactivate changes to a language in a targeted manner.

```
\yourrefname
\yourmailname
\myrefname
\customername
\invoicename
\subjectname
\ccname
\enclname
\headtoname
\headfromname
\datename
\pagename
\mobilephonename
\phonename
\faxname
\emailname
\wwwname
\bankname
```

These commands contain the language-dependent terms. These definitions can be modified to support a new language or for your private customization, as described in section 12.4. KOMA-Script sets these terms only in `\begin{document}`. Therefore they are not available in the preamble and cannot be redefined there. The default settings for `english` and `ngerman` are listed in table 22.1.

```
\captionsacadian
\captionsamerican
\captionsaustralien
\captionsaustrian
\captionsbritish
\captionscanadian
\captionscanadien
\captionscroatian
\captionsczech
\captionsdutch
\captionsenglish
\captionsfinnish
\captionsfrancais
\captionsfrench
\captionsgerman
\captionsitalian
\captionsnaustrian
\captionsnewzealand
\captionsngerman
\captionsnorsk
\captionsnswissgerman
\captionspolish
\captionsslovak
\captionsspanish
\captionsswedish
\captionsswissgerman
\captionsUKenglish
\captionsUSenglish
```

If you change the language of a letter, the language-dependent terms listed in table 22.1, page 511 are redefined using these commands. If your language-switching package does not support this, you can also use the above commands directly.

Table 22.1.: Defaults for language-dependent terms for the languages `english` and `ngerman`, if they are not already defined by the packages used for language switching

[l]

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
`\mobilephonename`	Mobile phone	Mobiltelefon
`\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, KOMA-Script does not redefine them. The actual wording may therefore differ and can be found in the documentation for the language package used.

```
\dateacadian
\dateamerican
\dateaustralien
\dateaustrian
\datebritish
\datecanadian
\datecanadien
\datecroatian
\dateczech
\datedutch
\dateenglish
\datefinnish
\datefrancais
```

Table 22.2.: Language-dependent forms of the date

Command	Date Example
\dateacadian	24. 12. 1993
\dateamerican	12/24/1993
\dateaustralien	24/12/1993
\dateaustrian	24. 12. 1993
\datebritish	24/12/1993
\datecanadian	1993/12/24
\datecanadien	1993/12/24
\datecroatian	24. 12. 1993.
\dateczech	24. 12. 1993
\datedutch	24. 12. 1993
\dateenglish	24/12/1993
\datefinnish	24.12.1993.
\datefrancais	24. 12. 1993
\datefrench	24. 12. 1993
\dategerman	24. 12. 1993
\dateitalian	24. 12. 1993
\datenaustrian	24. 12. 1993
\datenewzealand	24/12/1993
\datengerman	24. 12. 1993
\datenorsk	24.12.1993
\datenswissgerman	24. 12. 1993
\datepolish	24. 12. 1993
\dateslovak	24. 12. 1993
\datespanish	24. 12. 1993
\dateswedish	24/12 1993
\dateswissgerman	24. 12. 1993
\dateUKenglish	24/12/1993
\dateUSenglish	12/24/1993

can be found in table 22.2.

22.5. Obsolete Commands

Currently, additional information on this topic can be found at the same point in the German KOMA-Script book [Koh20a] only.

Japanese Letter Support for scrlttr2[1]

Since version 2.97e, scrlttr2 has provided 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 this support and provides a few examples of using the provided `lco` files to print 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 following 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 stationery sizes, listed in table A.3. You may come across these when buying stationery.

The ISO C-series is not a paper size as such but a standard developed for envelopes and intended for the corresponding A-series paper. It is discussed in the next subsection.

[1] This chapter was originally written 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 millimetres 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 non-standard 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 stationery

