Manual

The KOMA-Script bundle

Frank Neukam Markus Kohm Axel Kielhorn

2004-01-07

Authors of the manual:

Markus Kohm Jens-Uwe Morawski

No warranty:
There is no warranty for KOMA-Script or any part of it. The authors provide KOMA-Script *as is*, without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purposs. The entire risk as to the quality and performance of KOMA-Script is with you. Should KOMA-Script or any part of it prove defective, you assume the cost of all necessary servicing, repair or correction.
In no event unless agreed to in writing will the authors or any other party who may distribute and/or modify KOMA-Script as permitted in `LEGAL.TXT`, be liable to you for damages, including any general, special, incidental or consequential damages arising out of any use of KOMA-Script or out of inability to use KOMA-Script (including, but not limited to, loss of data, data being rendered inaccurate, or losses sustained by anyone as a result of any failure of KOMA-Script to operate with any other programs), even if a author or said other party has been advised of the possibility of such damages.
See `LEGAL.TXT` for more information. `LEGAL.TXT` is part of KOMA-Script and must not be removed from any KOMA-Script distribution.

KOMA-Script

ein wandelbares LaTeX 2$_\varepsilon$-Paket

Following people took part on KOMA-Script: **Stefan Brill** (proof-read), **Dr. Engelbert Buxbaum** (translation), **Christian Buss** (proof-read), **João Canas Ferreira** (translation), **Michael Dewey** (language adaptation), **Luzia Dietsche** (beta test), **Georg Grandke** (translation, proof-read), **Ralph J. Hangleiter** (language adaptation), **Henk Jongbloets** (language adaptation), **Axel Kielhorn** (scrlettr, addrconv, ideas), **Markus Kohm** (development, documentation, translation, support, release), **Torsten Krüger** (beta test, proof-read, necessary questions, ideas, motor), **Enrico Kunz** (documentation, proof-read), **Werner Lemberg** (translation), **Branka Lončarević** (language adaptation), **Alejandro López-Valencia** (language adaptation), **Colin Marquardt** (translation), **Peter Marx** (release), **Jens-Uwe Morawski** (documentation, translation, coordination, proof-read, ideas, motor), **Simone Naldi** (language adaptation), **Frank Neukam** (Script and Script2.0), **Thomas Neumann** (documentation, proof-read), **Rolf Niepraschk** (proof-read), **Carsten Schurig** (beta test), **Axel Sommerfeldt** (documentation) and a lot of friendly peoply, sending bug reports, nice emails, ideas and sometimes even implementations.

Contents

List of Tables

1 Introduction

1.1 Preface

KOMA-Script is a very complex bundle. You may see this, because it is not only one class or one package but a bundle of many classes and packages. The classes are counterparts to the standard classes (see chapter 3) but never they come with only the same commands, environments, options and optional possibilities like the standard classes nor they result in the same look-a-like. More about the history of KOMA-Script you will find at section 1.3.

KOMA-Script comes with a lot of classes, packages, commands, environments and possibilities. Some of these you may find also at the standard classes, many of them you wouldn't. Some are even supplements to the LaTeX kernel.

Because of all this KOMA-Script manual has to be large. Also there's no school for KOMA-Script. So there's no teacher knowing everyone of his students. Such a teacher should be able to change the lessons dependent on the kind of students he has to teach. It would be easy to write a manual which is good for one kind of a reader. But it is nearly impossible to write a manual which is good for every reader. Never the less the authors tried to do so. The result must be a compromise. Please remebers this, if you don't like the manual or parts of it, before you'll write a report about. We hope the compromise is a good one. Never the less, if you have solutions of improvements write us.

If you habe any problem with KOMA-Script please read the manual despite the size of it. Apart from the guide this means also all the other text files of the bundle. You will find a list of these as `readme.txt`.

1.2 Structure of the Manual

There's only less information for LaTeX beginners at the guide. Beginners should first read documents like [OPHS99] or [Tea99a]. It would also be good to read a book about LaTeX. You'll find literature at almost every faq like

[UK:02]. You should also have a look to these faqs, if you have a problem with LaTeX.

Informations about understanding LaTeX or KOMA-Script are written like this paragraph. You do not need this informations to use KOMA-Script, but if you have problems in using KOMA-Scriptyou should read this parts too. So read them before writing a bug report.

Some of you do not want to read the whole guide but only informations about one class or package. This should be possible, because you will find almost all informations about one package or one class at one chapter. Only the three main classes are described together at one chapter, because most informations about are valid for all of them. If something is only valid for one or two of the three you will find a note at the margin. If for example something is only valid for scrartcl this is marked at the margin like here.

scrartcl

1.3 History of KOMA-Script

At the early 90th Frank Neukam has to write a student script. At that time LaTeX2.09 was LaTeX. At LaTeX2.09 classes and packages has been all the same and where called *styles*. Frank thought the standard document styles not to be good enough for his work. He needed more and other commands and environments.

At the same time Frank was very interested in typography. After reading [Tsc87] he decided to write his own document style not only to write the one script but also to have a style family especialy for european and german typography. Script was born.

I, Markus Kohm, found Script at december 1992. While Frank used A4 paper format, I often had to use A5. At 1992/1993 the standard styles and Script had no options for A5 paper. So I've changed Script. These and other changes where also found at Script-2 released by Frank at december 1993.

At the middle of 1994 LaTeX 2ε could be found at CTAN. Most users of Script-2 not only wanted to use these styles at compatibility mode of LaTeX 2ε but have a native LaTeX 2ε Script — me too. So I have made new LaTeX 2ε classes a first package and released all together named KOMA-Script at july 7th 1994. Some month later Frank declared KOMA-Script to be the new Script. At this time there was no letter class at KOMA-Script. Axel Kielhorn wrote it and so since december 1994 it is part of KOMA-Script. Some time later Axel Sommerfeldt has written the first real scrguide. The old english scrguide was made by Werner Lemberg.

A lot of time is gone. There where minor changes at LaTeX but a lot of changes at the surroundings. There are many new packages and classes. KOMA-Script isn't the KOMA-Script from 1994, it's much more. While it was first made to have good classes for german authors, now it made to have classes with more flexibility than the standard classes. I've got a lot email from all over the world. Because of this I've written lots

of new macros. They all needs documentation. Some day we needed simply a new scrguide.

This new scrguide is not only made for A5 paper. Because of the history the implementation is german. The primary documentation is in german too. Maybe this isn't easy to understand. But see: Almost everything around LaTeX is english. KOMA-Script is german. But don't worry, the commands, environments, options etc. are english. Sometimes my english was not good enough to find the correct english name, never the less I tried. As result you'll find options called `pointlessnumbers` which means only "numbers without dot at the end" and should be called e.g. `dotlessnumbers`.

1.4 Thanks

In fact the thanks-giving is part of the addendum, but my thanks are not primarily for the authors of this guide. These thanks should be given by the readers!

I give my personal thanks to Frank Neukam. Without his SCRIPT family KOMA-Script wouldn't be.

I am indebted to the persons who have contributed to KOMA-Script in many different fashions. Vicariously for all those contributers I would like mention Jens-Uwe Morawski and Torsten Krüger. The english translation of the guide is , besides many other things, obliged to Jens' untiring engagement. Torsten was the best beta-tester I ever had. Especially his work has enhanced the usabillity of scrlttr2 und scrpage2.

Many thanks to all who have encouraged me to go on, make things better, less error-prone or to implement some additional features.

Thanks to DANTE, Deutschsprachige Anwendervereinigung TeX e.V, without the DANTE server, KOMA-Script couldn't be released and distributed. Thanks to everybody at the TeX news groups and mailing lists, who answer questions and help me to support KOMA-Script.

1.5 Legal Notes

KOMA-Script was released under LaTeX Project Public Licence. You will find it at the file `LEGAL.TXT`. Germans will also find a inofficial translation at `LEGALDE.TXT`. This german file is valid at every country with german being an official language.

There's no warranty for any documentation or any other part of the KOMA-Script bundle.

1.6 Installation

You'll find the documentation about installation at the files `readme.txt` and `INSTALL.TXT`. Read also the documentation of the TEX distribution you're using.

1.7 Bugreports

If you think, you've found a bug at the documentation, 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 at CTAN to see, if there is a new version of the part with the bug. In this case install the new part and try it again.

If you've found an error not at the documentation or a bug, which is still at the updated file, please write a short example LATEX document to show the problem. At this example you should only use these packages and definitions needed to demonstrate the problem. Don't use unusual packages.

Writing this example you will often find out, if the problem is a bug at KOMA-Script or not. Report only bugs at KOMA-Script to the author of KOMA-Script. Please use `komabug.tex` for the generation of the bug report. `komabug.tex` is an interactive LATEX document.

If you want to ask your question at a news group or mainling list, you should also write such a example as part of your question. But you need not use `komabug.tex` in this case. To get the versions of all used packages, simply put `\listfiles` in the preamble of your example and read the end of the `log`-file.

1.8 Authors

Every chapter will be finished with a section called "authors". You will find there the authors of the original chapter of the german scrguide. The main author is written using a bold font. The translator is shown using an italic font. If there is an email address, please send reports about this chapter of

the documentation to this address. If no translator is given, the authors are the translators.

- **Markus Kohm** <Markus.Kohm@gmx.de>

2 Construction of the Page Layout with typearea

2.1 Fundamentals of Page Layout

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

The literature offers and discusses different algorithms and heuristic approaches for constructing a good page layout. Often, they are mentioning an approach which involves diagonals and their intersections. The result is a page where the text block proportions relate 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 inner margin (the margin at the spine) should be the same as each of the two outer margins.

In the previous paragraph, we mentioned and emphasized the *page*. Erroneously, often it is thought that the format of the page would be the format of the paper. However, if you look at a bound document, it is obvious that part of the paper vanishes in the binding and is not part of the visible page. But the format of the paper is not important for the layout of a page, it is the impression of the visible page to the reader. Therefore, it is clear that the calculation if 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 *binding correction*.

The binding correction depends on the process of actually producing the document and thus can not 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 relation of the individual parts of a page. However, we do not know about the width and the height of the text block yet. 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.

$$\begin{aligned} \textit{textblock heigth} : \textit{textblock width} &= \textit{page height} : \textit{page width} \\ \textit{page width} &= \textit{paper width} - \textit{binding correction} \\ \textit{top margin} + \textit{foot margin} &= \textit{page height} - \textit{textblock height} \end{aligned}$$

$$\begin{aligned} \textit{top margin} : \textit{foot margin} &= 1:2 \\ \textit{left margin} : \textit{right margin} &= 1:1 \\ \textit{half inner margin} &= \frac{1}{2}\textit{outer margin} + \textit{binding correction} \end{aligned}$$

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

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

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

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

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

Literature gives different values for good line lengths, depending on the author. To some extent, this is due to the native language of the author. Since the eye jumps from word to word, short words make this task easier. Not considering language and font, a line length of 60 to 70 letters including spaces and punctuation is a usable compromise. This requires well-chosen line spacing, but LaTeX's default is usually good enough.

Before looking at the actual construction of the page layout, there are some minor things to know. LaTeX doesn't start the first line of the text block at the upper edge of the text block, but with a defined distance. Also, LaTeX knows the commands `\raggedbottom` and `\flushbottom`. `\raggedbottom` specifies that the last line of a page should be positioned wherever it was calculated. This means that the position of this line can be different on each page, up to the height of one line. In double-sided documents this is usually unwanted. `\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 not to stretch the paragraph skip on normal pages where it is the only stretchable glue, the height of the text block should be a multiple of the height of the text line, including the distance from the upper edge of the text block to the first line.

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

2.2 Page layout construction by Dividing

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

In KOMA-Script, this kind of construction is implemented in the typearea package. For A4 paper, *DIV* is predefined according to the font size (see Table 2.2). If there is no binding correction ($BCOR = 0\,\mathrm{pt}$), the results roughly match the values of Table 2.1.

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

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

2.3 Page layout construction by drawing a circle

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

A disadvantage of this late dark age method is that the width of the text area is not dependent on the font anymore. 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 senseless) *DIV* value or a special package option, a *DIV* value is chosen to match the perfect values as closely as possible. See also section 2.4.

2.4 Options and macros to influence the page layout

The package typearea offers two different user interfaces to influence type area construction. The first method is to load the package with options. For information on how to load packages and to give package options, please refer to the LaTeX literature, e.g. [OPHS99] and [Tea99a], 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).

BCOR*Correction*

With the BCOR*Correction* option you specify the absolute value of the binding correction, i.e., the width of the area that is used for the binding, thus "lost" from the paper width.

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

example: Assume you want to produce a financial report, which is to be printed on A4 paper and bound in a folder. The rim of the folder covers 7,5 mm. Since the report is thin, only an additional 0,75 mm are lost by folding when leafing through the pages. You would use the following commands:

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

or, using a KOMA-Script-class:

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

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

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

Please note: if you use one of the KOMA-Script classes, this option must be given as a class option. If you use another class, this only works if the class has explicit support for typearea. So when using the standard classes, you need to give the option when you load typearea. You can also use `\PassOptionsToPackage` (see [Tea99b]) before you are loading typearea, this always works.

`DIV`*Factor*

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

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

example: Imagine you are writing meeting minutes with the protocol[1]-class. The whole thing is supposed to be double sided. In your company, the Bookman font in 12 pt is used. This standard PostScript font is activated in LaTeX with the command `\usepackage{`*bookman*`}`. Bookman runs very wide, that means, the characters are wide in relation to its height. Because of that, the default for the *DIV* value in typearea is too small for you. Instead of 12, you want 15. The minutes will not be bound but punched and filed into a folder, so you don't need any binding correction. Thus, you write:

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

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

```
\documentclass[a4paper,twoside]{iso9000p}
\usepackage{bookman}
\usepackage[DIV15,BCOR12mm]{typearea}
```

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

```
\documentclass[a4paper,twoside,DIV15,BCOR12mm]{scrartcl}
\usepackage{bookman}
```

Please note: if you use one of the KOMA-Script classes, `BCOR` must be given as a class option. If you use another class, this only works if the class has explicit support for typearea. So when using the standard classes, you need to give `BCOR` when you load typearea. You can use `\PassOptionsToPackage` (see [Tea99b]) too before you are loading typearea, this always works.

`DIVcalc`
`DIVclassic`

As mentioned in section 2.2, only paper format A4 has fixed defaults for the *DIV* value. These are listed in Table 2.2. If you choose a different paper

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

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

Table 2.2: *DIV* defaults for A4

format, typearea calculates a good *DIV* value itself. Of course, you can also have it calculate that for A4: use `DIVcalc` instead of `DIV`*Factor*. This works for all other paper formats as well. If you want to use the automatic calculation, this is even very useful, since you can then override the defaults that are given in a configuration file (see section 2.7) with this option.

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

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

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

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

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

example: Let us assume again that we want to calculate a good page layout for the Bookman font. We also want to use a KOMA-Script class.

This is possible using the \typearea-command with DIVcalc = 1 as *DIV*-parameter:

```
\documentclass[a4paper,BCOR12mm,DIVcalc,twoside]{scrartcl}
\usepackage{bookman}
\typearea[12mm]{1}% same as class options above
```

It would be ridiculous if one had to use the \typearea-command with some pseudo-values, while the DIV-Option allows the use of DIVcalc and DIVclassic. Thus the \typearea also accepts symbolic values for the parameter *DIV*:

`calc` – re-calculate page layout and determine *DIV*.

`classic` – re-calculate page layout using the classical method (circle) .

`current` – re-calculate page layout with current value of *DIV*.

`default` – re-calculate page layout with default values for the current page- and font size. If no default values exist, apply `calc`.

`last` – re-calculate page layout using the same *DIV*-argument, which was set last time.

The \typearea also understands the following symbolic values for the parameter *BCOR*. Thus it is not neccesary to re-enter the current value.

`current` – Re-calculate page layout using the current value for *BCOR*.

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

```
\documentclass[a4paper,BCOR12mm,DIVcalc,twoside]{scrartcl}
\usepackage{bookman}
\typearea[current]{calc}
```

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

```
\documentclass[a4paper,BCOR12mm,DIV11,twoside]{scrartcl}
\usepackage{bookman}
\typearea[current]{current}
```

or the old method:

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

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

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

example: Assume you want to write a thesis and university regulations require a font size of 10 pt and a line spacing of 2 pt. Thus a stretch-factor of 1.25 is required. Let us also assume that $BCOR = 12\,\text{mm}$. Then you might use:

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

The command `\selectfont` is required here in order to activate the new line spacing before the page layout is calculated.

The same example again, using the setspace-package:

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

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

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

```
\documentclass[10pt,BCOR12mm,DIVcalc,twoside]{scrreprt}
\usepackage{setspace}
\onehalfspacing
\typearea[current]{calc}
\begin{document}
\title{Title}
\author{Markus Kohm}
\begin{spacing}{1}
  \maketitle
  \tableofcontents
\end{spacing}
\chapter{Ok}
\end{document}
```

See also the notes in section 2.8.

The command \typearea is currently defined in such a way that it is possible to change the page layout in the middle of a text. This however makes assumptions about the inner workings of the LaTeX-kernel and changes some internal values and definitions of that kernel. There is some probability, but no guarantee that this will also work in future versions of LaTeX. It must be assumed that this method will not give correct results in LaTeX3. However, as author of KOMA-Script I expect considerable incompatibilities when we change to LaTeX3.

```
headinclude
headexclude
footinclude
footexclude
```

So far we have discussed how the page layout is calculated and what the ratios are between the borders and between borders and text area. However, one important question has not been answered: What constitutes the borders? This question appears trivial: Borders are those parts on the right, left, top and bottom which remain empty. But this is only half of it. Borders are not always empty. There could be marginals, for example (for the \marginpar-command refer to [SKPH99] or subsection 3.6.5).

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

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

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

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

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

`mpinclude`
`mpexclude`

v2.8q

Besides documents where the head and foot is part of the text area, there are also documents where the margin-note area must be counted to the text body as well. The option `mpinclude` does exactly this. The effect is that one width-unit of the text-body is taken for the margin-note area. Using option `mpexclude`, the default setting, then the normal margin is used for the margin-note area. The width of that area is one or one and a half width-unit, depending on whether one-side or two-side page layout has been chosen. The option `mpinclude` is mainly for experts and so not recommended.

In the cases where the option `mpinclude` is used often a wider margin-note area is

required. In many cases not the whole margin-note width should be part of the text area, for example if the margin is used for quotations. Such quotations are typeset as ragged text with the flushed side where the text body is. Since ragged text gives no homogeneous optical impression the long lines can reach right into the normal margin. This can be done using option `mpinclude` and by an enlargement of length `\marginparwidth` after the type-area has been setup. The length can be easily enlarged with the command `\addtolength`. How much the the length has to be enlarged depends on the special situation and it requires some flair. Therefore the option is primarily for experts. Of course one can setup the margin-width to reach a third right into the normal margin, for example using `\setlength{\marginparwidth}{1.5\marginparwidth}` gives the desired result.

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

Value`headlines`

We have seen how to calculate the text layout and how to specify whether header and footer are part of the text body or the borders. However, we still have to specify the height in particular of the header. This is achieved with the option `headlines`, which is preceded by the number of lines in the header. typearea uses a default of 1.25. This is a compromise, large enough for underlined headers (see section 3.1) and small enough that the relative weight of the top border is not affected to 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 2-line header. Normally this would result in a "`overfull \vbox`" warning for each page. To prevent this from hapening, the typearea-package is told to calculate an appropriate page layout:

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

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

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

A tool that can be used to define the contents of a header with 2 lines is described in chapter 4.

If you use a KOMA-Script-class, this option must be given as class-option. With other classes this works only, if these classes explicitly supports typearea. If you use the standard classes, the option must be given when loading typearea. `\PassOptionsToPackage` will work in both cases (see also [Tea99b]).

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

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

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

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

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

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

2.5 Options and macros for paper format selection

The LaTeXstandard classes support the options `a4paper`, `a5paper`, `b5paper`, `letterpaper`, `legalpaper` and `executivepaper`.

`letterpaper`
`legalpaper`
`executivepaper`
`a`*X*`paper`
`b`*X*`paper`
`c`*X*`paper`
`d`*X*`paper`
`landscape`
`\isopaper[`*series*`]{`*number*`}`

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

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

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

```
\documentclass{article}
\usepackage[headexclude,footexclude,
            a8paper,landscape]{typearea}
\areaset{7cm}{5cm}
\pagestyle{empty}
\begin{document}
```

```
\section*{Paper-size Options}
letterpaper, legalpaper, executivepaper, a0paper,
a1paper \dots\ b0paper, b1paper \dots\ c0paper,
c1paper \dots\ d0paper, d1paper \dots
\end{document}
```

All a*X*paper-, b*X*paper-, c*X*paper- and d*X*paper-options need to be given as class options when KOMA-Script classes are used. For other classes this works only if they support typearea. For the standard LaTeX-classes these options need to be declared when typearea is loaded. `\PassOptionsToPackage` (see [Tea99b]) works in both cases.

