KOMA-Script

a versatile LaTeX 2ε bundle

Note: This document is part of KOMA-Script 3 but was written for KOMA-Script 2.98. Several features of KOMA-Script 2 are obsolete (but may still be used) with KOMA-Script 3. In this case this documentation sometimes describes obsolete things. Several features of KOMA-Script 3 are new but may not be found at this manual. This manual will be updated with respect to the available ressources. Any request for help shall be sent to komascript at gmx.info.

The Guide

KOMA-Script

Markus Kohm Jens-Uwe-Morawski

2012-01-01

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

Legal Notes:

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

Free screen version without any optimization of paragraph and page breaks

To All Friends of Typography!

Contents

List of Tables

Introduction

1.1. Preface

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

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

1.2. Structure of the Guide

The KOMA-Script user guide is not intended to be a LaTeX primer. Those new to LaTeX should look at *The Not So Short Introduction to LaTeX 2ε* [OPHS99] or *LaTeX 2ε for Authors* [Tea05b] or a LaTeX reference book. You will also find useful information in the many LaTeX FAQs, including the *TeX Frequently Asked Questions on the Web* [FAQ11].

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

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

Like this.

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

1.3. History of KOMA-Script

In the early 1990s, Frank Neukam needed a method to publish an instructor's lecture notes. At that time LaTeX was LaTeX2.09 and there was no distinction between classes and packages — there were only *styles*. Frank felt that the standard document styles were not good enough for his work; he wanted additional commands and environments. At the same time he was

interested in typography and, after reading Tschichold's *Ausgewählte Aufsätze über Fragen der Gestalt des Buches und der Typographie* (Selected Articles on the Problems of Book Design and Typography) [Tsc87], he decided to write his own document style — and not just a one-time solution to his lecture notes, but an entire style family, one specifically designed for European and German typography. Thus Script was born.

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

Beginning in mid-1994, LaTeX 2ε became available and brought with it many changes. Users of Script-2 were faced with either limiting their usage to LaTeX 2ε's compatibility mode or giving up Script altogether. This situation led Markus to put together a new LaTeX 2ε 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 und scrpage2. Many thanks to all who encouraged me to go on, to make things better and less error-prone, or to implement additional features.

Thanks go as well to DANTE, Deutschsprachige Anwendervereinigung TeX e.V, (the German-Language TeX User Group). Without the DANTE server, KOMA-Script could not have been released and distributed. Thanks as well to everybody in the TeX newsgroups and

mailing lists who answer questions and have helped me to provide support for KOMA-Script.

1.5. Legal Notes

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

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

1.6. Installation

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

1.7. Bugreports and Other Requests

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

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

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

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

1.8. Additional Information

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

Construction of the Page Layout with typearea

Many LaTeX classes, including the standard classes, present the user with the largely fixed configuration of margins and typearea. With the standard classes, the configuration determined is very much dependent on the chosen font size. There do exist separate packages, such as geometry (see [Ume00]), which give the user complete control, but also full responsibility, for the settings of typearea and margins.

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

It should be noted that the typearea package makes use of the scrbase package. The latter is explained in the expert section of this document in **??** from **??** onwards. The majority of the rules documented there are however not directed at the user, but rather at class- and package authors.

2.1. Fundamentals of Page Layout

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

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

In the previous paragraph, we mentioned and emphasized *the page*. Erroneously, it is often thought that with the page format the page is also meant the paper format. However, if you look at a bound document, it is obvious that part of the paper vanishes in the binding and is no longer part of the visible page. For the page layout, it is not the format of the paper which is important, it is the impression of the visible page to the reader. Therefore, it is clear that the calculation of the page layout must account for the "lost" paper in the binding and add this amount to the width of the inner margin. This is called the *binding correction*. The binding correction is therefore calculated as part of the *gutter*, not however of the visible inner margin.

The binding correction depends on the process of actually producing the document and thus

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

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

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

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

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

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

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

With leading is meant the vertical distance between individual lines of text. In LaTeX, the leading is set at about 20% of the font size. With commands like `\linespread` or, better, packages like

setspace (see [Tob00]), the leading can be changed. A wider leading helps the eye to follow the line. A very wide leading, on the other hand, disturbs reading because the eye has to move a wide distance between lines. Also, the reader becomes uncomfortable because of the visible stripe effect. The uniform gray value of the page is thereby spoiled. Still, with a wider leading, the lines can be longer.

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

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

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

2.2. Page Layout Construction by Dividing

The easiest way to make sure that the text area has the same ratios as the page is as follows: first, one subtracts the part *BCOR*, required for the binding correction, from the inner edge of the paper, and divides the rest of the page vertically into *DIV* rows of equal height; next, one divides the page horizontally into the same number (*DIV*) of columns; then, one takes the uppermost row as the upper margin and the two lowermost rows as the lower margin (if one is printing double-sided, one must similarly take the innermost column as the inner margin and the two outermost columns as the outer margin); then, one adds the binding correction *BCOR* to the inner margin. What now

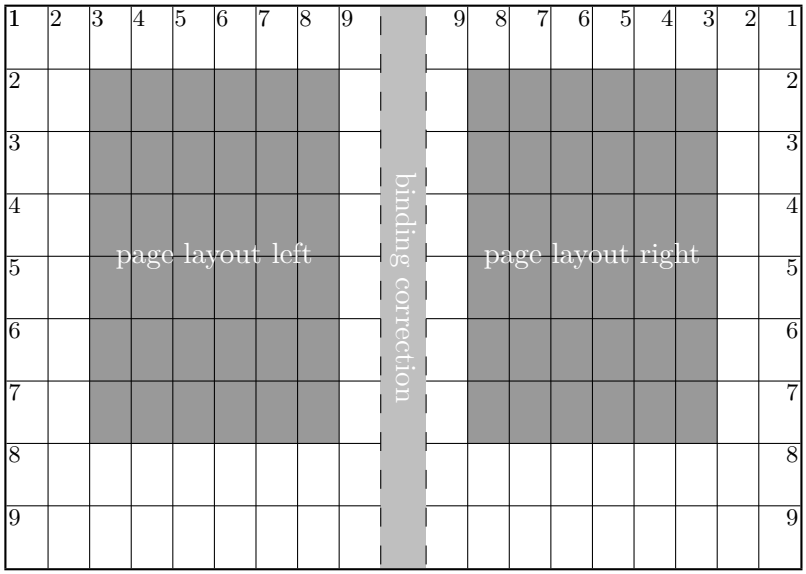


Figure 2.1.: Double-sided layout with the box construction of the classical division factor of 9, after subtraction of a binding correction

remains of the page is the text area. The width and the height of the text area and margins result automatically from the number of rows and columns *DIV*. Since the margins always need three stripes, *DIV* must be necessarily greater than three. In order that the text area occupy at least twice as much space as the margins, *DIV* should really be equal to or greater than 9. With this value, the construction is also known as the *classical division factor of 9* (see figure 2.1).

In KOMA-Script, this kind of construction is implemented in the typearea package, where the bottom margin may drop any fractions of a line in order to conform with the minor condition for the text area height mentioned in the previous paragraph, and thereby to minimize the mentioned problem with `\flushbottom`. For A4 paper, *DIV* is predefined according to the font size (see table 2.2, page 24). If there is no binding correction (*BCOR* = 0 pt), the results roughly match the values of table 2.1, page 23.

In addition to the predefined values, one can specify *BCOR* and *DIV* as options when loading the package (see section 2.4, from page 21 onwards). There is also a command to explicitly calculate the type area by providing these values as parameters (also see section 2.4, page 27).

The typearea package can automatically determine the optimal value of *DIV* for the font and leading used. Again, see section 2.4, page 24.

2.3. Page Layout Construction by Drawing a Circle

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

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

In the typearea package this construction is changed slightly. By using a special (normally meaningless) *DIV* value or a special package option, a *DIV* value is chosen to match the perfect values of the late Middle Age method as closely as possible. See also section 2.4, page 24.

2.4. Early or late Selection of Options

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

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

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

v3.00

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

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

v3.00 KOMA-Script offers most class and package options the opportunity to change the value of options even after loading of the class or package. One may than change the values of a list of options at will with the `\KOMAoptions` command. Each option in the *option list* has the form *option*`=`*value*.

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

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

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

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

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

2.5. Options and Macros to Influence the Page Layout

The package typearea offers two different user interfaces to influence type area construction. The more important method is to load the package with options. For information on how to load packages and to give package options, please refer to the LaTeX literature, e. g. [OPHS99] and [Tea05b], or the examples given here. Since the typearea package is loaded automatically when using the KOMA-Script main classes, the package options can be given as class options (see section 3.1).

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

`BCOR=`*correction*

v3.00 With the aid of the option `BCOR=`*correction* one may specify the absolute value of the binding correction, i. e., the width of the area which will be lost from the paper width in the binding process. This value is then automatically taken into account in the page layout construction and in the final output is added to the inner (or the left) margin. For the *correction* specification any measurement unit understood by TeX is valid.

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

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

or

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

when using BCOR as a global option.

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

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

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

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

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

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

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

`DIV=`*`Factor`*

v3.00 With the aid of the option DIV=*Factor* the number of stripes into which the page is divided horizontally and vertically during the page layout construction is set. The exact construction method is found in section 2.2. Of importance is that the larger the *Factor*, the larger the text block and the smaller the margins. Any integer value greater than 4 is valid for *Factor*.

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

	Type-area		Margins	
DIV	width [mm]	height [mm]	top [mm]	inner [mm]
6	105,00	148,50	49,50	35,00
7	120,00	169,71	42,43	30,00
8	131,25	185,63	37,13	26,25
9	140,00	198,00	33,00	23,33
10	147,00	207,90	29,70	21,00
11	152,73	216,00	27,00	19,09
12	157,50	222,75	24,75	17,50
13	161,54	228,46	22,85	16,15
14	165,00	233,36	21,21	15,00
15	168,00	237,60	19,80	14,00

Please note that large values can lead to unfulfillment of various minor conditions in the type-area, depending on further options chosen. Thus, in an extreme case, the header may fall outside of the page. Users applying the option `DIV=`*`Factor`* are themselves responsible for fulfillmen of the marginal conditions and setting of a typographically aesthetic line length.

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

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

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

On completion, it is decided that the protocols will from now on be collected and bound quarterly into book format. The binding is to be a simple glue binding, because it is only done to conform with ISO 9000 and nobody is actually going to read them. For the binding including space lost in turning the pages, an average

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

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

of 12 mm is required. Thus, one may change the options of the typearea package accordingly, and use the class for protocols conforming to ISO 9000 regulations:

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

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

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

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

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

`DIV=calc`
`DIV=classic`

v3.00

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

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

Example: In the example using the Bookman font with the `DIV=`*`Factor`* option, exactly that problem of choosing a more appropriate *DIV* value for the font arose. As a

variation on that example, one could simply leave the choice of such a value to the typearea package:

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

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

`DIV=current`
`DIV=last`

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

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

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

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

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

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

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

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

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

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

A summary of all possible symbolic values for the `DIV` option can be founf in table 2.3. At this point it is noted that the use of the fontenc package can also lead to LaTeX loading a different font.

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

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

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

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

When using the setspace package (see [Tob00]), the same example would appear as follows:

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

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

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

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

See further also the notes in section 2.7.

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

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

If the DIV option or the BCOR option is set after loading of the typearea package, then internally the command \typearea is called. When setting the DIV option the symbolic value current

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

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

is used internally for *BCOR*, which for reasons of completeness is found also in table 2.4. When setting the BCOR option, the symbolic value `last` is used internally for *DIV*. If it is instead desired that the text block and margins should be recalculated using the symbolic value `current` for *DIV*, then `\typearea{current}{current}` can be used directly.

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

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

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

Example: If one finds the notation

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

or

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

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

```
\recalctypearea
```

`twoside=`*switch*
`twoside=semi`

As already explained in section 2.1, the margin configuration is dependent on whether the document is to be typeset single- or double-sided. For single-sided typesetting, the left and right margins are equally wide, whereas for double-sided printing the inner margin of one page is only half as wide as the corresponding outer margin. In order to implement this distinction, the typearea package must be given the `twoside` option, if the document is to be typeset

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

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

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

double-sided. Being a *switch*, any of the standard values for simple switches in table 2.5 are valid. If the option is passed without a value, the value `true` is assumed, so double-sided typesetting is carried out. Deactivation of the option leads to single-sided typesetting.

v3.00

Apart from the values in table 2.5 the value `semi` can also be given. The value `semi` results in a double-sided typesetting with single-sided margins and single-sided, i. e., not alternating, margin notes.

The option can also be passed as class option in `\documentclass`, as package option to `\usepackage`, or even after loading of the typearea package with the use of per `\KOMAoptions` or `\KOMAoption`. Use of the option after loading the typearea package results automatically to a recalculation of the type-area using `\recalctypearea` (see page 27). If double-sided typesetting was active before the option was set, then before the recalculation a page break is made to the next odd page.

`twocolumn=`*switch*

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

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

The option can also be passed as class option in `\documentclass`, as package option to `\usepackage`, or even after loading of the typearea package with the use of per `\KOMAoptions` or `\KOMAoption`. Use of the option after loading the typearea package results automatically to a recalculation of the type-area using `\recalctypearea` (see page 27).

headinclude=*switch*
footinclude=*switch*

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

At first glance the question appears trivial: Margins are those parts on the right, left, top and bottom which remain empty. But this is only half the story. Margins are not always empty. There may be margin notes, for example (see `\marginpar` command in [OPHS99] or section 3.6.5).

One could also ask, whether headers and footers belong to the upper and lower margins or to the text. This can not be answered unambiguously. Of course an empty footer or header belong to the margins, since they can not be distinguished from the rest of the margin. A header or footer, that contains only a page number[1], will optically appear more like a margin. For the optical appearance it is not important whether headers or footers are easily recognized as such during reading. Important is only, how a well filled page appears when viewed *out of focus*. One could use the glasses of one's far-sighted grand parents, or, lacking those, adjust one's vision to infinity and look at the page with one eye only. Those wearing spectacles will find this much easier, of course. If the footer contains not only the page number, but other material like a copyright notice, it will optically appear more like a part of the text body. This needs to be taken into account when calculating text layout.

For the header this is even more complicated. The header frequently contains running headings [2]. In case of running headings with long chapter and section titles the header lines will be very long and appear to be part of the text body. This effect becomes even more significant when the header contains not only the chapter or section title but also the page number. With material on the right and left side, the header will no longer appear as an empty margin. It is more difficult if the pagination is in the footer, and the length of the titles varies, so that the header may appear as a margin on one page and as text on another. However, these pages should not be treated differently under any circumstances, as this would lead to vertically jumping headers. In this case it is probably best to count the header as part of the text.

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

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

v3.00

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

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

header or footer is counted as part of the text.

Readers who are unsure about the the correct setting should re-read the above explanations. Default is usually `headinclude=false` and `footinclude=false`, but this can change depending on KOMA-Script class and KOMA-Script packages used (see section 3.1 and chapter 4).

Please note that when using these options with one of the KOMA-Script classes as in the example above, they must be used either as a class option, or passed via `\KOMAoptions` or `\KOMAoption` after loading the class. Changing of these options after loading the typearea package does not result in an automatic recalculation of the type-area. Instead, the changes only take effect at the next recalculation of the type-area. For recalculation of the type-area, refer to the `DIV` option with the values `last` or `current` (see page 25) or the `\recalctypearea` command (see page 27).

`mpinclude=`*`switch`*

v2.8q

v3.00

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

The effect of `mpinclude=true` is that one width-unit of the text body is taken for the margin-note area. Using option `mpinclude=false`, the default setting, then the normal margin is used for the margin-note area. The width of that area is one or one and a half width-unit, depending on whether one-sided or double-sided page layout has been chosen. The option `mpincludetrue` is mainly for experts and so not recommended.

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

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

gives the desired result.

Currently there is no option to enlarge the margin by a given amount. The only solution is to either not use the option `mpinclude` or to set `mpinclude` to `false`, and instead after the type-area has been calculated one reduces the width of the text body `\textwidth` and enlarges the margin

width \marginparwidth by the same amount. Unfortunately, this can not be combined with automatic calculation of the *DIV* value. In contrast DIV=calc (see page 24) heeds mpinclude.

Please note that when using this option with one of the KOMA-Script classes as in the example above, it must be used either as a class option, or passed via \KOMAoptions or \KOMAoption after loading the class. Changing of this option after loading the typearea package does not result in an automatic recalculation of the type-area. Instead, the changes only take effect at the next recalculation of the type-area. For recalculation of the type-area, refer to the DIV option with the values **last** or **current** (see page 25) or the \recalctypearea command (see page 27).

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

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

v3.00

The option **headlines** is set to the number of header lines. The typearea package uses a default of 1.25. This is a compromise, large enough for underlined headers (see section 3.1) and small enough that the relative weight of the top margin is not affected too much when the header is not underlined. Thus in most cases you may leave **headlines** at its default value and adapt it only in special cases.

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

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

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

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

Commands that can be used to define the contents of a header with two lines are described in chapter 4.

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

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

\KOMAoption after loading the class. Changing of these options after loading the typearea package does not result in an automatic recalculation of the type-area. Instead, the changes only take effect at the next recalculation of the type-area. For recalculation of the type-area, refer to the DIV option with the values last or current (see page 25) or the \recalctypearea command (see page 27).

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

So far we have seen how a good or even very good type-area is calculated and how the typearea package can support these calculations, giving you at the same time the freedom to adapt the layout to your needs. However, there are cases where the text body has to fit exactly some specified dimensions. At the same time the margins should be well spaced and a binding correction should be possible. The typearea package offers the command \areaset for this purpose. As parameters this command accepts the binding correction and the width and height of the text body. Width and position of the margins will then be calculated automatically, taking account of the options headinclude, headinclude=false, footinclude and footinclude=false where needed. On the ther hand, the options headlines and headheight are ignored!

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

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

You need only 29 instead of 30, because the base line of the topmost text line is \topskip below the top margin of the type area, as long as the height of the topmost line is less than \topskip. Thus, the uppermost line does not require any height. The descenders of characters on the lowermost line, on the other hand, hang below the dimensions of the type-area.

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

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

DIV=areaset

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

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

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

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

2.6. Paper Format Selection

The paper format is a definitive characteristic of any document. As already mentioned in the description of the supported page layout constructions (see section 2.1 to section 2.3 from page 16 onwards), the entire page division and document layout depends on the paper format. Whereas the LaTeX standard classes are restricted to a few formats, KOMA-Script supports in conjunction with the typearea package even exotic paper sizes.

paper=*format*

v3.00 The option paper is the central element for format selection in KOMA-Script. *Format* supports first of all the american formats letter, legal and executive. In addition, it supports the ISO formats of the series A, B, C and D, for example A4 or — written in lowercase — a4. Landscape formats are supported by specifying the option again, this time with the value landscape. Additionally, the *format* can also be specified in the form *height*:*width*.

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

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

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

```
\documentclass{article}
\usepackage[headinclude=false,footinclude=false,%
            paper=A8,paper=5cm:3cm]{typearea}
\areaset{4cm}{2.4cm}
\pagestyle{empty}
\begin{document}
\section*{Supported Paper Sizes}
letter, legal, executive, a0, a1 \dots\ %
b0, b1 \dots\ c0, c1 \dots\ d0, d1 \dots
\end{document}
```

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

Please note that when using this options with one of the KOMA-Script classes, it must be used either as a class option, or passed via `\KOMAoptions` or `\KOMAoption` after loading the class. Changing of this option after loading the typearea package does not result in an automatic recalculation of the type-area. Instead, the changes only take effect at the next recalculation of the type-area. For recalculation of the type-area, refer to the `DIV` option with the values `last` or `current` (see page 25) or the `\recalctypearea` command (see page 27).

`pagesize=`*output driver*

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

The option `pagesize=dvips` writes the paper size as a `\special` into die DVI data file. This `\special` is then evaluated by, for example, `dvips`. On the other hand, the option `pagesize=pdftex` writes the paper size at the start of the document into the pdfTeX page registers `\pdfpagewidth` and `\pdfpageheight`, so that later when viewing the PDF file the correct format is chosen. The option `pagesize=auto` is more flexible and, depending on whether a PDF or DVI data file is output, uses the mechanism of the option `pagesize=dvips` or `pagesize=pdftex`. With the option `pagesize=automedia` it may be that `\mediawidth` and `\mediaheight` are also set appropriately. In this way even the demands of VTeX are

supported. If the option `pagesize` is used with specifying an *`output driver`* then `auto` is used.

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

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

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

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

2.7. Tips

In particular for theses many rules exist that violate even the most elementary rules of typography. The reasons for such rules include typographical incompetence of those making them, but also the fact that they were originally meant for mechanical typewriters. With a typewriter or a primitive text processor dating back to the early ´80s it was not possible to produce typographically correct output without extreme effort. Thus rules were created that appeared to be achievable and still allowed easy correction. To avoid short lines made worse by ragged margins, the margins were kept narrow and the line spacing was increased to 1.5 for corrections. Before the advent of modern text processing systems, single-spaced would have been the only alternative — other than with TEX. In such a single-spaced document even correction signs would have been difficult to add. When computers became more widely available for text processing, some students tried to use a particularly “nice” font to make their work look better than it really was. They forgot however that such fonts are often more difficult to read and therefore unsuitable for this purpose. Thus two bread-and-butter fonts became widely used which neither fit together nor are particularly suitable for the job. In particular Times is a relatively narrow font which was developed at the beginning of the 20^{th} century for the narrow columns of British newspapers. Modern versions usually are somewhat improved. But still the Times font required in many rules does not really fit to the margin sizes prescribed.

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

To some extend the questionable rules are difficult to implement in LATEX. A fixed number of characters per line can be kept only when a non-proportional font is used. There are very few good non-proportional fonts available. Hardly a text typeset in this way looks really good.

In many cases font designers try to increase the serifs on the ‘i’ or ‘l’ to compensate for the different character width. This can not work and results in a fragmented and agitated-looking text. If one uses LaTeX for one’s paper, some of these rules have to be either ignored or at least interpreted generously. For example one may interpret “60 characters per line” not as a fixed, but as an average or maximal value.

As executed, record regulations are usually intended to obtain a usable result even if the author does not know what needs to be considered. Usable means frequently: readable and correctable. In the author’s opinion the type-area of a text set with LaTeX and the typearea package meets these criteria well right from the start. Thus if one is confronted with regulations which deviate obviously substantially from it, then the author recommends submitting an extract from the text to the responsible person and inquiring whether it is permitted to submit the work despite deviations in the format. If necessary the type area can be moderately adapted by modification of option `DIV`. The author advises against the use of `\areaset` for this purpose however. In the worst case one may make use of the geometry package (see [Ume00]), which is not part of KOMA-Script, or change the type-area parameters of LaTeX. One may find the values determined by typearea in the `log` file of one’s document. Thus moderate adjustments should be possible. However, one should make absolutely sure that the proportions of the text area correspond approximately to those of the page including consideration of the binding correction.

If it should prove absolutely necessary to set the text with a line spacing of 1.5, then one should not under any circumstances redefine `\baselinestretch`. Although this procedure is recommended all too frequently, it has been obsolete since the introduction of LaTeX 2_ε in 1994. In the worst case one may use the instruction `\linespread`. The author recommends the package setspace (see [Tob00]), which is not part of KOMA-Script. Also one should let typearea recalculate a new type-area after the conversion of the line spacing. However, one should switch back to the normal line spacing for the title, preferably also for the table contents and various listings — as well as the bibliography and the index. The setspace package offers for this a special environment and its own instructions.

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

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

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

The Main Classes scrbook, scrreprt and scrartcl

NOTE: Almost all commands of the chapter for experts are missing, because that chapter is still missing. Some are at this chapter instead.

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

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

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

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

3.1. The Options

NOTE: Since version 3.00 the main classes understand command `\KOMAoptions` (see section 6.2, page 134). In the course of the development many new options were implemented and old became obsolete. Only the new options may be used with `\KOMAoptions`. Unfortunately most of them are documented not yet. You may find the obsolete and corresponding new options at table 3.2.

NOTE: Following options are still missing in this chapter: `bibliography=`*`setting`*, `bibliography=openstyle`, `bibliography=oldstyle`, `captions=bottombeside`, `captions=centeredbeside`, `captions=innerbeside`, `captions=leftbeside`,

captions=outerbeside, captions=rightbeside, captins=topbeside, fontsize=*size*, footnotes=multiple, footnotes=nomultiple, headings=onelineappendix, headings=twolineappendix, headings=onelinechapter, headings=twolinechapter, listof=chapterentry, listof=chaptergapline, listof=chaptergapsmall, listof=leveldown, listof=nochaptergap, numbers=autoendperiod, toc=bibliography, toc=bibliographynumbered, toc=index, toc=listof, toc=listofnumbered, toc=nobibliography, toc=noindex, toc=nolistof, version=*value* (see section 6.2.2, page 134).

Table 3.2.: Obsolete vs. Recommended Options

obsolete option	recommended option
abstracton	abstract
abstractoff	abstract=false
parskip-	parskip=full-
parskip+	parskip=full+
parskip*	parskip=full*
halfparskip	parskip=half
halfparskip-	parskip=half-
halfparskip+	parskip=half+
halfparskip*	parskip=half*
tocleft	toc=flat
tocindent	toc=graduated
listsleft	listof=flat
listsindent	listof=graduated
cleardoubleempty	cleardoublepage=empty
cleardoubleplain	cleardoublepage=plain
cleardoublestandard	cleardoublepage=current
pointednumber	numbers=enddot
pointlessnumber	numbers=noenddot
nochapterprefix	chapterprefix=false
noappendixprefix	appendixprefix=false
bigheadings	headings=big
normalheadings	headings=normal
smallheadings	headings=small
headnosepline	headsepline=false
footnosepline	footsepline=false
liststotoc	listof=totoc
liststotocnumbered	listof=numbered

...

Table 3.2.: Obsolete vs. Recommended Options (*continuation*)

obsolete Option	recommended option
`bibtotoc`	`bibliography=totoc`
`bibtotocnumbered`	`bibliography=totocnumbered`
`idxtotoc`	`index=totoc`
`tablecaptionabove`	`captions=tableheading`
`tablecaptionbelow`	`captions=tablesignature`
`onelinecaption`	`captions=oneline`
`noonelinecaption`	`captions=nooneline`

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

Table 3.3 lists those options that are set by default in at least one of the KOMA-Script classes. The table shows for each KOMA-Script main class if the option is set by default and if it is even defined for that class. An undefined option cannot be set, either by default or by the user.

Allow me an observation before proceeding with the descriptions of the options. It is often the case that at the beginning of a document one is often unsure which options to choose for that specific document. Some options, for instance the choice of paper size, may be fixed from the beginning. But already the question of which *DIV* value to use could be difficult to answer initially. On the other hand, this kind of information should be initially irrelevant for the main tasks of an author: design of the document structure, text writing, preparation of figures, tables and index. As an author you should concentrate initially on the contents. When that is done, you can concentrate on the fine points of presentation. Besides the choice of options, this means correcting things like hyphenation, page breaks, and the distribution of tables and figures. As an example consider table 3.3, which I moved repeatedly between the beginning and the end of this section. The choice of the actual position will only be made during the final production of the document.

3.1.1. Options for Compatibility

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

Table 3.3.: Default options of the KOMA-Script classes

Option	scrbook	scrreprt	scrartcl
abstract=	*undefined*	false	false
captions=	tablesignature	tablesignature	tablesignature
chapteratlists=	10pt	10pt	*undefined*
chapterprefix=	false	false	*undefined*
draft=	false	false	false
fontsize=	11pt	11pt	11pt
footsepline=	false	false	false
headings=	big	big	big
headsepline=	false	false	false
listof=	graduated	graduated	graduated
open=	right	any	*undefined*
paper=	a4	a4	a4
parindent	default	default	default
titlepage=	true	true	false
toc=	graduated	graduated	graduated
twocolumn=	*false*	*false*	*false*
twoside=	true	false	false
version=	first	first	first

`version`
`version=`*`value`*

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

```
version=first
```

or

```
version=2.9
```

or

```
version=2.9t.
```

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

```
version=last
```

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

The question of compatibility is first of all a question of line and page breaking. New capabilities, which do not affect page breaks, are also available if the option of compatibility to an older version is selected. The option has no effect on changes in the page breaking when using a newer version, which result purely through the correction of errors. If absolute compatibility including errors is required, then the requisite KOMA-Script version should be archived along with the document source.

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

3.1.2. Options for Page Layout

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

`letterpaper`
`legalpaper`
`executivepaper`
`a`*`X`*`paper`
`b`*`X`*`paper`
`c`*`X`*`paper`
`d`*`X`*`paper`
`landscape`

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

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

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

3.1.3. Options for Document Layout

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

`open=`*value*

scrbook, scrreprt

These option has the same effects like the standard options `openany` and `openright`. They affect the choice of the page where a chapter can begin, so they are not available with the scrartcl class, since the next largest unit below "part" is "section". The chapter level is not available in scrartcl.

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

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

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

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

`titlepage=`*switch*

The values of the option (see table 2.5, page 29) have the same effect as the standard options `titlepage` and `notitlepage`. The option `titlepage=true` makes LaTeX use separate pages for the titles. These pages are set inside a `titlepage` environment and normally have neither header nor footer. In comparison with standard LaTeX, KOMA-Script expands the handling of the titles significantly (see section 3.3).

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

```
parskip=full
parskip=full*
parskip=full+
parskip=full-
parskip=half
parskip=half*
parskip=half+
parskip=half-
parindent
```

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

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

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

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

The default behaviour of KOMA-Script follows the `parindent` option. In this case, there is

no spacing between paragraphs, only an indentation of the first line by 1 em.

`headsepline=`*switch*
`footsepline=`*switch*

In order to have a line separating the header from the text body use the option `headsepline` (see table 2.5, page 29). The option `headsepline=false` has the reverse effect. These options have no effect with the page styles `empty` and `plain`, because there is no header in this case. Such a line always has the effect of visually bringing header and text body closer together. That doesn't mean that the header must now be moved farther from the text body. Instead, the header should be considered as belonging to the text body for the purpose of page layout calculations. KOMA-Script takes this into account by automatically passing the option `headinclude` to the typearea package whenever the `headsepline` option is used.

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

`chapterprefix`
`chapterprefix=false`

scrbook, scrreprt

With the standard classes book and report a chapter title consists of a line with the word "Chapter"[1] followed by the chapter number. The title itself is set left-justified on the following lines. The same effect is obtained in KOMA-Script with the class option `chapterprefix`. The default however is `chapterprefix=false`. These options also affect the automatic running titles in the headers (see section 3.2.2).

`appendixprefix=`*switch*

scrbook, scrreprt

Sometimes one wishes to have the chapter titles in simplified form according to `chapterprefix=false`. But at the same time, one wishes a title of an appendix to be preceded by a line with "Appendix" followed by the appendix letter. This is achieved by using the `appendixprefix` option (see table 2.5, page 29). Since this results in an inconsistent document layout, I advise against using this option.

The reverse option `appendixprefix=false` exists only for completeness' sake. I don't know of any sensible use for it.