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, 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; non-standard
C1	648×917	for flat A1 sheet; non-standard
C2	458×648	for flat A2 sheet, A1 sheet folded in half; non-standard
C3	324×458	for flat A3 sheet, A2 sheet folded in half; non-standard
B4	250×353	C4 envelope
C4	229×324	for flat A4 sheet, A3 sheet folded in half; very common; non-standard
B5	176×250	C5 envelope
C5	162×229	for flat A5 sheet, A4 sheet folded in half; very common; non-standard
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; non-standard
C7	81×114	for A5 sheet folded in quarters; uncommon; non-standard
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; non-standard
Yes	Chou 2	119×277	for B5 folded in half lengthwise; non-standard
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 ; non-standard
	Kaku 0	287×382	for B4 flat, B3 folded in half; non-standard
	Kaku 1	270×382	for B4 flat, B3 folded in half; non-standard
Yes	Kaku 2	240×332	for A4 flat, A3 folded in half; non-standard
	Kaku Kokusai A4	229×324	for A4 flat, A3 folded in half; same size as ISO C4; non-standard
Yes	Kaku 3	216×277	for B5 flat, B4 folded in half; non-standard
Yes	Kaku 4	197×267	for B5 flat, B4 folded in half; non-standard
Yes	Kaku 5	190×240	for A5 flat, A4 folded in half ; non-standard
Yes	Kaku 6	162×229	for A5 flat, A4 folded in half; same size as ISO C5; non-standard
Yes	Kaku 7	142×205	for B6 flat, B5 folded in half; non-standard
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); non-standard
	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 fold marks required for different envelope sizes and subvariants with different window positions and sizes.

The `lco` files provided together with the envelope types that they 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, 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 catalogue.
[2] Given as width by height in millimetres.
[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

Suppose 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 notice that the window dimensions and positions of your particular windowed envelope differ significantly (more than approximately 2 mm) from those of any of the supported subvariants, please contact the author of KOMA-Script to obtain support as soon as possible, and in the meantime, 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 fold marks as required. Another, independent, mark is the hole-punch 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

In this list of changes, you will find all significant changes to the user interface of the KOMA-Script bundle at the last few versions. The list was sorted by the names of the classes and packages and their version. The numbers after the version are the pages where the changes are described. In the margins of these pages, you will find corresponding version marks.

Bibliography

In the following, you will find many references. All of them are referenced in the main text. In many cases the reference points to documents or directories which can be accessed via the 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 replace the prefix with https://mirror.ctan.org/. For LaTeX packages, it is also important to mention that we have tried to give a version number appropriate to the text that cites the reference. But for some packages 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, be sure that the package is the most up-to-date version and check first if the package is already available on your system.

[Ame02] American Mathematical Society:
User's guide for the amsmath package, February 2002.
CTAN://macros/latex/required/amslatex/math/.

[BB13] Johannes Braams and Javier Bezos:
Babel, December 2013.
CTAN://macros/latex/required/babel/.

[BCJ+05] Johannes Braams, David Carlisle, Alan Jeffrey, Leslie Lamport, Frank Mittelbach, Chris Rowley, and Rainer Schöpf:
The LaTeX2e Source, December 2005.

[BJ04] Johannes L. Braams and Theo Jurriens:
The supertabular environment, February 2004.
CTAN://macros/latex/contrib/supertabular.

[Car99a] David Carlisle:
The keyval package, March 1999.
CTAN://macros/latex/required/graphics/.

[Car99b] David Carlisle:
The tabularx package, January 1999.
CTAN://macros/latex/required/tools/.

[Car04] David Carlisle:
The longtable package, February 2004.
CTAN://macros/latex/required/tools/.

[Car17] David P. Carlisle:
Packages in the 'graphics' bundle, June 2017.
CTAN://macros/latex/required/graphics/.

[Dal10] Patrick W. Daly:
Natural sciences citations and references, September 2010.
CTAN://macros/latex/contrib/natbib/.

[DUD96] DUDEN:
Die deutsche Rechtschreibung. DUDENVERLAG, Mannheim, 21st edition, 1996.

[Fai11] Robin Fairbairns:
footmisc — a portmanteau package for customising footnotes in LaTeX, June 2011.
CTAN://macros/latex/contrib/footmisc/.

[FAQ13] *Tex frequently asked questions on the web*, June 2013.
http://www.tex.ac.uk/faq.

[Fel17] Felix999:
Paket adrconf verlangt nicht vorhandene Datei scrpage.sty, December 2017.
https://komascript.de/node/2154.

[Gau07] Bernard Gaulle:
Les distributions de fichiers de francisation pour latex, May 2007.
CTAN://language/french/.

[Gre12] Enrico Gregorio:
The xpatch package, extending etoolbox patching commands, October 2012.
CTAN://macros/latex/contrib/xpatch/.

[KDP] *KOMA-Script Homepage*.
http://www.komascript.de.

[Keh97] Roger Kehr:
XINDY, A Flexible Indexing System, 1997.

[Ker07] Dr. Uwe Kern:
Extending LaTeX's color facilities: the xcolor package, January 2007.
CTAN://macros/latex/contrib/xcolor/.