`\paperwidth`
`\paperheight`

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

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

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

`dvips`
`pdftex`
`pagesize`

These mechanisms will set internal LaTeX dimensions to values that header, text body and footer can be printed on paper of the given size. However, the specifications of the DVI-format do not allow the paper format to be specified. If DVI is translated directly into a low level printer language like PCL (Hewlett-Packard printers) or Esc-P (Epson), this is usually not important, because in all these cases the origin is the upper left corner. If however the DVI-source is translated into languages like PostScript or PDF, that have

an origin in a different position and also contain the paper size explicitly, the required information is not available in the DVI-file. To solve this problem the DVI-driver will use the default paper size, which the user may set per option or in the TEX-source. In case of the DVI-driver dvips this can be done with a \special-command. For pdfTEX two dimensions are set instead.

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

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

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

If the file is run through pdfTEX then the lengths \pdfpagewidth and \pdfpageheight will be set to appropriate values. If on the other hand you create a DVI-file – either with LATEX or pdfLATEX – a \special will be written to the beginning of the file.

2.6 Odd bits without direct relevance to text layout

`\ifpdfoutput{`*then*`}{`*else*`}`

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

example: As you may know pdfLATEX will produce a DVI-file instead of a PDF-file, if the counter \pdfoutput is assigned the value 0. Only is the counter is assigned a value different from 0 output is switched to PDF. Since \pdfoutput is unknown when LATEX is used instead of pdfLATEX, \pdfoutput can not be set to 0 generally, if you want

DVI-output. A simple solution to this problem is to execute following command:

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

This only works after loading typearea package. If you want the line above to be executed after af package, which set's `\pdfoutput` to 1 whenever the counter exist, you may combine it with the `\AfterPackage` command from scrlfile package (see chapter 9).

2.7 Local defaults in the file typearea.cfg

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

`\SetDIVList{`*List*`}`

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

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

2.8 Hints

In particular for thesis many rules exist that violate even the most elementary rules of typography. The reasons for such rules include typographical incompetence of those making them, but also the fact that they were originally meant for mechanical typewriters. With a typewriter or a primitive text processor dating back to the early ´80s it is not possible to produce typographically correct output without extreme effort. Thus rules were created that appeared to be achievable and still allowed easy correction. To avoid short lines made worse by ragged margins the borders were kept narrow, and the line spacing increased to 1.5 for corrections. In a single spaced document even correction signs would have been difficult to add. When computers became widely available for text processing, some students tried to use a particularly "nice" font to make their work

look better than it really was. They forgot however that such fonts are often more difficult to read and therefore unsuitable for this purpose. Thus two bread-and-butter fonts became widely used which neither fit together nor are particularly suitable for the job. In particular Times is a relatively narrow font which was developed at the beginning of the 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 border sizes prescribed.

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

To some 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 around. Hardly a text typeset in this way looks really good. In many cases font designers try to increase the serifes on the 'i' or 'l' to compensate for the different character width. This can not work and results in a fragmented and agitated looking text. If you use LaTeX for your paper, some of these rules have to be either ignored or at least interpreted generously. For example you may interpret "60 characters per line" not as a fixed, but average or maximal value.

As executed, record regulations are usually meant to obtaining an useful result even if the author does not know, what to be consider thereby. Usefully means frequently: readable and correctable. In my opinion the type-area of a text set with LaTeX and the typearea package becomes well done from the beginning fair. Thus if you are confronted with regulations, which deviate obviously substantially from it, then I recommend to submit a text single dump to the responsible person and inquire whether it is permitted to supply the work despite the deviations in this form. If necessary the type-area can be moderately adapted by modification of option `DIV`. I advise against use of `\areaset` for this purpose however. At worst you may use geometry package (see [Ume00]), which is not part of KOMA-Script, or change the type-area parameters of LaTeX. You may find the values determined by typearea at the log file of your document. Thus moderate adjustments should be possible. However absolutely make sure that the proportions of the text area correspond approximate with those the page with consideration of the binding correction.

If it should be absolutely necessary to set the text one-and-a-half-lined then you should not redefine `\baselinestretch` under any circumstances. Although this procedure is recommended very frequently, it is however obsolet since the introduction of LaTeX 2_ε in 1994. Use at least the instruction `\linespread`. I recommend package setspace (see [Tob00]), which is not part

of KOMA-Script. Also you should use typearea to calculate a new type-area after the conversion of the line space. However you should switch back to the normal line space for the title, better also for the directories — as well as the bibliography and the index. The setspace package offers for this a special environment and own instructions.

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

The question is asked to me not rarely, why I actually talk section by section about a type-area calculation, while it would be very many simpler, only to give you a package, with which one can adjust the edges as during a text processing. Often also one states, such a package would be anyway the better solution, since everyone knew, how good edges are to be selected, and the edges from KOMA-Script anyway would not be well. I take the liberty of translating a suitable quotation from [WF00]. You may find the original german words at the german scrguide.

> *The making by oneself is long usually, the results are often doubtful, because layman typographers do not see, what is correct and cannot not know, on what it important. Thus one gets accustomed to false and bad typography.* [...] *Now the objection could come, typography is nevertheless taste thing. If it concerned decoration, perhaps one could let apply the argument, since it concerns however primarily information with typography, errors cannot only disturb, but even cause damage.*

2.9 Autoren

The authors listed below are responsible for this chapter or have contributed to this chapter in different ways.

- Frank Neukam
- **Markus Kohm** <Markus.Kohm@gmx.de>
- Axel Sommerfeldt

- *Colin Marquardt*
- *Dr. Engelbert Buxbaum*

3 The Main Classes scrbook, scrrprt and scrartcl

The main classes of the KOMA-Script bundle are designed as counterparts to the standard LaTeX classes. This means that the KOMA-Script bundle contains replacements for the three standard classes book, report and article. There is also a replacement for the standard class letter. The document class for letters is described in a separate chapter, because it is fundamentally different from the three main classes (see chapter 6).The names of the KOMA-Script classes are composed of the prefix "scr" and the abbreviated name of the corresponding standard class. In order to restrict the length of the names to eight letters, the vowels, starting with the last one, are left off if necessary. The Table 3.1 shows an overview. The table also includes the names of the LaTeX2.09 style files that were used in KOMA-Script.

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.

Standard class	KOMA-Script class	Script Style (LaTeX2.09)
article	scrartcl	`script_s`
report	scrreprt	`script`
book	scrbook	`script`
letter	scrlettr	`script_l`

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

3.1 The Options

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

only describe those aspects whose implementation differs from the standard one.

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

Option	scrbook	scrreprt	scrartcl
11pt	default	default	default
a4paper	default	default	default
abstractoff	*undefined*	default	default
bigheadings	default	default	default
final	default	default	default
footnosepline	default	default	default
headnosepline	default	default	default
listsindent	default	default	default
nochapterprefix	default	default	*undefined*
onelinecaption	default	default	default
notitlepage			default
onecolumn	default	default	default
oneside		default	default
openany		default	default
openright	default		
parindent	default	default	default
tablecaptionbelow	default	default	default
titlepage	default	default	
tocindent	default	default	default
twoside	default		

Table 3.2: Default options of the KOMA-Script classes

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 the Table 3.2 that I have 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 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 use the command `\usepackage{typearea}` explicitly.

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

The basic options for the choice of paper format are not processed directly by the classes. 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 options of the standard classes that have the same name and define the same paper format. The page layout calculated for each is different, however.

The options for the A, B, C or D format are actually not processed by the typearea, 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 correcting, the divisor and the number of header lines.

`BCOR`*correction*
`DIV`*factor*
`DIVcalc`
`DIVclassic`
Value`headlines`

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

3.1.2 Options for Document Layout

This subsection collects all the options that affect the document layout, not only the page layout. Strictly speaking all page layout options (see section 3.1.1) are also document layout options. The reverse is also partially true.

`oneside`
`twoside`

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

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

`onecolumn`
`twocolumn`

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

`openany`
`openright`

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

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

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

`cleardoublestandard`
`cleardoubleplain`
`cleardoubleempty`

If one wishes the empty pages created by the `\cleardoublepage` command to have no headers or footers while using the standard classes, the only possibility is to redefine the command appropriately. KOMA-Script provides options that avoid this. The option `cleardoublestandard` enables the default `\cleardoublepage` behaviour. If the option `cleardoubleplain` is used, then the `plain` page style is applied to the empty left page. The option `cleardoubleempty` causes the `empty` page style to be used. The page styles are described in subsection 3.2.2.

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

In order to have a line separating the header from the text body use the option `headsepline`. The option `headnosepline` has the reverse effect. These options have no effect with the page styles `empty` and `plain`, because there is no header in this case. Such a line always has the effect of visually approximating header and text body. That doesn't mean that the header must be put farther apart from the text body. Instead, the header should be considered to belong

to the text body for the purpose of page layout calculations. KOMA-Script takes this into account by automatically passing the option `headinclude` to the typearea package whenever the `headsepline` option is used.

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

`titlepage`
`notitlepage`

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

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

`chapterprefix`
`nochapterprefix`

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 `nochapterprefix`. These options also affect the automatic running titles in the headers (see subsection 3.2.2).

`appendixprefix`
`noappendixprefix`

scrbook, scrreprt

Sometimes one wishes to have the chapter titles in simplified form according to `nochapterprefix`. But at the same time, one wishes a title of an appendix

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

to be preceded by a line with "Appendix" followed by the appendix letter. This is achieved by using the `appendixprefix` option. Since this results in an inconsistent document layout, I advise against using this option.

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

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

The standard classes normally set paragraphs indented and without any vertical inter-paragraph space. This is the best solution when using a regular page layout, like the ones produced with the typearea package. If there where no indentation and no vertical space, only the length of last line would give the reader a reference point. In extreme cases, it is very difficult to detect if a line is full or not. Furthermore, it is found that a marker at the paragraph's end tends to be easily forgotten. A marker at the paragraph's beginning is 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 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 should be rejected. The indentation is perfectly sufficient by itself. The only drawback of indentation is the reduction of the line length. The use of inter-paragraph spacing is therefore justified when using short lines, for instance in a newspaper.

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

The four `parskip` options define an inter-paragraph spacing of one line. The four `halfparskip` options use just a spacing of half a line. In order to avoid a change of paragraph going unnoticed, for instance after a page break, three of the options of each set ensure that the last line of a paragraph is not

full. The variants without plus or star sign ensure a free space of 1 em. The plus variant ensures that at least a third of the line is free and the star variant ensures that at least a fourth of the line is free. The minus variants make no special provision for the last line of a paragraph.

All eight options change the spacing before, after and inside list environments. This avoids the problem of having these environments or the paragraphs inside them with 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.

`onelinecaption`
`noonelinecaption`

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

The avoidance of a special treatment for the caption has an additional effect that is sometimes greatly desired. Footnotes that appear inside a `\caption` command often have a wrong number assigned to them. This happens because the footnote counter is incremented once when the line is measured. When the `noonelinecaption` option is used no such measurement is made. The footnote numbers are therefore correct.

But since KOMA-Script version 2.9 you don't need the option `noonelinecaption` to avoid the above described effect. KOMA-Script classes contain a workaround, so if you have footnotes at captions you simply should put the contents of the figure or table into a minipage and everything will be nice.

3.1.3 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, so that the user must provide any other class option files. The package extsizes (see [Kil99]) can be used for that task. Very big font sizes may lead to arithmetic overflow inside the page layout calculations of the typearea package.

`smallheadings`
`normalheadings`
`bigheadings`

The font size used for the titles is relatively big, both with the standard classes and with KOMA-Script. Not everyone likes this choice; moreover it is specially problematic for small paper sizes. Consequently KOMA-Script provides, besides the large title font size defined by the `bigheadings` option, the two options `normalheadings` and `smallheadings`, that allow for smaller title font sizes. The spacing before and after chapter titles is also influenced by these options. Chapter titles are also influenced by the options `chapterprefix` and `nochapterprefix`, and appendix titles by the options `appendixprefix` and `noappendixprefix`, all of them are described in subsection 3.1.2.

scrbook, scrreprt

3.1.4 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 of the contents can be obtained with the options provided.

`liststotoc`
`idxtotoc`
`bibtotoc`
`bibtotocnumbered`
`liststotocnumbered`

Lists of tables and figures, index and bibliography are not normally included in the table of contents. These entries are omitted in classical typography because it is silently

assumed that, if at all, lists of figures and tables come after the table of contents, the index comes at the end and the bibliography comes before the index. Books with all these parts often include ribbons that can be used to mark the location of these parts 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 certainly 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 aforementioned lists in the table of contents, there exists only one option `liststotoc` that causes entries for both types of lists to be included. This also includes any lists produced with version 1.2e or later of the float package (see [Lin01]). All these lists are unnumbered, since they contain entries that reference other sections of the document.

The option `idxtotoc` causes an entry for the index to be included in the table of contents. The index is also unnumbered since it 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. On these reasons it could be argued that it is a different chapter (or section) and, as such, should be numbered. The option `bibtotocnumbered` has this effect, including the generation of the corresponding entry in the table of contents. I think that a similar reasoning would lead us to consider a classical list of sources to be a separate chapter. Also, the bibliography is not something that was written by the document's author. In view of this, the bibliography merits nothing more than an unnumbered entry in the table of contents, and that can be achieved with the `bibtotoc` option.

The author of KOMA-Script doesn't like option `bibtotocnumbered`. He almost detests option `liststotocnumbered`. Because of this you won't find a detailed description of this option. Nevertheless it exists.

`tocindent`
`tocleft`

v2.8q

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

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

In order to calculate automatically the space taken by the unit numbers when using the option `tocleft` it is necessary to redefine some macros. It is improbable but not impossible that this leads to problems when using other packages. If you think this may be causing problems, you should try the alternative option `tocindent`, since it does not make any redefinitions. When using packages that affect the format of the table of contents, it is possible that the use of options `tocleft` and `tocindent` may lead to problems. In that case, one should use neither of these global (class) option.

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

3.1.5 Options for Lists of Floats

The best known lists of floats are the list of figures and the list of tables. With help from the float package, for instance, it is possible to produce new float environments with the 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 subsection 3.1.4.

`listsindent`
`listsleft`

Lists of figures and tables are generally setup so that their numbering uses a fixed space. This corresponds to the use of option `listsindent`. v2.8q

If the numbers get too large, for instance because many tables are used, it may happen that the available space is exceeded. Therefore KOMA-Script supplies an option called `listsleft` that is similar to the `tocleft` option. The width of the numbers is automatically determined and the space for them

correspondingly adjusted. Concerning the mode of operation and the side effects, the observations made in subsection 3.1.4 for the `tocleft` option are equally valid in this case. Please note that when using the `listsleft` option several LaTeX runs are necessary before the lists of floats achieve their final form.

3.1.6 Options Affecting the Formatting

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

`abstracton`
`abstractoff`

scrreprt, scrartcl

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

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

`pointednumbers`
`pointlessnumbers`

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

In some cases the mechanism that switches the end point may fail or other languages may have different rules. Therefore it is possible to activate the

use of the end point with the option `pointednumbers` or to deactivate it with `pointlessnumbers`.

Please note that the mechanism only takes effect on the next LaTeX run. When trying to use these options to control the numbering format, a 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.

`leqno`

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

`fleqn`

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

`tablecaptionbelow`
`tablecaptionabove`

As described in subsection 3.6.6, the `\caption` command acts with figures like the `\captionbelow` command. The behaviour with tables depends on two options. With the default `tablecaptionbelow` options, the `\caption` macro acts like the `\captionbelow` command. With the `tablecaptionabove` option, `\caption` acts like the `\captionabove` command.

Note that when using the float package, the options `tablecaptionbelow` and `tablecaptionabove` cease to act correctly when the `\refloatstyle` is applied to tables. More details of the float package's `\refloatstyle` can be obtained from [Lin01]. float

`origlongtable`

The package longtable (see [Car98]) sets table captions internally by calling the command `\LT@makecaption`. In order to ensure that these table captions match the ones used with normal tables, the KOMA-Script classes normally redefine that command. See subsection 3.6.6 for more details. The redefinition is performed with help of the command `\AfterPackage` immediately after the loading of package longtable. If the package longtable

caption2 (see [Som95]) 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, activate the `origlongtable` option.

`openbib`

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

`draft`
`final`

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

The two options are also processed by other packages and affect their workings. 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 they 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. 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 that the interface was mainly used by document authors. So a new, simpler interface was created. However, the author advises explicitly 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 are usually 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], [Tea99a] or [Tea00]. 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` commands extends the existing definition.

It is not recommended to use both statements in the same document. Usage examples can be found in the paragraphs on the corresponding element. Names and meanings of the individual items are listed in Table 3.3. The default values are shown in the corresponding paragraphs.

The command `\usekomafont` can change the current font specification to the one 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}}
```

Element	Description
`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`
`descriptionlabel`	Labels, i.e. the optional argument of `\item`, in the `description` environment
`dictumauthor`	Author of a smart saying; used according to the element `dictumtext`
`dictumtext`	Text of a smart saing (see command `\dictum`)
`pagefoot`	The foot of a page, but also the head of a page
`footnote`	Footnote text and marker
`footnotelabel`	Mark of a footnote; used according to the element `footnote`
`footnotereference`	Footnote reference in the text
`pagehead`	The head of a page, but also the foot of a page
`pagenumber`	Page number in the header or footer
`paragraph`	Title of the sectional unit `\paragraph`
`part`	Title of the `\part` sectional unit, without the line with the part number
`partnumber`	Line with the part number in a title of the sectional unit `\part`
`section`	Title of the sectional unit `\section`
`sectioning`	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
`subparagraph`	Title of the sectional unit `\subparagraph`
`subsection`	Title of the sectional unit `\subsection`
`subsubsection`	Title of the sectional unit `\subsubsection`
`title`	Main title of the document, i.e. the argument of `\title`

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

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

3.2.2 Page Style

```
\pagestyle{empty}
\pagestyle{plain}
\pagestyle{headings}
\pagestyle{myheadings}
\thispagestyle{local 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. 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 head and only a page number in the foot. With the standard classes this page number is always centered in the foot. With KOMA-Script the page number appears on double-sided layout on the outer side of the foot. The one-sided page style behaves like the standard setup.

headings is the page style with running titles in the head. With the classes scrbook and scrreprt the titles of chapters and sections are repeated in the head – with KOMA-Script on the outer side, with the standard classes on the inner side. The page number comes with KOMA-Script on the outer side of the foot, with the standard classes it comes on the inner side of the head. In one-sided layouts only the headings of the chapters are used and are, with KOMA-Script, centered in the head. The page numbers are set with KOMA-Script centered in the foot. scrartcl behaves similarly, but starting a level deeper in the section hierarchy with sections and subsections, because the chapter level does not exist in this case. scrartcl

While the standard classes automatically set running headings always in capitals, KOMA-Script applies the style of the title. This has several typographic reasons. Capitals as a decoration are actually too strong. If one applies them nevertheless, they should be set with a smaller type

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

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

size and tighter spacing. The standard classes don't take these points in consideration.

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

Besides, the form of the page styles `headings` and `myheadings` is affected by each of the four class options `headsepline`, `headnosepline`, `footsepline` and `footnosepline` (see subsection 3.1.2). 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 one uses the `\thispagestyle` command. This also happens at some places in the document automatically. 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 not performed by page style changes when using the scrpage2 package, but instead special instructions must be used. The page styles `headings` and `myheadings` should not be used together with this package (see chapter 4).

v2.8p

In order to change the type style used in the head, foot or for the page number, please use the interface described in subsection 3.2.1. The same element is used for head and foot, which you can designate equivalently with `pagehead` or `pagefoot`. The element for the page number within the head or foot is called `pagenumber`. The default settings can be found in Table 3.4.

example: Assume that you want to set head and foot 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 horribly, you can reach this as follows:

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

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

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

As you can see, the last example uses the element `pagefoot`. You can achieve the same result using `pagehead` instead (see Table 3.3 on Page 52).

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).

If you define your own page styles, the commands `\usekomafont{pagehead}` and `\usekomafont{pagenumber}` can find a meaningful use. If you use for that not the KOMA-Script package scrpage2 (see chapter 4), but, for example, the package fancyhdr (see [Oos00]), you can use these commands in your definitions. Thereby you can remain compatible with KOMA-Script. 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. The default values for all four cases is `plain`. The meaning of these macros can be taken from the following list:

`\titlepagestyle` – Page style for a title page when using *in-page* titles.