`captions=oneline`
`captions=nooneline`

The standard classes differentiate between one-line and multi-line table or figure captions. One-line captions are centered while multi-line captions are left-justified. This behavior, which

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

is also the default with KOMA-Script, corresponds to the option `captions=oneline`. There is no special handling of one-line captions when the `captions=nooneline` option is given.

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

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

3.1.4. Options for Font Selection

Font options are those options that affect the font size of the document or the fonts of individual elements. Options that affect the font style are also theoretically font options. However KOMA-Script currently has no such options.

`10pt`
`11pt`
`12pt`
X`pt`

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

`headings=small`
`headings=normal`
`headings=big`

The font size used for the titles is relatively big, both with the standard classes and with KOMA-Script. Not everyone likes this choice; moreover it is specially problematic for small paper sizes. Consequently, KOMA-Script provides, besides the large title font size defined by the `headings=big` option, the two options `headings=normal` and `headings=small`, that allow for smaller title font sizes. The font sizes for headings resulting from these options for scrbook and scrreprt are shown in table 3.10, page 74. For scrartcl smaller font sizes are generally used.

scrbook, scrreprt

The spacing before and after chapter titles is also influenced by these options. Chapter titles are also influenced by the options `chapterprefix` and `chapterprefix=false`, and appendix titles by the options `appendixprefix` and `appendixprefix=false`, all of which are described in section 3.1.3, page 46.

3.1.5. Options Affecting the Table of Contents

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

```
listof=totoc
index=totoc
bibliography=totoc
bibliography=totocnumbered
listof=numbered
```

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

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

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

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

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

The bibliography is a different kind of listing. It does not list the contents of the present document but refers instead to external documents. For that reason, it could be argued

that it qualifies as a chapter (or section) and, as such, should be numbered. The option `bibliography=totocnumbered` has this effect, including the generation of the corresponding entry in the table of contents. I personally think that this reasoning would lead us to consider a classical list of sources also to be a separate chapter. On the other hand, the bibliography is finally not something that was written by the document's author. In view of this, the bibliography merits nothing more than an unnumbered entry in the table of contents, and that can be achieved with the `bibliography=totoc` option.

v2.8q As the author of KOMA-Script already views the option `listof=totoc` with open skepticism, and frankly detests option `bibliography=totocnumbered`, it should come as no surprise that he implemented option `listof=numbered` only under extreme duress. He fears that as a next step someone will want the table of contents numbered and entered in the table of contents. Therefore, those looking in this documentation for a detailed description of option `listof=totoc` will search in vain. A similar option for the index would be just as silly, so its implementation has been determinedly refused so far.

`toc=graduated`
`toc=flat`

v2.8q The table of contents is normally set up so that different sectional units have different indentations. The section number is set left-justified in a fixed-width field. This setup is selected with the option `toc=graduated`.

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

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

If the `toc=flat` option is active, the width of the field for unit numbering is determined when outputting the table of contents. After a change that affects the table of contents, at most three LaTeX runs are necessary to obtain a correctly set table of contents.

3.1.6. Options for Lists of Floats

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

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

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

`listof=graduated`
`listof=flat`

v2.8q

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

If the numbers become too large, for instance because many tables are used, it may happen that the available space is exceeded. Therefore KOMA-Script supplies an option called `listof=flat` that is similar to the `toc=flat` option. The width of the numbers is automatically determined and the space for them correspondingly adjusted. Concerning the mode of operation and the side effects, the observations made in section 3.1.5, page 49 for the `toc=flat` option are equally valid in this case. Please note that when using the `listof=flat` option several LaTeX runs are necessary before the lists of floats achieve their final form.

`chapteratlists`
`chapteratlists=`*`value`*

scrbook, scrreport

v2.96a

Normally, every chapter entry generated with `\chapter` introduces vertical spacing into the lists of floats. Since version 2.96a this applies also for the command `\addchap`, if no compatibility option to an earlier version was chosen (see option `version` in section 3.1.1, page 42).

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

This option can be changed with `\KOMAoptions{`*`chapteratlists`*`}` or `\KOMAoptions{`*`chapteratlists=value`*`}` even inside the document. It takes effect from the next heading onwards. However, changes to the option will only become effective in the lists following two more LaTeX runs.

3.1.7. Options Affecting the Formatting

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

`abstract=`*switch*

scrreprt, scrartcl

In the standard classes the `abstract` environment sets the text "Abstract" centered before the summary text. This was normal practice in the past. In the meantime, newspaper reading has trained readers to recognize a displayed text at the beginning of an article or report as the abstract. This is even more true when the text comes before the table of contents. It is also surprising when precisely this title appears small and centered. KOMA-Script provides the possibility of including or excluding the abstract's title with the options `abstract=true` and `abstract=false` (see table 2.5,page 29).

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

`numbers=enddot`
`numbers=noenddot`

In German, according to DUDEN, the numbering of sectional units should have no dot at the end if only arabic numbers are used (see [DUD96, R 3]). On the other hand, if roman numerals or letters are appear in the numbering, then a dot should appear at the end of the numbering (see [DUD96, R 4]). KOMA-Script has an internal mechanisms that tries to implement this somewhat complex rule. The resulting effect is that, normally, after the sectional commands `\part` and `\appendix` a switch is made to numbering with an ending dot. The information is saved in the `aux` file and takes effect on the next LaTeX run.

In some cases the mechanism for placing or leaving off the ending dot may fail, or other languages may have different rules. Therefore it is possible to activate the use of the ending dot manually with the option `numbers=enddot` or to deactivate it with `numbers=noenddot`.

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

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

`leqno`

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

`fleqn`

Displayed equations are normally centered. The standard option `fleqn` causes the standard option file `fleqn.clo` to be loaded. Displayed equations are then left-justified. This option may not be used at the argument of `\KOMAoptions` but at the optional argument of `\documentclass`.

`captions=tablesignature`
`captions=tableheading`

As described in section 3.6.6, page 96, the `\caption` command acts with figures like the `\captionbelow` command. The behaviour with tables, however, depends on these two options. In the default setting, `captions=tablesignature`, the `\caption` macro acts also with tables like the `\captionbelow` command. With the `captions=tableheading` option, `\caption` acts like the `\captionabove` command.

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

float

Note that when using the float package, the options `captions=tablesignature` and `captions=tableheading` cease to act correctly when `\restylefloat` is applied to tables. More details of the float package and `\restylefloat` can be found in [Lin01]. Additional support in KOMA-Script for the float package may be found at the explanation of `komaabove` in section 3.6.6, page 99.

`origlongtable`

longtable

The package longtable (see [Car04]) sets table captions internally by calling the command `\LT@makecaption`. In order to ensure that these table captions match the ones used with normal tables, the KOMA-Script classes normally redefine that command. See section 3.6.6, page 97 for more details. The redefinition is performed with help of the command `\AfterPackage` immediately after the loading of package longtable. If the package caption2 (see [Som08]) has been previously loaded, the redefinition is not made in order not to interfere with the caption2 package.

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

`openbib`
`bibliography=openstyle`
`bibliography=oldstyle`

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

`draft=`*switch*

The option `draft` (see table 2.5, page 29) is normally used to distinguish between the draft and final versions of a document. In particular, the option `draft=true` activates small black boxes that are set at the end of overly long lines. The boxes help the untrained eye to find paragraphs that have to be treated manually. With the `draft=false` option no such boxes are shown.

Option `draft` without value is also processed by other packages and affect their operation. For instance, the graphics and the graphicx packages don't actually output the graphics when the option `draft` is specified. Instead they output a framed box of the appropriate size containing only the graphic's filename (see [Car99b]).

3.2. General Document Characteristics

Some document characteristics do not apply to a particular section of the document like the titling, the text body or the bibliography, but do affect the entire document. Some of these characteristics were already described in section 3.1.

3.2.1. Changing Fonts

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

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

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

v2.8p

With the help of the two commands `\setkomafont` and `\addtokomafont` it is possible to define the *commands* that change the characteristics of a given *element*. Theoretically all possible statements including literal text could be used as *commands*. You should however absolutely limit yourself to those statements that really switch only one font attribute. This will usually be the commands `\normalfont`, `\rmfamily`, `\sffamily`, `\ttfamily`, `\mdseries`, `\bfseries`, `\upshape`, `\itshape`, `\slshape`, `\scshape` and the font size commands `\Huge`, `\huge`, `\LARGE`, etc. The description of these commands can be found in [OPHS99], [Tea05b] or [Tea05a]. Color switching commands like `\normalcolor` (see [Car99b]) are also acceptable. The behavior when using other commands, specially those that make redefinitions or generate output, is not defined. Strange behavior is possible and does not represent a bug.

The command `\setkomafont` provides a font switching command with a completely new definition. In contrast to this the `\addtokomafont` command merely extends an existing definition. It is recommended to not use both commands in the same document. Usage examples can be found in the paragraphs on the corresponding element. Names and meanings of the individual items are listed in table 3.4. The default values are shown in the corresponding paragraphs.

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

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

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

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

Table 3.4.: Elements, whose type style can be changed with the KOMA-Script command `\setkomafont` or `\addtokomafont`

`caption`	Text of a table or figure caption
`captionlabel`	Label of a table or figure caption; used according to the element `caption`
`chapter`	Title of the sectional unit `\chapter`
`chapterentry`	Table of contents entry of the sectional unit `\chapter`

Table 3.4.: Elements, whose type style can be changed (*continuation*)

`chapterentrypagenumber`
: Page number of the table of contents entry of the sectional unit `\chapter`, variation on the element `chapterentry`

`descriptionlabel`
: Labels, i. e., the optional argument of `\item` in the `description` environment

`dictum`
: wise saying (see command `\dictum`)

`dictumauthor`
: Author of a wise saying; used according to the element `dictumtext`

`dictumtext`
: Another name for `dictum`

`disposition`
: All sectional unit titles, i. e., the arguments of `\part` down to `\subparagraph` and `\minisec`, including the title of the abstract; used before the element of the corresponding unit

`footnote`
: Footnote text and marker

`footnotelabel`
: Mark of a footnote; used according to the element `footnote`

`footnotereference`
: Footnote reference in the text

`labelinglabel`
: Labels, i. e., the optional argument of `\item` in the `labeling` environment

`labelingseparator`
: Separator, i. e., the optional argument of the `labeling` environment; used according to the element `labelinglabel`

`minisec`
: Title of `\minisec`

`pagefoot`
: The foot of a page, but also the head of a page

...

Table 3.4.: Elements, whose type style can be changed (*continuation*)

`pagehead`
: The head of a page, but also the foot of a page

`pagenumber`
: Page number in the header or footer

`pagination`
: Another name for `pagenumber`

`paragraph`
: Title of the sectional unit `\paragraph`

`part`
: Title of the `\part` sectional unit, without the line containing the part number

`partentry`
: Table of contents entry of the secitonal unit `\part`

`partentrypagenumber`
: Page number of the table of contents entry of the sectional unit `\part` variation on the element `partentry`

`partnumber`
: Line containing the part number in a title of the sectional unit `\part`

`section`
: Title of the sectional unit `\section`

`sectionentry`
: Table of contents entry of sectional unit `\section` (only available in scrartcl)

`sectionentrypagenumber`
: Page number of the table of contents entry of the sectional unit `\section`, variation on element `sectionentry` (only available in scrartcl)

`sectioning`
: Another name for `disposition`

`subject`
: Categorization of the document, i. e., the argument of `\subject` on the main title page

`subparagraph`
: Title of the sectional unit `\subparagraph`

...

Table 3.4.: Elements, whose type style can be changed (*continuation*)

`subsection`	Title of the sectional unit `\subsection`
`subsubsection`	Title of the sectional unit `\subsubsection`
`subtitle`	Subtitle of the document, i. e., the argument of `\subtitle` on the main title page
`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 from page 66)

3.2.2. Page Style

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

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

Usually one distinguishes four different page styles.

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

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

headings is the page style with running headings in the header. These are headings for which titles are automatically inserted into the header. With the classes scrbook and scrreprt the titles of chapters and sections are repeated in the header for double-sided layout — with KOMA-Script on the outer side, with the standard classes on the inner side. The page number is set on the outer side of the footer with KOMA-Script, with the standard classes it is set on the inner side of the header. In one-sided layouts only the titles of the chapters are used and are, with KOMA-Script, centered in the header. The page numbers are set centered in the footer with KOMA-Script. scrartcl behaves similarly, but

scrbook, scrreprt

scrartcl

Table 3.5.: Default values for the elements of a page style

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

starting a level deeper in the section hierarchy with sections and subsections, because the chapter level does not exist in this case.

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

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

Besides, the form of the page styles `headings` and `myheadings` is affected by each of the four class options `headsepline`, `headsepline=false`, `footsepline` and `footsepline=false` (see section 3.1.3, page 46). The page style starting with the current page is changed by the command `\pagestyle`. On the other hand `\thispagestyle` changes only the style of the current page.

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

Please note that the change between automatic and manual running headings is no longer performed by page style changes when using the scrpage2 package, but instead via special instructions. The page styles `headings` and `myheadings` should not be used together with this package (see chapter 4, page 114).

v2.8p

In order to change the type style used in the header, footer or for the page number, please use the interface described in section 3.2.1. The same element is used for header and footer, which you can designate equivalently with `pagehead` or `pagefoot`. The element for the page number within the header or footer is called `pagenumber`. The default settings can be found in table 3.5.

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

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

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

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

As you can see, the last example uses the element `pagefoot`. You can achieve the same result using `pagehead` instead (see table 3.4 on page 54).

It is not possible to use these methods to force capitals to be used automatically for the running headings. For that, please use the scrpage2 package (see chapter 4, page 122).

If you define your own page styles, the commands `\usekomafont{pagehead}` and `\usekomafont{pagenumber}` can be useful. If you do not use the KOMA-Script package scrpage2 (see chapter 4) for that, but, for example, the package fancyhdr (see [vO00]), you can use these commands in your definitions. Thereby you can remain compatible with KOMA-Script as much as possible. If you do not use these commands in your own definitions, changes like those shown in the previous examples have no effect. The packages scrpage and scrpage2 take care to keep the maximum possible compatibility with other packages.

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

For some pages a different page style is chosen with the help of the command `\thispagestyle`. Which page style this actually is, is defined by these four macros, of which `\partpagestyle` and `\chapterpagestyle` are found only with classes scrbook and scrreprt, but not in scrartcl. The default value for all four cases is `plain`. The meaning of these macros can be taken from table 3.6. The page styles can be redefined with the `\renewcommand` macro.

scrbook, scrreprt

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

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

As mentioned previously on page 57, the page style `empty` is exactly what is required in this example. Naturally you can also use a user-defined page style.

Table 3.6.: Macros to set up page style of special pages

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

Assume you have defined your own page style for initial chapter pages with the package scrpage2 (see chapter 4). You have given to this page style the fitting name `chapter`. To actually use this style, you must redefine the macro `\chapterpagestyle` accordingly:

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

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

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

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

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

```
\endgroup
```

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

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

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

will be surprised at the results. For sure, the page style `headings` is thereby applied to the initial page of a chapter. But nevertheless no running headings appear when using the `openright` option. The reason for this behaviour can be found in the LaTeX core. There, the command `\rightmark`, that generates the marks for right-hand pages, is defined with;

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

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

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

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

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

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

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

The LaTeX core contains the `\clearpage` command, which takes care that all not yet output floats are output, and then starts a new page. There exists the instruction `\cleardoublepage` which works like `\clearpage` but which, in the double-sided layouts (see layout option `twoside` in section 2.4, page 28) starts a new right-hand page. An empty left page in the current page style is output if necessary.

With \cleardoublestandardpage KOMA-Script works as described above. The \cleardoubleplainpage command changes the page style of the empty left page to plain in order to suppress the running heading. Analogously, the page style empty is applied to the empty page with \cleardoubleemptypage, suppressing the page number as well as the runnning heading. The page is thus entirely empty. However, the approach used by \cleardoublepage is dependent on the layout options cleardoublepage=current, cleardoublepage=plain and cleardoublepage=empty described in section 3.1.3, page 44 and acts according to the active option.

\ifthispageodd{*true*}{*false*} \ifthispagewasodd*true*\else*false*\fi

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

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

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

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

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

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

The output would then be:

> This is a page with an even page number.

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

There are situations where the \ifthispageodd command never leads to the correct result. Suppose that the command is used within a box. A box is set by LaTeX always as a whole. No

Table 3.7.: 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

page breaks take place inside. Assume further that the *true* part is very big, but the *false* part is empty. If we suppose further that the box with the *false* part still fits on the current, even page, but that with the *true* part it does not. Further assume that KOMA-Script heuristically decides for the first run that the *true* part applies. The decision is wrong and is revised in the next run. The *false* part is thereby processed, instead of the *true* part. The decision must again be revised in the next run and so on.

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

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

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

This command works the same way in KOMA-Script as in the standard classes. More precisely it is a command from the LaTeX kernel. You can specify with this command the *numbering style* of page numbers. The changes take effect immediately, hence starting with the page that contains the command. The possible settings can be found in table 3.7. Using the command `\pagenumbering` also resets the page counter. Thus the page number of the next page which TeX outputs will have the number 1 in the style *numbering style*.

3.3. Titles

After having described the options and some general issues, we begin the document where it usually begins: with the titles. The titles comprise everything that belongs in the widest sense to the title of a document. Like already mentioned in section 3.1.3, page 44, we can distinguish between title pages and *in-page* titles. Article classes like article or scrartcl have by default *in-page* titles, while classes like report, book, scrreprt and scrbook have title pages as default. The defaults can be changed with the class option `titlepage`.

`titlepage`

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

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

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

Simple? Right.

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

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

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

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

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

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

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

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

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

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

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

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

Table 3.8.: Font defaults for the elements of the title

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

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

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

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

v2.8p The *Title* is output with a very large font size. Besides the change of size, the settings for the element `title` also take effect. By default these settings are identical to the settings for the element `disposition` (see table 3.4, page 54). The font size is however not affected (see table 3.4, page 67). The default settings can be changed with the commands of section 3.2.1.

v2.97c The *Subtitle* is set just below the title, in a font determined by the element `subtitle`. The default, seen in table 3.8 can be changed with the help of the commands in section 3.2.1.

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

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

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

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

With the exception of *titlehead* and possible footnotes, all the items are centered hori-

Table 3.9.: Font size and horizontal positioning of the elements in the main title page in the order of their vertical position from top to bottom when typeset with `\maketitle`

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

zontally. The information is summarised in table 3.9.

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

```
\documentclass{scrbook}
\begin{document}
\titlehead{{\Large Unseen University
    \hfill SS~2002\\}
  Higher Analytical Institute\\
  Mythological Rd\\
  34567 Etherworld}
\subject{Dissertation}
\title{Digital space simulation with the DSP\,56004}
\subtitle{short but sweet?}
\author{Fuzzy George}
\date{30. February 2002}
\publishers{Adviser Prof. John Eccentric Doe}
\maketitle
\end{document}
```

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

However if one takes a book and opens it, one notices very quickly at least one title page under the cover within the so-called inner book. Precisely these title pages are produced by `\maketitle`.

As is the case with the half-title, the full title page belongs to the inner book, and therefore should have the same page layout as the rest of the document. A cover is actually something that should be created in a separate document. The cover often has a very individual format. It can also be designed with the help of a graphics or DTP program. A separate document should also be used because the cover will be printed on a different medium, possibly cardboard, and possibly with another printer.

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

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

`\dedication{`*`dedication`*`}`

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

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

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

Please use your own favorite pet names.

`abstract`

scrartcl, scrreprt

Particularly with articles, more rarely with reports, there is a summary directly under the

title and before the table of contents. Therefore, this is often considered a part of the titles. Some LaTeX classes offer a special environment for this summary, the `abstract` environment. This is output directly, at it is not a component of the titles set by `\maketitle`. Please note that `abstract` is an environment, not a command. Whether the summary has a heading or not is determined by the option `abstract` (see section 3.1.7, page 51)

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

3.4. The Table of Contents

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

`\tableofcontents`
`\contentsname`

The production of the table of contents is done by the `\tableofcontents` command. To get a correct table of contents, at least two LaTeX runs are necessary after every change. The option `listof=totoc` causes the lists of figures and tables to be included in the table of contents. `index=totoc` is the corresponding option for the index. This is rather uncommon in classical typography. One does find the bibliography included in the table of contents slightly more frequently. This can be obtained with the options `bibliography=totoc` and `bibliography=totocnumbered`. These options are explained in section 3.1.5, page 48.

The table of contents is set as an unnumbered chapter and is therefore subject to the side effects of the standard `\chapter*` command, which are described in section 3.6.2, page 75. However, the running headings for left and right pages are correctly filled with the heading of the table of contents.

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

There are two variants for the construction of the table of contents. With the standard variant, the titles of the sectional units are indented so that the unit number is flush left to the edge of the text of the next upper sectional unit. However, the space for the numbers is thereby limited and is only sufficient for a little more than 1.5 places per unit level. Should this become a problem, the option `toc=flat` can be used to set the behaviour such that all entries in the table of contents are set flush left under one another. As explained in section 3.1.5, page 49, several LaTeX runs are needed.

The entry for the highest sectional unit below `\part`, i. e., `\chapter` with scrbook and scrreprt or `\section` with scrartcl is not indented. The font style is however affected by the settings of

the element `disposition` (see table 3.4, page 54). There are no dots between the text of the sectional unit heading and the page number. The typographic reasons for this are that the font is usually different, and the desire for appropriate emphasis. The table of contents of this manual is a good example of these considerations.

`tocdepth`

Normally, the units included in the table of contents are all the units from `\part` to `\subsection` (for the classes scrbook and scrreprt) or from `\part` to `\subsubsection` (for the class scrartcl). The inclusion of a sectional unit in the table of contents is controlled by the counter `tocdepth`. This has the value −1 for `\part`, 0 for `\chapter`, and so on. By setting, incrementing or decrementing the counter, one can choose the lowest sectional unit level to be included in the table of contents. The same happens with the standard classes.

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

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

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

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

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

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

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

KOMA-Script has always attempted to avoid page breaking directly between a sectional unit and the adjacent next lower unit, for example, between a chapter title and its first section title. v2.96a However, the mechanism worked poorly or not at all until version 2.96. In version 2.96a the mechanism was much improved and should now always work correctly. There can be changes in the page breaking in the table of contents as a result though. Thus, the new mechanism is only active, if the compatibility option is not set to version 2.96 or less (see option `version`,

section 3.1.1, page 42). The mechanism also does not work if the commands to generate the table of contents are redefined, for example, by the use of the package tocloft.

3.5. Lists of Floats

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

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

These commands generate a list of tables or figures. Changes in the document that modify these lists will require two LaTeX runs in order to take effect. The layout of the lists can be influenced by the options `listof=graduated` and `listof=flat` (see section 3.1.6, page 50). Moreover, the options `listof=totoc` and `listof=numbered` have indirect influence (see section 3.1.5, page 48).

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

3.6. Main Text

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

scrbook

3.6.1. Separation

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

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

The macro `\frontmatter` introduces the front matter in which roman numerals are used for the page numbers. Chapter headings in a front matter are not numbered. The section titles would be numbered, start at chapter 0, and would be consecutively numbered across chapter

boundaries. However, this is of no import, as the front matter is used only for the title pages, table of contents, lists of figures and tables, and a foreword. The foreword can thus be set as a normal chapter. A foreword should never be divided into sections but kept as short as possible. Therefore in the foreword there is no need for a deeper structuring than the chapter level.

v2.97e In case the user sees things differently and wishes to use numbered sections in the chapters of the front matter, as of version 2.97e the section numbering no longer contains the chapter number. This change only takes effect when the compatibility option is set to at least version 2.97e (see option `version`, section 3.1.1, page 42). It is explicity noted that this creates a confusion with chapter numbers! The use of `\addsec` and `\section*` (see section 3.6.2, page 75 and page 75) are thus, in the author's opinion, far more preferable.

v2.97e As of version 2.97e the numbering of float environments, such as tables and figures, and equation numbers in the front matter also contain no chapter number part. To take effect this too requires the corresponding compatibility setting (see option `version`, section 3.1.1, page 42).

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

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

3.6.2. Structuring the Document

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

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

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

The title of the level part (`\part`) is distinguished from other sectioning levels by being numbered independently from the other parts. This means that the chapter level (in scrbook or

scrreprt), or the section level (in scrartcl) is numbered consecutively over all parts. Furthermore, for classes scrbook and scrreprt the title of the part level together with the corresponding preamble (see `\setpartpreamble`, page 81) is set on a separate page.

scrartcl

`\chapter` only exists in book or report classes, that is, in classes book, scrbook, report and scrreport, but not in the article classes article and scrartcl. In addition to this, the command `\chapter` in KOMA-Script differs substantially from the version in the standard class. In the standard classes the chapter number is used together with the prefix "Chapter", or the corresponding word in the appropriate language, on a separate line above the actual chapter title test. This overpowering style is replaced in KOMA-Script by a simple chapter number before the chapter heading text, can however be reverted by the option `chapterprefix` (see section 3.1.3, page 46).

scrbook, scrreprt

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

v2.8p

The font of all headings can be changed with the commands `\setkomafont` and `\addtokomafont` described in section 3.2.1. In doing this, generally the element `disposition` is used, followed by a specific element for every section level (see table 3.4, page 54). The font for the element `disposition` is predefined as `\normalfont\normalcolor\sffamily\bfseries`. The default font size for the specific elements depends on the options `headings=big`, `headings=normal` and `headings=small` (see section 3.1.4, page 47). The defaults are listed in table 3.10

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

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

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

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

or simply using:

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

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

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

Table 3.10.: Default font sizes for different levels of document structuring in scrbook and scrreprt

class option	element	default
headings=big	part	\Huge
	partnumber	\huge
	chapter	\huge
	section	\Large
	subsection	\large
	subsubsection	\normalsize
	paragraph	\normalsize
	subparagraph	\normalsize
headings=normal	part	\huge
	partnumber	\huge
	chapter	\LARGE
	section	\Large
	subsection	\large
	subsubsection	\normalsize
	paragraph	\normalsize
	subparagraph	\normalsize
headings=small	part	\LARGE
	partnumber	\LARGE
	chapter	\Large
	section	\large
	subsection	\normalsize
	subsubsection	\normalsize
	paragraph	\normalsize
	subparagraph	\normalsize

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

Please be warned of misusing the possibilities of font switching to mix fonts, font sizes and font attributes excessively. Picking the most suitable font for a given task is a hard task even for professionals and has almost nothing to do with the personal tastes of non-experts. Please refer to the citation at the end of section 2.7, page 37 and to the following explanation.

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

There is a rule in typography which states that one should mix as few fonts as possible. Using

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

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

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

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

scrbook, scrreprt

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

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

v2.8p

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

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

In addition to the commands of the standard classes KOMA-Script offers the new commands `\addsec` and `\addchap`. They are similar to the standard commands `\chapter` and `\section`

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

scrartcl

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

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

v2.8p

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

`\minisec{`*Heading*`}`

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

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

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

```
\minisec{Items needed}

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

```
\minisec{Assembly}
At first one searches the mouse-hole and puts the drawing pin
directly behind the hole.  Thus the mouse cannot escape during ↩
 the
following actions.

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

Which gives:

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

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

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

2.96a

The font type of the structuring command `\minisec` be changed using the element `disposition` (see table 3.4, page 54) and `minisec`. Default setting of element `minisec` is empty, so the default of the element `disposition` is active.

`\raggedsection`

In the standard classes headings are set as justified text. That means that hyphenated words can occur and headings with more than one line are stretched up to the text border. This is a rather uncommon approach in typography. KOMA-Script therefore formats the headings left aligned with hanging indentation using `\raggedsection` with the definition:

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

This command can be redefined with `\renewcommand`.

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

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

or more compactly:

```
\let\raggedsection\relax
```

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

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

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

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

scrbook, scrreprt

As described in section 3.1.7, page 51, dots in section numbers should be handled for the German-speaking region according to the rules given in [DUD96]. The command `\autodot` in KOMA-Script ensures that these rules are being followed. In all levels except for `\part` a dot is followed by a further `\enskip`. This corresponds to a horizontal skip of 0.5 em.

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

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

```
\newcommand*{\partformat}{\partname~\thepart\autodot}
\newcommand*{\chapterformat}{%
  \chapappifchapterprefix{\ }\thechapter\autodot\enskip}
```

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

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

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

Strictly speaking, you could do without `\autodot` at this point and insert a fixed dot instead. As `\part` is numbered with roman numerals, according to [DUD96] a dot has to be applied. However, you thereby give up the possibility to use one of the options `numbers=enddot` and `numbers=noenddot` and optionally depart from the rules. More details concerning class options can be found in section 3.1.7, page 51.

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

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

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

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

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

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

scrbook, scrreprt (v2.96)

These two commands are not only used internally by KOMA-Script but are also provided to the user. Later it will be shown how they can be used for example to redefine other commands. Using the layout option `chapterprefix` (see section 3.1.3, page 46) `\chapappifchapterprefix` outputs the word "Chapter" in the main part of the document in the current language, followed by *additional text*. In the appendix, the word "Appendix" in the current language is output instead, followed by *additional text*. If the option `chapterprefix=false` is set, then nothing is output.

The command `\chapapp` always outputs the word "Chapter" or "Appendix". In this case the options `chapterprefix` and `chapterprefix=false` have no effect.

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

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

As mentioned in section 3.2.2 the page style `headings` works with automatic running headings. For this, the commands `\chaptermark` and `\sectionmark`, or `\sectionmark` and `\subsectionmark`, respectively, are defined. Every structuring command (`\chapter`, `\section` ...) automatically carries out the respective `\...mark` command. The parameter passed contains the text of the section heading. The respective section number is added automatically in the `\...mark` command. The formatting is done according to the section level with one of the three commands `\chaptermarkformat`, `\sectionmarkformat` or `\subsectionmarkformat`. Of course there is no command `\chaptermark` or `\chaptermarkformat` in scrartcl. Accordingly, `\subsectionmark` and `\subsectionmarkformat` exist only in scrartcl. This changes when you use the scrpage2 package (see chapter 4).

scrbook, scrreprt scrartcl

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

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

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

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

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

`secnumdepth`

Per default, in the classes scrbook and scrreprt the section levels from `\part` down to `\subsection` and in the class scrartcl the levels from `\part` down to `\subsubsection` are

numbered. This is controlled by the LaTeX counter `secnumdepth`. The value −1 represents `\part`, 0 the level `\chapter`, and so on. By defining, incrementing or decrementing this counter you can determine down to which level the headings are numbered. The same applies in the standard classes. Please refer also to the explanation concerning the counter `tocdepth` in section 3.4, page 70.