[Kie10] Axel Kielhorn:
adrconv, April 2010.
CTAN://macros/latex/contrib/adrconv/.

[Knu90] Donald E. Knuth:
The TeXbook, volume A of *Computers and Typesetting*. Addison-Wesley Publishing Company, Reading, Mass., 19th edition, 1990.

[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.

[Koh12] Markus Kohm:
Non-floating margin notes with marginnote, March 2012.
CTAN://macros/latex/contrib/marginnote/.

[Koh14] Markus Kohm:
Creating more than one index using splitidx and splitindex, April 2014.
CTAN://macros/latex/contrib/splitindex/.

[Koh15] Markus Kohm:
chapteratlists=entry automatisch nur für Kapitel mit Einträgen, July 2015.
https://komascript.de/comment/5070#comment-5070.

[Koh20a] Markus Kohm:
KOMA-Script. Edition DANTE. Lehmanns Media, Berlin, 7th edition, 2020, ISBN 978-3-96543-097-6. Print-Ausgabe.

[Koh20b] Markus Kohm:
KOMA-Script. Edition DANTE. Lehmanns Media, Berlin, 7th edition, 2020, ISBN 978-3-96543-103-4. eBook-Ausgabe.

[Lam87] Leslie Lamport:
MakeIndex: An index processor for LaTeX, February 1987.
CTAN://indexing/makeindex/doc/makeindex.pdf.

[Lap08] Olga Lapko:
The floatrow package, August 2008.
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, November 2001.
CTAN://macros/latex/contrib/float/.

[Mit11] Frank Mittelbach:
An environment for multicolumn output, June 2011.
CTAN://macros/latex/required/tools/.

[Nie15] Rolf Niepraschk:
The eso-pic package, July 2015.
CTAN://macros/latex/contrib/eso-pic/.

[Obe10] Heiko Oberdiek:
The engord package, March 2010.
CTAN://macros/latex/contrib/oberdiek/.

[OPHS11] Tobias Oetker, Hubert Partl, Irene Hyna, and Elisabeth Schlegl:
The Not So Short Introduction to LaTeX 2ε, April 2011.
CTAN://info/lshort/english/.

[Pac] Jean Marie Pacquet:
KomaLetter2; Example by Jean-Marie Pacquet (French style). Wiki.
http://wiki.lyx.org/Examples/KomaLetter2#toc6.

[Rai98a] Bernd Raichle:
german package, July 1998.
CTAN://language/german/.

[Rai98b] Bernd Raichle:
ngerman package, July 1998.
CTAN://language/german/.

[Sch09] Martin Schröder:
The ragged2e package, June 2009.
CTAN://macros/latex/contrib/ms/.

[Sch13] R Schlicht:
The microtype package: An interface to the micro-typographic extensions of pdfTeX, May 2013.
CTAN://macros/latex/contrib/microtype/.

[Som13] Axel Sommerfeldt:
Anpassen der Abbildungs- und Tabellenbeschriftungen mit Hilfe des caption-Paketes, May 2013.
CTAN://macros/latex/contrib/caption/.

[Tea98] The NTS Team:
The ε-TeX manual, February 1998.
CTAN://systems/e-tex/v2/doc/etex_man.pdf.

[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.

[TF11] Geoffrey Tobin and Robin Fairbairns:
setspace LaTeX package, December 2011.
CTAN://macros/latex/contrib/setspace/.

[The19] The LaTeX Project Team:
The iftex package, November 2019.
CTAN://macros/latex/contrib/iftex.

[Tsc87] Jan Tschichold:
Ausgewählte Aufsätze über Fragen der Gestalt des Buches und der Typographie.
Birkhäuser Verlag, Basel, 2nd edition, 1987.

[Ume10] Hideo Umeki:
The geometry package, September 2010.
CTAN://macros/latex/contrib/geometry/.

[vO04] Piet van Oostrum:
Page layout in LaTeX, March 2004.
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 in this index. The numbers in bold show the pages where the topic is defined or explained. The numbers in ordinary type show the pages of where the topic is mentioned.

General Index

Index of Commands, Environments, and Variables

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Index of Lengths and Counters

Index of Elements Capable of Adjusting Fonts

Index of Files, Classes, and Packages

Index of Class and Package Options

M

N

O

P

R

S

T

U

V

Index of Do-Hooks

H