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

scrbook, scrreprt

`\chapterpagestyle` – Page style for the first page of a chapter.

`\indexpagestyle` – Page style for the first page of the index.

The page styles can be redefined with the `\renewcommand` macro.

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

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

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

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

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

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

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

The redefinition of the other page styles is done in the same way.

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:

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

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 of the fact that all floats that have not yet been output and starts a new page. There exists the instruction `\cleardoublepage` which works like `\clearpage` but that, in the double-sided layouts (see layout option `twoside` in subsection 3.1.2) 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 title. The page is thus entirely empty. However, the approach used by \cleardoublepage is dependent on the layout options cleardoublestandard, cleardoubleplain and cleardoubleempty described in subsection 3.1.2 and acts according to the active option.

\ifthispageodd{*true*}{*false*}

A peculiarity of LaTeX consists in 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 TeXtest 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.

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

Table 3.5: Available numbering styles of page numbers

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

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

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

`\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 3.5. 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 subsection 3.1.2, 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 options `titlepage` and `notitlepage`.

`titlepage`

With the standard classes and with KOMA-Script all title pages are defined in a special environment, the `titlepage` environment. This environment always starts a new page – in the two-sided layout a new right page. For this page, the style is changed as by `\thispagestyle{empty}`, so that neither page number nor running title 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 makes 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, the 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 subsection 3.2.2.

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, precisely the *half-title*. Nowadays the extra page is often applied before the real full title and contains information on the publisher, series number and similar information.

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

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

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

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

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

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

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

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

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

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

v2.8p

The *Title* is output with a very large font size. Besides the change of size, the settings for the element `title` also take effect. By default these settings are identical to the settings for the element `sectioning` (see Table 3.3). The font size is however not affected (see Table 3.3). The default settings can be changed with the commands of subsection 3.2.1.

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

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

Table 3.6: 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`

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

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

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

With the exception of *titlehead* and possible footnotes, all the items are centered horizontally. The information is summarised in Table 3.6.

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 advisor 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 Institut\\
  Mythological Rd\\
```

```
  34567 Etherworld}
\subject{Dissertation}
\title{Digital space simulation with the DSP\,56004}
\author{Fuzzy George}
\date{30. February 2002}
\publishers{Advisor Prof. John Excentric Doe}
\maketitle
\end{document}
```

A frequent misunderstanding concerns the role of the full title side. It is often erroneous 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 have equally large left and right margins.

However if one takes a book and opens it, one hits very quickly on at least one title page under the cover within the so-called inner book. Precisely these title pages are produced by `\maketitle`. Like it happens 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\\
  in eternal love.}
\maketitle
\end{document}
```

`abstract`

Particularly with articles, more rarely with reports, there is a summary directly under the title and before the table of contents. Therefore, this is often considered a part of the titles. Some 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 options `abstracton` and `abstractoff` (see subsection 3.1.6)

scrartcl, scrreprt

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 subsection 3.6.6).

`\tableofcontents`
`\contentsname`

The production of the table of contents is done by the `\tableofcontents` command. To get a correct table of contents, at least two LaTeX runs are necessary after every change. The option `liststotoc` causes the lists of figures and tables to be included in the table of contents. `idxtotoc` is the corresponding option for the index. This is rather uncommon in classical typography. One finds the bibliography included in the table of contents a little bit more frequently. This can be obtained with the options `bibtotoc` and `bibtotocnumbered`. These options are explained in subsection 3.1.4.

The table of contents is not set as a numbered chapter and is therefore subjected to the side effects of the standard `\chapter*` command, which are described in subsection 3.6.2. However, the running titles 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`.

There is only one variant for the construction of the table of contents. The titles of the sectional units are indented so that the unit number is flush left to the edge of the title of the next upper unit. However, the place for the numbers is thereby limited and is only sufficient for a little bit more than 1.5 places per level. Should this become a problem, help can be found in [RNH02].

The entry for the highest sectional unit below `\part`, i.e., `\chapter` with scrbook and scrreprt or `\section` with scrartcl is not indented. It is however affected by the settings of the element `sectioning` (see Table 3.3). There is no point 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`. It has the value -1 for `\part`, 0 for `\chapter` and so on. Since the class scrartcl has no `\chapter`, the counter starts with 0 for the `\part`. 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 up 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 take from the `tocdepth` counter can also be found by looking at the table of contents after the first LaTeX run.

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 even can be found in the appendix. However, the author of this manual prefers the location after the table of contents, therefore it is discussed here.

`\listoftables`
`\listoffigures`

These commands generate a list of tables or figures. Changes in the document, that modify these lists will require two LaTeX runs in order to take effect. The layout of the lists can be influenced by the options `listsindent` and `listsleft`, see subsection 3.1.5. Moreover, the options `liststotoc` and `liststotocnumbered` have indirect influence, see subsection 3.1.4).

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.

3.6.1 Separation

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

scrbook

Before getting to the main text eventually 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 *prefix*, the *main text* and the *appendix* in a book.

The macro `\frontmatter` introduces the prefix in which roman numbers are used for the page numbers. Chapter headings in a prefix don't have any numbers. The foreword can be set as a normal chapter. A foreword should never be divided into sections but kept short. Therefore there is no need for a deeper structuring than chapter.

`\mainmatter` introduces the main text. If there is no prefix this command can be omitted. The default in the main text is arabic page numbering (re)starting with 1.

The appendix is introduced with `\backmatter`. Opinions differ in what should be part of the prefix. So in some cases you will only find the bibliography, in some cases only the index and in other cases both of it in the appendix. Besides, the appendix is similar to the prefix.

3.6.2 Structuring the Document

```
\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 quite similar to the standard classes. Normally the section headings show up in the table of contents exactly as they are entered in the text. The entry for the table of contents can be specified as an optional argument in front of the actual heading. `\chapter` only exists in book or report classes but not in article classes. In addition to this, the command `\chapter` in KOMA-Script differs substantially from the version in the standard class. While in the standard classes every chapter number is used together with the prefix "Chapter" on a separate line above the heading, KOMA-Script only places the chapter number in front of the heading.

scrartcl

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

The font of all headings can be changed with the commands `\setkomafont` and `\addtokomafont` described in subsection 3.2.1. First of all the element `sectioning` is used, which is followed by a specific element for every section level (see Table 3.3). The font for the element `sectioning` is predefined as `\normalfont\normalcolor\sffamily\bfseries`. The default font size for the specific elements depends on the options `bigheadings`, `normalheadings` and `smallheadings` (see subsection 3.1.3). The defaults are listed in Table 3.7

v2.8p

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

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

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

Table 3.7: Default font sizes for different levels of document structuring

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

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

or using:

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

The last version is to be preferred because it gives you the correct result even when you change the `sectioning` element as follows:

```
\setkomafont{sectioning}{\normalcolor\bfseries}
```

With this change it is possible to set all section levels at once to not longer use sans serif fonts.

Please be warned of using 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 thing even for professionals and has nothing to do with personal taste of beginners. Please refer to the citation at the end of section 2.8 and the following explanation.

It is possible to use different font types for different section levels in KOMA-Script. As a typographical beginner you should refrain from using these possibilities for typographical reasons. There is a rule which states that you should mix fonts only as few as possible. Using sans serif for headings seems already a breach of this rule. However, you should know that bold, huge and serif letters are much to heavy for headings. In general you would have to use a normal instead of a bold or semi bold font. However, in deeper levels of the structuring a normal font appears rather leight weighted. On the other hand sans serif fonts in headings have a very pleasant appearance and are to be used only in headings. That's why sans serif is the default in KOMA-Script. More variety should be avoided. Font mixing is only for professionals. In case you want to use other fonts than the standard TEX-Fonts – no matter whether using CM or EC fonts – you should consult an expert or redefine the font for the element `sectioning` as seen in the example above. You often find the combinations Times and Helvetica or Palatino with Helvetica. The author of this documentation does not favour these combinations.

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

All sectioning commands exist as "starred" versions. They produce section headings which do not show up in the table of contents, in the page header and which are not numbered. Not using a headline often has an unwanted side effect. For example, if a chapter which is set using `\chapter*` spans over several pages the headline of the chapter before comes up again. KOMA-Script
scrartcl offers a solution which is described below. `\chapter*` only exists in book and report classes which includes 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 `\partpagestyle` and `\chapterpagestyle` in KOMA-Script (see subsection 3.2.2).

v2.8p As for the possibilities of font switching the same explanations apply which were given above with the normal sectioning commands. The elements of structuring are named in the same way as the "unstarred" versions.

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

In addition to the standard classes KOMA-Script offers the new commands `\addsec` and `\addchap`. They are similar to the standard commands `\chapter` und `\section` except the missing numbering. The produce both a running headline and an entry in the table of contents. The starred versions `\addchap*` and `\addsec*` are similar to the standard commands `\chapter*` and `\section*` apart from a tiny but important difference: The headlines are deleted. This eliminates the side effect of obsolete headers mentioned above.
scrartcl `\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.

Similar to this, the command `\addpart` produces an unnumbered document part with an entry in the table of contents. Since the headers are already deleted by `\part` and `\part*` the problem of obsolete headers doesn't exist. The starred version `\addpart*` is identical to `\part*` and is only defined for consistency reasons.

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

As for the possibilities of font switching the same explanations apply which were given above with the normal sectioning commands. The elements of structuring are named in the same way as the "unstarred" versions. v2.8p

`\minisec{`*Heading*`}`

Sometimes a heading is wanted which is highlighted but closely linked to the following text. Such a heading shouldn't be separated by a vertical skip.

The command `\minisec` is designed for this situation. This heading isn't linked to a level of structuring. 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 the things needed and an assembly description. Using `\minisec` 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 meanwhile the following
actions.
```

```
Afterwards one knocks in the spring-plug with the hammer
in 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 with the two screws
employing the knife on the mouse-hole.
Instead of the knife one can use a screw-driver as well.
```

Which gives:

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

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

Afterwards one knocks in the spring-plug with the hammer in 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 with the two screws employing the knife on the mouse-hole. Instead of the knife one can use a screw-driver as well.

The font type of the sectioning command `\minisec` can only be changed using the element `sectioning` (see Table 3.3). There is no specific element for `\minisec`. This means you can't change the font size manually.

`\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 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. You write in the preamble of your document:

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

or short:

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

You will get a formatting of the headings which is very close to the standard classes. Even closer it will get when you combine this change with the change of the element `sectioning` mentioned above.

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

As you might know, for every counter in LaTeX there is a command \the*counter name*, which gives you 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 higher level followed by a dot and the arabic number of the *counter name* of the respective level.

KOMA-Script has added to the output of the section number a further logical level. The counter for the respective heading are formatted using \partformat, \chapterformat and \othersectionlevelsformat. Of course the command \chapterformat doesn't exist in the class scrartcl. scrbook, scrreprt

As described in subsection 3.1.6 KOMA-Script handles dots in section numbers according to [DUD96]. The command \autodot makes sure that these rules are being followed. Except from \part in all levels a dot is followed by a \enskip. This is similar to a horizontal skip of 0.5 em.

The command \othersectionlevelsformat takes the name of the section level, such as "*section*", "*subsection*" ..., as parameter. As default, only the levels \part and \chapter have formatting commands on their own, while all other section levels are covered by one formatting command only. This has historical reasons. At the time 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 you don't 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}
```

In fact, you could do without `\autodot` at this point and insert a fixed point instead. As `\part` is numbered with roman numbers, according to [DUD96] a dot has to be applied. However, you would give up the possibility to use one of the options `pointednumbers` und `pointlessnumbers` then. More details concerning class options you can find in subsection 3.1.6).

An additional possibility could be to place the section numbers in the left margin. That can be done in 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 almost unknown command `\llap` in the definition above, puts its argument left to the current possition without changing the position.

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

scrbook, scrreprt

These two commands are used internally by KOMA-Script and are also pro-

v2.8o

vided at the user interface. Using the layout option `chapterprefix` (see subsection 3.1.2) `\chapappifchapterprefix` issues the word "Chapter" in the main part of your document in the current language followed by *additional text*. In the appendix instead, the word "Appendix" in the current language followed by *additional text* is issued. Having set the option `nochapterprefix` there is no additional output.

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

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

`\chaptermark{`*Running head*`}`
`\sectionmark{`*Running head*`}`
`\subsectionmark{`*Running head*`}`
`\chaptermarkformat`
`\sectionmarkformat`
`\subsectionmarkformat`

As mentioned in subsection 3.2.2 the page style `headings` works with running heads. For this, the commands `\chaptermark` and `\sectionmark` as well as `\sectionmark` and `\subsectionmark` respectively are defined. Every sectioning command (`\chapter`, `\section`, `\subsection` ...) automatically carries out the respective `\...mark` command. The parameter handed over takes the text of the section heading. The respective section number is added automatically to the `\...mark` command. The formatting is done according to the section level with the command `\chaptermarkformat`, `\sectionmarkformat` or `\subsectionmarkformat`. Of course there is no command `\chaptermark` or `\chaptermarkformat` in scrartcl. Accordingly `\subsectionmark` and the command `\subsectionmarkformat` only exist 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 at scrartcl), `\sectionmarkformat` and the command `\subsectionmarkformat` (only at scrartcl) define the formatting of the section numbers in running heads. They can be adapted to your personal needs with `\renewcommand`. The original definitions from 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 combine the chapter number in the header with the word "Chapter". For example you could insert in the preamble of your document the following definition:

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

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

`secnumdepth`

As 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. Since in scrartcl there is no `\chapter` the counting in this class starts with 0 at the level `\part`. By way of defining, decrementing or incrementing 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 explanations concerning the counter `tocdepth` in section 3.4.

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

scrbook, scrreprt

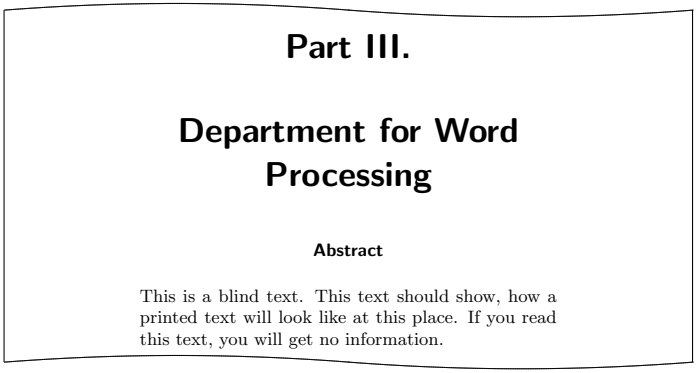
Parts and chapters in KOMA-Script can be started with a *preamble*. This is particularily useful when you are using a two column layout with the class option `twocolumn`. Together with the heading the *preamble* is always set in a one column layout. The *preamble* can comprise more than one paragraph. The command for issuing the *preamble* has to be put in front of the respective `\part`, `\addpart`, `\chapter` or `\addchap` command.

example: You are writing a report on the situation 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 a short abstract on the title page. You could write the following:

```
\setpartpreamble{%
  \begin{abstract}
```

```
    This is a blind text. This text should show, how a
    printed text will look like at this place. If you
    read this text, you will get no information.
  \end{abstract}
}
\part{Department for Word Processing}
```

Depending on the settings for the heading (see subsection 3.1.3) size and the `abstract` environment (see subsection 3.1.6), 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).

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: v2.8p

`o:` above the heading

`u:` below the heading

You can insert a preamble both above and 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 issue the text of the *preamble* 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 this case the option for horizontal positioning will have no effect. You can combine one letter from the vertical with one letter from the horizontal positioning.

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

scrbook, scrreprt

v2.8q

Apart from an introducing 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 [Tea99a]) of the width `\dictumwidth`. Yet `\dictumwidth` is not a length which is 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`. The command `\raggeddictum` can be redefined using `\renewcommand`.

Within the box the *dictum* is set using `\raggeddictumtext`. Default setting is `\raggedright`. Similar to `\raggeddictum` it can be redefined with `\renewcommand`. The output uses the default font which is set for the element `dictumtext`. It can be changed with the commands from subsection 3.2.1. Default settings are listed in Table 3.8.

If there is an *author* it is separated from the *dictum* by a line with the 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)}
```

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

Table 3.8: Default settings for the elements of a dictum

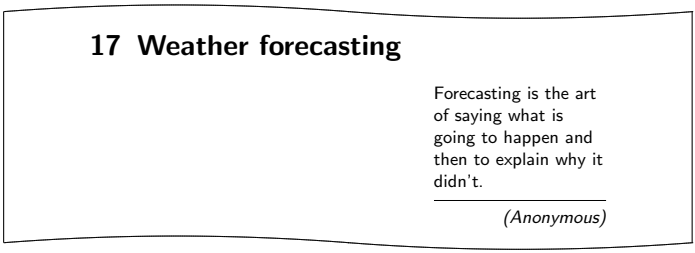
Thus the *author* is set in in round parenthesis. For the element `dictumauthor` a different font as for the element `dictumtext` can be defined. Default settings are listed in Table 3.8. Changes can be made using the commands from subsection 3.2.1. If `\dictum` is used within the macro `\setchapterpreamble` 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 of a 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 you should separate them by an additional vertical space. You could easely use the command `\bigskip` for that.

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 prefer the dictum to span over only a quarter of the text width you can redefine `\dictumwidth`:

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

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

3.6.3 Footnotes

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

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

Similar to the standard classes footnotes in KOMA-Script are produced with the `\footnote` command. An alternative is the usage in pairs 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 near the bottom of a page thus leaving LaTeX no choice as to break the page at this point.

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

Formatting footnotes in KOMA-Script is slightly different to the standard classes. As in the standard classes the footnote mark in the text is formed as a small number in superscript. The same formatting is used in the footnote itself.

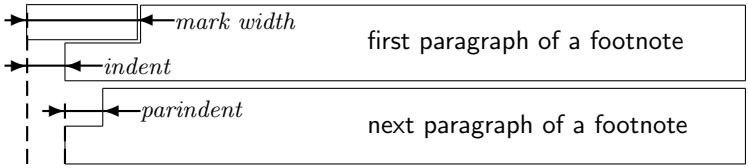


Figure 3.1: Parameters that control the footnote layout

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* was not specified, then it defaults to *indent*. If the footnote consists of more than one paragraph, then the first line of a paragraph is indented in addition to *indent* by the value of *parindent*.

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

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

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

The font element `footnote` determines the font of the footnote including the footnote mark. Using the element `footnotelabel` the font of the footnote mark can be changed separately with the commands mentioned in subsection 3.2.1 Please refer also to the Table 3.3. Default setting is no changing of the font. v2.8q

The footnote mark in the text is defined separately with `\deffootnotemark`. Default setting is:

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

Above the font for the element `footnotereference` is applied (see Table 3.3). Thus the footnote marks in the text and the footnote itself are identical. The font can be changed with the commands described in subsection 3.2.1. v2.8q

example: A feature often asked for are footnote marks which are neither in superscript nor in a smaller font size. They should not touch

the footnote text but have a small space in between. This can be accomplished as follows:

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

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

Another often requested footnote layout are left-aligned footnote marks. These can be reached with:

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

If you want however change the font for all footnotes, for example to sans serif, you can simply solve this problem using the commands from subsection 3.1.3:

```
\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. Even more than three levels are hard to perceive. Recommendation in these cases is to split your huge list in several small ones.

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

The most simple form of a list is an `itemize` list. Depending on the level KOMA-Script uses the following marks: "‘•”, "‘–”, "‘∗” and "‘·”. The definition of these symbols is specified in the macros `\labelitemi`, `\labelitemii`,

`\labelitemiii` and `\labelitemiv`. All of this macros 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 aeroplans
  \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 helicopter
  \end{itemize}
  \item automobiles
  \begin{itemize}
    \item racing car
    \item private car
    \item lorry
  \end{itemize}
  \item bicycles
\end{itemize}
```

As output you get:

Vehicles

- aeroplans
  - biplane
  - jets
  - transport planes
    - single-engined
      - jet-driven
      - propeller-driven
    - multi-engined
  - helicopter
- automobiles
  - racing car
  - private car
  - lorry
- 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 occurence of an `itemize` environment with an `enumerate` environment in the example above we get the following result:

Vehicles

1. aeroplans
   a) biplane
   b) jets
   c) transport planes
      i. single-engined
         A. jet-driven
         B. propeller-driven
      ii. multi-engined
   d) helicopter
2. automobiles
   a) racing car
   b) private car
   c) lorry
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 plane with `\label{xmp:jets}`. The `\ref` value is then 1(c)iA.