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

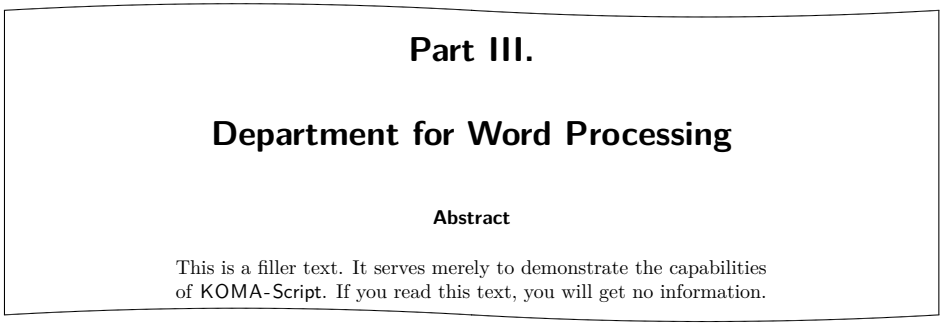
scrbook, scrreprt

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

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

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

Depending on the settings for the heading font size (see section 3.1.4, page 47) and the options for the `abstract` environment (see section 3.1.7, page 51), the result would look similar to:



Please note that it is *you* who is responsible for the spaces between the heading, preamble and the following text. Please note also that there is no `abstract` environment in the class scrbook (see section 3.3, page 68).

v2.8p

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

o: above the heading

u: below the heading

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

l: left-aligned

r: right-aligned

c: centered

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

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

scrbook, v2.8q, scrreprt

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

The dictum together with an optional *author* is inserted in a `\parbox` (see [Tea05b]) of width `\dictumwidth`. Yet `\dictumwidth` is not a length which can be set with `\setlength`. It is a macro that can be redefined using `\renewcommand`. Default setting is `0.3333\textwidth`, which is a third of the textwidth. The box itself is positioned with the command `\raggeddictum`. Default here is `\raggedleft`, that is, right justified. The command `\raggeddictum` can be redefined using `\renewcommand`.

Table 3.11.: Default settings for the elements of a dictum

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

Within the box the *dictum* is set using \raggeddictumtext. Default setting is \raggedright, that is, left justified. Similarly to \raggeddictum this can be redefined with \renewcommand. The output uses the default font setting for the element dictumtext, which can be changed with the commands from section 3.2.1. Default settings are listed in table 3.11.

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

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

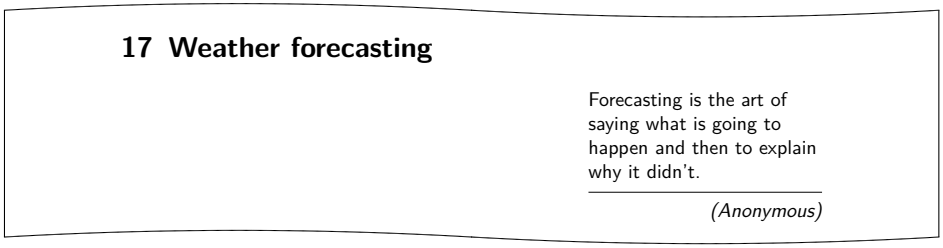
Thus the *author* is set enclosed in rounded parenthesis. For the element dictumauthor a different font than for the element dictumtext can be defined. Default settings are listed in table 3.11. Changes can be made using the commands from section 3.2.1. If \dictum is used within the macro \setchapterpreamble or \setpartpreamble you have to take care of the following: the horizontal positioning is always done with \raggeddictum. Therefore, the optional argument for horizontal positioning which is implemented for these two commands has no effect. \textwidth is not the width of the whole text corpus but the actually used text width. If \dictumwidth is set to .5\textwidth and \setchapterpreamble has an optional width of .5\textwidth too, you will get a box with a width one quarter of the text width. Therefore, if you use \dictum it is recommended to refrain from setting the optional width for \setchapterpreamble or \setpartpreamble.

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

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

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

The output would look as follows:



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

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

For a somewhat more sophisticated formatting of left- or right-aligned paragraphs including hyphenation you can use the package ragged2e [Sch09].

3.6.3. Footnotes

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

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

Similarly to the standard classes, footnotes in KOMA-Script are produced with the `\footnote` command, or alternatively the pairwise usage of the commands `\footnotemark` and `\footnotetext`. As in the standard classes it is possible that a page break occurs within a footnote. Normally this happens if the footnote mark is placed so near the bottom of a page as to leave LaTeX no choice but to break the footnote onto the next page.

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

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

All following lines will be indented by the length of *indent*. If the optional parameter *mark width* is not specified, it defaults to *indent*. If the footnote consists of more than one

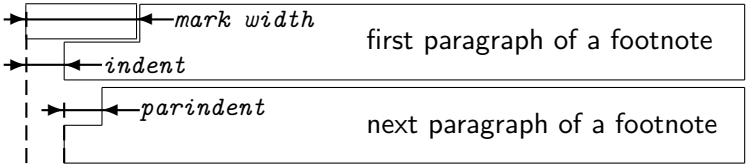


Figure 3.1.: Parameters that control the footnote layout

paragraph, then the first line of a paragraph is indented, in addition to *indent*, by the value of *parindent*.

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

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

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

v2.8q The font element `footnote` determines the font of the footnote including the footnote mark. Using the element `footnotelabel` the font of the footnote mark can be changed separately with the commands mentioned in section 3.2.1 Please refer also to table 3.4, page 54. Default setting is no change in the font.

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

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

v2.8q In the above the font for the element `footnotereference` is applied (see table 3.4, page 54). Thus the footnote marks in the text and in the footnote itself are identical. The font can be changed with the commands described in section 3.2.1.

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

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

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

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

```
\deffootnote{1.5em}{1em}{%
```

```
\makebox[1.5em][l]{\thefootnotemark}}
```

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

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

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

3.6.4. Lists

Both LaTeX and the standard classes offer different environments for lists. Though slightly changed or extended all these list are of course offered in KOMA-Script as well. In general all lists — even of different kind — can be nested up to four levels. From a typographical view, anything more would make no sense, as more than three levels can no longer be easily perceived. The recommended procedure in such a case is to split the large list into several smaller ones.

```
itemize
\item
\labelitemi
\labelitemii
\labelitemiii
\labelitemiv
```

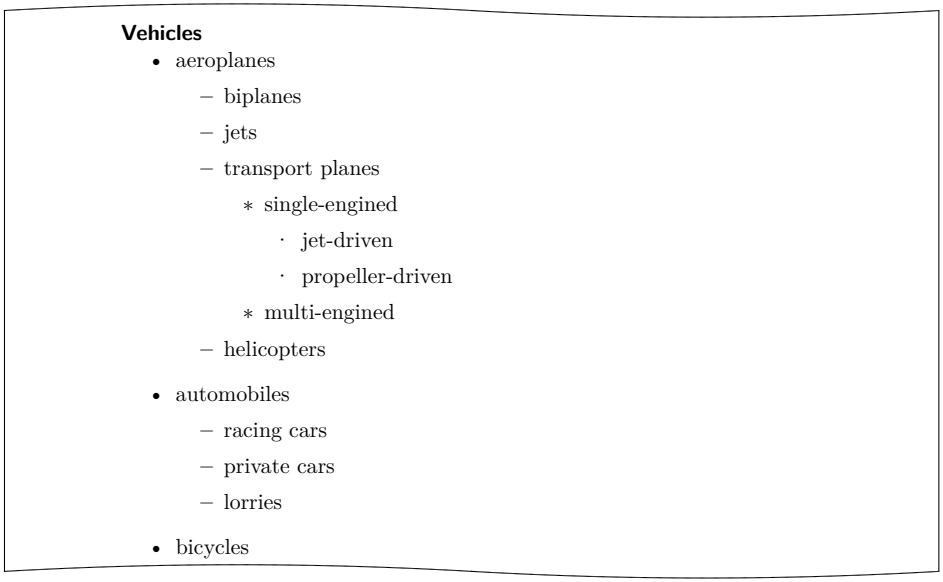
The simplest form of a list is an `itemize` list. The users of a certain disliked word processing package often refer to this form of a list as *bulletpoints*. Presumably, these users are unable to envisage that, depending on the level, a different symbol from a large dot could be used to introduce each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

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

```
\minisec{Vehicles}
\begin{itemize}
  \item aeroplanes
  \begin{itemize}
    \item biplane
    \item jets
    \item transport planes
    \begin{itemize}
      \item single-engined
```

```
        \begin{itemize}
          \item jet-driven
          \item propeller-driven
        \end{itemize}
        \item multi-engined
      \end{itemize}
      \item helicopters
    \end{itemize}
    \item automobiles
    \begin{itemize}
      \item racing cars
      \item private cars
      \item lorries
    \end{itemize}
    \item bicycles
  \end{itemize}
```

As output you get:



Vehicles

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

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

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

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

Vehicles

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

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

`description`
`\item[`*item*`]`

A further list form is the description list. Its main use is the description of several items. The item itself is an optional parameter in `\item`. The fontwhich is responsible for emphasizing the item can be changed with the commands for the element `descriptionlabel` (see table 3.4, page 54) described in section 3.2.1. Default setting is `\sffamily\bfseries`.

v2.8p

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

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

you redefine the font accordingly.

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

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

This abbreviated version gives:

empty is the page style without any header or footer.

plain is the page style without headings.

headings is the page style with running headings.

myheadings is the page style for manual headings.

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

An additional form of a description list is only available in the KOMA-Script classes: the `labeling` environment. Unlike the `description` environment, you can provide a pattern whose length determines the indentation of all items. Furthermore, you can put an optional *delimiter* between the item and its description. The fontwhich is responsible for emphasizing the item and the separator can be changed with the commands for the element `labelinglabel` and `labelingseparator` (see table 3.4, page 54) described in section 3.2.1.

v3.01

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

```
\setkomafont{labelinglabel}{\ttfamily}
\setkomafont{labelingseparator}{\normalfont}
```

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

As result we get:

`empty` – Page style without header and footer

`plain` – Page style for chapter beginnings without headings

`headings` – Page style for running headings

`myheadings` – Page style for manual headings

As can be seen in this example, a font changing command may be set in the usual way. But if you don't want the font of the separator be changed in the same way like the font of the label, you have to set the font of the separator different.

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

`verse`

Usually the `verse` environment is not perceived as a list environment because you do not work with `\item` commands. Instead, fixed line breaks are used within the `flushleft` environment. Yet internally in both the standard classes as well as KOMA-Script it is indeed a list environment.

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

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

```
\begin{verse}
  As soon as Wolf began to feel\\*
```

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

The result is as follows:

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

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

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

where a line break occurs within a line of verse:

> Both the philosopher and the house-owner have always something to repair.
>
> Don't trust a man, my son, who tells you that he has never lied.

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

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

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

quote
quotation

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

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

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

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

  Little Bird sat on twig. ‘‘Whillo, Tom! I heed you.
  I’ve a guess, I’ve a guess where your fancies lead you.
  Shall I go, shall I go, bring him word to meet you?’’
\end{quotation}
```

The result is:

A small example for a short anecdote:

> The old year was turning brown; the West Wind was calling;
>
> Tom caught the beechen leaf in the forest falling. “I’ve caught the happy day blown me by the breezes! Why wait till morrow-year? I’ll take it when me pleases. This I’ll mend my boat and journey as it chances west down the withy-stream, following my fancies!”
>
> Little Bird sat on twig. “Whillo, Tom! I heed you. I’ve a guess, I’ve a guess where your fancies lead you. Shall I go, shall I go, bring him word to meet you?”

Using a `quote` environment instead you get:

A small example for a short anecdote:

> The old year was turning brown; the West Wind was calling;
>
> Tom caught the beechen leaf in the forest falling. “I’ve caught the happy day blown me by the breezes! Why wait till morrow-year? I’ll take it when me pleases. This I’ll mend my boat and journey as it chances west down the withy-stream, following my fancies!”
>
> Little Bird sat on twig. “Whillo, Tom! I heed you. I’ve a guess, I’ve a guess where your fancies lead you. Shall I go, shall I go, bring him word to meet you?”

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

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

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

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

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

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

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

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

You define yourself the following environment:

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

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

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

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

Incidentally, whether a page is going to be on the left or right side of the book can not be determined for certain in the first LaTeX run. For details please refer to the explanation of the commands `\ifthispageodd` and `\ifthispagewasodd` in section 3.2.2, page 62.

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

3.6.5. Margin Notes

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

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

Example: In the introduction, the class name scrartcl can be found in the margin. This can be produced[2] with:

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

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

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

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

Unfortunately `\marginpar` does not always work correctly in two-sided layout. Whether a margin note will end up in the left or right margin is already decided while evaluating the command `\marginpar`. If the output routine now shifts the margin note onto the next page the formatting is no longer correct. This behaviour is deeply rooted within LaTeX and was therefore declared a feature by the LaTeX3 team. `\marginline` suffers from this "feature" too. The package mparhack (see [SU03]) offers a standard solution for this problem which naturally benefits also `\marginpar` and `\marginline`.

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

3.6.6. Tables and Figures

With the floating environments LaTeX offers a very capable and comfortable mechanism for automatic placement of figures and tables. But often these floating environments are slightly misunderstood by beginners. They often ask for a fixed position of a table or figure within the text. However, since these floating environments are being referenced in the text this is not necessary in most cases. It is also not sensible because such an object can only be set on the page if there is

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

enough space left for it. If this is not the case the object would have to be shifted onto the next page, thereby possibly leaving a huge blank space on the page before.

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

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

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

In the standard classes caption text *title* of tables and figures is inserted with the `\caption` command below the table or figure. In general this is correct for figures. Opinions differ as to whether captions of tables are to be placed above or, consistent with captions of figures, below the table. That is the reason why KOMA-Script, unlike the standard classes, offers `\captionbelow` for captions below and `\captionabove` for captions above tables or figures. Using `\caption` for figures always produces captions below the figure, whereas with tables the behaviour of `\caption` can be modified using the options `captions=tableheading` and `captions=tablesignature` (see section 3.1.7, page 52). For compatibility reasons the default behaviour of `\caption` used with tables is similar to `\captionbelow`.

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

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

Then you would get the unsatisfying result:

Table 30.2: This is an example table.

This	is	an	example.
This	is	an	example.
This	is	an	example.

Using KOMA-Script you write instead:

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

Then you get:

Table 30.2: This is just an example table

This	is	an	example.
This	is	an	example.
This	is	an	example.

Since you want all your tables typeset with captions above, you could of course use the option `captions=tableheading` instead (see section 3.1.7, page 52). Then you can use `\caption` as you would in the standard classes. You will get the same result as with `\captionabove`.

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

If the longtable package is used, KOMA-Script ensures that captions above tables which are placed within a `longtable` environment have the same appearance as those in a normal `table` environment. This also means that you can apply the same settings as in a `table` environment. Please note that in the longtable package the maximum width of a table caption can be limited and the default is set to 4 in (see [Car04]). Used together with KOMA-Script this mechanism in longtable works only if the class option `origlongtable` is set (see section 3.1.7, page 52). If the caption2 or caption package (see [Som08]) is loaded, table captions are handled by this package.

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

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

v2.8q

Apart from captions above and below the figure, one often finds captions, in particular with small figures, which are placed beside the figure. In general in this case both the baseline of

the figure and of the caption are aligned at the bottom. With some fiddling and the use of two \parbox commands this could also be achieved in the standard classes. However, KOMA-Script offers a special environment for this which can be used within the floating environment. The first optional parameter *entry* and the obligatory parameter *title* mean the same as the corresponding parameters of \caption, \captionabove or \captionbelow. The caption text *title* is placed beside the content of the environment in this case.

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

l – left

r – right

i – inner margin in two-sided layout

o – outer margin in two-sided layout

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

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

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

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

Example: An example for the usage of the captionbeside environment can be found in figure 3.2. This figure was typeset with:

```
\begin{figure}
  \begin{captionbeside}[Example for a figure description]%
    {A figure description which is neither above nor
     below, but beside the figure}[i][\linewidth][2em]*
    \fbox{%
      \parbox[b][5\baselineskip][c]{.25\textwidth}{%
        \hspace*{\fill}\KOMAScript\hspace*{\fill}\par}}
  \end{captionbeside}
```

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

```
  \label{fig:maincls.captionbeside}
\end{figure}
```

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

v2.8p

The font style for the description and the label — "Figure" or "Table", followed by the number and the delimiter — can be changed with the commands described in section 3.2.1. The respective elements for this are `caption` and `captionlabel` (see table 3.4, page 54). First the font style for the element `caption` is applied to the element `captionlabel` too. After this the font style of `captionlabel` is applied on the respective element. The default settings are listed in table 3.12.

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.

`komaabove`
`komabelow`

float

If you use the float package the appearance of the float environments is solely defined by the

Table 3.12.: Font defaults for the elements of figure or table captions

element	default
`caption`	`\normalfont`
`captionlabel`	`\normalfont`

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

`\captionformat`

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

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

This too can be changed with `\renewcommand`.

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

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

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

`\figureformat`
`\tableformat`

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

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

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

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

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

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

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

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

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

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

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

Whether one-line captions are set as captions with more than one line or are treated separately is specified with the class options `captions=oneline` and `captions=nooneline`. For details please refer to the explanations of these options in section 3.1.3, page 46.

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

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

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

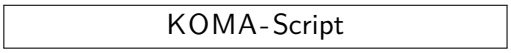


Figure 3.3.: Equivalent to the standard setting, similar to the usage of `\setcaphanging`

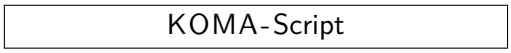


Figure 3.4.: With slightly hanging indentation starting at the second line using `\setcapindent{1em}`

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

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

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

v2.8q

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

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

`l` – left-aligned

`c` – centered

`r` – right-aligned

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

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

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

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

with the same problem with justification inside or outside as mentioned with the macro \setcapwidth. Furthermore, the usage of \setcapmargin or \setcapmargin* switches on the option captions=nooneline (see section 3.1.3, page 46) for the captions which are typeset with this margin setting.

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

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

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

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

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

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

Having done this you can calculate the width directly with the LaTeX command \settowidth (see [Tea05b]) in many cases. A possible solution would look as follows:

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

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

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

3. outputting the argument

in:

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

Using this command the example abbreviates to:

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

However, commands have the disadvantage that errors in the macros of the argument in case of arguments with more than one line are not reported with the very accurate line numbers by LaTeX. Thus in some cases the use of an environment has advantages. Then, however, the question arises of how the width of the content of the environment can be determined. The solution involves the `lrbox` environment, described in [Tea05b]:

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

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

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

Admittedly, the environment in this example is not necessary. However, its defini-

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

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

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

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

```
\begin{figure}
  \centering%
  \begin{FigureSetsCaptionWidth}
    \fbox{\rule{0pt}{5\baselineskip}%
      \hspace{1em}\KOMAScript\hspace{1em}}
  \end{FigureSetsCaptionWidth}
  \parbox[b]{\FigureWidth}{%
    \caption{Example of a centered caption
             below the figure}
  }
\end{figure}
```

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

3.6.7. Logical Markup of Text

LaTeX offers different possibilities for logical markup of text. Strictly speaking, a heading is a kind of markup too. However, in this section we are only concerned with direct markup, i. e., markup which does not have an additional semantic meaning and which can be used for different purposes. More details on the normally defined possibilities can be found in [OPHS99], [Tea05b] and [Tea05a].

`\textsubscript{`*text*`}`

In section 3.6.3, page 84, the command `\textsuperscript` was already introduced as an integral part of the LaTeX kernel. Unfortunately, LaTeX itself does not offer a command to produce text in subscript instead of superscript. KOMA-Script defines `\textsubscript` for this purpose.

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

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

Using this you then write:

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

The output looks as follows:

> The cell produces its energy from reaction of $C_6H_{12}O_6$ and O_2 to produce H_2O and CO_2. However, arsenic (As) has a quite detrimental effect on the metabolism.

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

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

Now the output in the whole document changes to:

> The cell produces its energy partly from reaction of $\mathsf{C_6H_{12}O_6}$ and $\mathsf{O_2}$ to produce $\mathsf{H_2O}$ and $\mathsf{CO_2}$. However, arsenic (As) has a quite detrimental effect on the metabolism.

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

3.7. Appendix

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

`\appendix`

The appendix in the standard as well as the KOMA-Script classes is introduced with `\appendix`. This command switches, among other things, the chapter numbering to upper case letters, also ensuring that the rules according to [DUD96] are followed (for German-speaking regions). These rules are explained in more detail in the description of the class options `numbers=enddot` and `numbers=noenddot` in section 3.1.7, page 51.

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

`\appendixmore`

There is a peculiarity within the `\appendix` command in the KOMA-Script classes. If the command `\appendixmore` is defined, this command is executed also by the `\appendix` command. Internally the KOMA-Script classes scrbook and scrreprt take advantage of this behaviour to implement the options `appendixprefix` and `appendixprefix=false` (see section 3.1.3, page 46). You should take note of this in case you decide to define or redefine the `\appendixmore`. In case one of these options is set, you will receive an error message when using `\newcommand{\appendixmore}{...}`. This behaviour is intended to prevent you from disabling options without noticing it.

Example: You do not want the chapters in the main part of the classes scrbook or scrreprt to be introduced by a prefix line (see layout options `chapterprefix` and `chapterprefix=false` in section 3.1.3, page 46). For consistency you also do not want such a line in the appendix either. Instead, you would like to see the word "Chapter" in the language of your choice written in front of the chapter letter and, simultaneously, in the page headings. Instead of using the either layout option `appendixprefix` or `appendixprefix=false`, you would define in the document preamble:

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

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

It is also possible to get a similar behaviour of the appendix for the class scrartcl. You would write in the preamble of your document:

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

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

Redefined commands are explained in more detail in section 3.6.2, page 78 and page 80.

`\setbibpreamble{`*preamble*`}`

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

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

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

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

`\setindexpreamble{`*preamble*`}`

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

page numbers in different ways.

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

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

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

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

3.8. Obsolete Commands

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

`\sectfont`

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

`\capfont`
`\caplabelfont`

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

`\descfont`

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

Adapting Page Headers and Footers with scrpage2

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

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

Included as part of the basic functionality of scrpage2 are various predefined and configurable page styles.

4.1. Basic Functionality

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

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

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

4.1.1. Predefined Page Styles

scrheadings
scrplain

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

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

The page styles include three boxes in both the header and the footer. The commands modifying the content of these boxes can be seen in figure 4.1. Commands in the middle column modify the box contents on both odd and even pages.

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

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

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

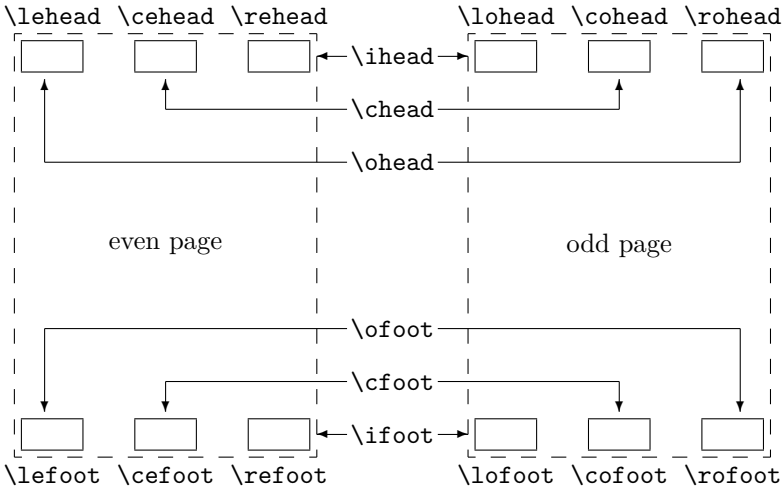


Figure 4.1.: Commands for modification of page styles `scrheadings` and `scrplain` and their relationship to header and footer elements

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

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

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

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

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

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

As mentioned above, there is a new `plain` page style which corresponds to `scrheadings`. Since it should also be possible to customize this style, the commands support an optional

argument with which the contents of the appropriate fields of this `plain` page style can be modified.

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

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

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

`\clearscrheadings`
`\clearscrplain`
`\clearscrheadfoot`

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

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

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

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

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

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

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

`\leftmark`
`\rightmark`

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

`\headmark`

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

`\pagemark`

This command returns the formatted page number. The formatting can be controlled by `\pnumfont`, introduced in section 4.1.3, page 115, which `\pagemark` heeds automatically. Alternatively, `\setkomafont` can be used if a KOMA-Script class is used (see section 3.2.1).

`useheadings`

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

4.1.2. Manual and Running Headings

Usually there is a *my*-version of the `headings` page style. If such a page style is active, then the running headings are no longer updated no longer automatically and become manual headings. With scrpage2 a different path is taken. Whether the headings are running or manual is determined by the instructions `\automark` and `\manualmark`, respectively. The default can be set already while loading of the package, with the options `automark` and `manualmark` (see section 4.1.4, page 121).

`\manualmark`

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

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

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

v2.2

The optional argument *right page* is understandably meant only for two-sided documents. In the one-sided case one should normally not use it. With the help of the option `autooneside` one can also set that the optional argument in one-sided mode is ignored automatically (see section 4.1.4, page 122).

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

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

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

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

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

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

4.1.3. Formatting of Header and Footer

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

`\headfont`
`\pnumfont`

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

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

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

From version 2.8p of the KOMA-Script classes a new unified user interface scheme is implemented for font attributes. If scrpage2 is used together with one of these classes, then it is recommended to set up font attributes in the manner described in section 3.2.1.

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

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

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

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

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

`paper` – the width of the paper

`page` – the width of the page

`text` – the width of the text body

`textwithmarginpar` – the width of the text body including margin

`head` – the current header width

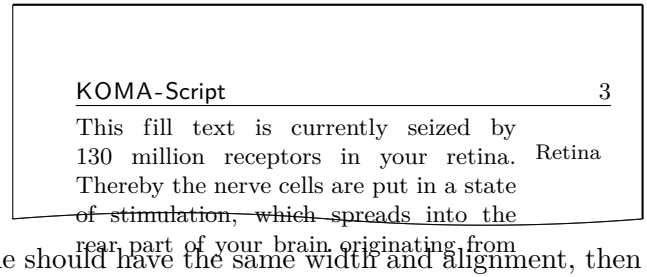
`foot` – the current footer width

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

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

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

which appears like this on an odd page:



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

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

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

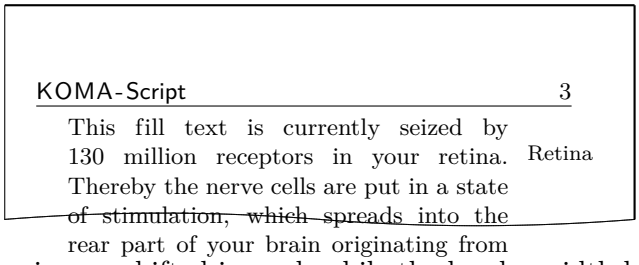
```
\setfootwidth[0pt]{head}
```

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

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

```
\setheadwidth{textwithmarginpar}
```

which appears like this on an odd page:



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

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

Corresponding to the size configuration parameters of header and footer there are commands to modify the lines above and below the header and footer.

`\setheadtopline` – configures the line above the header

`\setheadsepline` – configures the line below the header

`\setfootsepline` – configures the line above the footer

`\setfootbotline` – configures the line below the footer

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

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

v2.2 The optional argument *commands* may be used to specify additional commands to be executed before the respective line is drawn. For example, such commands could be used for changing the color of the line. When using a KOMA-Script class you could also use `\setkomafont` to specify commands for one of the elements `headtopline`, `headsepline`, `footsepline`, `footbottomline`, or `footbotline`. These can then be extended via `\addtokomafont`. See section 3.2.1 for details on the `\setkomafont` and `\addkomafont` commands.

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

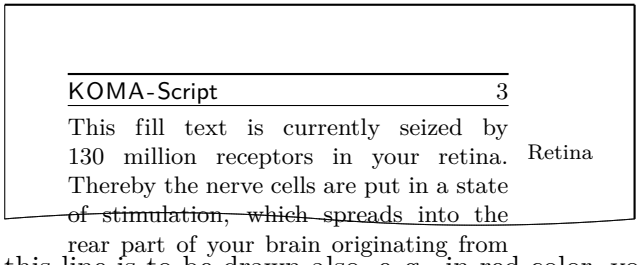
The arguments shown here for the command `\setheadtopline` are of course valid for the other three configuration commands too.

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

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

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

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



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

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

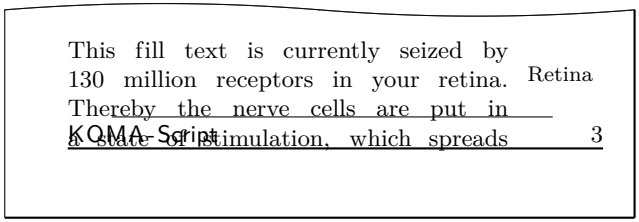
In this example, as well as in the following one, line color is activated by applying the syntax of the color package, so this package must of course be loaded. Since scrpage2 comes without built-in color handling, any package providing color support may be used.

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

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

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

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



Now not everyone will like the alignment of the line above the footer; instead, one would expect the line to be left-aligned. This can only be achieved with a global package option, which will be described together with other package options in the next section 4.1.4.

4.1.4. Package Options

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

These options determine whether the page header or the page footer are considered as part of the page body for the calculation of the type area. The adjustments necessary by the use of these parameters are made by the package typearea (see section 2.4), if this package is loaded after scrpage2. Important here is that when using a KOMA-Script class, these options must be given for the document class and not for the package scrpage2, in order to be effective.

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

Basic adjustment of the lines under and over header and footer can be made with these options. These adjustments are then considered the default for all page styles defined with scrpage2. If one of these options is used, then a line thickness 0.4 pt is set.

Since there is a corresponding `plain` page style to the page style `scrheadings`, the corresponding line in the plain style can also be configured with the `plain...` options. These `plain` options do however work only if the corresponding options without `plain` are activated. Thus, `plainheadtopline` shows no effect without the `headtopline` option set.

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

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

`ilines`
`clines`
`olines`

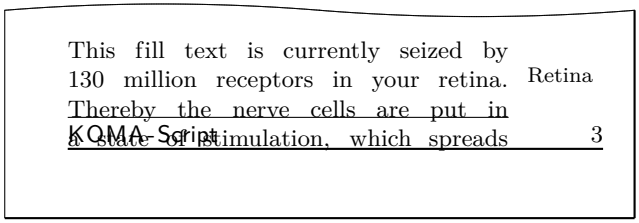
With the definition of the line lengths the case can arise where the lengths are set correctly, but the justification is not as desired because the line will be centered in the header or footer area. With the package options presented here, this specification can be modified for all page styles defined with scrpage2. The option `ilines` sets the justification in such a way that the

lines align to the inside edge. The option `clines` behaves like the default justification, and `olines` aligns at the outside edge.

Example: The next example illustrates the influence of the option `ilines`. Please compare to the example for `\setfootsepline` on page 119.

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

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



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

`automark`
`manualmark`

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

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

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

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