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

Another list form is the description list. Its main use is the description of several items. The item itself is an optional parameter in `\item`. The font, which is responsible for emphasizing the item can be changed with the commands for the element `descriptionlabel` (see Table 3.3) described in subsection 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 subsection 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 running headline.
  \item[headings] is the page style with running headline.
  \item[myheadings] is the page style for manual headline.
\end{description}
```

This abbreviated version gives:

empty is the page style without any header or footer

plain is the page style without running headline.

headings is the page style with running headline.

myheadings is the page style for manual headline.

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

An additional form of a description list in KOMA-Script is the `labeling` environment. In difference to the `description` environment you can provide a pattern, which determines the indentation of all items. Furthermore you can put an optional *delimiter* between item and description.

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

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

As result we get:

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

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

Originally this environment was implemented for things like "Given is..., Asked is..., Solution" that are often used in 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`

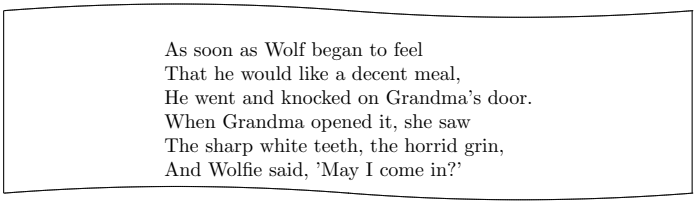
Normally the `verse` environment isn't perceived as a list environment because you don't work with `\item` commands. Instead fixed line breaks are used like within the `flushleft` environment. Yet internally in both the standard classes as well as KOMA-Script it is a list environment.

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

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

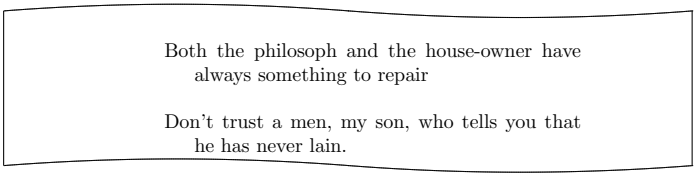
```
\begin{verse}
  As soon as Wolf began to feel\\*
  that he would like a decent meal,\\*
  He went and knocked on Grandma's door.\\*
  When Grandma opened it, she saw\\*
  The sharp white teeth, the horrid grin,\\*
  And Wolfie said, 'May I come in?'
\end{verse}
```

The result would like 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?'

Yet if you have very long lines `\\*` can not prevent a page break within a verse. That would be possible here for example:



Both the philosoph and the house-owner have
always something to repair

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

These two verses were separated by a `\bigskip`.

`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 matter how 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 get around this behaviour you have to insert a `\noindent` command in front.

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. Different to the first two environments using `addmargin` the user can influence the width of the indentation. Furthermore this environment doesn't change the indentation of the first line and 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 *indentation* is given the value *left indentation* is added to *indentation* at the left margin.

The starred `addmargin*` only differs from the normal version in a two-side layout. In addition the difference only occurs if the optional argument *inner indentation* is used. In this case this value is added to the normal inner indentation. 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 slightly spanning 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 way 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 use a length of, for example, `2\parindent` of course.

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 is 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` is broken at the end of a page. Unfortunately this is not without disadvantages: In some cases pages can't be filled correctly which has the effect of several warnings.

By the way, whether a page is going to be on the left or right side of the book can't be determined in the first LaTeX compiling for sure. For details please refer to the explanation for the command `\ifthispageodd`.

One concluding note to the list environments: In the internet and support it is often asked why such an environment is followed by a indented paragraph. In fact this is the result of 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 separate this environment from the rest of your text nevertheless, you can insert a comment line which only consists of a percent character in the LaTeX source.

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 oneside layout the right border is used. Though `\marginpar` optionally can take a different margin note in case the output is on the left margin, margin notes are always in justified layout. But many users prefer left- or right-aligned margin notes instead. KOMA-Script offers the command `\marginline` for that.

example: At several places in this documentation you find the classes mentioned written in the margin. This can be produced[2] with:

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

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

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

Eventually `\marginline` is only an abbreviating writing of the code above.

Unfortunately `\marginpar` doesn't always work correctly in the twoside layout. Whether a margin note is going to show up on the left or right is already decided while evaluating the command `\marginpar`. If the output routine now shifts the margin note onto the next page the alignment isn't correct anymore. This behaviour is deeply founded within LaTeX and was therefore declared a feature by the LaTeX3 team. `\marginline` suffers from this "feature" too.

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

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

or figure within the text. As these floating environments are being referenced in the text this is not necessary in most cases. It is not sensible too because such an object can only be set on the page if there is enough space left. If this is not the case the object would have to be shifted onto the next page leaving a huge space on the page before.

Often in many documents the same optional argument for positioning an object is found with every floating object. This also makes no sense. In such cases you should change the standard parameters globally. For more details refer to [RNH02].

One last important note before starting this section: The most mechanisms described here which extend the capabilities of the standard classes do not work correctly when used together with packages which interfere with the typesetting of captions of figures and tables. This should be without saying but is often neglected.

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

In the standard classes captions of tables and figures are inserted with the `\caption` command below the table or figure. In general this is correct with figures. Opinions differ as to whether captions of tables are to be placed above or together with captions of figures below the table. That's the reason why KOMA-Script, unlike the standard classes, offers `\captionbelow` for captions below and `\captionabove` for captions above tables or figures. Using `\caption` together with figures always produces captions below the figure whereas the behaviour of `\captionbelow` can be modified using the options `tablecaptionabove` and `tablecaptionbelow` (see subsection 3.1.6). For compatibility reasons the default behaviour of `\caption` together with tables is similar to `\captionbelow`.

example: Instead of using captions below the table you want to place your captions above it, because you have tables which span over 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 `tablecaptionabove` instead (see subsection 3.1.6). 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` from the topcapt package (see [Fai99]). But that is not the case. The command `\topcaption` is neglected by packages which directly redefine the `\caption` macro. The hyperref package (see [Rah01]) is one example for this. In KOMA-Script `\captionabove` and `\captionbelow` are implemented so, that the changes have an effect on both of these commands.

If the longtable package is used KOMA-Script makes sure that captions above tables which are placed within a `longtable` environment have the same appearance as in a normal `table` environment. This also means that you can apply the same settings as in a `table` environment. Please note that in the longtable package the maximum width of a table caption can be limited and the default is set to 4 in (see [Car98]). Using KOMA-Script this mechanism in longtable only works when the class option `origlongtable`

is set (see subsection 3.1.6). If caption2 (see [Som95]) 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 same behaviour as described in [Lin01] for the `\caption` command. In this case, only the float style determines whether it is a caption 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 you often find captions, in particular with small figures, which are placed beside the figure. In general in this case both the baseline of the figure and the caption are aligned at the bottom. With some fiddling and the use of two `\parbox` commands this could be achieved in the standard classes. But KOMA-Script offers a special environment for this problem. This environment can be used within the floating environment. The first optional parameter *entry* and the obligatory parameter *title* are similar to the parameters of `\caption`, `\captionabove` or `\captionbelow`. The *title* is placed beside the content of the environment in this case.

Whether the *title* is placed left or right can be determined by the parameter *placement*. One of the following letters is accepted:

- `l` – left
- `r` – right
- `i` – inner margin in twoside layout
- `o` – outer margin in twoside layout

Default setting is at the right side of the content of the environment. If either `o` or `i` are used you have to run LaTeX twice to get the correct placement.

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

When supplying a *width* the used width is 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.

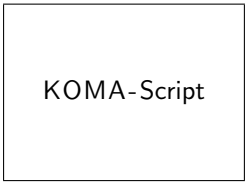


Figure 3.2: A figure description which is neither above nor below, but beside the figure

Adding a star to the optional parameter *offset* the value means a shift relative to the right margin on left pages in double 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}
  \label{fig:maincls.captionbeside}
\end{figure}
```

Thus, the width is the current available width `\linewidth`. However, this width is shifted `2em` to the outside. The title or the description is placed inside beside the figure. Therefore, the figure itself is shifted 2 em into the 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 mentioned in subsection 3.2.1. The respective elements for this are `caption` and `captionlabel` (see Table 3.3). First the font style for the element `caption` is applied on the element `captionlabel` too. After this the font style of `captionlabel` is applied on the respective element. The default settings are listed in Table 3.9.

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

Table 3.9: Font defaults for the elements of figure or table captions

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`

Using the float package the appearance of the float environments is solely defined by the *float* style. This includes the fact 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 both these styles can be activated as the styles `plain`, `boxed` or `ruled` in float are defined. For details refer to [Lin01]. The style `komaabove` inserts `\caption`, `\captionabove` and `\captionbelow` above whereas `komabelow` inserts them below the float content. float

`\captionformat`

In KOMA-Script there are different ways to change the formatting of the description. 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 difference to all other `\...format` commands in this case it doesn't contain the counter but 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 you should put in the preamble of your document.

`\figureformat`
`\tableformat`

It was already mentioned that `\captionformat` doesn't contain a formatting for the label itself. But this label shouldn't in any case 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, for example, of the list of figures. For this case 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 label texts without any label and delimiter are wanted. In KOMA-Script it takes only the following definitions to achieve this:

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

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

As mentioned previously, in the standard classes the captions are set in a not-hanging style. That means that in descriptions with more than one line

the second and subsequent lines start directly beneath the label. There is no straight way in the standard classes 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.

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

If you want a line break before the label and the description you define the indentation *xindent* of the description with the starred version of the command instead: `\setcapindent*`.

Using a negative value of *indent* instead, a page break is inserted and only the first line but not the subsequent lines are indented by $-$*indent*.

Whether one-line captions are set as captions with more than one line or are treated separately is specified with the class options `onelinecaption` and `noonelinecaption`. For details please refer to the explanations of this options in subsection 3.1.2.

example: As examples please refer to the figures 3.3 to 3.6. As you can see the usage of a complete hanging indentation is not preferable together with a small column width:

```
\begin{figure}
  \setcapindent{1em}
  \fbox{\parbox{.95\linewidth}{\centering{\KOMAScript}}}
  \caption{Examples 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.

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

Using these three commands you can specify the width and justification of the label text. In general the whole text or column width is available for the description. v2.8q

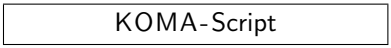


Figure 3.3: Equivalent to the standard setting, similar to the usage of `\setcaphanging`

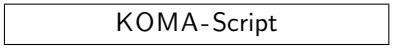


Figure 3.4: With slightly hanging indentation starting at the second line using `\setcapindent{1em}`

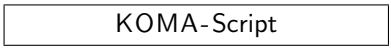


Figure 3.5:
With hanging indentation starting at the second line and line break before the description using `\setcapindent*{1em}`

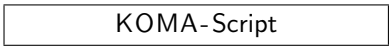


Figure 3.6:
With indentation in the second line only and line break before the description using `\setcapindent{-1em}`

With the command `\setcapwidth` you can decrease this *width*. The obligatory argument determines the *with* of the description. As an optional argument you can supply 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 a double sided output
- `o` – alignment at the outer margin in a 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 doesn't work correctly. In particular the captions of tables of subsequent pages are aligned corresponding to the first part of the table. This is a problem which has its roots 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 the option `noonelinecaption` (see subsection 3.1.2) for the descriptions which are typeset with this margin setting.

longtable places the description in a box, which is issued again at the subsequent pages if needed. While treating a box the macros needed for the creation of it aren't run through again. That's why it is not possible for KOMA-Script to swop margin settings in double sided layout on even pages. This would be necessary to produce a justification which is shifted towards the outside or inside.

You can also submit negative values for *margin* and *margin right* or *margin outside*. This has the effect of the description spanning into the margin.

example: A rather odd problem is a figure caption which is both centered and of the same width as the figure itself. If the width of the figure is known the solution with KOMA-Script is quite easy. Suppose the figure has a width of 8 cm, it only takes:

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

directly in front of `\caption` or `\captionbelow`. If it is unknown first you 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 [Tea99a]) in many cases. A possible solution would look as follows:

```
\begin{figure}
  \centering%
  \settowidth{\figurewidth}{%
    \fbox{\hspace{1em}\KOMAScript\hspace{1em}}%
    }%
  \fbox{\hspace{1em}\KOMAScript\hspace{1em}}%
  \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 use `\setcapwidth` with every figure. But nothing is easier than defining a new command in the preamble of your document which hides the three steps:

1. Defining the width of the argument
2. Specifying the width of the caption
3. Output of the argument

in:

```
\newcommand{\figure2}[1]{%
  \settowidth{\figurewidth}{#1}%
  \setcapwidth[c]{\figurewidth}%
  #1}
```

Using this command the example abbreviates to:

```
\begin{figure}
  \centering%
  \figure2{\fbox{\hspace{1em}\KOMAScript\hspace{1em}}}%
  \caption{Example of a centered caption below the figure}
\end{figure}
```

But a command has the disadvantage that errors in the macros of the argument in case of arguments with more than one line aren't reported with the very correct line number by LaTeX. In these cases the usage of an environment has advantages. But then the question is raised how the width of the content of the environment can be determined. The solution offers the `lrbox` environment, which is described in [Tea99a]:

```
\newsavebox{\figurebox}
\newenvironment{FigureDefinesCaptionWidth}{%
  \begin{lrbox}{\figurebox}%
}{%
  \end{lrbox}%
  \global\setbox\figurebox=\box\figurebox%
  \aftergroup\setfigurebox%
}
\newcommand{\setfigurebox}{%
  \figure2{\usebox{\figurebox}}}
```

This definition uses the macro `\figure2` 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}
```

Admittingly, the environment in this example is not necessary. But its definition using `\global` is quite clever. Most users wouldn't be able to define such an environment without help. But as this can be very useful, it was introduced in the example above.

If the `captionbeside` environment wouldn't 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}%
  }
```

As the next step 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 unvisible 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. In a sense, a heading is a kind of markup too. However, in this section we are only concerned with direct markup, i.e. markup which doesn't have an additional meaning and which can be used for different purposes. More details to the normally defined possibilities you can find in [OPHS99], [Tea99a] and [Tea00].

`\textsubscript{`*text*`}`

In subsection 3.6.3 the command `\textsuperscript` was already introduced which is a part of the LaTeX kernel. Unfortunately LaTeX itself doesn't offer a command to produce a text in subscript instead of superscript. KOMA-Script defines `\textsubscript` for this.

example: You are writing a text on the human metabolism. From time to time you have to mention some simple sum formulas in which the numbers are in subscript. Being convinced from logical markup you first define in the document preamble or in a separate package:

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

Using this you then write:

```
The cell produces its energy from reaction of
\Molek C6\Molek H{12}\Molek O6 and \Molek O2 to
\Molek H2\Molek O{} and \Molek C{}\Molek O2.
However, Arsenic (\Molek{As}{}) has a 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 H_2O and CO_2. However, Arsenic (As) has a detrimental effect on the metabolism.

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

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

Now the output in the whole document changes to:

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

In the example above the writing "`\Molek C6`" is used. This makes use of the fact that arguments which consist of only one character doesn't have to be enclosed in parenthesis. That's why "`\Molek C6`" is similar to "`\Molek{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 thus making sure that the rules according to [DUD96] are being followed. These rules are explained in more detail in the description of the class options `pointednumbers` and `pointlessnumbers` in subsection 3.1.6.

Please note that `\appendix` is a command, *not* an environment! This command does not need an argument. The sectioning commands in the appendix are used in the same way as in the main part.

`\appendixmore`

There is a peculiarity within the `\appendix` command in the KOMA-Script classes. In case the command `\appendixmore` is defined, `\appendix` is executed too. Internally the KOMA-Script classes scrbook and scrreprt take advantage of this behaviour for implementing the options `appendixprefix` and `noappendixprefix` (see subsection 3.1.2). You should take care of this in case you are going to define or redefine the `\appendixmore` by yourself. In case one of these options is set, you will receive an error message when

using \newcommand{\appendixmore}{...}. This is thought to prevent you from changing options without noticing.

example: You do not want the chapters in the main part of the classes scrbook or scrreprt to be introduced by a prefix line (see layout options chapterprefix and nochapterprefix in subsection 3.1.2). For being consistent you 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 layout options appendixprefix or noappendixprefix, you would define in the document preamble:

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

In case you are going to change your mind and want to use the option appendixprefix at a later state, you will get an error message because of the already defined \appendixmore command. This prevents the definition mentioned above from changing the settings already set using chapterprefix and nochapterprefix.

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 subsection 3.6.2.

`\setbibpreamble{`*Präambel*`}`

The command `\setbibpreamble` can be used for setting a preamble for the bibliography. This can be achieved by placing the preamble before the command for issuing the bibliography. However, it doesn't have to be directly in front of it. For example it could be placed at the beginning of the document. Similar to the class options `bibtotoc` and `bibtotocnumbered` this command can only be successful if you haven't loaded a package which prevents this by redefining the `thebibliography` environment. Though the natbib package unauthorized uses internal macros of KOMA-Script it could be made sure 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 occuring 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.}
```

The `\bigskip` command makes sure that the preamble and the first reference are separated by a big skip.

Another usage of the preamble in the bibliography would be setting the references ragged right. Just put the preamble as follows:

```
\setbibpreamble{\raggedright}
```

You can have a look at the result in the bibliography of this guide.

`\setindexpreamble`

Similar to the bibliography you can use a preamble in 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 an 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 in the index:

```
\setindexpreamble{In \textbf{bold} printed page numbers
  are references to the definition of terms.\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 subsection 3.2.2). 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 more. They are part of older KOMA-Script versions. For compatibility reasons they can still be used in the new KOMA-Script release. There are new mechanisms and user interfaces however, which you should use instead. The reason for listing the obsolete macros in this documentation is to aid users with understanding old documents. Furthermore, package authors are free to use these macros in the future.

`\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. Please use element `caption` and `captionlabel` of the current KOMA-Script instead which are described in subsection 3.2.1.

`\descfont`

This macro sets the font for the optional item arguments of an `description` environment. Please use element `descriptionlabel` instead, which are described in section 3.2.1.

`\sectfont`

This macro sets the font which is used for all section headings, the main title an the highest level below `\part` in the table of contents. Use element `sectioning` instead, which is described in more detail in subsection 3.2.1.

3.9 Authors

The authors listed below are responsible for this chapter or have contributed to this chapter in different ways.

- Frank Neukam
- **Markus Kohm** <Markus.Kohm@gmx.de>
- Axel Sommerfeldt
- *João Canas Ferreira*
- *Georg Grandke*

4 Adapt Head and Foot with scrpage2

In chapter 2 this guide referenced a package to customise the head and foot lines of a document. This package scrpage2, the successor of scrpage, enables the user to create versatile head and foot layouts with less effort, due to a simple but powerful user interface.

The package's focus is its good integration into the whole KOMA-Script-bundle, thus it extents the base functionality of KOMA-Script perfectly. It is very flexible in either its layout and usage, compared to other packages often to be seen like fancyhdr[Oos00].

Neither scrpage2 nor scrpage must be used with KOMA-Script, the package can be used in any other document-class environment. New documents shall base upon the package scrpage2, but the package scrpage is always distributed for compatibility reasons. The current chapter points to scrpage2 only. Commands available in both versions are signed (see margin), if the commands of the new version differ from the former one an advice can be found in marked paragraphs.

scrpage

scrpage!

LaTeXs head and foot mechanism is a little complicated, thus a brief view in its depth is needed. Basically the LaTeX-kernel defines the chief page styles `empty` and `plain`. The latter writes only a page number in the foot, in contrast using `empty` results in blank head and foot. Besides, many document classes provide the style `headings`, which allows more complex style settings. The `headings` style often has a subvariant, the *my*-variant. In contrast to the `headings` style the `myheadings` switches off the automatic update of the running head, thus it is the users task keeping headings in sync with the current document content. A more detailed discription can be found in section 3.2.2.

Another important note is that some LaTeX-commands switch to the pagestyle `plain` for the current page, independent from what pagestyle was choosen by the author, consequently the document needs an appropriate `plain` pagestyle.

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 his own page style in combination with `scrplain`, the page style `scrplain` has to be activated first, i.e. `\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 another way to deal with automatic and manual headings. This way is more flexible and allows configurations usually difficult to

implement for unexperienced users. The required commands to work with the scrpage2 implementation are introduced at the end of 4.1.1 and the begin of 4.1.2.

4.1 Basic Functionality

4.1.1 Predefined Page Styles

`scrheadings`
`scrplain`

Package scrpage2 delivers an own pagestyle, named `scrheadings`. The command `\pagestyle{scrheadings}` activates this page style, likewise after activation an appropriate `plain` pagestyle is available. In this case *appropriate* meens that the `plain` page style is also configureable by the commands introduced in subsection 4.1.3, which, for example, configure the head and foot width. Neither the activation of `scrheadings` nor `scrplain` influences the mode of manual or automatic headings, see subsection 4.1.2.

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

The page styles include three boxes either in head and foot. The commands modifying the content of these boxes can be seen in Figure 4.1. Commands in the middle column modify the boxes content on both the odd and even pages.

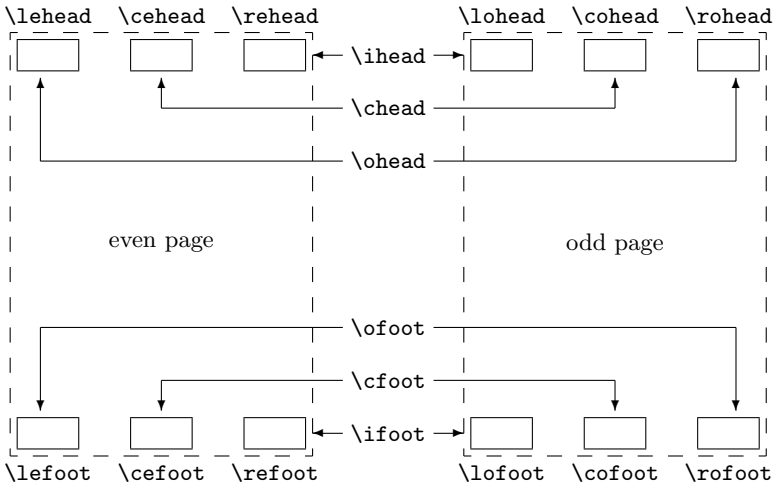


Figure 4.1: Commands for modification of pagestyles `scrheadings` and `scrplain` and their association to head and foot elements

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

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

The next example shows how to place both running head and page number in the head; the running head inside and the page number outside.

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

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

Using the commands which are associated with only one item allows more advanced settings.

example: Assuming one has the order to write an annual report for his company, he could use commands like this:

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

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

As mentioned above, there is an `plain`-pagestyle which corresponds to `scrheadings`. Since it should also be possible to adapt this style, the commands support an optional argument. Thus the contents of the appropriate field of the `plain`-pagestyle can be modified.

example: The position of the page number for the pagestyle `scrheadings` can be declared as follows:

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

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

`\clearscrheadings`
`\clearscrplain`
`\clearscrheadfoot`

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

example: If one wants to reset the page style to the default KOMA-Script settings, independent from 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 are required.

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

Of course, assuming a special configuration some of them can be dropped.

`\leftmark`
`\rightmark`

These two instructions make it possible to access the running headlines, which are normally meant for the left or for the right page. These two instruction are not made available by scrpage or scrpage2, but directly by the LaTeX kernel. If in this section running headline of the left side or the right page are mentioned, then the contents of `\leftmark` or `\rightmark` is meant.

In previous examples two commands already have been used, which haven't been introduced yet. Now, the description of these commands shall follow.

`\headmark`

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

`\pagemark`

This command returns the formatted page number. The formatting can be *scrpage*

controlled by \pnumfont introduced in subsection 4.1.3 or by \setkomafont if a newer version KOMA-Script is used, see 3.2.1.

useheadings

The package scrpage2 is meant primarily for the fact that the supplied styles are used or own styles are defined. However it can be necessary to shift back also to the style provided by the document class. It would be obvious to do this with \pagestyle{headings}, but this has however the disadvantage that commands \automark and \manualmark discussed in the following do not function as expected. For this reason one should shift back to the original styles using \pagestyle{useheadings}. Such a switching has then no effect on it whether one operates with manual or automatic running headlines.

4.1.2 Manual and Automatic Headings

Normally there is a *my*-version of the headings-style. If such a style is active, then the running headlines are updated no longer automatically. With scrpage2 another path is taken. Whether the running headlines are living or not, determines the instructions \automark and \manualmark. The default can be also already influenced while the loading of the package with the options automark and manualmark, see subsection 4.1.4 on Page 125.

\manualmark

As the name already clarifies, \manualmark switches off the updating of the running headlines. It is left to the user to provide for updating or for contents of the running headlines. For that purpose the instructions \markboth und \markright are available.

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

The macro \automark however activates the automatic updating. For the two parameters the designations of the document level are to be used, whose title in appropriate place is to appear. Valid values for the parameters are: chapter, section, subsection, subsubsection, paragraph und subparagraph.

The optional argument *right page* is meant understandably only for two-sided documents. In the one-sided case you should normally not use it. With the help of the option autooneside one can also adjust that the optional argument in one-sided mode is ignored automatically, see subsection 4.1.4.

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

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

restores the original behaviour. If one prefers one of the lower section levels in running head following can be used.

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

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

The data of the headings is set by the command `\markboth` for the top-most section levels, for the lower levels `\markright` or `\markleft` are used. These commands are called indirectly by the sectioning-commands. The macro `\markleft` is provided by the package scrpage2. Its definition is similar to `\markright` originating from the LaTeX kernel. Although `\markleft` is not defined as an internal command, the direct use is not recommended.

4.1.3 Formatting of Heading and Footing

In the previous paragraph it concerned mainly contentwise things. Is not sufficient naturally, in order to satisfy the formative ambitions. Therefore it is to turn in this paragraph exclusive therefore.

`\headfont`
`\pnumfont`

The command `\headfont` contains the commands which determine the formatting of head and foot lines. The style of the page number defines the command `\pnumfont`. *scrpage*

example: If, for example, one wants the head and foot to be typeset in bold sans-serif and the page number in a slanted style, then it can be used the configuration:

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

Since version 2.8p of the KOMA-Script classes a new configuration scheme is implemented. If scrpage2 is used together with these classes then it is recommended to setup font attributes the way described in subsection 3.2.1.

example: This interfaces implements the command `\setkomafont` in order to configure the font attributes. The previous definition 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 heading and footing line 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 up the value for the width of the page head or foot, *shift* is a longitudinal dimension for the shift of the appropriate item 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 head width

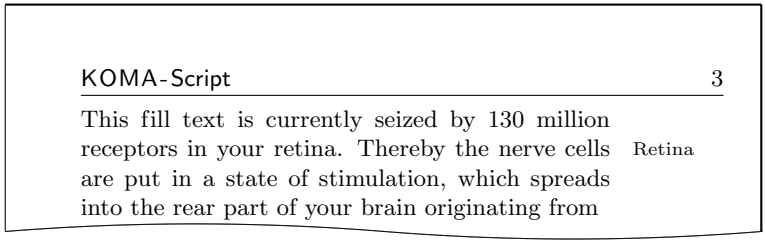
`foot` – the current foot width

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

example: The layout of the *LaTeX Companion*, the head-line projects into the margin, can be obtained with:

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

and looks on an odd page like this:



If the foot-line shall have the same width and alignment, then two ways are possible. The first simply repeats the settings for the foot-line:

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

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

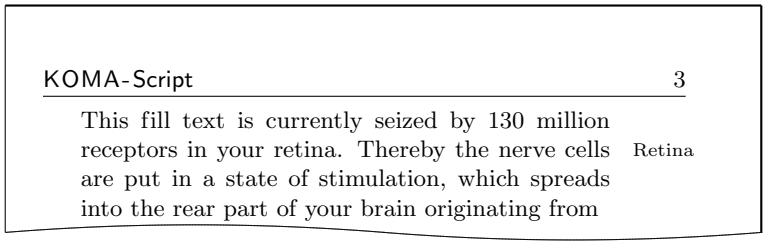
```
\setfootwidth[0pt]{head}
```

If no *shift* is indicated, i.e. without the optional argument, then the heading or the foot appears symmetrically on the page arranged. A value for the *shift* is determined automatically, which corresponds to the current page shape.

example: According to the previous example the optional argument is abandoned:

```
\setheadwidth{textwithmarginpar}
```

and looks on an odd page like this:



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

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

According to the configuration of head and foot there are commands to modify the lines above and below the head and foot.

`\setheadtopline` – configures the line above the head

`\setheadsepline` – configures the line below the head

`\setfootsepline` – configures the line above the foot

`\setfootbotline` – configures the line below the foot

The mandatory argument *thickness* determines, how strongly the line is drawn. The optional argument *length* accepts the same symbolic values as *width* with `\setheadwidth`, as also a normal length expression. As long as in the document *length* value was not assigned to the optional argument, the appropriate line length adapts automatically the width of the heading or the foot.

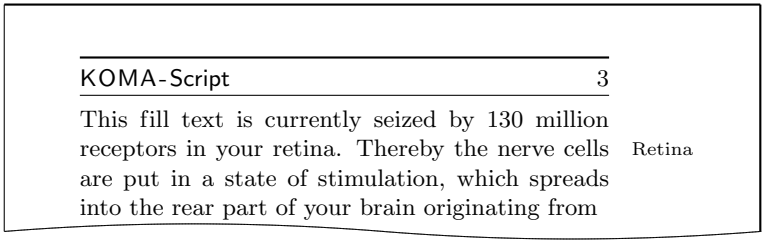
If one wants to restore this automatism, then the symbolic value `auto` in the length argument can be useful.

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

The arguments, here illustrated with 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 thicknes of the line keeps unchanged. This can be used in order to modify the length but not the thickness.

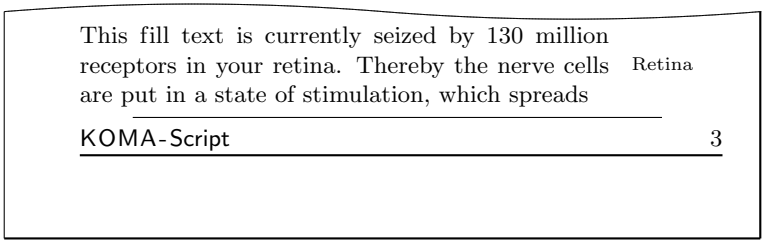
example: Shall, for example, the line above the head be a strong line with 2 pt and the line between head head and body with 0.4 pt, then it can be achieved with:

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



The automatic adjustment of the head and foot width is illustrated in the following example:

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



Now not everybody will like the alignment of the line above the foot, instead one will expect the line left-aligned. This can only be achieved with a global package option, which will be described together with other package option in the next subsection 4.1.4.

4.1.4 Package Options

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

These options intend whether the page-header or that page-footing are reckoned in with 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 2.4), if this package is loaded after scrpage2. Important is here that on use of a KOMA-Script class these options must be indicated for the document class and not for scrpage2, in order to obtain an effect.

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

A basic adjustment for the lines under and over heading and footing can be made with these options. These adjustments are considered then as default to all page styles defined with scrpage2. If one of these options is used, then a line thickness is used by 0.4 pt.

Since there is an appropriate `plain`-style to the page style `scrheadings`, with the `plain...`-options also the appropriate line of the plain style can be configured. These `plain`-options work however only, even if the corresponding option 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, heading or footing, is reckoned in with the text-area for the calculation of the type-area in case a line has been activated. This means, if with `headsepline` the separation-line between heading and text is activated, then the packge 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 automatism. That means that the package typearea must be loaded to 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 heading or footing line in the text-area.

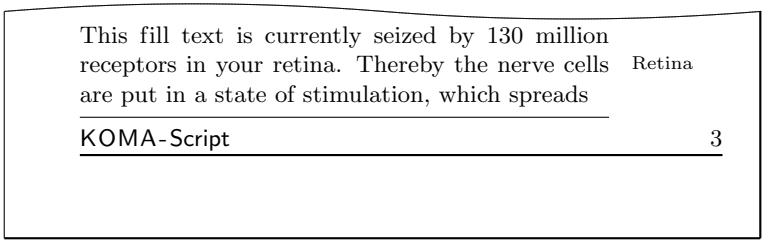
`ilines`
`clines`
`olines`

The definition of the line lengths can lead to an undesired adjustment, since the line is centered in the heading or footing area. With the package options presented here, this specification can be modified for all page styles defined with scrpage2. The option `ilines` sets the adjustment in such a way that the lines align to the inside edge. The option `clines` behaves like the standard adjustment 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 123.

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

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



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

`automark`
`manualmark`

These options set the adjustment at the beginning of the document whether automatic updating of the running headlines takes place. The option `automark` switches the automatic updating on, `manualmark` deactivates it. Without use of one of the two options the adjustment is preserved, which was valid while the loading of the package.

example: One loads 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 active yet. Futhermore `plain` means manual headings. If one now activates the page style `scrheadings` with

```
\pagestyle{scrheadings}
```

then the manual headings are always active.

If one uses the document class scrbook instead, 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 present. The marking-commands of scrbook continue to be used.

However, the use of

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

activates the automatic update of the running heading independent from the used document class. The option does not effect the used page style `plain` of the class scrreprt. The headings are not visible until the page style has been changed to `scrheadings`, `useheadings` or another self-defined page style with headings.

`autooneside`

This option ensures that the optional parameter of `\automark` will be ignored automatically in one-side mode. See the explanation of the command `\automark` in subsection 4.1.2.

`komastyle`
`standardstyle`

These options determine the kind of the pre-defined page style `scrheadings`. The option `komastyle` takes up a configuration like the KOMA-Script classes. This is the default for KOMA-Script classes and can this way also be set for other classes.

A configuration expected from the standard classes can be defined using option `standardstyle`. Automatically the option `markuppercase` will be activated, but only if option `markusedcase` is not given.

`markuppercase`
`markusedcase`

The package scrpage2 has to modify internal commands, which are used by document structuring commands, in order to get the function of `\automark` working. 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 represents the headings.

Option `markuppercase` shows scrpage2 that the document class uses uppercase letters. If the document class does not represent the headings in uppercase letters 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 behaviour of the document class.

`nouppercase`

In the previous paragraph about `markuppercase` and `markusedcase` it has been already stated that some document classes represent the running headings in uppercase letters using the commands `\MakeUppercase` or `\uppercase`. Setting the option `nouppercase` allows to disable these both commands. The option `nouppercase` only is valid as long page styles defined by scrpage2 are used, including `scrheadings` and its appropriate `plain` page style.

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

example: If a document uses the standard class book, but the uppercase headings are not desired, then the preamble of the 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 non-uppercase headings.

4.2 Defining Own Page Styles

4.2.1 The Interface for Beginners

Now one would not always be bound only to the provided page styles, but moreover there will be the wish to define own page styles. Sometimes there will be a special need, since a specific *Corporate Identity* requires the declaration of own page styles. The easiest way to deal with is:

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

scrpage

The parameters have the following meaning:

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

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

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

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

HC – contents of the centered box in the page head

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

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

FC – contents of the centered box in the page foot

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

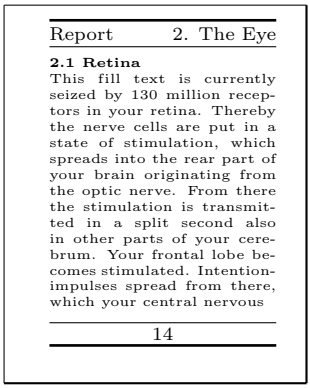
The command `\deftripstyle` surely represents the simplest possibility of defining page styles. Unfortunately also restrictions are connected with since in a page range with one via deftripstyle defined page style no modification of the lines above and below heading and footing can take place.

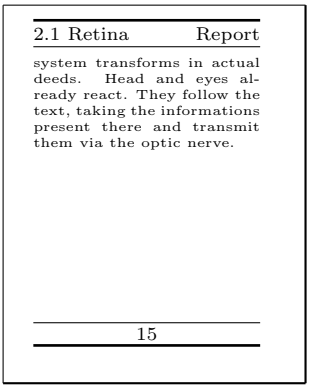
example: Assuming a two-side layout, where the running headings are placed inside. Furthermore the document title, here "'Report"', shall be placed outside in the head, the page number is centered in the foot.

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

Shall moreover the lines above the head and below the foot be drawn with a thickness of 2 pt and the text body be separated from head and foot 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, how they can be defined with `\deftripstyle`, are rare according to experience. Either a professor requires that the thesis looks in such a way like its own, and who wants to contradict it seriously, or a company would like that the half financial department emerges in the page footing. No problem, the solution is:

```
\defpagestyle{name}{head-definition}{foot-definition}
\newpagestyle{name}{head-definition}{foot-definition}
\renewpagestyle{name}{head-definition}{foot-definition}
\providepagestyle{name}{head-definition}{foot-definition}
```

scrpage

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

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

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

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

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

The syntax of the four commands is explained on command `\defpagestyle` examplary.

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

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

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

It can be seen, that head and foot declaration are identical. The parameters have the following meaning:

ALL – above line length: (head = outside, foot = separation line)

ALT – above line thickness

EP – definition for *even* pages

OP – definition for *odd* pages

OS – definition for *one-side* layout

BLL – below line length: (head = separation line, foot = outside)

BLT – below line thickness

If the optional line-parameters are omitted, then the line behaviour keeps configurable by the commands introduced in subsection 4.1.3. In the old version scrpage the line-parameters are mandatory. *scrpage!*

The three elements *EP*, *OP* and *OS* are boxes with the width of page head or foot respectively. The definition occur on the left side in the box, thus the space between two text elements has to be stretched using `\hfill`, in order to write the first text element on the left edge *and* the second text element on the right edge.

```
{\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 no horizontal space. Only this way the middle text is really centered.

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

This and more examples of the expert's interface and other commands provided by scpage2 follow now in the final example.

example: This examples uses the document class scrbook, which means that the default page layout is two-side. While the loading of the package scrpage2 the options `automark` and `headsepline` are given. The first switches on the automatic update of running headings, the second determines that a separation line between head and text-body is drawn in the `scrheadings` page style.

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

The expert's 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 head and 0 pt for the separation-line between head 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)
}
```

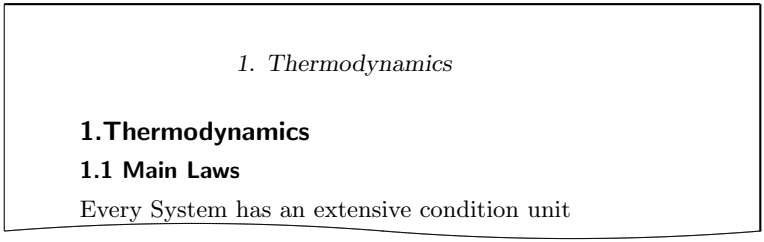
At the document's begin the page style `scrheadings` is choosen. The command `\chapter` starts a new chapter and sets automatically the page style for this page to `plain`.

The command `\chead` shows how running headings can be represented even on a `plain` page. Running headings on chapter start-pages are not usual, since in this case the page looses its emphasis-character. Basically it is more important to show that

a new chapter starts here than that a section of this page has a special title.

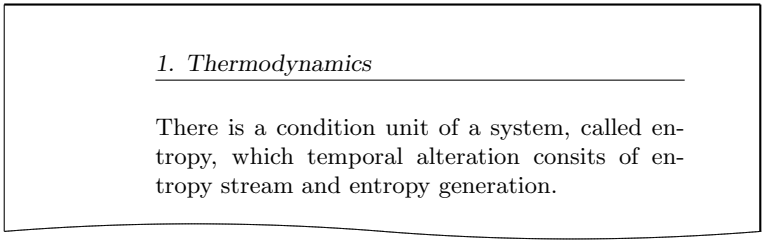
Instead of `\leftmark` one would expect the use of `\rightmark` in the parameter of `\chead`, since the chapter starts on an even page. But, because of internal LaTeX definitions, this does not work. It only returns an empty string.

```
\begin{document}
\pagestyle{scrheadings}
\chapter{Thermodynamics}

\chead[\leftmark]{}

\section{Main Laws}
Every system has an extensive condition unit called
Energy. In a closed system the energy is constant.
```

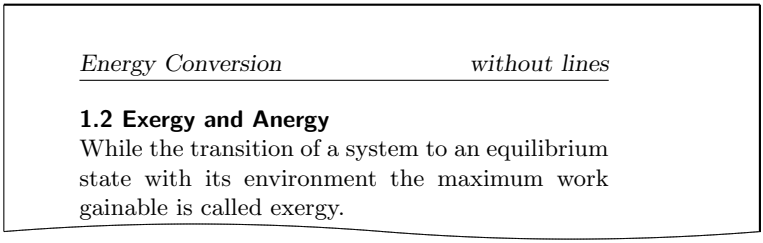


After starting a new page the page style `scrheadings` is active and thus the separation line below the heading is visible.