```
\pagestyle{scrheadings}
```

then the manual headings are nevertheless still active.

If you instead use the document class scrbook, then after

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

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

However, the use of

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

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

`autooneside`

This option ensures that the optional parameter of `\automark` will be ignored automatically in one-sided mode. See also the explanation of the command `\automark` in section 4.1.2, page 114.

`komastyle`
`standardstyle`

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

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

`markuppercase`
`markusedcase`

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

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

`nouppercase`

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

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

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

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

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

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

4.2. Defining Own Page Styles

4.2.1. The Interface for Beginners

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

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

The individual parameters have the following meanings:

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

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

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

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

HC – contents of the centered box in the page header

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

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

FC – contents of the centered box in the page footer

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

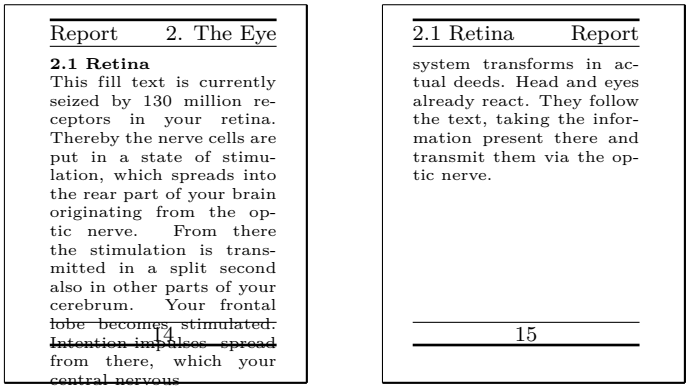
The command `\deftripstyle` definitely represents the simplest possibility of defining page styles. Unfortunately, there are also restrictions connected with this, since in a page range using a page style defined via deftripstyle, no modification of the lines above and below header and footer can take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

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

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

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

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



4.2.2. The Interface for Experts

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

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

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

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

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

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

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

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

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

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

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

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

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

ALT – above line thickness

EP – definition for *even* pages

OP – definition for *odd* pages

OS – definition for *one-sided* layout

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

BLT – below line thickness

If the optional line-parameters are omitted, then the line behaviour remains configurable by the commands introduced in section 4.1.3, page 117.

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

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

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

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

This and the use of the expert interface in connection with other commands provided by scpage2 follows now in the final example.

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

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

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

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

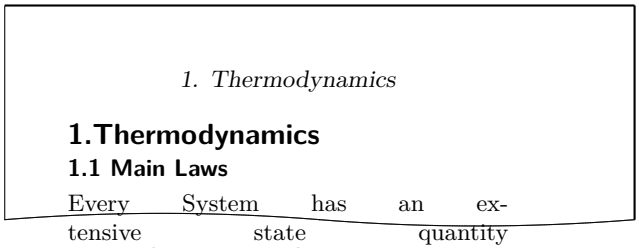
```
\defpagestyle{withLines}{%
  (\textwidth,1pt)
  {with lines\hfill\headmark}{\headmark\hfill with lines}
  {\rlap{\KOMAScript}\hfill \headmark\hfill%
   \llap{with lines}}
  (0pt,0pt)
}{%
  (\textwidth,.4pt)
  {\pagemark\hfill}{\hfill\pagemark}
  {\hfill\pagemark\hfill}
  (\textwidth,1pt)
}
```

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

Instead of `\leftmark` one would expect the use of `\rightmark` in the parameter of `\chead`, since the chapter starts on an even page. But, because of internal LaTeX definitions, this does not work. It only returns an empty string.

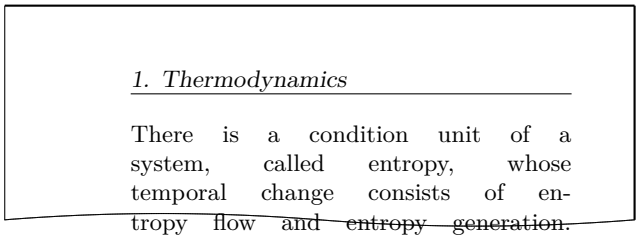
```
\begin{document}
```

```
\pagestyle{scrheadings}
\chapter{Thermodynamics}

\chead[\leftmark]{}

\section{Main Laws}
Every system has an extensive state quantity called
Energy. In a closed system the energy is constant.
```

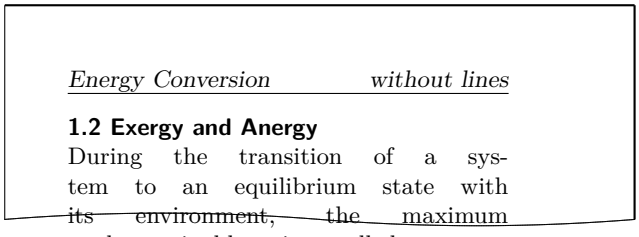


After starting a new page the page style `scrheadings` is active and thus the separation line below the header is visible.

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



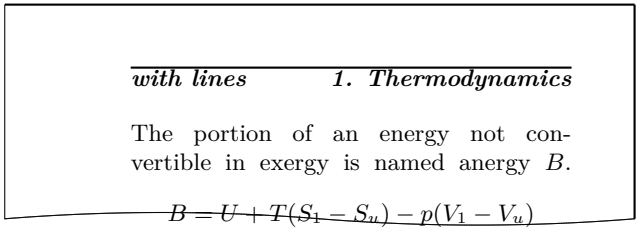
After switching to the next page, the automatic update of the running headings is disabled using `\manualmark`, and the page style `withoutLines` becomes active. Since no line parameters are given in the definition of this page style, the default configuration is used, which draws a separation line between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



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

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

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



4.2.3. Managing Page Styles

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

Weekday and Time Using scrdate and scrtime

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

5.1. The Name of the Current Day of the Week Using scrdate

The first problem is the question of the current day of the week. The answer may be given using the package scrdate.

`\todaysname`

You should know that with `\today` one obtains the current date in a language-dependent spelling. scrdate offers you the command `\todaysname` with which one can obtain the name of the current day of the week in a language-dependent spelling.

Example: In your document you want to show the name of the weekday on which the `dvi` file was generated using LaTeX. To do this, you write:

```
I have done the {\LaTeX} run of this document on a \↩
todaysname.
```

This will result in, e. g.:

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

Note that the package is not able to decline words. The known terms are the nominative singular that may be used, e. g., in the date of a letter. Given this limitation, the example above can work correctly only for some languages.

Tip: The names of the weekdays are saved in capitalized form, i. e., the first letter is a capital letter, all the others are lowercase letters. But for some languages you may need the names completely in lowercase. You may achieve this using the standard LaTeX command `\MakeLowercase`. You simply have to write `\MakeLowercase{\todaysname}`.

`\nameday{`*name*`}`

Analogous to how the output of `\today` can be modified using `\date`, so the output of `\todaysname` can be changed to *name* by using `\nameday`.

Example: You change the current date to a fixed value using `\date`. You are not interested in the actual name of the day, but want only to show that it is a workday. So you set:

```
\nameday{workday}
```

After this the previous example will result in:

I have done the LaTeX run of this document on a workday.

Currently the package scrdate knows the languages english (english, american, USenglish, UKenglish and british), german (german, ngerman, austrian, and naustrian), french, italian, spanish, croatian, finnish, and norsk. If you want to configure it for other languages, see `scrdate.dtx`.

In the current implementation it does not matter whether you load scrdate before or after german, ngerman, babel or similar packages. The current language will be set up at `\begin{document}`.

To explain a little bit more exactly: while you are using a language selection which works in a compatible way to babel or german, the correct language will be used by scrdate. If you are using another language selection you will get (US) english names. In `scrdate.dtx` you will find the description of the scrdate commands for defining the names.

5.2. Getting the Time with Package scrtime

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

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

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

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

Example: The line

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

results, for example, in:

Your train departs at 11:30.

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 30 minutes old.

or:

This day is already 12 hours and 25 minutes old.

`\settime{`*time*`}`

`\settime` sets the output of `\thistime` and `\thistime*` to the value *time*. Now the optional parameter of `\thistime` or `\thistime*` is ignored, since the result of `\thistime` or `\thistime*` was completely determined using `\settime`.

`12h`
`24h`

Using the options `12h` and `24h` one can select whether the result of `\thistime` and `\thistime*` is in 12- or in 24-hour format. The default is `24h`. The option has no effect on the results of `\thistime` and `\thistime*` if `\settime` is used.

The New Letter Class scrlttr2

v2.8q

Since the June 2002 release KOMA-Script provides a completely rewritten letter class. Although part of the code is identical to that of the main classes described in chapter 3, letters are quite different from articles, reports, books, and suchlike. That alone justifies a separate chapter about the letter class. But there is another reason for a chapter on scrlttr2. This class has been redeveloped from scratch and provides a new user interface different from every other class the author knows of. This new user interface may be uncommon, but the author is convinced both experienced and new KOMA-Script users will benefit from its advantages.

6.1. Looking Back on the Old Letter Class

With the June 2002 release the old letter class scrlettr became obsolete. It is recommended not to use that class for new applications. There is no more active development of the old letter class, and support is very restricted. However, if you really need the documentation of the old letter class, you can still find it in the file `scrlettr.dtx`, but only in German. You should run it through LaTeX several times, like this:

```
latex scrlettr.dtx
latex scrlettr.dtx
latex scrlettr.dtx
```

Then you obtain the file `scrlettr.dvi` containing the old German manual.

To facilitate the transition to the new class, there is a compatibility option. In general, the complete older functionality still remains in the new class. Without that compatibility option, the user interface and the defaults will be different. More details on this option are provided in section 6.2.9, table 6.10 and section 6.9.

6.2. Options

The letter class scrlttr2 uses the package keyval to handle options. This is part of the graphics package (see [Car99b]). Since graphics is part of the *required* section of LaTeX, it should be found in every LaTeX distribution. Should your TeX distribution contain LaTeX, but not the packages graphics and keyval, please complain to your TeX distributor. If you want to use scrlttr2, you will have to install the graphics package yourself in that case.

The special feature of the keyval package is that options can have values. Thus, you not only need a lot less options, but perhaps also fewer optional arguments. You will see that when discussing the `letter` environment in section 6.4.4, page 173. The class will automatically load the keyval package. If you need to supply options to the keyval package, you should use the `\PassOptionsToPackage` command before `\documentclass`.

6.2.1. Defining Options Later

This section anticipates a feature of the new letter class. The meaning of this feature will not become clear until the structure of a document with more than one letter inside and another feature of scrlttr2 is understood. However, to keep the number of forward references low, it is reasonable to describe the feature this early.

`\KOMAoptions{`*`option list`*`}`

A special feature of the scrlttr2 class is the possibility to change many options even after loading the class. The `\KOMAoptions` command serves this purpose, taking options and their values as arguments. You can list multiple options, separated by commas, just like in the optional argument of `\documentclass`. If an option is only available when loading the class, i. e., as an optional argument to `\documentclass`, this will be explicitly mentioned in the option's description.

If you set an option to an illegal value within the *`option list`*, LaTeX will stop and show an error message. By entering "`h`" you will get an explanation that will also list possible values for that particular option.

6.2.2. Options for Compatibility

People who archive their letters in source code format generally place the highest priority on obtaining exactly the same results in future LaTeX runs. In some cases however, improvements and corrections to the class can lead to changes in behaviour, particularly as regards page breaking.

`version=`*`value`*

v2.9t With scrlttr2 there is the choice, whether a source file should give, as far as is technically possible, the same in any future LaTeX runs, or whether the document should be set according to the latest version of the class. To which version compatibility should be retained is determined by the option `version`. The default is version 2.9t. The same effect can be obtained with

```
version=first
```

or

```
version=2.9
```

or

```
version=2.9t
```

If an unknown version is entered for *`value`*, a warning is output and for safety's sake `version=first` is assumed. With

```
version=last
```

the current newest version can be selected. In this case future compatibility is waived. If the option is used without any value, then `last` is assumed as well.

The question of compatibility concerns first and foremost page breaking. New features which have no effect on page breaking will be available even when compatibility to an earlier version is chosen by this option. The option also has no effect on changes in page breaking which result from the removal of old errors in the new version class. If page breaking compatibility is required absolutely, to the point of incorporating previous class errors, then the document should rather be archived together with the relevant version of KOMA-Script.

It should be noted that the option `version` can no longer be changed after loading of the class.

6.2.3. Page Layout Options

In contrast to the old scrlettr class, but in agreement with the other KOMA-Script classes, the scrlttr2 class refers to the typearea package for the construction of the page layout (see chapter 2). The package will be loaded by the class automatically, and the class then controls the package. The necessary options will be explained in this section.

`enlargefirstpage`

As described later in this chapter, the first page of a letter always uses a different page layout. The scrlttr2 class provides a mechanism to calculate height and vertical alignment of header and footer of the first page independently of the following pages. If, as a result, the footer of the first page would reach into the text area, this text area is automatically made smaller using the `\enlargethispage` macro. On the other hand, if the text area should become larger, supposing that the footer on the first page allows that, you can also use this option. At best, a little more text will then fit on the first page. See also the description of the pseudo-length `firstfootvpos` in section 6.4.3, page 169. This option can take the standard values for simple switches, as listed in table 2.5, page 29. Default is `false`.

6.2.4. Other Layout Options

In this subsection, you will find all options, except the specific page layout options, that have an influence on the layout in general. Strictly speaking, all page layout options (see 6.2.3) are also layout options, and vice versa for some of them.

`cleardoublepage=`*`style`*

If you want pages inserted by the `\cleardoublepage` command to just contain a page number in the header and footer, or to be empty, this can be accomplished with this option. There are three different styles supported that are listed at table 6.1. Default is `standard`.

Table 6.1.: Possible values of option `cleardoublepage` for selection of page style of empty left pages with scrlttr2

`empty`	switches to page style `empty` for inserted pages
`plain`	switches to page style `plain` for inserted pages
`standard`	keeps the current page style for inserted pages

`headsepline=`*switch*
`footsepline=`*switch*

These two options insert a separator line below the header or above the footer, respectively, on consecutive pages. In the terminology of this manual, all pages of a letter except the first one are consecutive pages. This option can take the standard values for simple switches, as listed in table 2.5, page 29. Default is `false`. If one of the options is used without a value, like in the declaration above, this evaluates as `true`, so the separator line will be activated. When used as a `\documentclass` option, the option `headinclude` or `footinclude`, respectively, will be passed on to the typearea package (see section 2.4, page 29).

`pagenumber=`*position*

This option defines if and where a page number will be placed on consecutive pages. All pages without a letterhead are consecutive pages. This option affects the page styles `headings` and `plain`. It also affects the default page styles of the scrpage2 package, if set before loading the package (see chapter 4). It can take values only influencing horizontal, only vertical, or both positions. Possible value are shown in table 6.2. Default is `botcenter`.

`parskip=`*value*

Especially in letters you often encounter paragraphs marked not with indentation of the first line, but with a vertical skip between them. This is a matter of tradition. Apparently, it was easier for a secretary to operate the carriage return lever twice than to set an indentation using a tab stop or the space bar. Correct justification is almost impossible using a typewriter, so letters are traditionally typeset unjustified.

However, typographers like Jan Tschichold take the view that letters, written using means available to modern typesetting, should take advantage of their possibilities just like other documents do. Under these circumstances, letters should also be typeset using paragraph indentation and justification.

As a reaction to many serious requests, scrlttr2 offers the possibility to mark paragraphs not only by indentation of the first line, but alternatively by a vertical skip. You can choose

Table 6.2.: Possible values of option `pagenumber` for the position of the page number in page styles `headings` and `plain` with scrlttr2

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

between a full or half a line of vertical space. When using paragraph spacing, it is often useful to keep the last line of a paragraph shorter so that paragraph recognition will be eased. All these features are controlled by different values for the `parskip` option, shown in table 6.3. Default is `false`.

6.2.5. Font Options

Fonts options are any options which influence the size of the base font or of fonts for particular parts of the letter. In theory, options affecting the font type would also count as font options. At present there is only one option for font size in scrlttr2.

`fontsize=`*`size`*

Whereas in the main classes you choose the font size for the document using the `10pt`, `12pt`, etc., options, in the scrlttr2 class the desired *`size`* is set using the `fontsize` option. The functionality is however the same. This option can only be used with `\documentclass`, not with `\KOMAoptions`. Default is `12pt`.

6.2.6. Options for Letterhead and Address

The scrlttr2 class offers numerous extensions for the design of the letterhead. There are also options for address formatting, extending the possibilities of the standard letter class, although these features could already be found in the now obsolete scrlettr class.

`firsthead=`*`switch`*

v2.97e This option determines whether the letterhead will be typeset at all. The option undestands the standard values for simple keys, given in table 2.5, page 29. Default is for the letterhead to be set.

`fromalign=`*`value`*

v2.97e This option defines the placement of the return address in the letterhead of the first page. Apart from the various options for positioning the return address in the letterhead, there is also the option of adding the return address to the sender's extension. At the same time, this option serves as a switch to activate or deactivate the extended letterhead options. If these extensions are deactivated, some other options will have no effect. This will be noted in the explanations of the respective options. Possible values for `fromalign` are shown in table 6.4. Default is `left`.

`fromrule=`*`value`*

This option is part of the letterhead extensions (see option `fromalign` above). It allows you to place a horizontal line within the return address. The possible values are shown in table 6.5.

Table 6.3.: Possible values of option `parskip` to select the paragraph mark with scrlttr2

`false`, `off`
: paragraph indentation instead of vertical space; the last line of a paragraph may be arbitrarily filled

`full`, `on`, `true`
: one line vertical space between paragraphs; there must be at least 1 em free space in the last line of a paragraph

`full*`
: one line vertical space between paragraphs; there must be at least a quarter of a line free space at the end of a paragraph

`full+`
: one line vertical space between paragraphs; there must be at least a third of a line free space at the end of a paragraph

`full-`
: one line vertical space between paragraphs; the last line of a paragraph may be arbitrarily filled

`half`
: half a line vertical space between paragraphs; there must be at least 1 em free space in the last line of a paragraph

`half*`
: half a line vertical space between paragraphs; there must be at least a quarter of a line free space at the end of a paragraph

`half+`
: half a line vertical space between paragraphs; there must be at least a third of a line free space at the end of a paragraph

`half-`
: one line vertical space between paragraphs

Table 6.4.: Possible values of option `fromalign` for setting the position of the from address in the letterhead with scrlttr2

`center`, `centered`, `middle` return address centered; an optional logo will be above the extended return address; letterhead extensions will be activated
`false`, `no`, `off` standard design will be used for the return address; the letterhead extensions are deactivated
`left` left-justified return address; an optional logo will be right justified; letterhead extensions will be activated
`locationleft`, `leftlocation` return address is set left-justified in the sender's extension; a logo, if applicable, will be placed above it; the letterhead is automatically deactivated but can be reactivated using option `firsthead`.
`locationright`, `rightlocation`, `location` return address is set right-justified in the sender's extension; a logo, if applicable, will be placed above it; the letterhead is automatically deactivated but can be reactivated using option `firsthead`.
`right` right-justified return address; an optional logo will be left justified; letterhead extensions will be activated

Table 6.5.: Possible values of option `fromrule` for the position of the rule in the from address with scrlttr2

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

Default is `false`. You can not activate more than one line at a time. Regarding the length of the line, see section 6.4.8, page 168.

`fromphone=`*switch*

This option is part of the letterhead extensions (see option `fromalign` above). It defines whether the phone number will be part of the return address. This option can take the standard values for simple switches, as listed in table 2.5, page 29. Default is `false`.

`fromfax=`*switch*

This option is part of the letterhead extensions (see option `fromalign` above). It defines whether the facsimile number will be part of the return address. This option can take the standard values for simple switches, as listed in table 2.5, page 29. Default is `false`.

`fromemail=`*switch*

This option is part of the letterhead extensions (see option `fromalign` above). It defines whether the email address will be part of the return address. This option can take the standard values for simple switches, as listed in table 2.5, page 29. Default is `false`.

`fromurl=`*switch*

This option is part of the letterhead extensions (see option `fromalign` above). It defines whether the URL will be part of the return address. This option can take the standard values for simple switches, as listed in table 2.5, page 29. Default is `false`.

`fromlogo=`*switch*

This option is part of the letterhead extensions (see option `fromalign` above). It defines whether the logo will be part of the return address. This option can take the standard values for simple switches, as listed in table 2.5, page 29. Regarding the placement of the logo, see also the explanation of the option `fromalign` above. Default is `false`.

`addrfield=`*switch*

This option defines whether an address field will be set. Default is to use the address field. This option can take the standard values for simple switches, as listed in table 2.5, page 29. Default is `true`.

`backaddress=`*switch*

This option defines whether a return address for window envelopes will be set. Default is to use the return address. If the address field is suppressed (see option `addrfield`), there will be no return address either. This option can take the standard values for simple switches, as listed in table 2.5, page 29. Default is `true`.

Table 6.6.: Possible values of option `subject` for the position of the subject with scrlttr2

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

`subject=`*value*

v2.97c

This option serves two purposes: first, you can choose if your subject should have a title, given by the *`subject`* variable (see table 6.17, page 178); second, you can choose if the subject should be set before or after the opening. Furthermore, the formatting of the subject can be modified. Possible values for this option are shown in table 6.6. It is expressly noted that when using the setting `underlined`, the subject must fit on one line! Defaults are `beforeopening` and `untitled`.

`locfield=`*value*

scrlttr2 places a field with additional sender attributes next to the address field. This can be used, for example, for bank account or similar additional information. Depending on the `fromalign` option, it will also be used for the sender logo. The width of this field may be defined within an `lco` file (see section 6.2.9). If the width is set to 0 in that file, then the `locfield` option can toggle between two presets for the field width. See the explanation on the `locwidth` pseudo length in section 6.4.5, page 174. Possible values for this option are shown in table 6.7. Default is `narrow`.

Table 6.7.: Possible values of option `locfield` for setting the width of the field with additional sender attributes with scrlttr2

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

`foldmarks=`*value*

This option activates or deactivates foldmarks for vertical two-, three- or four-panel folding, and a single horizontal folding, of the letter, whereby the folding need not result in equal-sized parts. The position of the four horizontal and the single vertical marks are configurable via pseudo-lengths (see section 6.4.1 from page 165 onwards).

The user has a choice: Either one may use the standard values for simple switches, as described in table 2.5, page 29, to activate or deactivate at once all configured foldmarks on the left and upper edges of the paper; or one may specify by one or more letters, as listed in table 6.8, the use of the individual foldmarks independently. Also in the latter case the foldmarks will only be shown if they have not been switched off generally with one of `false`, `off` or `no`. The exact positioning of the foldmarks is specified in the user settings, that is, the `lco` files (see section 6.2.9) chosen for a letter. Default values are `true` and `TBMPL`.

v2.97e

The exact placement of the fold marks for three-panel letter folding depends on user settings, that is, the `lco` files (see section 6.2.9). The folding need not result in equal-sized parts. This option can take the standard values for simple switches, as listed in table 2.5, page 29. Default is `true`, which implies setting the fold marks.

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

```
\KOMAoption{foldmarks=blmt}
```

as long as the defaults have not been changed previously. If some changes might have been made before, then for added safety you may use:

```
\KOMAoption{foldmarks=true,foldmarks=blmtP}
```

`numericaldate=`*switch*

This option toggles between the standard, language-dependent date presentation, and a short, numerical one. KOMA-Script does not provide the standard presentation. It should be defined by packages such as german, babel, or isodate. The short, numerical presentation, on the other

Table 6.8.: Combined values for the configuration of foldmarks with the option `foldmarks`

`B`	activate upper horizontal foldmark on left paper edge
`b`	deactivate upper horizontal foldmark on left paper edge
`H`	activate all horizontal foldmarks on left paper edge
`h`	deactivate all horizontal foldmarks on left paper edge
`L`	activate left vertical foldmark on upper paper edge
`l`	deactivate left vertical foldmark on upper paper edge
`M`	activate middle horizontal foldmark on left paper edge
`m`	deactivate middle horizontal foldmark on left paper edge
`P`	activate punch or center mark on left paper edge
`p`	deactivate punch or center mark on left paper edge
`T`	activate lower horizontal foldmark on left paper edge
`t`	deactivate lower horizontal foldmark on left paper edge
`V`	activate all vertical foldmarks on upper paper edge
`v`	deactivate all vertical foldmarks on upper paper edge

Table 6.9.: Possible value of option `refline` for setting the width of the reference fields line with scrlttr2

`narrow`	reference fields line restricted to type area
`wide`	reference fields line corresponds to address and sender attributes

hand, is produced by scrlttr2 itself. This option can take the standard values for simple switches, as listed in table 2.5, page 29. Default is `false`, which results in standard date presentation. In the now obsolete scrlettr class, the opposite effect was achieved using the `orgdate` option.

`refline`*value*

With the scrlttr2 class, the header, footer, address, and sender attributes may extend beyond the normal type area to the left and to the right. This option defines whether that should also apply to the reference fields line. Normally, the reference fields line contains at least the date, but it can hold additional data. Possible values for this option are shown in table 6.9. Default is `narrow`.

6.2.7. Options for the Letterfoot

The letterfoot is the footer of the first page of the letter. There exist some special rules for its placement, which are given in the description of the option `enlargefirstpage` (see section 6.2.3, page 135) and the pseudo-length `firstfootvpos` (siehe section 6.4.3, page 169).

`firstfoot`*switch*

v2.97e

This option determines whether the letterfoot is set or not. If the letterfoot is not set then the pseudo-length `firstfootvpos` is also ignored, and instead scrlttr2 assumes that the value is equal to `\paperheight`. This has an effect when the option `enlargefirstpage` (see section 6.4.3, page 169) is used concurrently.

The option understands the standard values for simple switches, as given in table 2.5, page 29. Default is the setting of the letterfoot.

6.2.8. Formatting Options

Formatting options are those which influence form or formatting of the output and do not belong to another section. You might also call them the *miscellaneous options*.

`draft=`*switch*

This option toggles between the final and the draft version of a document. In particular, enabling the `draft` option activates little black boxes that will be drawn at the end of overfull lines. These boxes allow the unpracticed eye to more easily identify paragraphs that need manual intervention. When the `draft` option is disabled, there will be no such boxes. This option can take the standard values for simple switches, as listed in table 2.5, page 29. Default is `false`, as usual. However, I strongly recommend enabling the `draft` option when designing a letter, as for every other document.

6.2.9. The Letter Class Option Files

Normally, you would not redefine parameters like the distance between the address field and the top edge of the paper every time you write a letter. Instead, you would reuse a whole set of parameters for certain occasions. It will be much the same for the letterhead and footer used on the first page. Therefore, it is reasonable to save these settings in a separate file. For this purpose, the scrlttr2 class offers the `lco` files. The `lco` suffix is an abbreviation for l*etter c*lass *o*ption.

In an `lco` file you can use all commands available to the document at the time the `lco` file is loaded. Additionally, it can contain internal commands available to package writers. For scrlttr2, these are in particular the commands `\@newplength`, `\@setplength`, and `\@addtoplength` (see section 6.3.4).

v3.04 There are already some `lco` files included in the KOMA-Script distribution. The `DIN.lco`, `DINmtext.lco`, `SNleft.lco`, `SN.lco`, and `NF.lco` files serve to adjust KOMA-Script to different layout standards. They are well suited as templates for your own parameter sets. The `KOMAold.lco` file, on the other hand, serves to improve compatibility with the old letter class scrlettr. Since it contains internal commands not open to package writers, you should not use this as a template for your own `lco` files. You can find a list of predefined `lco` files in table 6.10, page 148.

If you have defined a parameter set for a letter standard not yet supported by KOMA-Script, you are explicitly invited to send this parameter set to the KOMA-Script support address. Please do not forget to include the permission for distribution under the KOMA-Script license (see the `lppl.txt` file). If you know the necessary metrics for an unsupported letter standard, but are not able to write a corresponding `lco` file yourself, you can also contact the KOMA-Script author, Markus Kohm, directly.

`\LoadLetterOption{`*name*`}`

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

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

Example: You write a document containing several letters. Most of them should comply with the German DIN standard. So you start with:

```
\documentclass{scrlttr2}
```

However, one letter should use the `DINmtext` variant, with the address field placed more toward the top, which results in more text fitting on the first page. The folding will be modified so that the address field still matches the address window in a DIN C6/5 envelope. You can achieve this as follows:

```
\begin{letter}{Markus Kohm\\
    Freiherr-von-Drais-Stra\ss e 66\\68535 Edingen-Neckarhausen↩
  }
  \LoadLetterOption{DINmtext}
  \opening{Hello,}
```

Since construction of the page does not start before the `\opening` command, it is sufficient to load the `lco` file before this. In particular, the laoding need not be done before `\begin{letter}`. Therefore the changes made by loading the `lco` file are local to the corresponding letter.

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

Example: You do not want to enter your sender address every time, so you create an `lco` file with the necessary data, like this:

```
\ProvidesFile{mkohm.lco}[2002/02/25 letter class option]
\setkomavar{fromname}{Markus Kohm}
\setkomavar{fromaddress}{Freiherr-von-Drais-Stra\ss e 66\\
                         68535 Edingen-Neckarhausen}
```

The command `\setkomavar` used above, and the principle of variables will be explained in detail insection 6.3.3, page 157. In the example given here, knowledge of the exact function of the command is not critical. It is only important to note what can be done with `lco` files, less so exactly how this might be accomplished. Please note that the German sharp s, “ß”, was entered using the TEX macro `\ss`, because directly after `\documentclass` no packages for input encoding, for example `\usepackage[latin1]{inputenc}` for Unix or `\usepackage[ansinew]{inputenc}` for Windows, and no language packages, like `\usepackage{ngerman}` for the new German orthography, are loaded.

However, if you would always use the same input encoding, you could also include it into your `lco` file. This would look as follows:

```
\ProvidesFile{mkohm.lco}[2002/02/25 letter class option]
```

```
\RequirePackage[latin1]{inputenc}
\setkomavar{fromname}{Markus Kohm}
\setkomavar{fromaddress}{Freiherr-von-Drais-Stra\ss e 66\\
                         68535 Edingen-Neckarhausen}
```

There is one distinct disadvantage with this usage: you can no longer load this `lco` file later in your document. If you want to have letters with different senders in one document, you should therefore refrain from loading packages in your `lco` file.

Let us further assume that I always typeset letters using the preset parameters `KOMAold`. Then I could add the following line to my `mkohm.lco` file:

```
\LoadLetterOption{KOMAold}
```

Anyway, now you can preset my sender address using

```
\documentclass[mkohm]{scrlttr2}
```

In table 6.10, page 148 you can find a list of all predefined `lco` files. If you use a printer that has large unprintable areas on the left or right side, you might have problems with the `SN` option. Since the Swiss standard SN 101 130 defines the address field to be placed 8 mm from the right paper edge, the headline and the sender attributes too will be set with the same small distance from the paper edge. This also applies to the reference fields line when using the `refline=wide` option (see section 6.2.6, page 145). If you have this kind of problem, create your own `lco` file that loads `SN` first and then changes `toaddrhpos` (see section 6.4.4, page 171) to a smaller value. Additionally, also reduce `toaddrwidth` accordingly.

Table 6.10.: The predefined `lco` files

`DIN`	parameter set for letters on A4-size paper, complying with German standard DIN 676; suitable for window envelopes in the sizes C4, C5, C6, and C6/5 (C6 long).
`DINmtext`	parameter set for letters on A4-size paper, complying with DIN 676, but using an alternate layout with more text on the first page; only suitable for window envelopes in the sizes C6 and C6/5 (C6 long).

...

Table 6.10.: The predefined `lco` files (*continuation*)

`KOMAold`
: parameter set for letters on A4-size paper using a layout close to the now obsolete scrlettr letter class; suitable for window envelopes in the sizes C4, C5, C6, and C6/5 (C6 long); some additional commands to improve compatibility with obsolete scrlettr commands are defined; scrlttr2 may behave slightly different when used with this `lco` file than with the other `lco` files.

`NF`
: parameter set for French letters, according to NF Z 11-001; suitable for window envelopes of type DL (110 mm to 220 mm) with a window of about 20 mm from right and bottom with width 45 mm and height 100 mm; this file was originally developed by Jean-Marie Pacquet, who provides an extended version and additional information on [Pac].

`NipponEL`
: parameter set for Japanese letters in A4 format; suitable for Japanese window envelopes of types Chou or You 3 or 4, in which the windows is approximately 90 mm wide, 45 mm high, and positioned 22 mm from the left and 12 mm from the top edge (see appendix A).

`NipponEH`
: parameter set for Japanese letters in A4 format; suitable for Japanese window envelopes of types Chou or You 3 or 4, in which the windows is approximately 90 mm wide, 55 mm high, and positioned 22 mm from the left and 12 mm from the top edge (see appendix A).

`NipponLL`
: parameter set for Japanese letters in A4 format; suitable for Japanese window envelopes of types Chou or You 3 or 4, in which the windows is approximately 90 mm wide, 45 mm high, and positioned 25 mm from the left and 12 mm from the top edge (see appendix A).

`NipponLH`
: parameter set for Japanese letters in A4 format; suitable for Japanese window envelopes of types Chou or You 3 or 4, in which the windows is approximately 90 mm wide, 55 mm high, and positioned 25 mm from the left and 12 mm from the top edge (see appendix A).

...

Table 6.10.: The predefined `lco` files (*continuation*)

`NipponRL`	parameter set for Japanese letters in A4 format; suitable for Japanese window envelopes of types Chou or You 3 or 4, in which the windows is approximately 90 mm wide, 45 mm high, and positioned 25 mm from the left and 24 mm from the top edge (see appendix A).
`KakuLL`	parameter set for Japanese letters in A4 format; suitable for Japanese window envelopes of type Kaku A4, in which the windows is approximately 90 mm wide, 45 mm high, and positioned 25 mm from the left and 24 mm from the top edge (see appendix A).
`SN`	parameter set for Swiss letters with address field on the right side, according to SN 010 130; suitable for Swiss window envelopes in the sizes C4, C5, C6, and C6/5 (C6 long).
`SNleft`	parameter set for Swiss letters with address field on the left side; suitable for Swiss window envelopes with window on the left side in the sizes C4, C5, C6, and C6/5 (C6 long).

`\LetterOptionNeedsPapersize{`*`option name`*`}{`*`paper size`*`}`

As mentioned in section 6.2.3, at present there exist only parameter sets and `lco` files for A4-sized paper. In order that you will at least be warned when using another *`paper size`*, you will find a `\LetterOptionNeedsPapersize` command in every `lco` file distributed with KOMA-Script. The first argument is the name of the `lco` file without the "`.lco`" suffix. The second argument is the paper size for which the `lco` file is designed.

If several `lco` files are loaded in succession, a `\LetterOptionNeedsPapersize` command can be contained in each of them, but the `\opening` command will only check the last given *`paper size`*. As shown in the following example, an experienced user can thus easily write `lco` files with parameter sets for other paper sizes. If you do not plan to set up such `lco` files yourself, you may just forget about this option and skip the example.

Example: Suppose you use A5-sized paper in normal, i. e., upright or portrait, orientation for your letters. We further assume that you want to put them into standard C6 window envelopes. In that case, the position of the address field would be the same as for a DIN standard letter on A4-sized paper. The main difference is that A5 paper needs only one fold. So you want to disable the upper and lower fold marks. The easiest way to achieve this is to place the marks outside of the paper area.

```
\ProvidesFile{paper=a5.lco}[2002/05/02 letter class option]
\LetterOptionNeedsPapersize{paper=a5}{a5}
\@setplength{tfoldmarkvpos}{\paperheight}
\@setplength{bfoldmarkvpos}{\paperheight}
```

Besides this, the placement of the foot, that is, the pseudo-length `firstfootvpos`, must be adjusted. It is left to the reader to find an appropriate value. When using such an `lco` file, you must only take care that other `lco` file options, like `SN`, are declared before the paper size, i. e., before loading "`paper=a5.lco`". Does this seem too complicated? Only before you have used it the first time. Anyway, how often do you write letters not using your standard formats for A4-size or letter-size paper?

By the way, the `DIN` `lco` file will always be loaded as the first `lco` file. This ensures that all pseudo-lengths will have more or less reasonable default values.

Please note that it is not possible to use `\PassOptionsToPackage` to pass options to packages from within an `lco` file that have already been loaded by the class. Normally, this only applies to the typearea, scrlfile, and keyval packages.

6.3. General Document Properties

Some document properties are not assigned to any particular part of the document such as to the letterhead or the letter body. Several of these properties have already been mentioned or explained in section 6.2.

6.3.1. Font Selection

Commands for defining, extending and querying the font of a specific element can be found in section 3.2.1. These commands work exactly the same in scrlttr2. The elements which can be influenced in this way are listed in table 6.11.

Table 6.11.: Alphabetical list of elements whose font can be changed in scrlttr2 using the commands `\setkomafont` and `\addtokomafont`

`addressee`	name und address in address field
`backaddress`	return address for a window envelope
`descriptionlabel`	label, i. e., the optional argument of `\item`, in a `description` environment

...

Table 6.11.: Elements whose font can be changed (*continuation*)

`foldmark` foldmark on the letter page; intended for color settings
`fromaddress` sender's address in the letterhead
`fromname` sender's address in the letterhead if different from `fromaddress`
`fromrule` line in return address field in letterhead; intended for color settings
`pagefoot` in most cases the footer, sometimes the header of a page
`pagehead` in most cases the header, sometimes the footer of page
`pagenumber` page number in the footer or header, inserted with `\pagemark`
`specialmail` mode of dispatch in address field
`subject` subject in the opening of the letter
`title` title in the opening of the letter
`toaddress` variation of the element `addressee` for setting the adressee address (less the name) in the adress field
`toname` variation of the element `addressee` for the name (only) of the addressee in the address field

6.3.2. Page Style

One of the general properties of a document is the page style. Please refer also to section 3.2.2 and chapter 4.

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

In letters written with scrlttr2 there are four different page styles.

empty is the page style, in which the header and footer of subsequent pages (all pages apart from the first) are completely empty. This page style is also used for the first page, because header and footer of this page are set by otehr means using the macro `\opening` (see section 6.4.2, section 6.4.3, as well as section 6.5.1, page 179).

plain is the page style with only page numbers in the header or footer on subsequent pages. The placement of these page numbers is determined by the option `pagenumber` (see section 6.2.4, page 136).

headings is the page style for running (automatic) headings on subsequent pages. The inserted marks are the sender's name from the variable `fromname` and the subject from the variable `subject` (see section 6.4.2, page 167 and section 6.4.7, page 177). At which position these marks and the page numbers are placed, depends on the option `pagenumber` (see section 6.2.4, page 136). The author can also change these marks manually after the `\opening` command. To this end, the commands `\markboth` and `\markright` are available as usual, and with the use of package scrpage2 also `\markleft` (see section 4.1.2, page 114) is available.

myheadings is the page style for manual page headings on subsequent pages. This is very similar to `headings`, but here the marks must be set by the author using the commands `\markboth` and `\markright`. With the use of package scrpage2 also `\markleft` can be utilized.

In the terminology of this manual, subsequent pages are all pages of a letter except for the first one.

Page styles are also influenced by the option `headsepline` and `footsepline` (see section 6.2.4, page 136). The page style beginning with the current page is switched using `\pagestyle`. In contrast, `\thispagestyle` changes only the page style of the current page. The letter class itself uses `\thispagestyle{empty}` within `\opening` for the first page of the letter.

For changing the font style of headers or footers you should use the user interface described in section 3.2.1. For header and footer the same element is used, which you can name either `pagehead` or `pagefoot`. The element for the page number within the header or footer is named `pagenumber`. Default settings are listed in table 3.5, page 58. Please have also a look at the example in section 3.2.2, page 59.

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

Please refer to section 3.2.2, page 61. The function of `\cleardoublepage` in scrlttr2 depends on the option `cleardoublepage` which is described in more detail in section 6.2.4, page 135.

6.3.3. Variables

Apart from options, commands, environments, counters and lengths, additional elements have already been introduced in KOMA-Script. A typical property of an element is the font style and the option to change it (see section 3.2.1). At this point we now introduce variables. Variables have a name by which they are called, and they have a content. The content of a variable can be set independently from time and location of the actual usage in the same way as the contents of a command can be separated from its usage. The main difference between a command and a variable is that a command usually triggers an action, whereas a variable only consists of plain text which is then output by a command. Furthermore, a variable can additionally have a description which can be set and output.

This section specifically only gives an introduction to the concept of variables. The following examples have no special meaning. More detailed examples can be found in the explanation of predefined variables of the letter class in the following sections. An overview of all variables is given in table 6.12.

Table 6.12.: Alphabetical list of all supported variables in scrlttr2

`backaddress`	return address for window envelopes (section 6.4.4, page 172)
`backaddressseparator`	separator within the return address (section 6.4.4, page 172)
`ccseparator`	separator between title of additional addressees, and additional addressees (section 6.6.2, page 182)
`customer`	customer number (section 6.4.6, page 176)
`date`	date (section 6.4.6, page 176)

...

Table 6.12.: Alphabetical list of all supported variables in scrlttr2 (*continued*)

`emailseparator`
: separator between e-mail name and e-mail address (section 6.4.2, page 168)

`enclseparator`
: separator between title of enclosure, and enclosures (section 6.6.2, page 182)

`faxseparator`
: separator between title of fax, and fax number (section 6.4.2, page 168)

`fromaddress`
: sender's address without sender name (section 6.4.2, page 167)

`frombank`
: sender's bank account (section 6.4.8, page 178)

`fromemail`
: sender's e-mail (section 6.4.2, page 167)

`fromfax`
: sender's fax number (section 6.4.2, page 167)

`fromlogo`
: commands for inserting the sender's logo (section 6.4.2, page 167)

`fromname`
: complete name of sender (section 6.4.2, page 167)

`fromphone`
: sender's telephone number (section 6.4.2, page 167)

`fromurl`
: a url of the sender (section 6.4.2, page 167)

`invoice`
: invoice number (section 6.4.6, page 176)

`location`
: more details of the sender (section 6.4.5, page 175)

`myref`
: sender's reference (section 6.4.6, page 176)

`place`
: place (section 6.4.6, page 176)

Table 6.12.: Alphabetical list of all supported variables in scrlttr2 (*continued*)

`placeseparator`
: separator between place and date (section 6.4.6, page 176)

`phoneseparator`
: separator between title of telephone, and telephone number (section 6.4.2, page 168)

`signature`
: signature beneath the ending of the letter (section 6.6.1, page 180)

`specialmail`
: mode of dispatch (section 6.4.4, page 173)

`subject`
: subject (section 6.4.7, page 177)

`subjectseparator`
: separator between title of subject, and subject (section 6.4.7, page 177)

`title`
: letter title (section 6.4.7, page 177)

`toname`
: complete name of addressee (section 6.4.4, page 173)

`toaddress`
: address of addressee without addressee name (section 6.4.4, page 173)

`yourmail`
: date of addressee's mail (section 6.4.6, page 176)

`yourref`
: addressee's reference (section 6.4.6, page 176)

```
\newkomavar[description]{name}
\newkomavar*[description]{name}
\removereffields
\defaultreffields
\addtoreffields{name}
```

With `\newkomavar` a new variable is defined. This variable is addressed via *`name`*. As an option you can define a *`description`* for the variable *`name`*. Using the command `\addtoreffields` you can add the variable *`name`* to the reference fields line (see section 6.4.6). The *`description`* and the content of the variable are added at the end of the reference fields line. The starred version `\newkomavar*` is similar to the unstarred version, with a subsequent call of the com-

mand \addtoreffields. Thus, the starred version automatically adds the variable to the reference fields line.

Example: Suppose you need an additional field for direct dialling. You can define this field either with

```
\newkomavar[Direct dialling]{myphone}
\addtoreffields{myphone}
```

or more concisely with

```
\newkomavar*[direct dialling]{myphone}
```

When you define a variable for the reference fields line you should always give it a description.

With the command \removereffields all variables in the ference line can be removed. This also includes the predefined variables of the class. The reference fields line is then empty except for the date which is always appended to the end. This can be useful, for example, if you wish to change the order of the variables in the reference fields line.

The command \defaultreffields acts to reset the reference fields line to its predefined format. In doing so, all custom-defined variables are removed from the reference fields line.

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

With the command \setkomavar you determine the *content* of the variable *name*. Using an optional argument you can at the same time change the *description* of the variable. In contrast, \setkomavar* can only set the *description* of the variable *name*.

Example: Suppose you have defined a direct dialling as mentioned above and you now want to set the content. You write:

```
\setkomavar{myphone}{-\,11}
```

In addition, you want to replace the term “direct dialling” with “Connection”. Thus you add the description:

```
\setkomavar*{myphone}{Connection}
```

or you can combine both in one command:

```
\setkomavar{myphone}[Connection]{-\,11}
```

By the way: You may delete the content of a variable using an empty *content* argument. You can also delete the description using an empty *description* argument.

Example: Suppose you have defined a direct dialling as mentioned above and you now no longer want a description to be set. You write:

```
\setkomavar*{myphone}{}
```

You can combine this with the definition of the content:

```
\setkomavar{myphone}[]{-\,11}
```

So you may setup the content and delete the description using only one command.

```
\usekomavar[command]{name}
\usekomavar*[command]{name}
```

v2.9i In some cases it is necessary for the user to access the content or the description of a variable, and not to leave this only up to the class. This is specially important when you have defined a variable which is not added to the reference fields line. Using the command `\usekomavar` you have access to the content of the variable *name*, whereas the starred version `\usekomavar*` allows you to access the description.

The commands `\usekomavar` and `\usekomavar*` are, similarly to all commands where a starred version exists or which can take an optional argument, not fully expandable. Nevertheless, if used within `\markboth`, `\markright` or similar commands, you need not insert a `\protect` before using them. Of course this is also true for `\markleft` if using package scrpage2. However, these kinds of commands can not be used within commands like `\MakeUppercase` which directly influence their argument. `\MakeUppercase{\usekomavar{`*name*`}}` would result in `\usekomavar{`*NAME*`}`. To avoid this problem you may use commands like `\MakeUppercase` as an optional argument to `\usekomavar` or `\usekomavar*`. Then you will get the uppercase content of a variable using `\usekomavar[\MakeUppercase]{`*name*`}`.

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

v2.9i With these commands you may check whether or not the expanded content or description of a variable is empty. The *true* argument will be executed if the content or description is empty. Otherwise the *false* argument will be executed. The starred variant handles the description of a variable, the unstarred variant handles the contents.

It is important to know that the content or description of the variable will be expanded as far as this is possible with `\edef`. If this results in spaces or unexpandable macros like `\relax`, the result will be not empty even where the use of the variable would not result in any visible output.

Both variants of the command also must not be used as the argument of `\MakeUppercase` or other commands which have similar effects to their arguments (see the description of `\usekomavar` above for more information about using commands like `\usekomavar` or `\ifkomavarempty` at the argument of `\MakeUppercase`). However, they are robust enough to be used as the argument of, e. g., `\markboth` or `\footnote`.

6.3.4. The Pseudo-Lengths

TEX works with a fixed number of registers. There are registers for tokens, for boxes, for counters, for skips and for dimensions. Overall there are 256 registers for each of these categories. For LATEX

lengths, which are addressed with \newlength, skip registers are used. Once all these registers are in use, you can not define any more additional lengths. The letter class scrlttr2 would normally use up more than 20 of such registers for the first page alone. LaTeX itself already uses 40 of these registers. The typearea package needs some of them too; thus, approximately a quarter of the precious registers would already be in use. That is the reason why lengths specific to letters in scrlttr2 are defined with macros instead of lengths. The drawback of this approach is that computations with macros is somewhat more complicated than with real lengths.

It can be pointed out that the now recommended LaTeX installation with ε-TeX no longer suffers from the above-mentioned limitation. However, that improvement came too late for scrlttr2.

A list of all pseudo-lengths in scrlttr2 is shown in table 6.13 starting at page 159. The meaning of the various pseudo-lengths is shown graphically in figure 6.1. The dimensions used in the figure correspond to the default settings of scrlttr2. More detailed description of the individual pseudo-lengths is found in the individual sections of this chapter.

Table 6.13.: Pseudo-lengths provided by class scrlttr2

`backaddrheight`	height of the return address at the upper edge of the address field (section 6.4.4, page 172)
`bfoldmarkvpos`	vertical distance of lower foldmark from top paper edge (section 6.4.8, page 165)
`firstfootvpos`	vertical distance of letterfoot from top paper edge (section 6.4.3, page 169)
`firstfootwidth`	width of letterfoot; letterfoot is centered horizontally on letter paper (section 6.4.3, page 170)
`firstheadvpos`	vertical distance of letterhead from top paper edge (section 6.4.2, page 167)
`firstheadwidth`	width of letter head; letterhead is centered horizontally on letter paper (section 6.4.2, page 167)
`foldmarkhpos`	horizontal distance of all foldmarks from left paper edge (section 6.4.8, page 166)
`fromrulethickness`	Thickness of an optional horizontal line in the letterhead (section 6.4.2, page 168)

...

Table 6.13.: Pseudo-lengths provided by class scrlttr2 (*continued*)

`fromrulewidth`
: length of an optional horizontal rule in letterhead (section 6.4.2, page 168)

`locwidth`
: width of supplemental data field; for zero value width is calculated automatically with respect to option `locfield` that is described in section 6.2.6 (section 6.4.5, page 174)

`refaftervskip`
: vertical skip below reference fields line (section 6.4.6, page 176)

`refhpos`
: horizontal distance of reference fields line from left paper edge; for zero value reference fields line is centered horizontally on letter paper (section 6.4.6, **??**)

`refvpos`
: vertical distance of reference fields line from top paper edge (section 6.4.6, page 175)

`refwidth`
: width of reference fields line (section 6.4.6, page 176)

`sigbeforevskip`
: vertical skip between closing and signature (section 6.6.1, page 181)

`sigindent`
: indentation of signature with respect to text body (section 6.6.1, page 181)

`specialmailindent`
: left indentation of mode of dispatch within address field (section 6.4.4, page 173)

`specialmailrightindent`
: right indentation of mode of dispatch within address field (section 6.4.4, page 173)

`tfoldmarkvpos`
: vertical distance of upper foldmark from top paper edge (section 6.4.8, page 165)

`toaddrheight`
: height of address field (section 6.4.4, page 172)

`toaddrhpos`
: horizontal distance of address field from left paper edge, for positive values; or negative horizontal distance of address field from right paper edge, for negative values (section 6.4.4, page 171)

...

Table 6.13.: Pseudo-lengths provided by class scrlttr2 (*continued*)

`toaddrindent`	left and right indentation of address within address field (section 6.4.4, page 172)
`toaddrvpos`	vertical distance of address field from top paper edge (section 6.4.4, page 171)
`toaddrwidth`	width of address field (section 6.4.4, page 172)

`\@newplength{`*`name`*`}`

This command defines an new pseudo-length. This new pseudo-length is uniquely identified by its *`name`*. If with this command a redefinition of an already existing pseudo-length is attempted, the commands exits with an error message.

Since the user in general does not define own pseudo-lengths, this command is not intended as a user command. Thus, it can not be used within a document, but it can, for example, be used within an `lco` file.

`\@setplength[`*`factor`*`]{`*`pseudo-length`*`}{`*`value`*`}`
`\@addtoplength[`*`factor`*`]{`*`pseudo-length`*`}{`*`value`*`}`

Using the command `\@setplength` you can assign the multiple of a *`value`* to a *`pseudo-length`*. The *`factor`* is given as an optional argument (see also `\setlengthtoplength`). The command `\@addtoplength` adds the *`value`* to a *`pseudo-length`*. To assign, or to add the multiple of, one *`pseudo-length`* to another pseudo-length, the command `\useplength` is used within *`value`*. To subtract the value of one pseudo-length from another *`pseudo-length`* a minus sign, or `-1`, is used as the *`factor`*.

Since the user in general does not define own pseudo-lengths, this command is not intended as a user command. Thus, it can not be used within a document, but can, for example, be used within an `lco` file.

`\useplength{`*`name`*`}`

Using this command you can access the value of the pseudo-length with the given *`name`*. This is one of the few user commands in connection with pseudo-lengths. Of course this command can also be used with an `lco` file.

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

While you can simply prepend a factor to a length, this is not possible with pseudo-lengths. Suppose you have a length `\test` with the value 2 pt; then `3\test` gives you the value 6 pt. Using pseudo-lengths instead, `3\useplength{test}` would give you 32 pt. This is especially annoying if you want a real *`length`* to take the value of a *`pseudo-length`*.

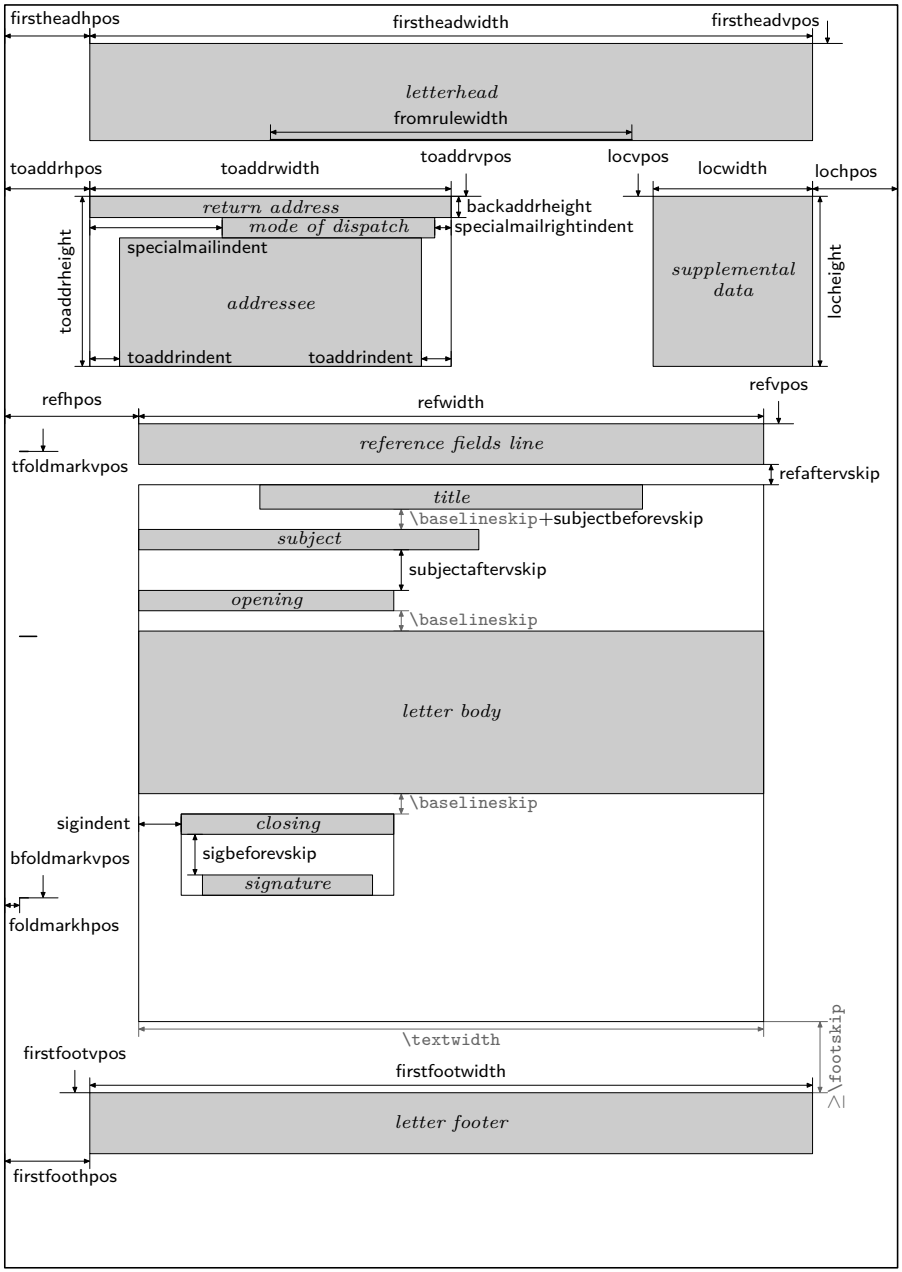


Figure 6.1.: Schematic of the pseudo-lengths for a letter

Using the command \setlengthtoplength you can assign the multiple of a *pseudo-length* to a real *length*. Here, instead of prepending the *factor* to the *pseudo-length*, it is given as an optional argument. You should also use this command when you want to assign the negative value of a *pseudo-length* to a *length*. In this case you can either use a minus sign or -1 as the *factor*. The command \addtolengthplength works very similarly; it adds the multiple of a *pseudo-length* to the *length*.

6.3.5. The General Structure of a Letter Document

The general structure of a letter document differs somewhat from the structure of a normal document. Whereas a book document in general contains only one book, a letter document can contain several letters. As illustrated in figure 6.2, a letter document consists of a preamble, the individual letters, and the closing.

The preamble comprises all settings that in general concern all letters. Most of them can also be overwritten in the settings of the individual letters. The only setting which can not be changed within a single letter is compatibility to prior versions of scrlttr2 (see option version in section 6.2.2, page 134).

It is recommended that only general settings such as the loading of packages and the setting of options be placed before \begin{document}. All settings that comprise the setting of variables or other text features should be done after \begin{document}. This is particularly recommended when the babel package (see [Bra01]) is used, or language-dependent variables of scrlttr2 are to be changed.

The closing usually consists only of \end{document}. Of course you can also insert additional comments at this point.

As shown in figure 6.3, every single letter itself consists of an introduction, the letter body, and the closing. In the introduction, all settings pertaining only to the current letter are defined. It is important that this introduction always ends with \opening. Similarly, the closing always starts with \closing. The two arguments *opening* and *closing* can be left empty, but both commands must be used and must have an argument.

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

As already mentioned, all general settings used in the preamble of a letter document, with the exception of font size, can also be in the preamble of the individual letters. Therefore, you will not find more detailed explanations for the possible settings in this section. Please refer to section 6.4.

```
\documentclass[...]{scrlttr2}
...
                    settings for all letters
...
\begin{document}
...
                    settings for all letters
...
```

```
\begin{letter}{addressee}
...
                content of the individual letter
...
\end{letter}
```

⋮

```
\end{document}
```

Figure 6.2.: General structure of a letter document with several individual letters (the structure of a single letter is shown in figure 6.3)

```
\begin{letter}[options]{addressee}
...
                    settings for this letter
...
\opening{opening}
```