```
There is a condition unit of a system, called entropy,
which temporal alteration consits of entropy stream
and entropy generation.
```



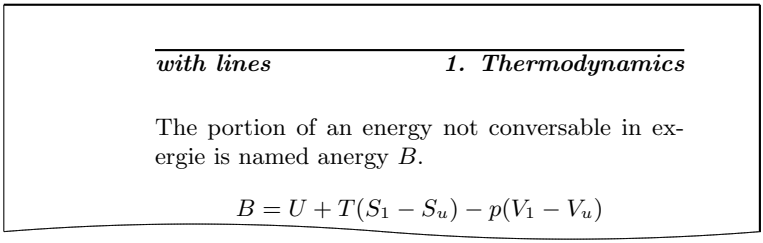
After switching to the next page, the automatic update of the running headings is disabled using `\manualmark`. The page style `withoutLines` becomes active. Since no line parameters are given in the definition of this page style, the default configuration is used, which draws a separation line between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




```
\manualmark
\pagestyle{withoutLines}
\section{Exergy and Anergy}\markright{Energy Conversion}
While 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 conversable in exergie
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, scrpage2 reads after initialisation the file `scrpage.cfg`. This file can contain a set of user-defined page styles, which many projects can share.

4.3 Authors

The authors listed below are responsible for this chapter or have contributed to this chapter in different ways.

- Markus Kohm <Markus.Kohm@gmx.de>
- **Jens-Uwe Morawski**

5 Week-Day and Time Using scrdate and scrtime

There are two packages at KOMA-Script to improve and extend the handling of date and time. So you have not only the standard commands `\today` and `\date`. Like all the other packages from KOMA-Script bundle these two packages may be used not only with KOMA-Script classes but also with standard and many other classes.

5.1 The Name of the Current Day of Week Using scrdate

`\todaysname`

You should know, that you may get the current date with `\today` in a language dependend spelling. scrdate offers you the command `\todaysname`. This results in the name of the current day of week in a language dependend spelling.

example: At your document you want to show the name of the week-day at which the `dvi`-file was generated using LaTeX. To do this, you write:

```
I've done the \LaTeX-run of this document on a \todaysname.
```

This will result in e.g.:

I've done the LaTeX-run of this document on a Sunday.

Tip: The names of the week-days are saved in capitalization. So the first letter is a capital letter, all the others are small letters. But at some languages you also need the names with a first letter in lower case. You may achieve this using the standard LaTeX command `\MakeLowercase`. You simply have to write `\MakeLowercase{\todaysname}`.

`\nameday{`*name*`}`

You should know, that you may change the output of `\today` using `\date`. In a analogous way you can change the output of `\todaysname` using `\nameday` into *name*.

example: You're changing the current date into a fix value using `\date`. You are not interested in the name of the day, but you want to show, that it is a workday. So you set:

```
\nameday{workday}
```

After this the previous example will result in:

I've done the LaTeX-run of this document on a workday.

Package scrdate knows the languages english (english, american, USenglish, UKenglish and british), german (german, ngerman and austrian), french, italian, spanish and croatian. If you want to configure it for other languages, see `scrdate.dtx`.

At current implementation it doesn't matter, if you're loading scrdate before or after german, ngerman, babel or similar packages. The current language will be setup at `\begin{document}`.

To explain it 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 english caption names. At `scrdate.dtx` you will find the description of the scrdate-commands for defining the names.

5.2 Getting the Time with Package scrtime

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

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

`\thistime*` works in the same way as `\thistime`. The difference between both is that the value of the minute using `\thistime*` is not preceded with zero when its value is less than 10, thus using `\thistime` the minute-value has always two places.

example: The line

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

results for example in:

Your train departs at 13:40.

or:

Your train departs at 23:09.

In contrast to the prevous example a line like:

```
This day is already \thistime*[\ hours and\ ] minutes old.
```

results in:

This day is already 13 hours and 40 minutes old.

or:

This day is already 12 hours and 25 minutes old.

`\settime{`*Time*`}`

`\settime` sets the output `\thistime` and `\thistime*` on the value of *Time*. Afterwards the optinal 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` has been used.

5.3 Authors

The authors listed below are responsible for this chapter or have contributed to this chapter in different ways.

- Frank Neukam
- Markus Kohm <Markus.Kohm@gmx.de>
- Axel Sommerfeldt

6 The New Letter Class scrlttr2

v2.8q

Since the June 2002 release, KOMA-Script contains a completely rewritten letter class. Although part of the code is identical to the classes from chapter 3, letters are quite different from articles, reports, books, and such. That alone justifies a separate chapter about the letter class. But there is another reason for a chapter on scrlttr2. The class has been redeveloped from scratch. And the user interface differs from every other class the author knows. You can dispute the sense or nonsense of that new usage concept. The author just thinks that this new user interface offers some advantages.

6.1 Looking Back on the Old Letter Class

With the June 2002 release the old letter class scrlettr becomes obsolete. It should better not be used for new letters. There is no development on the old letter class anymore, 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 can run it through LaTeX some times, like that:

```
latex scrlettr.dtx
latex scrlettr.dtx
latex scrlettr.dtx
```

You get 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 functionality still remains in the new class. Without that compatibility option, the user interface and the defaults will be different. More detail on this option is provided in sections 6.2 and 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 the possibility to accompany options by values. You do not only need a lot less options, but maybe even fewer optional arguments. You will see that when discussing the `letter` environment in subsection 6.4.3. 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 will be understood. But to keep the number of forward references low, it is reasonable to describe them this early.

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

The possibility to change many options after loading the class is a special feature of the scrlttr2 class. The `\KOMAoptions` command serves this purpose, taking options and their values as arguments. You can list multiple options, separated by commas, 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`, there will be an explicit remark 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 Page Layout Options

In contrast to the old scrlettr class, but in correspondence 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 controls the package. The necessary options will be explained in this section.

`paper=`*format*

This option defines the paper format. Theoretically, all paper formats the typearea package knows about are supported. But you have to leave out the suffix `paper` when entering a value. So, for letter format you would use the

value `letter`. The formats of the ISO A, B, C, and D series must be entered with small letters, e.,g. `a4` for ISO A4. See also section 2.5.

Although every paper size supported by typearea can be used, several formats may result in unexpected results on the first page of a letter by now. That is not a matter of the class concept, but there exist only parameter sets for ISO A4 at this time. Unfortunately, there are no general rules to define the placement of the address field and similar for an arbitrary paper size. But it is possible to define additional parameter sets. See subsection 6.2.7 for more information.

`BCOR`=*length*
`DIV`=*value*
`headlines`=*count*

The options for the divisor, the binding correction, and the number of headlines will be translated directly into the corresponding options of the typearea package. If the options are set using `\KOMAoptions` and not as class options, the `\typearea` command from the typearea package will be used instead. See section 2.4.

`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 head and foot of the first page independently of the following pages. If, as a result, the foot of the first page would reach into the text area, this text area would automatically be made smaller using the `\enlargethispage` macro. On the other hand, if the text area should become larger, supposed the foot on the first page allows that, you could use this option. At best, some more text would fit on the first page. See also the description of pseudo length `firstfootvpos` at subsection 6.4.2. This option can take the standard values for simple switches, as listed in Table 6.1. Default is `false`.

6.2.3 Other Layout Options

In this subsection, you will find all options that have influence on the layout in general, except page layout. Strictly speaking, all page layout options (see 6.2.2) are also layout options, and vice versa for some of them.

Value	Description
`true`	activates the option
`on`	activates the option
`false`	deactivates the option
`off`	deactivates the option

Table 6.1: Standard values for simple switches in scrlttr2

`twoside`

From the author's point of view, double-sided letters do not make much sense. Therefore, the option `twoside` only partially switches to double-sided layout. You get the possibility to have different margins on left and right pages, but it is not used. So this option really means *activate the possibilities of a double-sided document, but stay with the one-sided layout as far and as long as possible.* This option can take the standard values for simple switches, as listed in Table 6.1. Default is `false`.

By the way, double-sided letters are not supported, because they seem unreasonable.

`cleardoublepage`=*style*

If you want pages inserted by the `\cleardoublepage` command to just contain a page number in head and foot or to be empty, this can be accomplished with this option. There are three different styles supported:

`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.

The page styles will be discussed in subsection 6.3.2. Default is `standard`.

`headsepline`
`footsepline`

These two options insert a separator line below the head or above the foot, resp., on consecutive pages. In the lingo 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 6.1. Default is `false`. If one of the

options is used without value, like in the declaration above, this evaluates as `true`, so the separator line will be activated. When used as a `\documentclass` option, the typearea package will be called with the option `headinclude` or `footinclude`, resp. (see 2.4).

`mpinclude`
`mpexclude`

These two options of the typearea package should not be used with the scrlttr2 class, because the first page in particular does not take this option into account. To anticipate any complaints, a warning will be issued when this option is used. If you feel adventurous you could try how these options, especially `mpinclude`, interact with other class options.

`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 effects the page layouts `headings` and `plain`. It also effects 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:

`bot` – page number in foot, horizontal position not changed

`botcenter` – page number in foot, centered

`botcentered` – same as `botcenter`

`botleft` – page number in foot, left justified

`botmiddle` – same as `botcenter`

`botright` – page number in foot, right justified

`center` – page number centered horizontally, vertical position not changed

`centered` – same as `center`

`false` – no page number

`foot` – same as `bot`

`footcenter` – same as `botcenter`

`footcentered` – same as `botcenter`

`footleft` – same as `botleft`

`footmiddle` – same as `botcenter`

`footright` – same as `botright`

`head` – page number in head, horizontal position not changed

`headcenter` – page number in head, centered

`headcentered` – same as `headcenter`

`headleft` – page number in head, left justified

`headmiddle` – same as `headcenter`

`headright` – page number in head, right justified

`left` – page number left, vertical position not changed

`middle` – same as `center`

`no` – same as `false`

`off` – same as `false`

`right` – page number right, vertical position not changed

`top` – same as `head`

`topcenter` – same as `headcenter`

`topcentered` – same as `headcenter`

`topleft` – same as `headleft`

`topmiddle` – same as `headcenter`

`topright` – same as `headright`

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. It is a matter of tradition. Apparently it has been easier for a secretary to operate the carriage return lever twice than setting 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 of modern typesetting, should take advantage of their possibilities like other documents do. Under these circumstances, letters should also be typeset using paragraph indentation and justification. The author of scrlttr2 shares this point of view, but nevertheless refrains from imposing to many restrictions upon the user.

As as reaction to many serious requests, scrlttr2 offers the possibility to mark paragraphs not only by indentation of the first line, but alternatively by vertical skip. You can choose between full or half a line of vertical space. When using paragraph spacing, it seems 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:

`false` – paragraph indentation instead of vertical space; the last line of a paragraph may be arbitrarily filled.

`full` – 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; the last line of a paragraph may be arbitrarily filled.

`off` – same as `false`

`on` – same as `full`

`true` – same as `full`

Default is `false`.

6.2.4 Font Options

Fonts options are any options with influence on the size of the base font or of fonts for letter parts. 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*

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 the same. This option can only be used with `\documentclass`, not with `\KOMAoptions`. Default is `12pt`.

6.2.5 Options for Letterhead and Address

The scrlttr2 class offers lots of extensions for the design of the letterhead. There are also options for address formatting, extending the possibilities of the standard letter class, although these features could already be found in the now obsolete scrlettr class.

`fromalign`

This option defines the placement of the from address in the letterhead of the first page. At the same time, this option serves as a switch to activate or deactivate the extended letterhead options. If these extensions are deactivated, some other options will have no effect. This will be noted with the respecting options. Possible values for `fromalign` are:

`center` – return address centered; an optional logo will be on top of the extended return address; letterhead extensions will be activated.

`centered` – same as `center`

`false` – 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.

`middle` – same as `center`

`no` – same as `false`

`off` – same as `false`

`right` – right justified return address; an optional logo will be left justified; letterhead extensions will be activated.

Default is `left`.

`fromrule`

This option is part of the letterhead extensions (see option `fromalign` above). It allows you to place a horizontal line within the return address. The possible values are:

`afteraddress` – line below the return address

`aftername` – line right below the sender's name

`below` – same as `afteraddress`

`false` – no line

`no` – same as `false`

`off` – same as `false`

`on` – same as `afteraddress`

`true` – same as `afteraddress`

`yes` – same as `afteraddress`

Default is `false`. You can not activate more than one line at a time.

`fromphone`

This option is part of the letterhead extensions (see option `fromalign` above). It defines whether the phone number will be part of the return address. This option can take the standard values for simple switches, as listed in Table 6.1. Default is `false`.

`fromfax`

This option is part of the letterhead extensions (see option `fromalign` above). It defines whether the facsimile number will be part of the return address. This option can take the standard values for simple switches, as listed in Table 6.1. Default is `false`.

`fromemail`

This option is part of the letterhead extensions (see option `fromalign` above). It defines whether the email address will be part of the return address. This option can take the standard values for simple switches, as listed in Table 6.1. Default is `false`.

`fromurl`

This option is part of the letterhead extensions (see option `fromalign` above). It defines whether the URL will be part of the return address. This option can take the standard values for simple switches, as listed in Table 6.1. Default is `false`.

`fromlogo`

This option is part of the letterhead extensions (see option `fromalign` above). It defines whether the logo will be part of the return address. This option can take the standard values for simple switches, as listed in Table 6.1. Default is `false`.

`addrfield`

This option defines whether an address field will be set. Default is to use the address field. This option can take the standard values for simple switches, as listed in Table 6.1. Default is `true`.

`backaddress`

This option defines whether a return address for window envelopes will be set. Default is to use the return address. If the address field is suppressed (see option `addrfield`), there will be no return address either. This option can take the standard values for simple switches, as listed in Table 6.1. Default is `true`.

`subject`

This option serves two purposes: First, you can choose if your subject should have a title, given by the *subject* variable (see Table 6.8). Second, you can choose if the subject should be set before or after the opening. Possible values for this option are:

`afteropening` – set subject after opening

`beforeopening` – set subject before opening

`titled` – add title to subject

`untitled` – do not add title to subject

Defaults are `beforeopening` and `untitled`.

`locfield`

scrlttr2 places a field with additional sender attributes next to the address field. This can be used for bank accounts or similar. 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 subsection 6.2.7). 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 subsection 6.4.4. Possible values for this option are:

`narrow` – small sender supplement field

`wide` – large sender supplement field

Default is `narrow`.

`foldmarks`

This option activates fold marks for two or three panel folding of the letter. The exact placement of the fold marks for three panel letter fold depends on user settings or the `lco` file, resp. (see subsection 6.2.7). The folding need not result in equal sized parts. This option can take the standard values for simple switches, as listed in Table 6.1. Default is `true`, which implies setting the fold marks.

`numericaldate`

This option toggles between the standard, language-dependent date presentation and a short, numerical one. KOMA-Script does not provide the standard presentation. It should be defined by packages like german, babel, or isodate. The short, numerical presentation will be produced by scrlttr2 itself. This option can take the standard values for simple switches, as listed in Table 6.1. Default is `false`, which results in standard date presentation. In the now obsolete scrlettr class, this was achieved using the `orgdate` option, but with opposite results.

`refline`

With the scrlttr2 class, the head, foot, address, and sender attributes may extend beyond the normal type area to the left and to the right. This option defines if that also applies to the reference line. Normally, the reference line contains at least the date, but it can hold additional data. Possible values for this option are:

`narrow` – reference line restricted to type area

`wide` – reference line corresponds to address and sender attributes

Default is `narrow`.

6.2.6 Format Options

Format options are those, which influence form or format of the output and do not belong to another section. You might also call them the *miscellaneous options*.

`draft`

This option toggles between the final and the draft version of a document. In particular, enabling the `draft` option activates little black boxes that will be drawn at the end of overfull lines. For the unpracticed eye, these boxes ease the identification of paragraphs that need manual improvement. When the `draft` option is disabled, there will be no such boxes. This option can take the standard values for simple switches, as listed in Table 6.1. Default is `false`, as usual. But I strongly recommend enabling the `draft` option when designing a letter, as for every other document.

6.2.7 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 foot 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`, `\@addtoplength` and `\addtolengthplength` (see subsection 6.3.4).

There are already some `lco` files included in the KOMA-Script distribution. The `DIN.lco`, `DINmtext.lco`, `SNleft.lco`, and `SN.lco` files serve to adjust KOMA-Script to different layout standards. They are well suited as templates for your own parameter sets. The `KOMAold.lco` file, however, serves to improve compatibility with the old letter class scrlettr. Because it contains internal commands not open to package writers, you should not use them as a template for your own `lco` files. You can find a list of predefined `lco` files in Table 6.2.

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 `LEGAL.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 me. You will find current addresses in the latest volume of the KOMA-Script-News or in section 1.8.

`\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.

But 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 gets 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}
```

But one letter should use the `DINmtext` variant, with the address field placed more to 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\\
    Fichtenstraße 63\\68535 Edingen-Neckarhausen}
  \LoadLetterOption{DINmtext}
  \opening{Hello,}
```

Since construction of the page does not start before the `\opening`, it is sufficient to load the `lco` file before the `\opening` command. In particular, this need not be done before `\begin{letter}`. So the changes made by loading the `lco` file are local to the corresponding letter.

If an `lco` file is loaded via `\documentclass`, it may nevertheless take the name of an option, provided it is an option that does not take an argument. However, it would be possible to give the name `fromalign=left.lco` to an `lco` file. It will get loaded every time the `\documentclass` option `fromalign` is used with the value `left`. Admittedly, this is quite academic. Of course you can use this feature only if your operating and file system support this kind of file names. Otherwise you have to choose another file name and add the corresponding option, if needed.

example: You do not want to enter your sender address every time, so you create an `lco` file with the necessary data, like this:

```
\ProvidesFile{mkohm.lco}[2002/02/25 letter class option]
\setkomavar{fromname}{Markus Kohm}
\setkomavar{fromaddress}{Fichtenstra\ss e 63\\
                         68535 Edingen-Neckarhausen}
```

Please note that the German sharp s, "ß", was entered using the TEX macro `\ss`, because right 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.

But if you would always use the same input encoding, you could also include it into your `lco` file. This would look like this:

```
\ProvidesFile{mkohm.lco}[2002/02/25 letter class option]
\RequirePackage[latin1]{inputenc}
\setkomavar{fromname}{Markus Kohm}
\setkomavar{fromaddress}{Fichtenstraße 63\\
                         68535 Edingen-Neckarhausen}
```

There is one problem with this usage: you cannot load this `lco` file later in your document. If you want to have letters with different senders in one document, you should refrain from loading packages in your `lco` file.

Let us further assume that I always typeset letters using the preset `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 6.2 you 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. The Swiss standard SN 101 130 defines the address field to be placed 8 mm from the right paper edge, so the headline and the sender attributes will be set with the same small distance to the paper edge. This also applies to the reference line when using the `refline=wide` option (see

lco file	Description and features
DIN	parameter set for letters on A4 size paper, complying with German standard DIN 676; suitable for window envelopes in the sizes C4, C5, C6, and C6/5 (C6 long).
DINmtext	parameter set for letters on A4 size paper, complying with DIN 676, but using an alternate layout with more text on the first page; only suitable for window envelopes in the sizes C6 and C6/5 (C6 long).
KOMAold	parameter set for letters on A4 size paper using a layout close to the now obsolete 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.
SN	parameter set for Swiss letters with address field on the right side, according to SN 010 130; suitable for Swiss window envelopes in the sizes C4, C5, C6, and C6/5 (C6 long).
SNleft	parameter set for Swiss letters with address field on the left side; suitable for Swiss window envelopes with window on the left side in the sizes C4, C5, C6, and C6/5 (C6 long).

Table 6.2: The predefined lco files

subsection 6.2.5). If you have this kind of problem, create your own lco file that loads SN first and then changes toaddrhpos (see subsection 6.4.3) to a smaller value. Please also reduce toaddrwidth accordingly.

\LetterOptionNeedsPapersize{*option name*}{*paper size*}

As mentioned in subsection 6.2.2, there are only parameter sets and lco files for A4 sized paper. But in every lco file distributed with KOMA-Script you will find a \LetterOptionNeedsPapersize command so that you will be warned when using another *paper size*. The first argument is the name of the lco file without the ".lco" suffix. Second argument is the paper size the lco file is designed for.

If several lco files are loaded, the \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 lco files yourself, you may just forget about this option and skip the example.

example: Supposed 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. Then, 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 the paper area.

```
\ProvidesFile{paper=a5.lco}[2002/05/02 letter class option]
\LetterOptionNeedsPapersize{paper=a5}{a5}
\@setplength{tfoldmarkvpos}{\paperheight}
\@setplength{bfoldmarkvpos}{\paperheight}
```

Besides, the placement of the foot 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". This seems too complicated? Only before you used it the first time. Anyway, how often do you write letters not using your standard format?

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 aren't assigned to a certain part of the document such as the letterhead or the letter body. Several of these properties have already been mentioned in section 6.2.