```
...
                        letter text
...
```

```
\closing{closing}
\ps
...
                        postscript
...
\encl{enclosures}
\cc{additional addressees}
\end{letter}
```

Figure 6.3.: General structure of a single letter within a letter document (see also figure 6.2)

6.4. The Letter Declaration

The letter declaration gives all settings for the letter itself, as well as for the first page of the body. The first page consists of more than just the preliminaries of the letter; in fact, it consists of several different parts.

6.4.1. Foldmarks

Foldmarks are short horizontal lines at the left edge, and short vertical lines at the upper edge of the paper. KOMA-Script at present supports three configurable horizontal and one configurable vertical foldmarks. In addition, there is support for a punchmark or center mark which cannot be shifted in the vertical direction.

`tfoldmarkvpos`
`mfoldmarkvpos`
`bfoldmarkvpos`

The letter class scrlttr2 knows a total of three vertically-placed configurable foldmarks. The position of the upper foldmark, taken from the upper edge of the paper, is governed by the pseudo-length `tfoldmarkvpos`, that of the middle foldmark by pseudo-length `mfoldmarkvpos`, and that of the lower foldmark by pseudo-length `bfoldmarkvpos`. With the addition of the punch or centermark, there is still a fourth horizontal mark. This one is however always placed at the vertical center of the paper.

v2.97e

The upper and lower foldmarks do not serve to divide the paper into exactly equal thirds. Instead, with their help, the paper should be folded such that the address field appears correctly in the space available in the chosen window envelope format, which is determined by choice of `lco` file. Several such files are available offering predefined formats. An anomaly is present with `DINmtext`: for this format, an envelope format of C6/5 (also known as “C6 long”) is assumed. Letters written with this option are not suited to envelopes of formats C5 or C4.

The middle foldmark is not normally required for Western letters. In Japan, however, a larger number of envelope formats exists, requiring one more foldmark (see the Japanese `lco` files). At this point attention is drawn to the fact that reference to “upper”, “middle”, and “lower” foldmarks is simply a convenience. In fact, it is not defined that `tfoldmarkvpos` must be smaller than `mfoldmarkvpos`, which in turn must be smaller than `bfoldmarkvpos`. If on the other hand one of the pseudo-lengths is set to null, then the corresponding foldmark will not be set even if the option `foldmarks` (see section 6.2.6, page 143) is explicity activated.

`tfoldmarklength`
`mfoldmarklength`
`bfoldmarklength`
`pfoldmarklength`

v2.97e

These four pseudo-lengths determine the lengths of the four horizontal foldmarks. One exceptional behaviour exists. If the length is given as null, then the three vertically-configurable

pseudo-lengths `tfoldmarklength`, `mfoldmarklength` and `bfoldmarklength` are set to 2 mm in length. The length of the punchmark, `pfoldmarklength`, is instead set to 4 mm.

`foldmarkhpos`

This pseudo-length gives the distance of all horizontal foldmarks from the left edge of the paper. Normally, this is 3.5 mm. This value can be changed in the user's own `lco` file, in case the user's printer has a wider unprintable left margin. Whether the foldmarks are typeset at all depends on the option `foldmarks` (see section 6.2.6, page 143).

`lfoldmarkhpos`

v2.97e Apart from the horizontal foldmarks there is also a vertical foldmark, whose position from the left margin is set via the pseudo-length `lfoldmarkhpos`. This foldmark is used, for example, in Japanese Chou- or You-format envelopes, when one wishes to use A4 size sheets with them. This can also be useful for envelopes in C6 format.

`lfoldmarklength`

v2.97e The length of the vertical foldmark is set via the pseudo-length `lfoldmarklength`. Here too there is an exceptional behaviour. When the length is set to null, a length of 4 mm is actually used.

`foldmarkvpos`

v2.97e This pseudo-length gives the distance of all vertical foldmarks from teh upper edge of the paper. Normally this is 3.5 mm, but the value can be changed in the user's personal `lco` file in case the user's printer has a wider unprintable top margin. Whether the foldmarks are typeset at all depends on the option `foldmarks` (see section 6.2.6, page 143). At present there is only one vertical foldmark, which is designated the left vertical foldmark.

`foldmarkthickness`

v2.97c This pseudo-length determines the thickness of all foldmarks. Default value is 0.2 pt, in other words a very thin hairline. In particular, if the color of the foldmarks is changed, this can be too thin!

`foldmark`

v2.97c Via this element the color of the foldmarks can be changed. To do so, the commands to change the font of the element are usd, as described in section 6.3.1, page 151. The default setting is no change.

6.4.2. The Letterhead

The term letterhead here refers to all of the data pertaining to the sender and which is set above the addressee's address. It is usually expected that this information is set via the page style settings. In fact, this was the case in the earlier incarnation of the letter class, scrlettr. But with scrlttr2, the letterhead is made independent of the page style setting, and is set by the command `\opening`. The position of the letterhead is absolute and independent of the type area. In fact, the first page of a letter, the page that holds the letterhead, is set using the page style `empty`.

`firstheadvpos`

The pseudo-length `firstheadvpos` gives the distance between the top edge of the paper and start of the letterhead. This value is set differently in the various predefined `lco` files. A typical value is 8 mm.

`firstheadwidth`

The pseudo-length `firstheadwidth` gives the width of the letterhead. This value is set differently in the various predefined `lco` files. While this value usually depends on the paper width and the distance between the left edge of the paper and the adressee address field, it was the type area width in `KOMAold`.

`fromname`
`fromaddress`
`fromphone`
`fromfax`
`fromemail`
`fromurl`
`fromlogo`

These variables give all information concerning the sender necessary to create the letterhead. Which variables will actually be used to finally build the letterhead can be chosen by use of the letterhead extensions (see option `fromalign` in section 6.2.6, page 138) and the options given there. The variables `fromname`, `fromaddress` and `fromlogo` will be set in the letterhead without their labels; the variables `fromphone`, `fromfax`, `fromemail` and `fromurl` will be set with their labels. The labels are shown in table 6.14, page 168.

An important hint concerns the sender's address: within the sender's address, parts such as street, P.O. Box, state, country, etc., are separated with a double backslash. Depending on how the sender's address is used, this double backslash will be interpreted differently and therefore is not strictly always a line break. Paragraphs, vertical spacing and the like are usually not allowed within the sender's address declaration. One has to have very good knowledge of scrlttr2 to use things like those mentioned above, intelligently. Another point to note is the

Table 6.14.: The sender's predefined labels for the letterhead

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

one should most certainly set the variables for return address (see section 6.4.4, page 172) and signature (see section 6.6.1, page 180) oneself.

It is possible, by the way, to load an external picture to use as a logo. For this purpose one can put as content of `fromlogo` an `\includegraphics` command. Naturally, the corresponding package, that is, either graphics or graphicx (see [Car99b]), has to be loaded in the preamble of the letter document (see section 6.3.5).

`fromrulethickness`
`fromrulewidth`

Depending on the class option `fromrule` (see section 6.2.6, page 138), a horizontal rule can be drawn the predefined letterheads under or within the sender address. If the pseudo-length `fromrulewidth` has a value of 0 pt, which is the default in the predefined `lco` files, the rule length is calculated automatically taking into account, e. g., letterhead width or an optional logo. Users can adjust rule length manually in their own `lco` files by setting this pseudo-length to positive values using `\setplength` (see section 6.3.4, page 161). The default thickness of the line, `fromrulethickness`, is 0.4 pt.

v2.97c

`phoneseparator`
`faxseparator`
`emailseparator`
`urlseparator`

With these variables, hyphens are defined. If applicable, they are used in the sender's data in the letterhead (see table 6.14). As a feature, they are labeled and the labels also used in the sender's details of the letterhead. To look up the predefined labels and their contents, see table 6.15.

Table 6.15.: predefined labels and contents of hyphens for sender's data in the letterhead

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

\firsthead{*construction*}

For most cases, scrlttr2 with its options and variables offers enough possibilities to create a letterhead. In very rare situations one may wish to have more freedom in terms of layout. In those situations one will have to do without predefined letterheads, which could have been chosen via options. Instead, one needs to create one's own letterhead from scratch. To do so, one has to define the preferred *construction* with the command \firsthead. Within \firsthead, and with the help of the \parbox command (see [Tea05b]), one can set several boxes side by side, or one underneath the other. An advanced user will thus be able to create a letterhead on his own. Of course the *construct* may use variables with the help of \usekomavar.

6.4.3. The Letterfoot

As the first page holds a letterhead of its own, it also holds a footer of its own. And, as with the letterhead, it will not be set by the page style settings, but directly with the use of \opening.

firstfootvpos

This pseudo-length gives the distance between the letterfoot and the upper edge of the paper. This value is set differently in the various predefined lco files. It also takes care of preventing text from jutting into the footer area. If needed, it can help to shorten the text height on the first page using \enlargethispage. Likewise, and if it is needed, the text height can conversely be extended with the help of the option enlargefirstpage. This way, the distance between text area and the first letterfoot can be reduced to the value \footskip. See also section 6.2.3, page 135.

2.9t With the compatibility option set up to version 2.9t (see version in section 6.2.2, page 134) the footer is set independently of the type area in all predefined lco files (see section 6.2.9) except for KOMAold. The option enlargefirstpage also loses its effect. From version 2.9u onwards the footer is set in a position at the bottom edge of the paper. In this situation, the height of the type area also becomes dependent on enlargefirstpage.

v2.97e

If the letterfoot be switched off using option `firstfoot` (siehe section 6.2.7, page 145), then the setting of `firstfootvpos` is ignored, and instead `\paperheight` is applied. Thus, there remains a mimimum bottom margin of length `\footskip`.

`firstfootwidth`

This pseudo-length gives the width of the letter's first page footer. The value is set equal to that of the pseudo-length `firstheadwidth` in the predefined `lco` files.

`\firstfoot{`*construction*`}`

The first page's footer is preset to empty. However, with the `\firstfoot` command, it is possible to create a *construction* the same way as when defining the letterhead with `\firsthead`.

Example: In the first page's footer, you may want to set the content of the variable `frombank` (the bank account). The double backslash should be exchanged with a comma at the same time:

```
\firstfoot{%
  \parbox[b]{\linewidth}{%
    \centering\def\\{, }\usekomavar{frombank}%
  }%
}
```

For the hyphen you might define a variable of your own if you like. This is left as an exercise for the reader.

Nowadays it has become very common to create a proper footer in order to obtain some balance with respect to the letterhead. This can be done as follows:

```
\firstfoot{%
  \parbox[t]{\textwidth}{\footnotesize
    \begin{tabular}[t]{l@{}}%
      \multicolumn{1}{@{}l@{}}{Partners:}\\
      Jim Smith\\
      Russ Mayer
    \end{tabular}%
    \hfill
    \begin{tabular}[t]{l@{}}%
      \multicolumn{1}{@{}l@{}}{Manager:}\\
      Jane Fonda\\[1ex]
      \multicolumn{1}{@{}l@{}}{Court Of Jurisdiction:}\\
      Great Plains
    \end{tabular}%
```

```
    \ifkomavarempty{frombank}{}{%
      \hfill
      \begin{tabular}[t]{l@{}}%
        \multicolumn{1}{@{}l@{}}{\usekomavar*{frombank}:}\\
        \usekomavar{frombank}
      \end{tabular}%
    }%
  }%
}
```

This example, by the way, came from Torsten Krüger. With

```
\setkomavar{frombank}{Account No. 12\,345\,678\\
  at Citibank\\
  bank code no: 876\,543\,21}
```

the bank account can be set accordingly. If the footer will have such a large height then it might happen that you have to shift its position. You can do this with the pseudo-length `firstfootvpos`, which is described above in this section.

In the previous example a multi-line footer was set. With a compatibility setting to version 2.9u (see `version` in section 6.2.2, page 134) the space will in general not suffice. In that case, you may need to reduce `firstfootvpos` (see page 169) appropriately.

6.4.4. The Address

The term address here refers to the addressee's name and address which are output in an address field. Additional information can be output within this address field, such as dispatch type or a return address; the latter is especially useful when using window envelopes. The address directly follows the letterhead.

`toaddrvpos`
`toaddrhpos`

These pseudo-lengths define vertical and horizontal position of the address field relative to the top-left corner of the paper. Values are set differently in the various predefined `lco` files, according to standard envelope window measures. A special feature of `toaddrhpos` is that with negative values the offset is that of the right edge of the address field relative to the right edge of the paper. This can be found, for instance, in the case of `SN`. The smallest value of `toaddrvpos` is found with `DINmtext`. Care must be taken to avoid overlap of letterhead and address field. Whether the address field is output or not can be controlled by class option `addrfield` (see section 6.2.6, page 141).

`toaddrheight`

The pseudo-length `toaddrheight` defines the height of the address field, including the dispatch type. If no dispatch type is specified, then the address is vertically centered in the field. If a dispatch type is specified, then the address is set below the dispatch type, and vertically centered in the remaining field height.

`toaddrwidth`

The pseudo-length `toaddrwidth` defines the width of the address field. Values are set differently in the various predefined `lco` files, according to standard envelope window measures. Typical values are between 70 mm and 100 mm.

Example: Assume that your printer has a very wide left or right margin of 15 mm. In this case, when using the option `SN`, the letterhead, sender's extensions and the address can not be completely printed. Thus, you create a new `lco` file with the following content:

```
\ProvidesFile{SNmmarg.lco}
[2002/06/04 v0.1 my own lco]
\LoadLetterOption{SN}
\@addtoplength{toaddrwidth}{%
  -\useplength{toaddrhpos}}
\@setplength{toaddrhpos}{-15mm}
\@addtoplength{toaddrwidth}{%
  \useplength{toaddrhpos}}
\endinput
```

Then, until you can obtain a printer with smaller page margins, you simply use the option `SNmmarg` instead of `SN`.

`toaddrindent`

Additional indentation of the address within address field can be controlled by the pseudo-length `toaddrindent`. Its value applies to both left and right margin. Default value is 0 pt.

`backaddress`
`backaddressseparator`
`backaddrheight`

When using window envelopes, the sender's return address is often included within the window, placed at the top above the addressee and dispatch type information, separated by a horizontal rule and set in a smaller font size. The contents of the return address, stored in the variable `backaddress`, are usually built automatically from the variables `fromname` and `fromaddress`.

Within the return address, double backslashes are replaced by the content of the variable `backaddressseparator`, whose default value is a comma followed by a non-breaking space.

The height reserved for the return address within the address field is defined by the pseudo-length `backaddrheight`. In the predefined `lco` files, this is typically set to 5 mm. Whether the return address is output or not is controlled by document class options `addrfield` (see section 6.2.6, page 141) and `backaddress` (see section 6.2.6, page 141).

`specialmail`
`specialmailindent`
`specialmailrightindent`

An optional dispatch type can be output within the address field between the return address and the addressee address, by setting the variable `specialmail`. Left and right alignment are determined by pseudo-lengths `specialmailindent` and `specialmailrightindent`, respectively. In the predefined `lco` files provided by KOMA-Script, `specialmailindent` is set to rubber length `\fill`, while `specialmailrightindent` is set to 1 em. Thus the dispatch type is set 1 em from the address field's right margin.

`toname`
`toaddress`

These two variables contain the addressee's name and address as output in the address field. Usually you will not access these variables directly, but their values are taken from the argument to the `letter` environment. Please see the important hint on address formatting given in section 6.4.2, page 167.

`letter[`*`options`*`]{`*`addressee`*`}`

The `letter` environment is only one of the key environments of the letter class. A special scrlttr2 feature are optional arguments to the `letter` environment. These *`options`* are executed internally via the `\KOMAoptions` command.

The *`addressee`* is a mandatory argument passed to the `letter` environment. Parts of the addressee contents are separated by double backslashes. The first part of *`addressee`* is stored in variable `toname`, while the rest is stored in variable `toaddress` for further use. These parts are output on individual lines in the address field. Nevertheless, the double backslash should not be interpreted as a certain line break. Vertical material such as paragraphs or vertical space is not permitted within *`addressee`*, and could lead to unexpected results and error messages, as is the case also for the standard letter class.

The `letter` environment does not actually start the letter output. This is done by the `\opening` command.

`\AtBeginLetter{`*`commands`*`}`

LaTeX enables the user to declare *`commands`* whose execution is delayed until a determined point. Such points are called *hooks*. Known macros for using hooks are `\AtBeginDocument`

and \AtEndOfClass. The letter class scrlttr2 provides an additional hook that can be used via the macro \AtBeginLetter. Originally, hooks were provided for package and class authors, so they are documented in [Tea06] only, and not in [Tea05b]. However, with letters there are useful applications of \AtBeginLetter as the following example may illustrate:

Example: It is given that one has to set multiple letters with questionnaires within one document. Questions are numbered automatically within single letters using a counter. Since, in contrast to page numbering, that counter is not known by scrlttr2, it would not be reset at the start of each new letter. Given that each questionnaire contains ten questions, question 1 would get number 11 in the second letter. A solution is to reset this counter at the beginning of each new letter:

```
\newcounter{Question}
\newcommand{\Question}[1]{%
  \refstepcounter{Question}\par
  \@hangfrom{\makebox[2em][r]{\theQuestion:~}}{#1}}
\AtBeginLetter{\setcounter{Question}{0}}
```

This way question 1 remains question 1, even in the 1001st letter. Of course definitions like those mentioned above need to be stated either between macros \makeatletter and \makeatother (see [Wik]) in letter declarations (see section 6.3.5 and figure 6.2, page 164), in a unique package, or in an lco file (see section 6.2.9).

6.4.5. The Sender's Extensions

Often, especially with business letters, the space for the letterhead or page footer seems to be too tight to include all you want. To give more details about the sender, often the space right beside the addressee's field is used. In this manual this field is called the *sender's extension*

```
locheight
lochpos
locvpos
locwidth
```

v2.97d

The pseudo-lengths locwidth and locheight set the width and height of the sender's extension field. The pseudo-lengths lochpos and locvpos determine the distances from the right and upper paper edges. These value is typically set to 0 pt in the predefined lco files. This does not mean that the sender's extension has no width; instead, it means that the actual width is set with \opening when the paper width, address window width, and the distance between the left and upper edges of the paper and the address window are known. The option locfield (see section 6.2.6, page 142) is also taken into account. As is the case for toaddrhpos, negative values of lochpos take on a special meaning. In that case, instead

of referring to a distance from the right edge of the paper, `lochpos` now means a distance from the left edge of the paper. The meaning is thus the opposite to that of `toaddrhpos` (see section 6.4.4, page 171).

`location`

The contents of the sender's extension field is determined by the variable `location`. To set this variable's content, it is permitted to use formatting commands like `\raggedright`. One has to consider that depending on the use of the options `fromalign` and `fromlogo`, a part of the space for the sender's extension may already be reserved for a logo or return address (see section 6.2.6, page 138 and page 141).

Example: Assume that you would like to put the names of your partners, manager, or court of jurisdiction in the sender's extension field. You can do this as follows:

```
\KOMAoptions{locfield=wide}
\setkomavar{location}{\raggedright
  \textbf{Partners:}\\
  \quad Hugo Mayer\\
  \quad Bernd Miller\\[1ex]
  \textbf{Manager:}\\
  \quad Liselotte Mayer\\[1ex]
  \textbf{Court of jurisdiction:}\\
  \quad Washington, DC
}
```

The option `locfield=wide` is set to make the details fit horizontally. Sender details like those mentioned in the above example can be written, together with the common sender address details, into your own `lco` file.

6.4.6. The Reference Fields Line

Especially with business letters, a line can be found that gives initials, dial code, customer number, invoice number, or a reference to a previous letter. In this manual this line is called the *reference fields line*. The reference fields line can consist of more than just one line and is set only if one of those variables mentioned above is given. Only those fields will be set that are given. To set a seemingly empty field, one needs to give as value at least a white space or `\null`. If you want to have your letter without a reference fields line, then instead of it the label and contents of the variable `date` will be set.

`refvpos`

This pseudo-length gives the distance between the upper edge of the paper and the reference fields line. Its value is set differently in the various predefined `lco` files. Typical values are between 80.5 mm and 98.5 mm.

`refwidth`
`refhpos`

This pseudo-length gives the width that is available for the reference fields line. The value is set typically to 0 pt in the predefined `lco` files. This value has a special meaning: in no way does it determine that there is no available width for the business line; instead, this value means that the width will be calculated with the `\opening`. Thus the calculated width depends on the determination of the options `refline` (see section 6.2.6, page 145). At the same time, `refhpos` will be set according to this option. With `refline=wide`, the reference fields line is centered, while with `refline=narrow` it is aligned on the left.

If `refwidth` non-null, i. e., the width of the reference fields line is therefore not determined by the option `refline`, then `refhpos` gives the distance of the reference fields line from the left edge of the paper. If this distance is null, then the reference fields line is set so that the ratio between its distances from the left and right edges of the paper equal the ratio of distance of the type area from the left and right edges of the paper. Thus, for a type area horizontally centered on the paper, the reference fields line too will be centered.

As a rule, these special cases are likely to be of little interest to the normal user. The simplest rule is as follows: either `refhpos` is left at null and so the width and alignment of the reference fields line are left to the option `refline`, or `refwidth` as well as `refhpos` are set by the user.

`refaftervskip`

This pseudo-length gives the vertical space that has to be inserted beneath the reference fields line. The value is set in the predefined `lco` files. It directly affects the text height of the first page. A typical value lies between one and two lines.

`yourref`
`yourmail`
`myref`
`customer`
`invoice`
`date`

These variables are typical reference fields. Their meanings are given in table 6.12 on page 154. Each variable has also a predefined label, shown in table 6.16. The field width that belongs to each variable, adjusts itself automatically to its label and content.

`place`
`placeseparator`

As said before in the introduction of this subsection, the reference fields line can be omitted. This happens if all variables of the business line are empty with the exception of the variable for the date. In this case, the content of `place` and `placeseparator` will be set, followed by the content of `date`. The predefined content of the `placeseparator` is a comma followed

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

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

by a non-breaking space. If the variable `place` has no value then the hyphen remains unset also. The predefined content of `date` is `\today` and depends on the setting of the option `numericaldate` (see section 6.2.6, page 143).

6.4.7. The Title and the Subject Line

Business letters very often carry a subject line. The subject line indicates briefly the respect of the letter. Usually the subject should be short and precise and not run across several lines. Apart fom the subject, such a letter may also carry a title. Titles find usage most often with irregular letters such as a warning, an invoice or a reminder.

`title`

With scrlttr2 a letter can carry an additional title. The title is centered and set with font size `\LARGE` directly after and beneath the reference fields line. The predefined font setup for this element (`\normalcolor\sffamily\bfseries`) can be changed with help of the interface described in section 3.2.1. Font size declarations are allowed.

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

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

This way the addressee will recognize a reminder as such.

`subject`
`subjectseparator`

In case a subject should be set, the contents of the variable `subject` need to be defined. Depending on what the option `subject` is set to, a label can be placed in front of the subject contents; also, the vertical position of the subject contents can be changed (see section 6.2.6, page 142). The predefined labels are shown in table 6.17. The predefined value of `subjectseparator` is a colon followed by a non-breaking space.

Table 6.17.: Predefined labels of subject-related variables.

name	label
subject	\usekomavar*{subjectseparator}% \usekomavar{subjectseparator}
subjectseparator	\subjectname

The subject line is set in a separate font. To change this use the user interface described in section 3.2.1. For the element subject the predetermined font in scrlttr2 is \normalfont\normalcolor\bfseries.

Example: Assume you are a board member and want to write a letter to another member of that board about a few internals of the organization. You want to clarify with your subject line what this letter is all about, but without labeling it thus. You can do this as follows:

```
\setkomavar{subject}[Subject ]{%
  organization's internals}
```

or easier:

```
\setkomavar{subject}[]{%
  about organization's internals}
```

Furthermore, if you want to set the subject line not only in bold but also in sans serif:

```
\addtokomafont{subject}{\sffamily}
```

As you can see, it is really easy to solve such problems.

6.4.8. Further Settings

In this paragraph variables and settings are listed which could not be assigned to any other part of the letter declaration but somehow belong to this section.

frombank

This variable at the moment takes on a special meaning: it is not used internally at this point, and the user can make use of it to set, for example, his bank account within the sender's extension field or the footer.

\nexthead{*construction*}
\nextfoot{*construction*}

The possibilities that are offered with variables and options in scrlttr2 should be good enough in most cases to create letterheads and footers for those pages that follow the first letter

page. Even more so since you can additionally change with `\markboth` and `\markright` the sender's statements that scrlttr2 uses to create the letterhead. The term "subsequent pages" in this manual refers to all pages following the first letter page. The commands `\markboth` and `\markright` can in particular be used together with pagestyle `myheadings`. If the package scrpage2 is used then this, of course, is valid also for pagestyle `scrheadings`. There the command `\markleft` is furthermore available.

At times one wants to have more freedom with creating the letterhead or footer of subsequent pages. Then one has to give up the possibilities of predefined letterheads or footers that could have been chosen via the option `pagenumber` (see section 6.2.4, page 136). Instead one is free to create the letterhead and footer of subsequent pages just the way one wants to have them set. For that, one creates the desired letterhead or footer *`construction`* using the command `\nexthead` or `\nextfoot`, respectively. Within `\nexthead` and `\nextfoot` you can, for example, have several boxes side by side or one beneath the other by use of the `\parbox` command (see [Tea05b]). A more advanced user should have no problems creating letterheads of footers of his own. Within *`construction`* you can of course also make use of the variables by using `\usekomavar`.

6.5. The Text

In contrast to an article, a report or a book, a letter normally has no chapter or section structure. Even float environments with tables and figure are unusual. Therefore, a letter has no table of contents, lists of figures and tables, index, bibliography, glossary or similar things. The letter text mainly consists of an opening and the main text. Thereupon follow the signature, a postscript and various listings.

6.5.1. The Opening

In the early days of computer-generated letters, programs did not have many capabilities, therefore the letters seldom had an opening. Today the capabilities have been enhanced. Thus personal openings are very common, even in mass-production advertising letters.

`\opening{`*`opening`*`}`

This is one of the most important commands in scrlttr2. For the user it may seem that only the opening is typeset, but the command also typesets the folding marks, letterhead, address field, reference fields line, subject, the page footer and others. In short, without `\opening` there is no letter.

6.5.2. Footnotes

In letters footnotes should be used more sparingly than in normal documents. However, scrlttr2 is equipped with all mechanisms mentioned in section 3.6.3 for the main document classes.

Therefore they will not be discussed here again.

6.5.3. Lists

Lists have the same validity in letters as in normal documents. Thus scrlttr2 provides the same possibilities as mentioned in section 3.6.4 for the main document classes. Therefore they will not be discussed here again.

6.5.4. Margin Notes

Margin notes are quite uncommon in letters. Therefore the option `mpinclude` is not actively supported by scrlttr2. However, scrlttr2 is equipped with all mechanisms mentioned in section 3.6.5 for the main document classes. Therefore they will not be discussed here again.

6.5.5. Text Emphasis

The distinction of text has the same importance in letters as in other documents. Thus the same rules apply, meaning: emphasize text sparingly. Even letters should be readable and a letter where each word is typeset in another font is quite unreadable.

The class scrlttr2 is equipped with all mechanisms mentioned in section 3.6.7 for the main document classes. Therefore it will not be discussed here again.

6.6. The Closing Part

A letter always ends with a closing phrase. Even computer-generated letters without signature have such a phrase. Sometimes this is a sentence such as, “This letter has been generated automatically and is valid without a signature.”. Sometimes a sentence like this will even be used as a signature. Thereupon can follow a postscript and various listings.

6.6.1. Closing

The closing consists of three parts: besides the closing phrase there are a hand-written inscription and the signature, which acts as an explanation for the inscription.

`signature`

The variable `signature` holds an explanation for the inscription. The content is predefined as `\usekomavar{fromname}`. The explanation may consist of multiple lines. The lines should then be separated by a double backslash. Paragraphs in the explanation are however not permitted.

`\closing{`*closing phrase*`}`

The command `\closing` not only typesets the closing phrase, but also the content of the variable `signature`. The closing phrase may consists of multiple lines, but paragraphs are not permitted.

`sigindent`
`sigbeforevskip`
`\raggedsignature`

Closing phrase and signature will be typeset in a box. The width of the box is determined by the length of the longest line of the closing phrase or signature.

The box will be typeset with indentation of the length set in pseudo-length `sigindent`. In the predefined `lco` files this length is set to 0 mm.

The command `\raggedsignature` defines the alignment inside the box. In the predefined `lco` files the command is either defined as `\centering` (all besides `KOMAold`) or `\raggedright` (`KOMAold`). In order to obtain flush-right or flush-left alignment inside the box, the command can be redefined in the same way as `\raggedsection` (see section 3.6.2, page 77).

Between closing phrase and signature a vertical space is inserted, the height of which is defined in the pseudo-length `sigbeforevskip`. In the predefined `lco` files this is set to 2 lines. In this space you can then write your inscription.

Example: You are writing as the directorate of a society a letter to all members. Moreover, you want on the one hand to elucidate that you are writing in the name of the board of directors, and on the other hand you want indicate your position on the board of directors.

```
\setkomavar{signature}{John McEnvy\\
 {\small (Vice-President ``The Other Society'')}}
\closing{Regards\\
  (for the board of directors)}
```

You can of coure set the variable `signature` in your private `lco` files. Otherwise it is advisable to define the variable in the letter preamble (see section 6.4).

6.6.2. Postscript, Carbon Copy and Enclosures

After the closing can follow some other statements. Besides the postscript, there are the distribution list of carbon copies, and the reference to enclosures.

`\ps`

In the time when letters were written by hand it was quite common to use a postscript because this was the only way to add information which one had forgotten to mention in the main part of the letter. Of course, in letters written with LaTeX you can insert additional lines easily. Nevertheless,

it is still popular to use the postscript. It gives one a good possibility to underline again the most important or sometimes the less important things of the particular letter.

This instruction merely switches to the postscript. Hence, a new paragraph begins, and a vertical distance — usually below the signature — is inserted. The command `\ps` is followed by normal text. If you want the postscript to be introduced with the acronym "PS:" , which by the way is written without a full stop, you have to type this yourself. The acronym is typeset neither automatically nor optionally by the class scrlttr2.

`\cc{`*`distribution list`*`}`
`ccseparator`

With the command `\cc` it is possible to typeset a *`distribution list`*. The command takes the *`distribution list`* as its argument. If the content of the variable `ccseparator` is not empty, then the name and the content of this variable is inserted before *`distribution list`*. In this case the *`distribution list`* will be indented appropriately. It is a good idea to set the *`distribution list`* `\raggedright` and to separate the individual entries with a double backslash.

Example: You want to indicate that your letter is sent to all members of a society and to the board of directors:

```
\cc{%
  the board of directors\\
  all society members}
```

This instruction should be written below the `\closing` instruction from the previous example, or below a possible postscript.

A vertical space is inserted automatically before the distribution list.

`\encl{`*`enclosures`*`}`
`enclseparator`

`Enclosures` have the same structure as the distribution list. The only difference is that here the enclosures starts with the name and content of the variable `enclseparator`.

6.7. Language Support

The document class scrlttr2 supports many languages. These include German (`german` for old German orthography, `ngerman` for the new orthography, and `austrian` for Austrian), English (`english` without specification as to whether American or British should be used, `american` and `USenglish` for American, and `british` and `UKenglish` for British), French, Italian, Spanish, Dutch, Croatian, Finnish, Norsk, and Swedish.

v3.08

6.7.1. Language Selection

If the package babel (see [Bra01]) is used, one can switch between languages with the command `\selectlanguage{`*`language`*`}`. Other packages like german (see [Rai98a]) and ngerman (see [Rai98b]) also define this command. As a rule though, the language selection takes place already as a direct consequence of loading such a package. Further information can be obtained in the documentation of the relevant packages.

There is one thing more to mention about language packages. The package french (see [Gau03]) redefines not only the terms of section 6.7.2, but also other, for instance it even redefines the command `\opening`, since it assumes that the definition of the standard letter is used. With scrlttr2 this is not the case, therefore the package french destroys the definition in scrlttr2 and does not work correctly with KOMA-Script. The author views this is a fault in the french package.

If one utilizes the babel package in order to switch to language `french` while the package french is simultaneously installed, then the same problems will likely occur, since babel employs definitions from the french package. If the package french is not installed then there are no problems. Aimilarly, there is no problem if for babel instead of `french` other languages like `acadian`, `canadien`, `francais` or `frenchb` are chosen.

From babel version 3.7j this problem only occurs when it is indicated explicitly by means of an option that babel should use the french package.

If it cannot be ascertained that a new version of babel is being used, it is recommended to use

```
\usepackage[...,frenchb,...]{babel}
```

in order to select french.

Other languages can possibly cause similar problems. Currently there are no known problems with the babel package for the german language and the various english language selections.

```
\captionsenglish
\captionsUSenglish
\captionsamerican
\captionsbritish
\captionsUKenglish
\captionsgerman
\captionsngerman
\captionsaustrian
\captionsfrench
\captionsitalian
\captionsspanish
\captionsdutch
\captionscroatian
\captionsfinnish
\captionsnorsk
\captionsswedish
```

If one switches the language then using these commands the language-dependent terms from section 6.7.2 are redefined. If the used language selection scheme does not support this then the commands above can be used directly.

```
\dateenglish
\dateUSenglish
\dateamerican
\datebritish
\dateUKenglish
\dategerman
\datengerman
\dateaustrian
\datefrench
\dateitalian
\datespanish
\datedutch
\datecroatian
\datefinnish
\datenorsk
\dateswedish
```

The numerical representation of the date (see option `numericaldate` in section 6.2.6) will be written depending on the selected language. Some examples can be found in table 6.18.

6.7.2. Language-Dependent Terms

As is usual in LaTeX, the language-dependent terms are defined by commands which are then redefined when one switches the language.

Table 6.18.: Language-dependent forms of the date

Command	Date example
\dateenglish	24/12/1993
\dateUSenglish	12/24/1993
\dateamerican	12/24/1993
\datebritish	24/12/1993
\dateUKenglish	24/12/1993
\dategerman	24. 12. 1993
\datengerman	24. 12. 1993
\dateaustrian	24. 12. 1993
\datefrench	24. 12. 1993
\dateitalian	24. 12. 1993
\datespanish	24. 12. 1993
\datedutch	24. 12. 1993
\datecroatian	24. 12. 1993.
\datefinnish	24.12.1993.
\datenorsk	24.12.1993
\dateswedish	24/12 1993

```
\yourrefname
\yourmailname
\myrefname
\customername
\invoicename
\subjectname
\ccname
\enclname
\headtoname
\headfromname
\datename
\pagename
\phonename
\faxname
\emailname
\wwwname
\bankname
```

The commands above contain the language-dependent terms. These definitions can be modified in order to support a new language or for private customization. How this can be done is described in section 6.7.3. The definitions become active only at `\begin{document}`. Therefore they are not available in the LaTeX preamble and cannot be redefined there. In table 6.19

Table 6.19.: Default settings for languages `english` and `ngerman`

Command	`english`	`ngerman`
`\bankname`	Bank account	Bankverbindung
`\ccname`[1]	cc	Kopien an
`\customername`	Customer no.	Kundennummer
`\datename`	Date	Datum
`\emailname`	Email	E-Mail
`\enclname`[1]	encl	Anlagen
`\faxname`	Fax	Fax
`\headfromname`	From	Von
`\headtoname`[1]	To	An
`\invoicename`	Invoice no.	Rechnungsnummer
`\myrefname`	Our ref.	Unser Zeichen
`\pagename`[1]	Page	Seite
`\phonename`	Phone	Telefon
`\subjectname`	Subject	Betrifft
`\wwwname`	Url	URL
`\yourmailname`	Your letter of	Ihr Schreiben vom
`\yourrefname`	Your ref.	Ihr Zeichen

[1] Normally these terms are defined by language packages like babel. In this case they are not redefined by scrlttr2 and may differ from the table above.

the default settings for `english` and `ngerman` can be found.

6.7.3. Defining and Changing Language-dependent Terms

Normally one has to change or define the language-dependent terms of section 6.7.1 in such a way that in addition to the available terms the new or redefined terms are defined. This is made more difficult by the fact that some packages like german or ngerman redefine those settings when the packages are loaded. This definitions unfortunately occurs in such a manner as to destroy all previous private settings. That is also the reason why scrlttr2 delays its own changes, with `\AtBeginDocument` until `\begin{document}`, that is, after package loading is completed. The user can also use `\AtBeginDocument`, or redefine the language-dependent terms after `\begin{document}`, that is, not put them in the preamble at all. The class scrlttr2 even provides some additional commands for defining language-dependent terms.

```
\providecaptionname{language}{term}{definition}
\newcaptionname{language}{term}{definition}
\renewcaptionname{language}{term}{definition}
```

Using one of the commands above, the user can assign a *definition* for a particular *language* to a *term*. The *term* is always a macro. The commands differ dependent on whether a given *language* or a *term* within a given *language* are already defined or not at the time the command is called.

If *language* is not defined, then `\providecaptionname` does nothing other than writes a message in the log file. This happens only once for each language. If *language* is defined but *term* is not yet defined for it, then it will be defined using *definition*. The *term* will not be redefined if the *language* already has such a definition; instead, an appropriate message is written to the log file.

The command `\newcaptionname` has a slightly different behaviour. If the *language* is not yet defined, then a new language command (see section 6.7.1) will be created and a message written to the log file. If *term* is not yet defined in *language*, then it will be defined using *definition*. If *term* already exists in *language*, then this results in an error message.

The command `\renewcaptionname` again behaves differently. It requires an existing definition of *term* in *language*. If neither *language* nor *term* exist or *term* is unknown in a defined language then a error message will be given. Otherwise, the *term* for *language* will be redefined according to *definition*.

The class scrlttr2 itself employs `\providecaptionname` in order to define the commands in section 6.7.2.

Example: If you prefer "Your message of" instead of "Your letter of", you have to redefine the definition of `\yourmailname`.

```
\renewcaptionname{english}{\yourmailname}{%
  Your message of}
```

Since only existing terms in available languages can be redefined, you have to put the command after `\begin{document}` or delay the command by using `\AtBeginDocument`. Furthermore, you will get an error message if there is no package used that switches language selection to *english*.

6.8. Address Files and Circular Letters

When people write circular letters one of the more odious tasks is the typing of many different addresses. The class scrlttr2, as did its predecessor scrlettr as well, provides basic support for this task. Currently there are plans for much enhanced support.

```
\adrentry{Lastname}{Firstname}{Address}{Phone}{F1}{F2}{Comment}{Key}
```

The class scrlttr2 supports the use of address files which contain address entries, very useful for circular letters. The file extension of the address file has to be `.adr`. Each entry is an

\adrentry command with eight parameters, for example:

```
\adrentry{McEnvy}
         {Flann}
         {Main Street 1\\ Glasgow}
         {123 4567}
         {male}
         {}
         {niggard}
         {FLANN}
```

The 5th and 6th elements, F1 and F2, can be used freely: for example, for the gender, the academic grade, the birthday, or the date on which the person joined a society. The last parameter *Key* should only consist of uppercase letters in order to not interfere with existing TEX or LATEX commands.

Example: Mr. McEnvy is one of your most important business partners, but every day you receive correspondence from him. Before long you do not want to bother typing his boring address again and again. Here scrlttr2 can help. Assume that all your business partners have an entry in your partners.adr address file. If you now have to reply to Mr. McEnvy again, then you can save typing as follows:

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

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

\addrentry{*Lastname*}{*Firstname*}{*Address*}{*Phone*}{*F1*}{*F2*}{*F3*}{*F4*}{*Key*}

Over the years people have objected that the \adrentry has only two free parameters. To cater to this demand, there now exists a new command called \addrentry — note the additional "d" — which supports four freely-definable parameters. Since TEX supports maximally nine parameters per command, the comment parameter has fallen away. Other than this difference, the use is the same as that of \adrentry.

Both \adrentry and \addrentry commands can be freely mixed in the adr files. However, it should be noted that there are some packages which are not suited to the use of \addrentry. For example, the adrconv by Axel Kielhorn can be used to create address lists from adr files,

but it has currently no support for command \addrentry. In this case, the only choice is to extend the package yourself.

Besides the simple access to addresses, the address files can be easily used in order to write circular letters. Thus, there is no requirement to access a complicated database system via TEX.

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

```
\documentclass{scrlttr2}
\begin{document}
\renewcommand*{\adrentry}[8]{
  \begin{letter}{#2 #1\\#3}
    \opening{Dear members,} Our next general meeting will be ↩
on
    Monday, August 12, 2002. The following topics are \dots
    \closing{Regards,}
  \end{letter}
}
\input{members.adr}
\end{document}
```

If the address file contains \addrentry commands too, than an additional definition for \addrentry is required before loading the address file:

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

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

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

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

```
\renewcommand*{\adrentry}[8]{
  \ifdim #6pt>0pt\relax
  % #6 is an amount greater than 0.
```

```
  % Thus, this selects all members with due subscription.
    \begin{letter}{#2 #1\\#3}
      \if #5m \opening{Dear Mr.\,#2,} \fi
      \if #5f \opening{Dear Mrs.\,#2,} \fi

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

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

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

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

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

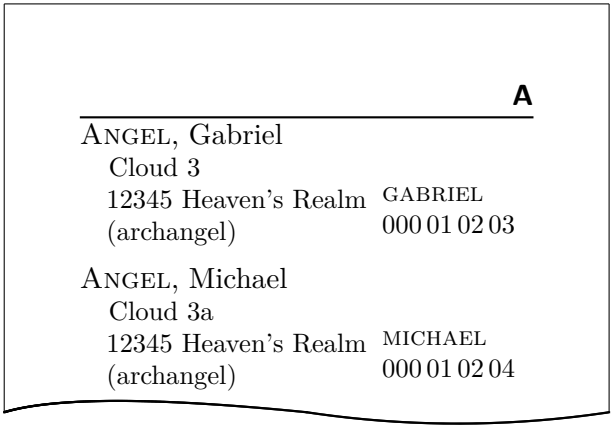
The address files have to be sorted already in order to obtain sorted lists. It is recommended to separate the sorted entries at each different initial letter of *Lastname*. As a separator, the commands `\adrchar` and `\addrchar` can be used.. These commands will be ignored if the address files are utilized in scrlettr2.

Example: Suppose you have the following short address file:

```
\adrchar{A}
\adrentry{Angel}{Gabriel}
         {Cloud 3\\12345 Heaven's Realm}
         {000\,01\,02\,03}{}{}{archangel}{GABRIEL}
\adrentry{Angel}{Michael}
         {Cloud 3a\\12345 Heaven's Realm}
         {000\,01\,02\,04}{}{}{archangel}{MICHAEL}
\adrchar{K}
\adrentry{Kohm}{Markus}
         {Freiherr-von-Drais-Stra\ss e 66\\68535 Edingen-↩
 Neckarhausen}
```

```
{+49~62\,03~1\,??\,??}{}{}{no angel at all}
{KOMA}
```

This address file can be treated with `adrdir.tex` of the adrconv package [Kie99]. The result should look like this:



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

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

6.9. From scrlettr to scrlttr2

The first step in the conversion of an old letter written with the scrlettr class is to load the appropriate `lco` file using option `KOMAold` at `\documentclass`. Thereupon most commands of the old class should work. However, you will encounter some differences in the output, since the page layout of the old class is not realized exactly. The reason is that the calculation of the type area in scrlettr has some minor bugs. For example, the position of the folding marks used to depend on the height of the page header, which again depended on the font size. That was unambiguously a design error.

There is no compatibility regarding the defined lengths in scrlettr. Thus, if a user has changed the page layout of scrlettr, then the relevant statements should be deleted or commented out. In some cases, the modification of a length can cause an error, since this length is no longer defined in scrlttr2. The user should delete or comment out such modifications as well.

After the switch from scrlettr to scrlttr2, the old letter example can be successfully compiled already alone through the setting of the option `KOMAold`:

```
\documentclass[10pt,KOMAold]{scrlttr2}
\name{{\KOMAScript} team}
\address{Class Alley 1\\12345 {\LaTeX} City}
\signature{Your {\KOMAScript} team}
\begin{document}
  \begin{letter}{{\KOMAScript} users\\
                 Everywhere\\world-wide}
    \opening{Dear {\KOMAScript} users,}
    the  {\KOMAScript} team is proud to announce \dots
    \closing{Happy {\TeX}ing}
  \end{letter}
\end{document}
```

The next step is that while wanting the old commands to be still be available, the layout of the old letter should no longer be used. If, for example, one wants to sue the layout of the letter class option `DIN`, then this option can be given in `\documentclass`, but is has to be specified *after* the option `KOMAold`:

```
\documentclass[10pt,KOMAold,DIN]{scrlttr2}
\name{{\KOMAScript} team}
\address{Class Alley 1\\12345 {\LaTeX} City}
\signature{Your {\KOMAScript} team}
\begin{document}
  \begin{letter}{{\KOMAScript} users\\
                 Everywhere\\world-wide}
    \opening{Dear {\KOMAScript} users,}
    the {\KOMAScript} team is proud to announce \dots
    \closing{Happy {\TeX}ing}
  \end{letter}
\end{document}
```

By using more options this way, you can further influence the layout. However, the author recommends a more inherent change right away.

The last step is to replace all old commands with their new equivalents, and to omit the option `KOMAold`. For this task, it may help to read the contents of `KOMAold.lco`. In that file the old commands are defined using the new commands and variables.

```
\documentclass{scrlttr2}
\setkomavar{fromname}{{\KOMAScript} team}
\setkomavar{fromaddress}{Class Alley 1\\
                         12345 {\LaTeX} City}
\setkomavar{signature}{Your {\KOMAScript} team}
\let\raggedsignature=\raggedright
```

```
\begin{document}
  \begin{letter}{{\KOMAScript} users\\
                 Everywhere\\
                 world-wide}
    \opening{Dear {\KOMAScript} users,}
    the {\KOMAScript} team is proud to announce \dots
    \closing{Happy {\TeX}ing}
  \end{letter}
\end{document}
```

This example shows also the possibility to change the alignment of the signature by redefining the command `\raggedsignature`. This is always recommended when the width of the explanation of the signature as defined by the command `\setkomavar{signature}{...}` is greater than the width of the argument of `\closing`.

Access to Address Files with scraddr

7.1. Overview

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

`\InputAddressFile{`*file name*`}`

The command `\InputAddressFile` is the main command of the scraddr, and reads the content of the address file given as its parameter. If the file does not exist the command returns an error message.

For every entry in the address file the command generates a set of macros for accessing the data. For large address files this will take a lot of 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 6.8 from page 187 onwards. The division of the address file with the help of `\adrchar` or `\addrchar`, also discussed therein, has no meaning for scraddr and is simply ignored.

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

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

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

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

7.2. Usage

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

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

The 4th parameter, the telephone number, has been left blank. If you know the story behind these addresses you will agree that a telephone number makes no sense here, and besides, it should simply be possible to leave them out. The command `\InputAddressFile` is used to load the address file shown above:

```
\InputAddressFile{lotr}
```

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

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

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

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

```
    Do you remember? Certainly Mithrandir has told you much
    about their deeds and adventures .
  \closing{``O spring-time and summer-time
             and spring again after!\\
           O wind on the waterfall,
             and the leaves' laughter!''}
\end{letter}
```

In the address of letters often both firstname and lastname are required. Thus, the command `\Name{Key}` is an abridgement for `\FirstName{Key} \LastName{Key}`.

The 5th and 6th parameters of the `\adrentry` or `\adrentry` commands are for free use. They are accessible with the commands `\FreeI` and `\FreeII`. In this example, the 5th parameter contains the name of a person who is the most important in the life of the entry's person, the 6th contains the person's passion. The 7th parameter is a comment or in general also a free parameter. The commands `\Comment` or `\FreeIII` give access to this data. Use of `\FreeIV` is only valid for `\addrentry` entries; for `\adrentry` entries it results in an error. More on this is covered in the next section.

7.3. Package Warning Options

As mentioned above, the command `\FreeIV` leads to an error if it is used for `\adrentry` entries. How scraddr reacts in such a situation is decide by package options.

```
adrFreeIVempty
adrFreeIVshow
adrFreeIVwarn
adrFreeIVstop
```

These four options allow the user to choose between *ignore* and *rupture* during the LaTeX run if `\FreeIV` has been used with an `\adrentry` entry.

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

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

`adrFreeIVwarn` – writes a warning in the logfile

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

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

Creating Address Files from a Address Database

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

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

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

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

Control Package Dependencies with scrlfile

The introduction of LaTeX 2ε in 1994 brought many changes in the handling of LaTeX extensions. Today the package author has many macros available to determine if another package or class is employed and whether specific options are used. The author can load other packages or can specify options in the the case that the package is loaded later. This has led to the expectation that the order in which package are loaded would not be important. Sadly this hope has not been fulfilled.

9.1. About Package Dependencies

More and more frequently, different packages either newly define or redefine the same macro again. In such a case the order in which a package is loaded becomes very important. For the user it sometimes becomes very difficult to understand the behaviour, and in some cases the user wants only to react to the loading of a package. This too is not really a simple matter.

Let us take the simple example of loading the package longtable with a KOMA-Script document class. The longtable package defines table captions very well suited to the standard classes, but the captions are totally unsuitable for documents using KOMA-Script and also do not react to the options of the provided configuration commands. In order to solve this problem, the longtable package commands which are responsible for the table captions need to be redefined. However, by the time the longtable package is loaded, the KOMA-Script class has already been processed.

Until the present, the only way for KOMA-Script to solve this problem was to delay the redefinition until the beginning of the document with help of the macro `\AtBeginDocument`. If the user wants to change the definitions too, it is recommended to do this in the preamble of the document. However, this is impossible since later at `\begin{document}` KOMA-Script will again overwrite the user definition with its own. Therefore, the user too has to delay his definition with `\AtBeginDocument`.

Actually, KOMA-Script should not need to delay the redefinition until `\begin{document}`. It would be enough to delay exactly until the package longtable has been loaded. Unfortunately, the LaTeX kernel does not define appropriate commands. The package scrlfile provides redress here.

Likewise, it might be conceivable that before a package is loaded one would like to save the definition of a macro in a help-macro, in order to restore its meaning after the package has been loaded. The package scrlfile allows this, too.

The employment of scrlfile is not limited to package dependencies only. Even dependencies on any other file can be considered. For example, the user can be warned if the not uncritical file `french.ldf` has been loaded.

Although the package is particularly of interest for package authors, there are of course applications for normal LaTeX users, too. Therefore, this chapter gives and explains examples for both

groups of users.

9.2. Actions Prior to and After Loading

scrlfile can execute actions both before and after the loading of files. In the commands used to do this, distinctions are made made between general files, classes, and packages.

```
\BeforeFile{file}{instructions}
\AfterFile{file}{instructions}
```

The macro `\BeforeFile` ensures that *instructions* are only executed before the next time *file* is loaded. `\AfterFile` works in a similar fashion, and the *instructions* will be executed only after the *file* has been loaded. If *file* is never loaded then the *instructions* will never be executed.

In order to implement those features scrlfile redefines the well known LaTeX command `\InputIfFileExists`. If this macro does not have the expected definition then scrlfile issues a warning. This is for the case that in future LaTeX versions the macro can have a different definition, or that another package has already redefined it.

The command `\InputIfFileExists` is used by LaTeX every time a file is to be loaded. This is independent of whether the actual load command is `\include`, `\LoadClass`, `\documentclass`, `\usepackage`, `\RequirePackage`, or similar. Exceptionally, the command

```
\input foo
```

loads the file `foo` without utilizing `\InputIfFileExists`. Therefore, one should always use

```
\input{foo}
```

instead. Notice the parentheses surrounding the file name!

```
\BeforeClass{class}{instructions}
\BeforePackage{package}{instructions}
```

These two commands work in the same way as `\BeforeFile`. The only difference is that the document class *class* and the LaTeX package *package* are specified with their names and not with their file names. That means that the file extensions `.cls` and `.sty` can be omitted.

```
\AfterClass{class}{instructions}
\AfterClass*{class}{instructions}
\AfterPackage{package}{instructions}
\AfterPackage*{package}{instructions}
```

The commands `\AfterClass` and `\AfterPackage` work in the same way as `\AfterFile`. The only difference is that the document class *class* and the LaTeX package *package* are specified with their names and not with their file names. That means that the file extensions `.cls` and

`.sty` can be omitted. The starred versions execute the *instructions* not only at next time that the class or package is loaded, but also immediately if the class or package has been loaded already.

Example: In the following, an example for class and package authors shall be given. It shows how KOMA-Script itself employs the new commands. The class scrbook contains:

```
\AfterPackage{hyperref}{%
  \@ifpackagelater{hyperref}{2001/02/19}{}{%
    \ClassWarningNoLine{scrbook}{%
      You are using an old version of hyperref package!%
      \MessageBreak%
      This version has a buggy hack at many drivers%
      \MessageBreak%
      causing \string\addchap\space to behave strange.%
      \MessageBreak%
      Please update hyperref to at least version
      6.71b}}}
```

Old versions of the hyperref package redefine a macro of the scrbook class in such a way that does not work with newer KOMA-Script versions. New versions of hyperref desist from making these changes if a new KOMA-Script version is detected. For the case that hyperref is loaded at a later stage, therefore, the code in scrbook verifies that a acceptable hyperref version is used. If not, the command issues a warning.

At other places in three KOMA-Script classes the following can be found:

```
\AfterPackage{caption2}{%
  \renewcommand*{\setcapindent}{%
```

After the package caption2 has been loaded, and only if it has been loaded, KOMA-Script redefines its own command `\setcapindent`. The exact code of the redefinition is not important. It should only be noted that caption2 takes control of the `\caption` macro and that therefore the normal definition of the `\setcapindent` macro would become ineffective. The redefinition improves the collaboration with caption2.

There are however also useful examples for normal LaTeX user. Suppose a document that should be available as a PS file, using LaTeX and dvips, as well as a PDF file, using pdfLaTeX. In addition, the document should contain hyperlinks. In the list of tables there are entries longer than one line. This is not a problem for the pdfLaTeX method, since here hyperlinks can be broken across multiple lines. However, if a hyperref driver for dvips or hyperTeX is used then this is not possible. In

this case one desires that for the hyperref setup `linktocpage` is used. The decision which hyperref driver to use happens automatically via `hyperref.cfg`. The file has, for example, the following content:

```
\ProvidesFile{hyperref.cfg}
\@ifundefined{pdfoutput}{\ExecuteOptions{dvips}}
                        {\ExecuteOptions{pdftex}}
\endinput
```

All the rest can now be left to `\AfterFile`.

```
\documentclass{article}
\usepackage{scrlfile}
\AfterFile{hdvips.def}{\hypersetup{linktocpage}}
\AfterFile{hypertex.def}{\hypersetup{linktocpage}}
\usepackage{hyperref}
\begin{document}
\listoffigures
\clearpage
\begin{figure}
  \caption{This is an example for a fairly long figure caption,↩
   but
     which does not employ the optional caption argument that ↩
  would
     allow one to write a short caption in the list of figures.}
\end{figure}
\end{document}
```

If now the hyperref drivers `hypertex` or `dvips` are used, then the useful hyperref option `linktocpage` will be set. In the pdfLaTeX case, the option will not be set, since in that case another hyperref driver, `hpdftex.def`, will be used. That means neither `hdvips.def` nor `hypertex.def` will be loaded.

Furthermore, the loading of package scrlfile and the `\AfterFile` statement can be written in a private `hyperref.cfg`. If you do so, then instead of `\usepackage` the macro `\RequirePackage` ought be used (see [Tea06]). The new lines have to be inserted directly after the `\ProvidesFile` line, thus immediately prior to the execution of the options `dvips` or `pdftex`.

Spare and Replace Files Using scrwfile[1]

TEX supports 18 write handles only. Handle 0 is used by TEX itself (log file). LATEX needs at least handle 1 for `\@mainaux`, handle 2 for `\@partaux`, one handle for `\tableofcontents`, one handle for `\listoffigures`, one handle for `\listoftables`, one handle for `\makeindex`. So there are 11 left. Seams a lot and enough. But every new float, every new index and several other packages, e.g., hyperref need write handles, too. Loading scrwfile minimizes the need of write handles for list of floats or tables of contents. Additionally it allows to clone entries from one file to another file.

10.1. The Idea

Table of contents, list of figures, list of tables and other content directories make use of a small amount of LATEX kernel macros to open helper files, write to those helper files and read them. The first macro is `\@starttoc`. It is used inside of `\tableofcontents`, `\listoffigure`, `\listoftable` and many `\listof` macros of several packages.

The primary macro `\@starttoc` reads the helper file with the contents of the directory. But this kernel macro also calls for a new write handle and even opens the helper file for writing. So every call of this macro consumes one of the rare write handles.

But while the document is processed nobody writes to that handle until `\end{document}`. Every write operation should be done using `\addtocontents` or `\addcontentsline`, that internally uses `\addtocontents`, too. The macro `\addtocontents` does not really write to the helper file, but writes a `\@writefile` macro to the main or part `aux` file.

At `\end{document}` the main `aux` file is closed and after closing LATEX inputs the main `aux` file. While this reading the `\@writefile` macros are processed and only then the helper files are written.

You see, there is no need to hold the helper files open while the document is processed. The helper files need to be opened only before reading the `aux` file at `\end{document}`. Even you do not need one write handle per helper file, if you only could write one after the other. In this case only one write handle would be needed. And that's the idea.

10.2. Using the Package

First of all you have to load the package. This may be simply done using

```
\usepackage{scrwfile}
```

or if you are a package author by using

[1] This chapter has been generated from `scrwfile.dtx`.

```
\RequirePackage{scrwfile}
```
This also activates the *single file feature*.

Note: The package scrwfile may be used with other files, that redefine `\@starttoc`. But if these files do a complete new definition of `\@starttoc` you should load them before scrwfile to avoid errors.

10.2.1. The Single File Feature

To activate the single file feature, that means, that `\starttoc` does not longer consumes write handles and every open and write to helper file action will be done at `\end{document}`, you only need to load package scrwfile.

See section 10.1 for more information about the idea of this feature.

Note: Package scrwfile uses package scrlfile to redefine `\@writefile` while `\end{document}`. Instead of directly writing to the helper files `\@writefile` itself will write to a new helper file `\jobname.wrt`. To write all the helper files this will file be input several times. One time for each helper file.

10.2.2. The Clone File Write Feature

Sometimes it is useful to input a file not only once but several times. As `\@starttoc` does not open files for writing any longer, this may be done by simply using `\starttoc` several times with the same extension. But sometimes you may have additional entries in only some of the content directories. Because of this, scrwfile allows to copy all entries of a file to another file, too. We call this cloning.

`\TOCclone[`*heading*`]{`*source*`}{`*destination*`}`

activates the clone feature for files with extensions *source* and *destination*. All entries to the file `\jobname.`*source* will be added to `\jobname.`*destination*, too.

If extension *destination* is a new one, *destination* will be added to the list of known extensions using the KOMA-Script package tocbasic.

If the optional argument *heading* is given, a new list-of macro `\listof`*destination* is defined. *heading* will be used as section (or chapter) heading of this list. In this case several tocbasic features of the *source* will be copied to *destination*, if and only if they have been set up when `\TOCclone` was used. Feature *nobabel* will always be set, because the language selection commands are part of the helper file and would be cloned, too.

Example: E.g., you want a short table of contents with only the chapter level but an additional entry with the table of contents:

```
\usepackage{scrwfile}
\TOCclone[Short \contentsname]{toc}{stoc}
\AtBeginDocument{%
  % aux first opened at \begin{document}. So wait until this:
  \addtocontents{toc}{% first toc entry:
    \protect\addcontentsline{stoc}{% write to Short Contents
      \protect\tableofcontents% Contents
    }%
  }%
}
\begin{document}
\setcounter{tocdepth}{1}% show chapters only
\listofstoc% Write short table of contents
\setcounter{tocdepth}{6}% show all levels
\tableofcontents% Write table of contents
```

You need at least three LaTeX runs to get a short table of contents and a detailed table of contents. The detailed one produces an entry at the short one but this entry will not be part of the detailed table of contents.

Package tocbasic for Class and Package Authors

Note: This is only a short version of the documentation. The german KOMA-Script guide does contain a long version with usefull examples, that should also be translated!

If a package creates it's list "list of something"—something like "list of figures", "list of tables", "list of listings", "list of algorithms", etc. also known as *toc-files*—have to do some operations, that are equal for all those packages. Also it may be usefull for classes and other packages to know about these additional toc-files. This packages implements some basic functionality for all those packages. Using this package will also improve compatibility with KOMA-Script and—let us hope—other classes and packages.

11.1. Legal Note

You are allowed to destribute this part of KOMA-Script without the main part of KOMA-Script. The files "`scrlogo.dtx`" and "`tocbasic.dtx`" may be distributed together under the conditions of the LaTeX Project Public License, either version 1.3c of this license or (at your option) any later version.

The latest version of this license is in http://www.latex-project.org/lppl.txt and version 1.3c or later is part of all distributions of LaTeX version 2005/12/01 or later.

11.2. Using Package tocbasic

This package was made to be used by class and package authors. Because of this the package has no options. If different packages would load it with different options a option clash would be the result. So using options wouldn't be a good idea.

This package may be used by by other class and package authors. It is also allowed to distribute it without the main part of KOMA-Script (see section 11.1. Because of this, it does not use any KOMA-Script package, that isn't allowed to be distributed without the main part of KOMA-Script, too. Currently tocbasic does only need package keyval from the graphics bundle.

There are two kind of commands. The first kind are basic command. Those are used to inform other packages about the extensions used for files that represent a list of something. Classes or packages may use this information e.g, for putting something to every of those files. Packages may also ask, if an extension is already in use. This does even work, if `\nofiles` was used. The second kind are commands to create the list of something.

11.2.1. Basic Commands

Basic commands are used to handle a list of all extensions known for files representing a list of something. Entries to such files are written using `\addtocontents` or `\addcontentsline` typically. There are also commands to do something for all known extensions. And there are commands to set or unset features of an extension or the file represented by the extension. Typically an extension also has an owner. This owner may be a class or package or a term decided by the author of the class or package using tocbasic, e.g., KOMA-Script uses the owner `float` for list of figures ans list of tables and the default owner for the table of contents.

`\ifattoclist{`*extension*`}{`*true part*`}{`*false part*`}`

This command may be used to ask, wether a *extension* is already a known extension or not. If the *extension* is already known the *true part* will be used, otherwise the *false part* will be used.

Example: Maybe you want to know if the extension "`foo`" is already in use to report an error, if you can not use it:

```
\ifattoclist{foo}{%
  \PackageError{bar}{%
    extension ‘foo’ already in use%
  }{%
    Each extension may be used only
    once.\MessageBreak
    The class or another package already
    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. If the optional argument, `[`*owner*`]`, was given this *owner* will be stored to be the owner of the *extension*. If you omit the optional argument, tocbasic tries to find out the filename of the current processed class or package and stores this as owner. This will fail if `\addtotoclist` was not used, loading a class or package but using a command of a class or package after loading this class or package. In this case the owner will be set to "`.`". Note that an empty *owner* is not the same like omitting the optional argument, but an empty owner.

Example: You want to add the extension "foo" to the list of known extension, while loading your package with file name "bar.sty":

```
\addtotoclist{foo}
```

This will add the extension "foo" with owner "bar.sty" to the list of known extensions, if it was not already at the list of known extensions. If the class or another package already added the extension you will get the error:

```
Package tocbasic Error: file extension ‘#2’ cannot be used ↩
  twice

See the tocbasic package documentation for explanation.
Type  H <return>  for immediate help.
```

and after typing `H <return>` you will get the help:

```
File extension ‘foo’ is already used by a toc-file, while bar.↩
  sty
tried to use it again for a toc-file.
This may be either an incompatibility of packages, an error at ↩
  a package,
or a mistake by the user.
```

Maybe you package has a command, that creates list of files dynamically. In this case you should use the optional argument of `\addtotoclist` to set the owner.

```
\newcommand*{\createnewlistofsomething}[1]{%
  \addtotoclist[bar.sty]{#1}%
  % Do something more to make this list of something available
}
```

If the user calls know, e.g.

```
\createnewlistofsomething{foo}
```

this would add the extension "foo" with the owner "bar.sty" to the list of known extension or report an error, if the extension is already in use. You may use any owner you want. But it should be unique! So, if you are the author of package float you may use for example owner "float" instead of owner "float.sty", so the KOMA-Script options for list of figure and list of table will also handle the

lists of this package, that are already added to the known extensions, when the option is used.

`\AtAddToTocList[`*owner*`]{`*commands*`}`

This command adds the *commands* to a internal list of commands, that should be processed, if a toc-file with the given *owner* will be added to the list of known extensions using `\addtotoclist`. If you omit the optional argument, tocbasic tries to find out the filename of the current processed class or package and stores this as owner. This will fail if `\AtAddToTocList` was not used, loading a class or package but using a command of a class or package after loading this class or package. In this case the owner will be set to "`.`". Note that an empty *owner* is not the same like omitting the optional argument. With an empty *owner* you may add `{`*commands*`}`, that will be processed at every succefull `\addtotoclist`, after processing the commands for the indivdual owner. While processing the `commands`, `\@currext` wil be set to the extension of the currently added extension.

Example: tocbasic itself uses

```
\AtAddToTocList[]{%
  \expandafter\tocbasic@extend@babel
  \expandafter{\@currext}%
}
```

to add every extension to the tocbasic-internal babel handling of toc-files. The `\expandafter` are needed, because the argument of `\tocbasic@extend@babel` has to expanded! See the description of `\tocbasic@extend@babel` at section 11.2.4, page 214 for more information.

`\removefromtoclist[`*owner*`]{`*extension*`}`

This command removes the *extension* from the list of known extensions. If the optional argument, `[`*owner*`]`, was given the *extension* will only be removed, if it was added by this *owner*. If you omit the optional argument, tocbasic tries to find out the filename of the current processed class or package and use this as owner. This will fail if `\removefromtoclist` was not used, loading a class or package but using a command of a class or package after loading this class or package. In this case the owner will be set to "`.`". Note that an empty *owner* is not the same like omitting the optional argument, but removes the *extension* without any owner test.

`\doforeachtocfile[`*owner*`]{`*commands*`}`

This command processes *commands* for every known toc-file of the given *owner*. While processing the *commands* `\@currext` ist the extension of the current toc-file for every known toc-file. If you omit the optional argument, `[`*owner*`]`, every known toc-file will be used. If the optional argument is empty, only toc-files with an empty owner will be used.

Example: If you want to type out all known extensions, you may simply write:

```
\doforeachtocfile{\typeout{\@currext}}
```

and if only the extensions of owner "`foo`" should be typed out:

```
\doforeachtocfile[foo]{\typeout{\@currext}}
```

`\tocbasicautomode`

This command redefines LaTeX kernel macro `\@starttoc` to add all not yet added extensions to the list of known extensions and use `\tocbasic@starttoc` instead of `\@starttoc`.

11.2.2. Creating a List of Something

At the previous section you've seen commands to handle a list of known extensions and to trigger commands while adding a new extension to this list. You've also seen a command to do something for all known extensions or all known extensions of one owner. In this section you will see commands to handle the file corresponding with an extension or the list of known extensions.

`\addtoeachtocfile[`*owner*`]{`*contents*`}`

This command writes *contents* to every known toc-file of *owner*. If you omit the optional argument, *contents* it written to every known toc-file. While writing the contents, `\@currext` is the extension of the currently handled toc-file.

Example: You may add a vertical space of one text line to all toc-files.