6.3.1 Font Selection

Commands for defining, extending and querying the font of a specific element can be found 3.2.1. These commands work exactly the same in scrlttr2. The elements which

Element	Description
`backaddress`	back address for a window envelope
`descriptionlabel`	label, i.e. the optional argument of `\item`, in a `description` environment
`fromaddress`	sender's address in the letterhead
`fromname`	sender's address in the letterhead if different from `fromaddress`
`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 which is inserted with `\pagemark`
`subject`	subject in the opening of the letter
`title`	headline in the opening of the letter

Table 6.3: Alphabetical list of the elements, whose font can be changed in scrlttr2 using the commands `\setkomafont` and `\addtokomafont`

can be influenced in this way are listed in Table 6.3.

6.3.2 Page Style

One of the general properties of a document is the page style. Please refer also the subsection 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 using the macro `\opening`.

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 subsection 6.2.3).

headings is the page style for automatic page headings of subsequent pages. The inserted marks are the sender's name from the variable `fromname` and the subject from the variable `subject` (see subsection 6.4.1 and subsection 6.4.6). At which position these marks and the page numbers are placed depends on the option `pagenumber` (see subsection 6.2.3). Apart from that, the author can change these marks after `\opening` manually.

myheadings is the page style for manual page headings of subsequent pages. This is very similar to `headings`, but here the marks are set by the author using the commands `\markboth` and `\markright`.

Page styles are also influenced by the option `headsepline` or `footsepline` (see subsection 6.2.3). The page style beginning with the current page is switched with `\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 subsection 3.2.1. For header and footer the same element is used which you can name either `pagehead` or `pagefoot`. The element for the page number within the header or footer is named `pagenumber`. Default settings are listed in Table 3.4. Please have also a look at the example in subsection 3.2.2.

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

Please refer to subsection 3.2.2. The function of `\cleardoublepage` in scrlttr2 depends on the option `cleardoublepage` which is described in more detail in subsection 6.2.3.

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 subsection 3.2.1). At this point we now are going to 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. The main difference between a command and a variable is that a command usually triggers an action whereas a variable only consist of plain text. Furthermore a variable can have an additional description, which can be set and issued.

This section only gives a short introduction to the term variable. 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 6.4.

```
\newkomavar[description]{name}
\newkomavar*[description]{name}
\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 (see subsection 6.4.5). The *description* and the content of the variable are added at the end of the reference fields. The starred version `\newkomavar*` is similar to the one without star with a subsequent call of the command `\addtoreffields`. Thus, the starred version automatically adds the variable to the reference fields.

example: Suppose you need an additional field for a direct dialling. You can define this field either with

```
\newkomavar[Direct dialling]{myphone}
\addtoreffields{myphone}
```

or more concise with

```
\newkomavar*[direct dialling]{myphone}
```

When you define a variable for the reference fields you should always give them a description.

Variable	Bedeutung
backaddress	back address for window envelopes
backaddressseparator	separator within the back address
ccseparator	separator between title of additional recipients and additional recipients
customer	customer number
date	date
emailseparator	separator between e-mail name and e-mail address
enclseparator	separator between title of enclosure and enclosures
faxseparator	separator between title of fax and fax number
fromaddress	sender's address without its name
frombank	sender's bank account
fromemail	sender's e-mail
fromfax	sender's fax number
fromlogo	commands for inserting the sender's logo
fromname	complete name of the sender
fromphone	sender's telephone number
fromurl	one url of the sender
invoice	invoice number
location	more details of the sender
myref	sender's reference
place	place
placeseparator	separator between place and date
phoneseparator	separator between title of telephone and telephone number
signature	signature beneath the ending of the letter
specialmail	special mail
subject	subject
subjectseparator	separator between title of subject and subject
title	letter title
toname	complete name of recipient
toaddress	address of recipient without its name
yourmail	date of recipient's mail
yourref	recipient's reference

Table 6.4: Alphabetical list of all supported variables in scrlttr2

```
\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 "Connexion". Thus you add the description

```
\setkomavar*{myphone}{Connexion}
```

or you can put both in one command:

```
\setkomavar{myphone}[Connexion]{-\,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 don't 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 that the user can access the content or the

description of a variable. This is specially important when you have defined a variable which is not added to the reference fields. Using the command `\usekomavar` you have access to the content of the variable *name* whereas the starred version `\usekomavar*` gives you the description.

The commands `\usekomavar` and `\usekomavar*` are, similar to all commands where a starred version or an optional argument exists, not fully expandable. Nevertheless if used within `\markboth`, `\markright` or similar commands you needn't insert a `\protect` before using them. Of course this is also true in scrpage2 for `\markleft`. But they couldn't be used at commands like `\MakeUppercase`. `\MakeUppercase{\usekomavar{`*name*`}}` would result in `\usekomavar{`*NAME*`}`. To avoid this problem you may use commands like `\MakeUppercase` as optional argument of `\usekomavar` or `\usekomavar*`. So you'll get the upper case content of a variable using `\usekomavar[\MakeUppercase]{`*name*`}`.

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

v2.9i

With these commands you may check wether or not the expanded contents or description of a variable is empty. The *true* argument will be executed if the contents or description is empty. Otherwise the *false* argument will be executed. The variant with star sign handles the description of a variable, the variant without star the contents.

Maybe it is important to know, that the contents or description of the variable will be expanded using `\edef`. If this results in spaces or unexpandable macros like `\relax`, it is not empty. This is even true, if the use of the variable would not result in any output.

Both variants of the command must not be used at the argument of `\MakeUppercase` or 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`. But they are robust enough to be used at 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 for each of them. For LATEX lengths, which are addressed with `\newlength` skip registers are used. If all of these registers are in use you can not define additional lengths. The letter class scrlttr2 only for the first page would use up more than 20 of such registers. LATEX itself already uses 40 of these registers. The typearea package needs some of them too. Thus approximately a quarter of the precious registers is already in use. That's the reason why lengths specific to letters in scrlttr2 are defined with macros instead of lengths. The drawback of this is that computations withs macros is somewhat more complicated than with real lengths.

`\@newplength{`*name*`}`

This command defines an new pseudo length. This new pseudo length can clearly identified with its *name*. It is made sure that every *name* can only be given once.

Since the user in general doesn't define its own pseudo lengths it is not intended as a user command. Thus, it can not be used within a document, but for example within a `lco` file.

`\useplength{`*name*`}`

Using this command you can access the value of the pseudo length with the given *name*. This is the only user command in connection with pseudo lengths. Of course this command can also be used with a `lco` file.

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

While you can combine a length with a factor 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* assign the value of a *pseudo length*.

Using the command `\setlengthtoplength` you can assign a multiple of a *pseudo length* to a real *length*. Instead of putting the *factor* in front of the *pseudo length* it is given as an optional argument. You should also use this command when you want to assign a 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 similar to that. It adds a multiple of *pseudo length* to *length*.

`\@setplength[`*factor*`]{`*pseudo length*`}{`*value*`}`
`\@addtoplength[`*factor*`]{`*pseudo length*`}{`*value*`}`

Using the command `\@setplength` you assign a multiple of a *value* to a *pseudo length*. The *factor* is given as an optional argument. The command `\@addtoplength` adds the *value* to a *pseudo length*. For assigning or adding the multiple of *pseudo length* to another pseudo length the command `\useplength` is used within *value*. To substract the value of a *pseudo length* from another *pseudo length* a minus sign or `-1` as *factor* is used.

Since the user in general doesn't define its own pseudo lengths it is not intended as a user command. Thus, it can not be used within a document, but for example within a `lco` file.

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 6.1 a letter document consists of a preamble, the individual letters and the closing.

The preamble comprises all settings that in general concern all letters. Most of them can be overwritten in the settings of the individual letters. The only setting which can not be changed within a single letter is the font size (see option `fontsize` in subsection 6.2.4). General settings such as the loading of packages and the setting of options should be placed before `\begin{document}` only. All settings that comprise the setting of variables or other text features should be done after `\begin{document}`. This is particularly recommend when the babel package (see [Bra01]) is used or language dependend variables of scrlttr2 are to be changed.

The closing usually consists only of the standard closing for the end of a document. Of course you can also insert additional comments at this point.

As shown in Figure 6.2 every single letter itself consists of an introduction, the letter body and the closing. In the introduction all settings for only this letter are defined. It is important that this introduction always ends with `\opening`. Similarily the closing starts with `\closing`. If needed both arguments *opening* and *closing* can be left empty. Nevertheless both commands have to be used and must have an argument.

There are further settings that can be changed between the individual letters. These settings have an effect on all subsequent letters. For reasons of maintainability of your letter documents it is not recommended to use further general settings with limited validity between the letters.

As already mentioned you can use all settings in the preamble of a letter document in the preamble of the individual letters apart from the font size. Therefore you will not find more detailed explanations for the possible settings at this point. Please refer to section 6.4.

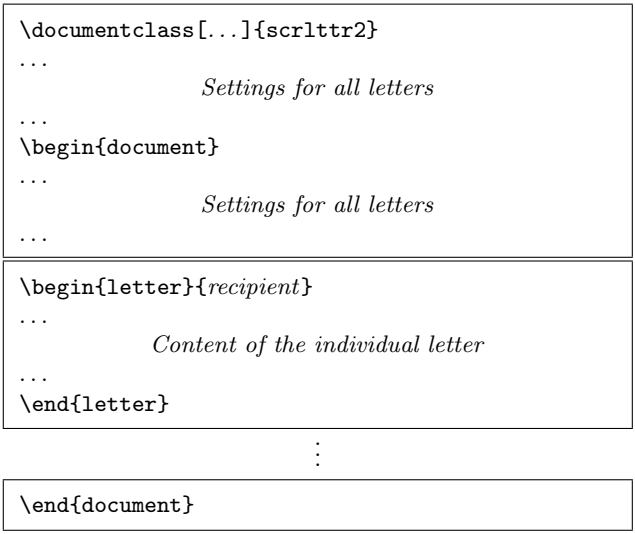


Figure 6.1: General structure of a letter document with several individual letters (the structure of a single letter is shown in Figure 6.2)

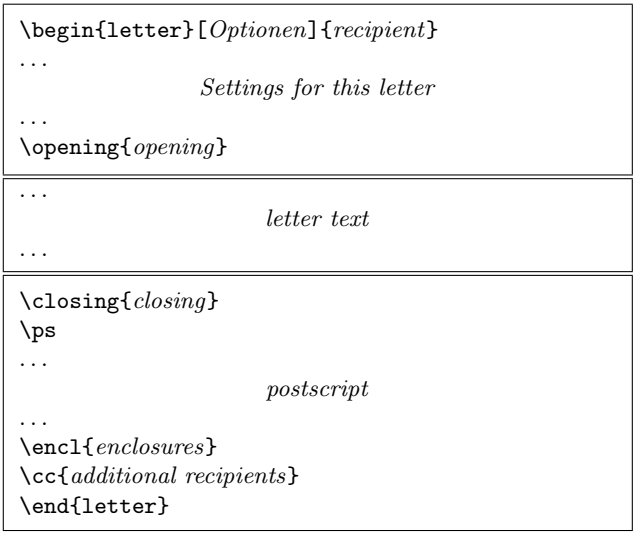


Figure 6.2: General structure of a single letter within a letter document (see also Figure 6.1)

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 prelims of the letter; it consists of several different parts.

6.4.1 The Letterhead

The term letterhead here refers to all of the sender's data and is printed above the recipient's address. It usually is expected to have all this set via the pagestyle setting. The earlier version of letterclass scrlettr worked that way. But with scrlttr2, the letterhead has gotten independent of the pagestyle setting and is run 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, that page that holds the letterhead, is set by `empty`.

`firstheadvpos`

The pseudolength `firstheadvpos` gives the distance between the top of the sheet and the letterhead. This value is set differently in the predefined `lco`-files. A typical value is 8 mm.

`firstheadwidth`

The pseudolength `firstheadwidth` gives the width of the letterhead. This value is set differently in the predefined `lco`-files. While this value usually depends on the paperwidth and the distance between the left side of the sheet and the address' field, it was the type area width in `KOMAold`.

`fromname`
`fromaddress`
`fromphone`
`fromfax`
`fromemail`
`fromurl`
`fromlogo`

These variables give all statements concerning the sender necessary to create the letterhead. Which variable will be used to build the letterhead in the end can be chosen by use of the letterhead extensions (see option `fromalign` in paragraph 6.2.5) and the options given there. The variables `fromname`, and

name	label
fromemail	\usekomavar*{emailseparator}% \usekomavar{emailseparator}
fromfax	\usekomavar*{faxseparator}% \usekomavar{faxseparator}
fromname	\headfromname
fromphone	\usekomavar*{phoneseparator}% \usekomavar{phoneseparator}
fromurl	\usekomavar*{urlseparator}% \usekomavar{urlseparator}

Table 6.5: The sender's predefined labels for the letterhead

fromaddress, and fromlogo will be set in the letterhead without their label; the variables fromphone, fromfax, fromemail and fromurl will be set with it's label. The labels are taken from Table 6.5.

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 not strictly as a line break. Paragraphs, vertical distances and the like usually aren't allowed within the sender's address declaration. One has to have very good knowledge of scrlttr2 to use things like those mentioned above, intelligently.

It's possible, by the way, to load an external picture to use it as a logo. In this case then put the content of fromlogo to a \includegraphics-command. The corresponding package (see [Car99b]) of course has to be given with the letter declaration (see paragraph 6.3.5).

phoneseparator
faxseparator
emailseparator
urlseparator

With these variables, hyphens are defined. If applicable they are used with the sender's data in the letterhead (see Table 6.5). As a feature, they are labeled and used in the sender's details of the letterhead. To look up the predefined labels and their contents, see Table 6.6.

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

Table 6.6: predefined labels and contents of hyphens for sender's datas in the letterhead

`\firsthead{`*construction*`}`

Mostly scrlttr2 and its variables offer enough possibilities to create a letterhead. In very rare situations one may wish to have more freedom to create. In those situations one will have to do without predefined letterheads, which could have been chosen via options. Instead one is to create freely. Therefore one has to define the preferred *construction* with the command `\firsthead`. Within `\firsthead` and with the help of the `\parbox`-command (see [Tea99a]) 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. And doing this of course the *construct* may use varibles with the help of `\usekomavar`.

6.4.2 The Footer

As the first page holds a letterhead of its own it holds a footer of its own too. And, same as with the letterhead, it will be printed independent of the pagestyle settings but with the use of `\opening`.

`firstfootvpos`

This pseudolength gives the distance between the footer and the upper border of the sheet. This value is set different in the predefined `lco`-files. Also it takes care of preventing text jutting into the footer area. If needed, it can help to shorten the textheight on the first page using `\enlargethispage`. Likewise and if it is needed the textheight can be extended with help of the option `enlargefirstpage`. This way, the distance between text area and the first page's footer can be reduced to the value `\footskip`. See also subsection 6.2.2.

`firstfootwidth`

This pseudolength gives the width of the letter's firstpage footer. The value is set in dependency of the pseudolength `firstheadwidth` in the `lco`-files.

`\firstfoot{`*construction*`}`

The first page's footer is preset to empty. But with the `\firstfoot` command it is possible to give definitions the same way as with defining the letterhead with `\firsthead`.

example: As the first page's footer, you may want to set the content of the variable `bank` (the bank account). The double backslash shall be exchanged with a comma at the same time:

```
\firstfoot{%
  \parbox[b]{\linewidth}{%
    \centering\def\\{, }\usekomavar{frombank}%
  }%
}
```

For the hyphen you might define a varible of your own if you like. Consider that has been left to the reader as an exercise.

Nowadays it has become very common to create a proper footer to have some balance to the letterhead. This can be done like this:

```
\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 a height like that it might happen that you have to shift its position. You do this with the pseudolength `firstfootvpos`, which is descripted above in this section.

6.4.3 The Address

The term address here usually refers to name and address of the recipient. But there are extensions. The first extension is the way of distribution e.g. "special mail". If window envelopes are used the return address is another extension since it has to show up within the envelope's window. The address immediately follows beneath the letterhead.

`toaddrvpos`
`toaddrhpos`

These pseudolengths give the distance between address window of a window envelope and the upper and left border of the sheet. They are set different in the predefined `lco`-files. The smallest value is `toaddrvpos` at `DINmtext`. It can happen easily that the letterhead juts into the address window field. Whether the address window field will be set at all depends on giving the option `addrfield` (see paragraph 6.2.5).

`toaddrwidth`

This pseudolength gives the width of the address window. It is set differently in the predefined `lco`-files according to the different standards that exist. Typically it is something between 70 mm and 100 mm.

`toaddrindent`

Sometimes it is wanted that the address isn't set exactly at the beginning of the left border of the address window, but a little indented to the right. The value of the indentiation can be set with the pseudolength `toaddrindent`. But typically it is 0 pt.

`backaddress`
`backaddressseparator`
`backaddrheight`

The sender's address often is printed in one line with small letters above the address if window envelopes are used. This line, called return address, is separated by a horizontal line, and usually build automaticly of the variables `fromname` und `fromaddress`. Double backslashes set within this return address will be exchanged by the settings of the variable `backaddressseparator`. Predefined here is a comma with a whitespace. The height this return address field can take is defined with the pseudolength `backaddrheight`. The value typically is 5 mm and set in the predefined `lco`-files. If the return address line is to be set or not depends on putting the options `addrfield` and `backaddress` in the letter declarations (see paragraph 6.2.5).

`specialmail`
`specialmailindent`
`specialmailrightindent`

Between the return address and addressee, the distribution like specialmail can be set optionally. It's set then if `specialmail` has a value. The both pseudolengths `specialmailindent` and `specialmailrightindent` determine the alignment. They determine the left and right indentation. The predefined `lco`-files have `specialmailindent` set as the elastic value `\fill`, while the pseudolength `specialmailrightindent` is set to 1 em. Thus distribution term is set 1 em off the right border of the address window.

`toname`
`toaddress`

These two variables that give name and address of the addressee usually aren't set directly by the user. Instead scrlttr2 takes it directly off the *recipient*-argument of the `letter`-environment. See also the hint about the sender's address in paragraph 6.4.1.

`\begin{letter}[`*options*`]{`*address*`}`

The letter environment is but one key point of the letter class. Especially with scrlttr2 it is possible to run the letter environment with additional *options*. They will be run according to the `\KOMAoptions`-commands. *address* will be given to the letter environment as an obligatory parameter. Here the double backslash works as a hyphen between parts of *address*. Those parts then are printed as single lines. The double backslashes here shouldn't be understood as strict line break commands. Paragraphs, vertical space, etc., aren't accepted within the address. They can lead to unexpected effects and error messages. By the way it is the same way with the standard letter class. The `letter`-environment itself doesn't start the letter. The letter is started with the `\opening`-command.

`\AtBeginLetter{`*command*`}`

It is possible with LaTeX to process additional *commands* while running the LaTeX-process at determined times. For that matter, the `\AtBeginDocument` or `\AtEndOfClass` exist; those points along the processing of LaTeX are called *hooks*. The class scrlttr2 adds another hook to this, that can be used with `\AtBeginLetter`. Originally those hooks are made for package or class authors, which can be understood by the fact that the descriptions for those hooks aren't documented in [Tea99a], but in [Tea99b]. With the letter class there could come some useful applications for `\AtBeginLetter` as the following example may show:

example: Given one has to set more letters than just one with one document and uses commands within the letter to set a form. The numbers of the questions in that form should be generated automatically with the help of a counter. Since, in contrary to the page numbering, that counter is not known by scrlttr2, it wouldn't be reset at each start of a new letter. Given each form holds ten questions, then question 1 would not be question 1 in the fifth letter, but instead

number 41. The solution is in having scrlttr2 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 given either braced (see [RNH02]) in the declaration of the letter (see paragraph 6.3.5 and Figure 6.1) with `\makeatletter` and `\makeatother`, or loaded as package of one's own, or as `lco`-file (see paragraph 6.2.7).

6.4.4 The Sender's Extensions

Often, especially with business letters, the space for letterhead or pagefoot seems to be too tight to put all you want to have set. 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 *sender's extension*

`locwidth`

This pseudolength `locwidth` declares the width of the sender's extensions. Its value is set typically 0 pt in the predefined `lco`-files. This value takes on a special position. It does not mean that the sender's extension has no width. Instead it's actual width is set with the `\opening` when paperwidth, address window width, and distance between left border of the sheet and address window is given. With it the option `locfield` (see paragraph 6.2.5) is taken into account.