```
\addtoeachtocfile{%
  \protect\addvspace{\protect\baselineskip}%
}
```

And if you want to do this, only for the toc-files of owner "`foo`":

```
\addtoeachtocfile[foo]{%
  \protect\addvspace{\protect\baselineskip}%
}
```

`\addcontentslinetoeachtocfile[`*owner*`]{`*level*`}{`*contentsline*`}`

This command is something like `\addcontentsline` not only for one file, but all known toc-files or all known toc-files of a given owner.

Example: You are a class author and want to write the chapter entry not only to the table of contents toc-file but to all toc-files, while `#1` is the title, that should be written to the files.

```
\addcontentslinetoeachtocfile{chapter}{%
  \protect\numberline{\thechapter}#1}
```

`\listoftoc*{`*extension*`}`
`\listoftoc[`*list of title*`]{`*extension*`}`

This commands may be used to set the "list of" of a toc-file. The star version `\listoftoc*` needs only one argument, the extension of the toc-file. It does setup the vertical and horizontal spacing of paragraphs, calls before and after hooks and reads the toc-file. You may use it as direct replacement of the LaTeX kernel macro `\@starttoc`.

The version without star, sets the whole toc-file with title, optional table of contents entry, and running heads. If the optional argument `[`*list of title*`]` was given, it will be used as title term, optional table of contents entry and running head. Note: If the optional argument is empty, this term will be empty, too! If you omit the optional argument, but `\listof`*extension*`name` was defined, that will be used.

Example: You have a new "list of algorithms" with extension `loa` and want to show it.

```
\listof[list of algorithm]{loa}
```

Maybe you want, that the "list of algorithms" will create an entry at the table of contents. You may set

```
\setuptoc{loa}{totoc}
```

But maybe the "list of algorithms" should not be set with a title. So you may use

```
\listof*{loa}
```

Note that in this case no entry at the table of contents will be created, even if you'd used the setup command above.

The default heading new following features using `\setuptoc`:

`totoc` writes the title of the list of to the table of contents

`numbered` uses a numbered headings for the list of

`leveldown` uses not the top level heading (e.g., `\chapter` with book) but the first sub level (e.g., `\section` with book).

\BeforeStartingTOC[*extension*]{*commands*}
\AfterStartingTOC[*extension*]{*commands*}

This commands may be used to process *commands* before or after loading the toc-file with given *extension* using \listoftoc* or \listoftoc. If you omit the optional argument (or set an empty one) the general hooks will be set. The general before hook will be called before the individuel one and the general after hook will be called after the individuel one. While calling the hooks \@currext is the extension of the toc-file and should not be changed.

\BeforeTOCHead[*extension*]{*commands*}
\AfterTOCHead[*extension*]{*commands*}

This commands may be used to process *commands* before or after setting the title of a toc-file with given *extension* using \listoftoc* or \listoftoc. If you omit the optional argument (or set an empty one) the general hooks will be set. The general before hook will be called before the individuel one and the general after hook will be called after the individuel one. While calling the hooks \@currext is the extension of the toc-file and should not be changed.

\listofeachtoc[*owner*]

This command sets all toc-files or all toc-files of the given *owner* using \listoftoc. You should have defined \listof*extension*name for each toc-file, otherwise you'll get a warning.

\MakeMarkcase

This command will be used to change the case of the letters at the running head. The default is, to use \@firstofone for KOMA-Script classes and \MakeUppercase for all other classes. If you are the class author you may define \MakeMarkcase on your own. If scrpage2 or another package, that defines \MakeMarkcase will be used, tocbasci will not overwrite that Definition.

\deftocheading{*extension*}{*definition*}

This command defines a heading command, that will be used instead of the default heading using \listoftoc. The heading command has exactly one argument. You may reference to that argument using #1 at your *defintion*.

\setuptoc{*extension*}{*featurelist*}
\unsettoc{*extension*}{*featurelist*}

This commands set up and unset features binded to an *extension*. The *featurelist* is a comma seperated list of single features. tocbasic does know following features:

totoc writes the title of the list of to the table of contents

numbered uses a numbered headings for the list of

leveldown uses not the top level heading (e.g., `\chapter` with book) but the first sub level (e.g., `\section` with book).

onecolumn switchs to internal one column mode, if the toc is set in internal two column mode and no `leveldown` was used.

nobabel prevents the extension to be added to the babel handling of toc-files. To make this work, you have to set the feature before adding the extension to the list of known extension.

noprotrusion prevents disabling character protrusion at the lists of something. Character protrusion at the lists will be disabled by default if package microtype or another package, that supports `\microtypesetup`, was loaded. So if you want protrusion at the lists, you have to set this feature. But note, that with character protrusion entries at the list may be set wrong. This is a known issue of character protrusion.

Classes and packages may know features, too, e.g, the KOMA-Script classes know following additional features:

chapteratlist activates special code to be put into the list at start of a new chapter. This code may either be vertical space or the heading of the chapter.

`\iftocfeature{`*extension*`}{`*feature*`}{`*true-part*`}{`*false-part*`}`

This command may be used, to test, if a *feature* was set for *extension*. If so the *true-part* will be processed, otherwise the *false-part* will be.

11.2.3. Generating new Lists of and new Floats

There is one simple command to generate either a new list of foo or a new list of a new float.

`\DeclareNewTOC[`*options*`]{`*extension*`}`

This command declares at least a new list of *extension*. Depending on the *options* it may also define a new float. The mandatory argument *extension* is the extension of the help file for that list. See the prior sections for more informations about this. The optional parameter *options* is a comma separated list of *key*=*value* pairs.

atbegin=*code*
: The *code* will be executed at the begin of the float or nonfloat environment.

atend=*code*
: The *code* will be executed at the end of the float or nonfloat environment.

`counterwithin=`*`parent counter`*
: If you define a float or nonfloat, the captions will be numbered. You may declare another counter to be the parent counter. In this case, the parent counter will be set before the float counter and the float counter will be reset whenever the parent counter is increased using `\stepcounter` oder `\refstepcounter`.

`float`
: If set, float environments for that type will be defined. The names of the environments are the value of *`type`* and for double column floats the value of *`type`* with postfix "`*`".

`floatpos=`*`float positions`*
: The default floating-position of the float. If no float position was given, "tbp" will be used like the standard classes do for figures and tables.

`floattype=`*`number`*
: The numericle float type of the defined floats. Float types with common bits cannot be reordered. At the standard classes figures has float type 1 and tables has float type 2. If no float type was given, 16 will be used.

`forcenames`
: If set, the names will be even defined, if they where already defined before.

`hang=`*`length`*
: The amount of the hanging indent of the entries for that list. If not given, 1.5 em will be used like standard classes use for entries to list of figures or list of tables.

`indent=`*`length`*
: The indent value for the entries of that list. If not given, 1 em will be used like standard classes use for entries to list of figures or list of tables.

`level=`*`number`*
: The level of the entries of that list. If not given level 1 will be used like standard classes use for entries to list of figures or list of tables.

`listname=`*`string`*
: The name of the list of foo. If not given the value of `types` with upper case first char prefixed by "List of " will be used.

`name=`*`string`*
: The name of an element. If no name is given, the value of `type` with upper case first char will be used.

`nonfloat`
: If set, a non floating environment will be defined. The name of the environment is the value of *`type`* with postfix "`-`".

owner=*string*
: The owner as described in the sections before. If no owner was given owner `float` will be used.

type=*string*
: sets the type of the new declared list. The type will be used e.g. to defined a `\listof`*string*. If no type is set up the extension from the mandatory argument will be used.

types=*string*
: the plural of the type. If no plural was given the value of `type` with postfix "s" will be used.

11.2.4. Internal Commands for Class and Package Authors

Commands with prefix `\tocbasic@` are internal but class and package authors may use them. But even if you are a class or package author you should not change them!

`\tocbasic@extend@babel{`*extension*`}`

This command extends the babel handling of toc-files. By default babel writes language selections only to `toc`, `lot` and `lof`. tocbasic adds every *extension* added to the list of known extensions (see `\addtotoclist`, section 11.2.1, page 206) using `\tocbasic@extend@babel`. Note: This should be called only once per *extension*. `\tocbasic@extend@babel` does nothing, if the feature `nobabel` was set for *extension* before using `\addtotoclist`.

`\tocbasic@starttoc{`*extension*`}`

This command is something like the LaTeX kernel macro `\@starttoc`, but does some additional settings before using `\@starttoc`. It does set `\parskip` zu zero, `\parindent` to zero, `\parfillskip` to zero plus one fil, `\@currext` to the *extension*, and processes hooks before and after reading the toc-file.

`\tocbasic@@before@hook`
`\tocbasic@@after@hook`

This macros are processed before and after loading a toc-file. If you don't use `\listoftoc` or `\listoftoc*` or `\tocbasic@starttoc` to load the toc-file, you should call these, too. But you should not redefine them!

`\tocbasic@`*extension*`@before@hook`
`\tocbasic@`*extension*`@after@hook`

This macros are processed before and after loading a toc-file. If you don't use `\listoftoc` or `\listoftoc*` or `\tocbasic@starttoc` to load the toc-file, you should call these, too. But you

should not redefine them! The first macro is processed just before `\tocbasic@@before@hook`, the second one just after `\tocbasic@@after@hook`

`\tocbasic@listhead{`*title*`}`

This command is used by `\listoftoc` to set the heading of the list, either the default heading or the indiviually defined heading. If you define your own list command not using `\listoftoc` you may use `\tocbasic@listhead`. In this case you should define `\@currext` to be the extension of the toc-file before using `\tocbasic@listhead`.

`\tocbasic@listhead@`*extension*`{`*title*`}`

This command is used in `\tocbasic@listhead` to set the individual headings, optional toc-entry, and running head, if it was defined. If it was not defined it will be defined and used in `\tocbasic@listhead`.

Japanese Letter Support for scrlttr2[1]

Since version 2.97e scrlttr2 provides support not only for European ISO envelope sizes and window envelopes, but also for Japanese envelopes, in the form of `lco` files which set the layout of the paper. This chapter documents the support, and provides a few examples of using the provided `lco` files for printing letters intended for Japanese envelopes.

A.1. Japanese standard paper and envelope sizes

The Japan Industrial Standard (JIS) defines paper sizes and envelope sizes for national use, which both overlap with the ISO and US sizes and include some metricated traditional Japanese sizes. Envelope window size and position have not been defined internationally as yet; hence, there exists a plethora of envelopes with differing window sizes and positions. The below subsections give some background on Japanese paper sizes and envelopes.

A.1.1. Japanese paper sizes

The JIS defines two main series of paper sizes:

1. the JIS A-series, which is identical to the ISO A-series, but with slightly different tolerances; and

2. the JIS B-series, which is not identical to the ISO/DIN B-series. Instead, the JIS B-series paper has an area 1.5 times that of the corresponding A-series paper, so that the length ratio is approximately 1.22 times the length of the corresponding A-series paper. The aspect ratio of the paper is the same as for A-series paper.

Both JIS A-series and B-series paper is widely available in Japan and most photocopiers and printers are loaded with at least A4 and B4 paper. The ISO/JIS A-series, and the different ISO and JIS B-series sizes are listed in table A.1.

There are also a number of traditional paper sizes, which are now used mostly only by printers. The most common of these old series are the Shiroku-ban and the Kiku paper sizes. The difference of these types compared to the JIS B-series are shown in table A.2. Finally, there are some common stationary sizes, listed in table A.3. You may come across these when buying stationary.

The ISO C-series is not a paper size as such, but is a standard developed for envelopes, intended for the corresponding A-series paper, and is discussed in the next subsection.

[1] This chapter has been written originally by Gernot Hassenpflug.

Table A.1.: ISO and JIS standard paper sizes

ISO/JIS A	W×H in mm	ISO B	W×H in mm	JIS B	W×H in mm
A0	841×1189	B0	1000×1414	B0	1030×1456
A1	594×841	B1	707×1000	B1	728×1030
A2	420×594	B2	500×707	B2	515×728
A3	297×420	B3	353×500	B3	364×515
A4	210×297	B4	250×353	B4	257×364
A5	148×210	B5	176×250	B5	182×257
A6	105×148[1]	B6	125×176	B6	128×182
A7	74×105	B7	88×125	B7	91×128
A8	52×74	B8	62×88	B8	64×91
A9	37×52	B9	44×62	B9	45×64
A10	26×37	B10	31×44	B10	32×45
A11	18×26			B11	22×32
A12	13×18			B12	16×22

[1] Although Japan's official postcard size appears to be A6, it is actually 100×148 mm, 5 millimeters narrower than A6.

A.1.2. Japanese envelope sizes

ISO (International Organization for Standardization) envelope sizes are the official international metric envelope sizes; however, Japan uses also JIS and metricated traditional envelope sizes. Sizes identified as nonstandard do not conform to Universal Postal Union requirements for correspondence envelopes.

Table A.2.: Japanese B-series variants

Format Size	JIS B-series W×H in mm	Shiroku-ban W×H in mm	Kiku W×H in mm
4	257×364	264×379	227×306
5	182×257	189×262	151×227
6	128×182	189×262	
7	91×128	127×188	

Table A.3.: Main Japanese contemporary stationary

Name	W×H in mm	Usage and Comments
Kokusai-ban	216×280	"international size" i. e., US letter size
Semi B5 or Hyoujun-gata	177×250	"standard size" (formerly called "Hyoujun-gata"), semi B5 is almost identical to ISO B5
Oo-gata	177×230	"large size"
Chuu-gata	162×210	"medium size"
Ko-gata	148×210	"small size"
Ippitsu sen	82×185	"note paper"

ISO envelope sizes

The ISO C-series envelope sizes, and possibly B-series envelope sizes, are available in Japan. C-series envelopes can hold the corresponding A-series paper, while B-series envelopes can hold either the corresponding A-series paper or the corresponding C-series envelope. The ISO envelope sizes commonly for Japan are listed in table A.4, with the corresponding paper they are intended for, and the folding required.

JIS and traditional envelope sizes

The JIS classifies envelopes into three categories based on the general shape of the envelope, and where the flap is located:

You: these envelopes are of the 'commercial' type, rectangular, and correspond largely to Western envelope sizes, and also have the flap on the long dimension ('Open Side') in 'commercial' or 'square' style. 'You-kei' means Western-style.

Chou: these are also 'commercial' type envelopes, with the same shape as the corresponding 'You' type, but with the flap on the short dimension ('Open End') in 'wallet' style. 'Chou-kei' means long-style.

Kaku: these envelopes are more square in appearance and are made for special use, and correspond to 'announcement' envelopes. The flap is on the long side, in the 'square' style. They generally do not fall under the ordinary envelope postage rates. 'Kaku-kei' means square-style.

The main JIS and traditional envelope sizes and the corresponding paper and its required folding are listed in table A.5.

Table A.4.: Japanese ISO envelope sizes

Name	W×H in mm	Usage and Comments
C0	917×1297	for flat A0 sheet; nonstandard
C1	648×917	for flat A1 sheet; nonstandard
C2	458×648	for flat A2 sheet, A1 sheet folded in half; nonstandard
C3	324×458	for flat A3 sheet, A2 sheet folded in half; nonstandard
B4	250×353	C4 envelope
C4	229×324	for flat A4 sheet, A3 sheet folded in half; very common; nonstandard
B5	176×250	C5 envelope
C5	162×229	for flat A5 sheet, A4 sheet folded in half; very common; nonstandard
B6	125×176	C6 envelope; A4 folded in quarters; very common
C6	114×162	for A5 sheet folded in half, A4 sheet folded in quarters; very common
C6/C5	114×229	A4 sheet folded in thirds; very common
C7/6	81×162	for A5 sheet folded in thirds; uncommon; nonstandard
C7	81×114	for A5 sheet folded in quarters; uncommon; nonstandard
C8	57×81	
C9	40×57	
C10	28×40	
DL[1]	110×220	for A4 sheet folded in thirds, A5 sheet folded in half lengthwise; very common

[1] Although DL is not part of the ISO C-series, it is a very widely used standard size. DL, probably at one time the abbreviation of DIN Lang (Deutsche Industrie Norm, long), is now identified as "Dimension Lengthwise" by ISO 269.

Table A.5.: Japanese JIS and other envelope sizes

JIS	Name	W× in mm	Usage and Comments
	Chou 1	142×332	for A4 folded in half lengthwise; nonstandard
Yes	Chou 2	119×277	for B5 folded in half lengthwise; nonstandard
Yes	Chou 3	120×235	for A4 folded in thirds; very common
	Chou 31	105×235	for A4 folded in thirds
	Chou 30	92×235	for A4 folded in fourths[3]
	Chou 40	90×225	for A4 folded in fourths[3]
Yes	Chou 4	90×205	for JIS B5 folded in fourths[3]; very common
	Kaku A3	320×440	for A3 flat, A2 folded in half ; nonstandard
	Kaku 0	287×382	for B4 flat, B3 folded in half; nonstandard
	Kaku 1	270×382	for B4 flat, B3 folded in half; nonstandard
Yes	Kaku 2	240×332	for A4 flat, A3 folded in half; nonstandard
	Kaku Kokusai A4	229×324	for A4 flat, A3 folded in half; same size as ISO C4; nonstandard
Yes	Kaku 3	216×277	for B5 flat, B4 folded in half; nonstandard
Yes	Kaku 4	197×267	for B5 flat, B4 folded in half; nonstandard
Yes	Kaku 5	190×240	for A5 flat, A4 folded in half ; nonstandard
Yes	Kaku 6	162×229	for A5 flat, A4 folded in half; same size as ISO C5; nonstandard
Yes	Kaku 7	142×205	for B6 flat, B5 folded in half; nonstandard
Yes	Kaku 8	119×197	pay envelope (for salaries, wages) ; common for direct mail

...

Table A.5.: Japanese JIS and other envelope sizes (*continued*)

JIS	Name	W× in mm	Usage and Comments
Yes	You 0[1] or Furusu 10	235×120	for A4 folded in thirds; same size as Chou 3 but with 'Open Side' style flap
	You 0[1]	197×136	for kyabine[1] (cabinet) size photos (165 mm×120 mm); nonstandard
	You 1[2]	176×120	for B5 folded in quarters
	You 1[2]	173×118	for B5 folded in quarters
Yes	You 2	162×114	for A5 folded in half, A4 folded in quarters; same size as ISO C6
Yes	You 3	148×98	for B6 folded in half
Yes	You 4	235×105	for A4 folded in thirds
Yes	You 5	217×95	for A4 folded in fourths[3]
Yes	You 6	190×98	for B5 folded in thirds
Yes	You 7	165×92	for A4 folded in quarters, B4 folded in quarters

[1] Because two different sizes are called You 0, the JIS You 0 is normally called Furusu 10; Furusu ('fools') derives from 'foolscap'; Kyabine is a metricated traditional Japanese size.
[2] Two slightly different sizes are sold as You 1; the smaller size (173 mm×118 mm) is the paper-industry standard size.
[3] Twice in the same direction.

Window variants

There are a large number of window subtypes existing within the framework explained in the previous subsection. The most common window sizes and locations are listed in table A.6.

A.2. Provided `lco` files

In scrlttr2 support is provided for Japanese envelope and window sizes through a number of `lco` files which customize the foldmarks required for different envelope sizes and subvariants with different window positions and sizes.

The provided `lco` files together with the envelope types for which they provide support are listed at table A.7. See table A.4 for the full list of Japanese envelopes and the paper they take, and table A.6 for the common window sizes and locations. The rightmost column indicates which `lco` file provides the support.

Table A.6.: Supported Japanese envelope types and the window sizes and locations.

Envelope type	Window name[1]	- size[2]	- location[3]	`lco` file[4]
Chou 3	A	90×45	l 23, t 13	`NipponEL`
Chou 3	F	90×55	l 23, t 13	`NipponEH`
Chou 3	Hisago	90×45	l 23, t 12	`NipponEL`
Chou 3	Mutoh 1	90×45	l 20, t 11	`NipponEL`
Chou 3	Mutoh 101	90×55	l 20, t 11	`NipponEH`
Chou 3	Mutoh 2	80×45	l 20, t 11	`NipponEL`
Chou 3	Mutoh 3	90×45	l 25, t 11	`NipponLL`
Chou 3	Mutoh 301	90×55	l 25, t 11	`NipponLH`
Chou 3	Mutoh 6	100×45	l 20, t 11	`NipponEL`
Chou 3	v.2[5]	90×45	l 24, t 12	`NipponLL`
Chou 40	A	90×45	l 23, t 13	`NipponEL`
Chou 4	A	90×45	l 23, t 13	`NipponEL`
Chou 4	B	80×45	l 98, t 28	`NipponRL`
Chou 4	C	80×45	l 21, t 13	`NipponEL`
Chou 4	K	80×45	l 22, t 13	`NipponEL`
Chou 4	Mutoh 1	80×45	l 40, b 11	—
Chou 4	Mutoh 2	80×45	l 20, t 11	`NipponEL`
Chou 4	Mutoh 3	90×45	l 20, t 11	`NipponEL`
Chou 4	Mutoh 6	100×45	l 20, t 11	`NipponEL`
Chou 4	v.2[5]	80×45	l 20, t 12	`NipponEL`
Chou 4	v.3[5]	90×45	l 20, t 12	`NipponEL`
Kaku A4	v.1[6]	95×45	l 20, t 24	`KakuLL`
You 0	Cruise 6	90×45	l 20, t 12	`NipponEL`
You 0	Cruise 601	90×55	l 20, t 12	`NipponEH`
You 0	Cruise 7	90×45	l 20, b 12	`NipponEL`
You 0	Cruise 8	90×45	l 24, t 12	`NipponLL`
You 0	v.2[5]	90×45	l 24, t 12	`NipponEL`
You 0	v.3[5]	90×45	l 23, t 13	`NipponEL`
You 4	A	90×45	l 23, t 13	`NipponEL`

[1] Names (acting as subtype information) are taken from the manufacturer catalog.
[2] Given as width by height in millimeters.
[3] Given as offset from left (l) or right (r), followed by offset from bottom (b) or top (t).
[4] The `lco` file, which provides support (see table A.7).
[5] In the absence of any other information, a numerical variation number for the subtype name is provided.
[6] Dimensions apply when envelope is held in portrait mode.

Table A.7.: `lco` files provided by scrlttr2 for Japanese window envelopes

`lco` file	Supported	Window size[1]	Window location[1]
`NipponEL`	Chou/You 3 and 4	90×45	l 22, t 12
`NipponEH`	Chou/You 3 and 4	90×55	l 22, t 12
`NipponLL`	Chou/You 3 and 4	90×45	l 25, t 12
`NipponLH`	Chou/You 3 and 4	90×55	l 25, t 12
`NipponRL`	Chou/You 3 and 4	90×45	l 98, t 28
`KakuLL`	Kaku A4	90×45	l 25, t 24

[1] Window size is given in width by height, location as offset from left (l) or right (r), followed by offset from bottom (b) or top (t). All Values in millimeters.

The tolerances for location is about 2 mm, so it is possible to accommodate all the envelope and window variants of table A.6 with just a small number of `lco` files. The difference between Chou/You 3 and Chou/You 4 is determined by paper size.

A.3. Examples of Japanese letter usage

Assume you want to write a letter on A4 size paper and will post it in a Japanese envelope. If the envelope has no window, then it is enough to determine whether the envelope dimensions match a European one — the standard `DIN.lco` style may suffice for many such cases.

If you wish to use a windowed envelope, please note that owing to the large variety, not all existing subvariants are currently supported. If you should note that you particular windowed envelope has its window dimensions and positions significantly (more than approximately 2 mm) different from any of the supported subvariants, please contact the author of KOMA-Script to obtain support as soon as possible, and in the meanwhile create a customized `lco` file for your own use, using one of the existing ones as a template and reading the KOMA-Script documentation attentively.

If your window envelope subvariant is supported, this is how you would go about using it: simply select the required `lco` file and activate the horizontal and vertical foldmarks as required. Another, independent, mark is the punching mark which divides a sheet in two horizontally for easy punching and filing.

A.3.1. Example 1:

Your favourite envelope happens to be a You 3 with window subvariant Mutoh 3, left over from when the company had its previous name, and you do not wish them to go to waste. Thus, you write your letter with the following starting code placed before the letter environment:

```
\LoadLetterOption{NipponLL}\setkomavar{myref}{NipponLL}
```

```
\begin{letter}{Martina Muster\\Address}
...
\end{letter}
```

A.3.2. Example 2:

You originally designed your letter for a You 3 envelope, but suddenly you get handed a used electrical company envelope with cute manga characters on it which you simply cannot pass up. Surprisingly, you find it conforms fairly closely to the Chou 4 size and C window subvariant, such that you realize you can alter the following in your document preamble:

```
\LoadLetterOption{NipponEL}\setkomavar{myref}{NipponEL}
\begin{letter}{Martina Muster\\Address}
...
\end{letter}
```

Now, scrlttr2 automatically reformats the letter for you to fit the required envelope.

Bibliography

In the following you can find many references. All of them are referenced from the main text. In many cases the reference points to documents or directories which can be accessed via Internet. In these cases the reference includes a URL instead of a publisher. If the reference points to a LaTeX package then the URL is written in the form "CTAN://*destination*". The prefix "CTAN://" means the TeX archive on a CTAN server or mirror. For example, you can substitute the prefix with ftp://ftp.ctan.org/tex-archive/. For LaTeX packages it is also important to mention that we have tried to give a version number appropriate to the text that cites the reference. But for some packages is is very difficult to find a consistent version number and release date. Additionally the given version is not always the current version. If you want install new packages take care that the package is the most up-to-date version and check first whether the package is already available on your system or not.

[Bra01] Johannes Braams:
Babel, a multilangual package for use with LaTeX's standard document classes, Februar 2001.
CTAN://macros/latex/required/babel/.

[Car99a] David Carlisle:
The ifthen package, September 1999.
CTAN://macros/latex/base/.

[Car99b] David P. Carlisle:
Packages in the 'graphics' bundle, Februar 1999.
CTAN://macros/latex/required/graphics/.

[Car04] David Carlisle:
The longtable package, Februar 2004.
CTAN://macros/latex/required/tools/.

[Dal99] Patrick W. Daly:
Natural sciences citations and references, Mai 1999.
CTAN://macros/latex/contrib/natbib/.

[DUD96] DUDEN:
Die deutsche Rechtschreibung. DUDENVERLAG, Mannheim, 21. Auflage, 1996.

[Fai99] Robin Fairbairns:
topcapt.sty, März 1999.
CTAN://macros/latex/contrib/misc/topcapt.sty.

[FAQ11] *Tex frequently asked questions on the web*, April 2011.
http://www.tex.ac.uk/faq/.

[Gau03] Bernard Gaulle:
Les distributions de fichiers de francisation pour latex, Dezember 2003.
CTAN://language/french/.

[KDP] *KOMA-Script Homepage*.
http://www.komascript.de.

[Kie99] Axel Kielhorn:
adrconv, November 1999.
CTAN://macros/latex/contrib/adrconv/.

[Kil99] James Kilfiger:
extsizes, a non standard LaTeX-package, November 1999.
CTAN://macros/latex/contrib/extsizes/.

[Lin01] Anselm Lingnau:
An improved environment for floats, Juli 2001.
CTAN://macros/latex/contrib/float/.

[OPHS99] Tobias Oetker, Hubert Partl, Irene Hyna und Elisabeth Schlegl:
The Not So Short Introduction to LaTeX 2ε, April 1999.
CTAN://info/lshort/.

[Pac] Jean Marie Pacquet:
KomaLetter2; Example by Jean-Marie Pacquet (French style). Wiki.
http://wiki.lyx.org/Examples/KomaLetter2#toc6.

[Rah01] Sebastian Rahtz:
Hypertext marks in LaTeX: the hyperref package, Februar 2001.
CTAN://macros/latex/contrib/hyperref/.

[Rai98a] Bernd Raichle:
german package, Juli 1998.
CTAN://language/german/.

[Rai98b] Bernd Raichle:
ngerman package, Juli 1998.
CTAN://language/german/.

[Sch09] Martin Schröder:
The ragged2e package, Juni 2009.
CTAN://macros/latex/contrib/ms/.

[Som08] Axel Sommerfeldt:
Anpassen der Abbildungs- und Tabellenbeschriftungen mit Hilfe des caption-Paketes, April 2008.
CTAN://macros/latex/contrib/caption/.

[SU03] Tom Sgouros und Stefan Ulrich:
The mparhack package, Mai 2003.
CTAN://macros/latex/contrib/mparhack/.

[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, Februar 2006.
CTAN://macros/latex/doc/clsguide.pdf.

[Tob00] Geoffrey Tobin:
setspace LaTeX package, Dezember 2000.
CTAN://macros/latex/contrib/setspace/.

[Tsc87] Jan Tschichold:
Ausgewählte Aufsätze über Fragen der Gestalt des Buches und der Typographie.
Birkhäuser Verlag, Basel, 2. Auflage, 1987.

[Ume00] Hideo Umeki:
The geometry package, Juni 2000.
CTAN://macros/latex/contrib/geometry/.

[vO00] Piet van Oostrum:
Page layout in LaTeX, Oktober 2000.
CTAN://macros/latex/contrib/fancyhdr/.

[WF00] Hans Peter Willberg und Friedrich Forssman:
Erste Hilfe in Typografie. Verlag Hermann Schmidt, Mainz, 2000.

[Wik] Wiki:
Deutsche TeX-FAQ.
http://projekte.dante.de/DanteFAQ/WebHome.

Index

There are two kinds of page numbers at this index. The bold printed numbers show the pages of declaration or explanation of the topic. The normal printed numbers show the pages of using a topic.

General Index

Index of Commands, Environments, and Variables

B

C

K

L

M

N

O

P

Q

R

S

Index of Lengths and Counters

Index of Elements with Capability of Font Adjustment

Index of Files, Classes, and Packages

Index of Class and Package Options

R

S

T

V