`location`

The content of the sender's extension is determined by the variable `location`. To set this variable's content it's allowed 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 be in use already (see paragraph 6.2.5).

example: Given you like to put the names of your partners, manager, or court of jurisdiction with the sender's extension, you can do this like:

```
\KOMAoptions{locfield=wide}
\setkomavar{location}{\raggedright
  \textbf{Partners:}\\
  \quad Hugo Mayer\\
  \quad Bernd Müller\\[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 in horizontally. Sender details like those mentioned in the above example, can be given together with the common sender address with your own `lco`-file.

6.4.5 The Business Line

Especially with business letters, a line can be found that gives initials, dial code, customer number, invoice number, or a reference to a previous letter. In this manual this line is called *business line*. The business line can consist of more than just one line and is set only if one of those variables mentioned above is given. Only those fields will be set that are given. To set a seemingly empty field, one needs to give as value at least a white space or `\null`. If you want to have your letter without a business line, then just the variable `date` will be set instead.

`refvpos`

This pseudolength gives the distance between the upper border of the sheet and the business line. It's value is set differently in the predefined `lco`-files. Typical values are between 80,5 mm and 98,5 mm.

`refwidth`

This pseudolength gives the width that is available for the business 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 paragraph 6.2.5).

`refaftervskip`

This pseudolength gives the vertical space that has to be inserted beneath the business line. The value is set in the predefined `lco`-files. It effects directly the textheight of the first page. A typical value is something between one and two lines.

`place`
`placeseparator`

As said before in the introduction of this paragraph, the business line can be left out. This happens if all variables of the business line are empty with the exception of the variable for the date. In this case, the content of `place` and `placeseparator` will be set, followed by the content of `date`. The predefined content of the `placeseparator` is a comma followed by a white space. If this variable `place` has no value then the hyphen remains unset also. The predefined subject of `date` is `\today` and depends on the setting of the option `numericaldate` (see paragraph 6.2.5).

`yourref`
`yourmail`
`myref`
`customer`
`invoice`
`date`

These variables are typical for business line fields. To find out about their meaning, see table 6.4 on Page 161. Each variable covers a predefined label that can be seen at Table 6.7. The field width that belongs to each variable, adjusts itself automatically to its label and content.

6.4.6 The Title and the Subject Line

Business letters most often carry a subject line. The subject line indicates in short of what that letter is about. Usually the subject should be short and

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

Table 6.7: predefined labels of the business line's typical variables. The content of the makros depend on language.

precise and not run across several lines. The letter may also carry a title. Titles find usage with unregular letters like an Invoice or a Reminder.

`title`

With scrlttr2 a letter can carry an additional title. The title becomes centered and set with letter size `\LARGE` right after and beneath the business line. The predefined font setup for this element (`\normalcolor\sffamily\bfseries`) can be changed with help of the interface described in paragraph 3.2.1. Font size declarations are allowed.

example: Given you are to write a reminder. Thus you put the title:

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

This way the addressee will recognize a reminder as such.

`subject`
`subjectseparator`

In case a subject should be set then you have to determine the variable `subject` fit. Depending on what the option `subject` is set to a label can be put in front of the subject issue; also the vertical position of the subject issue can be changed (see subsection 6.2.5). To see the predetermined labels look at Table 6.8. The predefined value of `subjectseparator` is a colon followed by a white space.

The subject line is set in a separate type face. To change this use the user interface descripted in paragraph 3.2.1. For the element `subject` the predetermined type face in scrlttr2 is `\normalfont\normalcolor\bfseries`.

name	label
subject	\usekomavar*{subjectseparator}% \usekomavar{subjectseparator}
subjectseparator	\subjectname

Table 6.8: Predefined Labels Of The Subject's Variables.

example: Given 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 that way:

```
\setkomavar{subject}[Subject ]{%
  organization's internals}
```

or easier:

```
\setkomavar{subject}[]{%
  about organization's internals}
```

More than just that you may want to have set the subject line not only bold but also with sans serif types:

```
\addtokomafont{subject}{\sffamily}
```

As you can see, it's really easy to solve this problem.

6.4.7 Further Issues

In this paragraph variables and settings are listed which couldn't be assigned to any other paragraph of the letter declaration but somehow belong to this chapter.

`tfoldmarkvpos`
`bfoldmarkvpos`

The letterclass scrlttr2 all in all has 3 fold marks. The one in the middle serves to halve the paper and is therefore printed always in the middle of paperheight. The position of the upper fold mark, seen from the upper sheet's border, is

determined by the pseudolength `tfoldmarkvpos`; the lower one is determined by the pseudolength `bfoldmarkvpos`. All three fold marks do not serve to exactly fold to a standard 3 panel letter fold. Instead the idea is to have the paper folded so that the address field is seen in the window of a window envelope. The settings therefore are different in the predefined `lco`-files. A special case is `DINmtext`. In this case the envelope format C6/5 (also called "C6 long") is necessary. Letters produced in this format aren't compatible with neither format C 5 nor C 4.

`foldmarkhpos`

This pseudolength gives the distance between all of the three fold marks and the sheet's left border. Usually it's 3,5 mm. In case you work with a printer with a broader unprintable left margin this value can be changed in your own `lco`-file. The length of the upper and lower fold mark is the same and 4 mm long. The thickness of all three is 2 pt. At the moment there are no plans to change this value. If the fold marks will be set at all depends on the option `foldmarks` (see paragraph 6.2.5).

`frombank`

This variable at the moment takes on a special position. Not in use internally up to this moment it can come into usage if one e.g. wants to have set 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 of the cases to create letterheads and foots for those pages that follow the first letter page. This even more since you can additionally change the sender's statements with `\markboth` and `\markright` that scrlttr2 uses to create the letter head. With the term "follow up pages" in this manual all pages are meant excepted the first letter page. The commands `\markboth` and `\markright` can be used all together with pagestyle `myheadings`. If the package scrpage2 is used this, of course, is valid also for pagestyle `scrheadings`. Then too the command `\markleft` is available.

At times one wants to have more freedom with creating letter head or foot of the follow up pages. Then one has to let go of the possibilities of predefined letter heads or foots, that could have been chosen via option. Instead one

is free to create it just the way one wants to have them set. Therefore one is to create the desired *construction* with use of the command `\nexthead` or `\nextfoot` respectively. Within `\nexthead` und `\nextfoot` for example you can have several boxes side by side or one beneath the other by use of `\parbox`-command (see [Tea99a]). With this a more advanced user should have no problems with creating letter heads of foots of his own. With *construction* of course you can make use of the variables of `\usekomavar` too.

6.5 The Text

In contrast to an article, a report or a book the letter 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 other things. The letter texts mainly consist of an opening and the main text. Thereupon follows the signature, a postscript and different listings.

6.5.1 The Opening

In the early days of computer generated letters the programs didn't 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 the most important command in scrlttr2. For the user it seems that only the opening will be typeset, but the command also typesets the folding marks, headings, address field, subject, the page foot and others. That means: 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 subsection 3.6.3 for the main document classes. Therefore it 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 subsection 3.6.4 for the main document classes.

6.5.4 Margin Notes

Margin notes are quite uncommon in letters. Therefore the option `mpinclude` is not supported by scrlttr2. However, scrlttr2 is equipped with all mechanisms mentioned in subsection 3.6.5 for the main document classes. Therefore it 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 that means: emphasize text sparingly. Even letters should be readable and a letter where each word is typeset in an other font is indeed unreadable.

The class scrlttr2 is equipped with all mechanisms mentioned in subsection 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 this phrase. Sometimes this is a sentence like "This letter has been generated automatically.". Sometimes a sentence like this will even be used as signature. Thereupon can follow a postscript and some listings.

6.6.1 Closing

The closing consists of three parts. Besides the closing phrase there are a hand-written inscription and the signature, an explanation for the inscription.

`signature`

The variable `signature` includes an explanation for the inscription. Their content is pre-defined as `\usekomavar{fromname}`. The explanation can con-

sist of multiple lines. The lines should be separated by a double backslash. Paragraphs in the explanation are not permitted.

`\closing{`*closing phrase*`}`

The command `\closing` does not only typeset the closing phrase, but moreover it typesets the phrase followed by a vertical space and the content of the variable `signature`. The closing phrase can consists of multiple lines, but paragraphs are not permitted.

`sigindent`
`sigbeforevskip`
`\raggedsignature`

Closing phrase, inscription 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 in pseudo-length `sigindent`. In the default `lco` file this length is set to 0 mm.

The command `\raggedsignature` defines the alignment inside the box. In the default `lco` file the command is either defined as `\centering` (all besides `KOMAold`) or `\raggedright` (`KOMAold`). In order to get flushright or flushleft alignment inside the box the command can be redefined in the same way as `\raggedsection` (see subsection 3.6.2).

Between closing phrase and signature a vertical space is inserted. The height of this space is defined in the pseudo-length `sigbeforevskip`. It defaults to 2 lines. In this space you can write your inscription.

example: You are writing as directorate of a society a letter to all members. Moreover you want in one respect elucidate that you are writing in the name of the board of directors and on the other hand you want indicate your position in the board of directors.

```
\setkomavar{signature}{John McEnvy\\
 {\small (Vize-President ``The Other Society'')}}
\closing{Regards\\
  (for the board of directors)}
```

Certainly you can set the variable `signature` in your private `lco` file. Usally you should prefer to define the variable in the letter preamble.

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 usual to use a postscript because this was the only way to add information which one had forget 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 a good possibility to underline again the most important or sometimes the less important things of this letter.

This instruction just switches to the postscript. Therefore 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:" you have to type the acronym inside the command. By the way, this acronym is been written without a full stop. The acronym is neither be typeset 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 gets the *distribution list* as argument. If the content of the variable `ccseparator` isn't empty then the name and the content of the variable is inserted prior to *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 lines by a double backslash.

example: You want to indicate that your letter is adressed to all members of a society and to the board of directors:

```
\cc{%
  the board of directors\\
  all society members\\
```

Write this instruction 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. There is just a single difference, the enclosures starts with the name and the content of the variable `enclseparator`.

6.7 Language Support

The document class scrlttr2 supports many languages. These are German `ngerman` (`german` for old German orthography), `austrian` for Austrian, English (`english` without specification whether American or British should be used, `american` and `USenglish` for American, `british` and `UKenglish` for British), French, Italian, Spanish, Dutch and Croatian.

6.7.1 Language Selection

If the package babel is used one can switch between languages with the command `\selectlanguage{`*language*`}`. Other packages like german and ngerman also define this command. As a rule the language selection takes place when such a package is loaded.

There is one thing more to mention about language packages. The package french re-defines not only the terms of subsection 6.7.2. The package even re-defines the command `\opening`, since it assumes that the definition of the standard letter is used. Therefore the package french spoils the definition of the scrlttr2 class. I think this is a fault of the french package.

If one utilizes the babel package in order to switch to language `french` and the package french is installed too, then the same problems occur since babel employs definitions from the french package. If the package french is not installed then there are no problems.

Additionally there is no problem if for babel instead of `french` other languages like `acadian`, `canadien`, `francais` or `frenchb` are chosen. Therefore I recommend `\usepackage[frenchb]{babel}` in order to select french.

Other languages can cause these problems too. Currently there are no problems known with the babel package for the german language and the various english language selections.

Most babel features at babel or original language definition files of babel (e.g. `frenchb.ldf`), which may cause problems with other packages or classes, can be switched off. This is a great advantage of babel. So if you have a problem, try to switch of such features or ask the authors of babel.

There are no problems known using the german or ngerman package for german selection with old or new orthography.

```
\captionsenglish
\captionsUSenglish
\captionsamerican
\captionsbritish
\captionsUKenglish
\captionsgerman
\captionsngerman
\captionsaustrian
\captionsfrench
\captionsitalian
\captionsspanish
\captionsdutch
\captionscroatian
```

If one switches the language then using these commands the language-terms from subsection 6.7.2 are re-defined. 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
```

The date in its numerical representation (see option `numericaldate` in subsection 6.2.5) will be written depending on the selected language. Some examples

Command	Date example
\dateenglish	1/12/1993
\dateUSenglish	12/1/1993
\dateamerican	12/1/1993
\datebritish	1/12/1993
\dateUKenglish	1/12/1993
\dategerman	1. 12. 1993
\datengerman	1. 12. 1993
\dateaustrian	1. 12. 1993
\datefrench	1. 12. 1993
\dateitalian	1. 12. 1993
\datespanish	1. 12. 1993
\datedutch	1. 12. 1993
\datecroatian	1. 12. 1993.

Table 6.9: Language-dependent forms of the date

can be found in Table 6.9.

6.7.2 Language-Dependent Terms

As usual in LaTeX, the language-dependent terms are defined by commands. These commands are re-defined when one switches the language.

```
\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 cutomization. How this can be done is described in subsection 6.7.3. The definitions become active at `\begin{document}`. Therefore they are not available in the LaTeX preamble. Thus they even can not be re-defined there. In Table 6.10 the default settings for `english` and `ngerman` can be found.

6.7.3 Defining Language Terms

Normally one has to change or define the language terms of subsection 6.7.1 in a way that additionally to the available terms even the new or re-defined terms are defined. Some packages like german or ngerman re-define those settings when they are loaded. These packages re-define the definitions in a way that spoils all previous private settings. That is also the reason, why scrlttr2 delays its own changes with `\AtBeginDocument` until `\begin{document}`. The user can also use `\AtBeginDocument` or re-define the language terms after `\begin{document}`. The class scrlttr2 provides some commands for defining language terms.

Command	english	ngerman
\bankname	Bank account	Bankverbindung
\ccname	cc	Kopien an
\customername	Customer no.	Kundennummer
\datename	Date	Datum
\emailname	Email	E-Mail
\enclname	encl	Anlagen
\faxname	Fax	Fax
\headfromname	From	Von
\headtoname	To	An
\invoicename	Invoice no.	Rechnungsnummer
\myrefname	Our ref.	Unser Zeichen
\pagename	Page	Seite
\phonename	Phone	Telefon
\subjectname	Subject	Betrifft
\wwwname	Url	URL
\yourmailname	Your letter of	Ihr Schreiben vom
\yourrefname	Your ref.	Ihr Zeichen

Table 6.10: Default settings for languages `english` and `ngerman`

```
\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 *language* to a *term* . The *term* is always a command. The commands differ dependent from whether a *term* is already define in *language* or not.

If *language* is not defined then `\providecaptionname` writes only a message in the log-file. This happens only once for each language. If *language* is defined but *term* isn't defined yet, then it will be defined using *definition*. The *term* will not be re-defined if the *language* already has a definition. Instaed a log-message will be written.

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 log-message will be written. If *term* is not yet defined in *language* then it will be defined with *definition*. If *term* already exists in *language* then this results in an error message.

The command \renewcaptionname requires an existing definition of *term* in *language*. In this case *term* for *language* will be re-defined according to *definition*. If neither *language* nor *term* exist or *term* is unknown in a defined language then a error message will be given.

The class scrlttr2 itself employs \providecaptionname in order to define the commands in subsection 6.7.2.

example: If you prefer "Your message of" instead of "Your letter of" you have to re-define the definition of \yourmailname.

```
\renewcaptionname{english}{\yourmailname}{%
  Your message of}
```

Since only available terms can be re-defined you have to delay the command until \begin{document} using \AtBeginDocument. Furthermore you will get an error message if there is no package used that defines a language selection command for *english*.

6.8 Address Files and Circular Letters

When people write circular letters they mostly dislike to type the many addresses. The class scrlttr2 and its predecessor scrlettr as well provide basic support for it. Currently there are plans for a much enhanced support.

\adrentry{*Lastname*}{*Firstname*}{*Address*}{*Telephone*}{*F1*}{*F2*}{*Comment*}{*Key*}

The class scrlttr2 supports to use address files. These address files contain address entries. Each entry is an \adrentry command with eight parameters as can be seen above. The file extension of the address file has to be .adr.

```
\adrentry{McEnvy}
         {Flann}
         {Main Street 1\\ Glasgow}
         {123 4567}
         {male}
         {}
         {niggard}
         {FLANN}
```

The 5th and 6th element, F1 and F2, can be used freely, for example for the gender, the academic grade, the birthday or the date the person has joined a society. The last parameter *Key* should only consist of uppercase letters in order to not interfere with other TEX or LATEX commands.

example: Mr. McEnvy is one of your most important business partners, but every day you get a reclamation from him. Before long you don't want to bother typing his boring address again and again. Here scrlttr2 can help. All your business partners have an entry in your partners.adr address file. If you now have to answer Mr. McEnvy again, then you can save typing as can be seen below:

```
\input{partners.adr}
\begin{letter}{\FLANN}
  Your today's reclamation  ...
\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 where you are running LATEX or configure your system to find the file in a special directory.

\addrentry{*Lastname*}{*Firstname*}{*Address*}{*Telephone*}{*F1*}{*F2*}{*F3*}{*F4*}{*Key*}

Over the years people objected that the \adrentry has only two free parameters. Since TEX supports at maximum nine parameters per command, there now exists a new command called \addrentry, note the additional "d". This command supports four freely definable parameters, that means one parameter more than \adrentry, since the comment parameter has been replaced with the fourth free parameter. The numbers of parameters is the only difference between both commands. Thus you can mix both entry types in one address file.

There are some packages which can employ adr files. For example adrconv by Axel Kielhorn can be used to create address lists from adr files. But it has currently no support for command \addrentry. The only choice is to extent the package yourself.

Besides the simple acccess to addresses the address files can be easily used in order to write circular letters. Thus there is no complicated data-base system and its connection to TEX required.

example: Suppose you are member of a society and want write a 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 ...
      \closing{Regards,}
    \end{letter}
  }
  \input{members.adr}
  \end{document}
```

If the address file contains `\addrentry` statements too, then even an additional definition for `\addrentry` is required.

```
\renewcommand*{\addrentry}[9]{
  \begin{letter}{#2 #1\\#3}
    \opening{Dear members,}
    our next general meeting will be on monday
    August 12, 2002. The topics of the meeting are ...
    \closing{Regards,}
  \end{letter}
}
```

In this simple example the extra freely definable parameter is not used.

With some additional programming one can let the contents depend on the address data. For this the free parameters can be used.

example: Suppose the fifth parameter of the `\adrentry` command contains the gender of a member (`m/f`). The sixth parameter contains what member subscribtion has still not been discharged by the member. If you would like to write a more personal reminder then the next example can help you.

```
\renewcommand*{\adrentry}[8]{
```

```
    \ifcase #6
    % #6 greater than 0?
    % this selects all members with open subscription
    \else
      \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 depending on attributes of the letter's recipient. The number of attributes is only restricted by number of the 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-Paket by Axel Kielhorn (see [Kie99]) is needed. This package contains interactive LaTeX documents which help to create those lists.

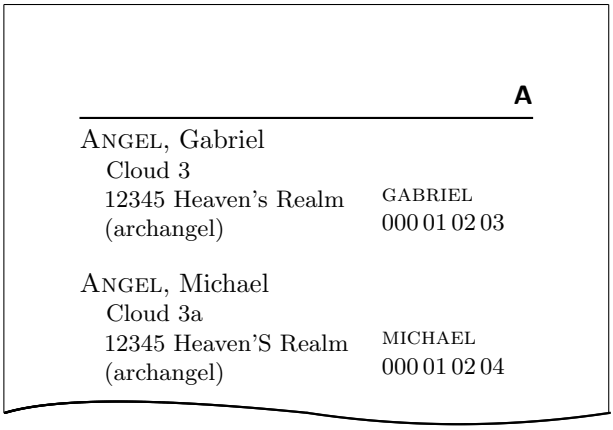
The address files have to be sorted already in order to get sorted lists. It is recommended to sort and separate the entries by the initial letter of *Lastname*. As a seperator the commands `\adrchar` and `\addrchar` can be used. These commands will be ignored if the address files are utilized in scrlettr2.

example: Suppose the following 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}
         {Fichtenstra\ss e 63\\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 heading is created by the `\adrchar` command, see the definition in `adrdir.tex`.

More about the adrconv package can be found in its documentation. There you should even find informations if the version of adrconv supports already 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. But you will encounter some differences in the output, since the page layout of both classes is not the same. 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 heading, which again has dependence to the font size. That was an unambiguous design fault.

There is no compatibility regarding the defined lengths in scrlettr. If one has changed the page layout of scrlettr then those statements have to be deleted or commented out. In some cases the modification of length can cause an error, since this length is not defined anymore. You should delete or comment these modifications as well.

The old letter example:

```
\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 annouce...
    \closing{Happy \TeX{}ing}
  \end{letter}
\end{document}
```

works as expected only by option `KOMAold`.

The next step is that the layout of the old letter will not be used anymore, but the old commands should still be available. If for example one wants the layout of `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 annouce...
```

```
    \closing{Happy \TeX{}ing}
  \end{letter}
\end{document}
```

Using more options this way you have further influence on the layout, but a more inherent change is really recommended.

That is to replace all old commands with its new representations and omit the option `KOMAold`. It can help to read the contents of `KOMAold.lco`. In that file the old commands are defined using the new ones.

```
\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 annouce...
    \closing{Happy \TeX{}ing}
  \end{letter}
\end{document}
```

This example shows also the possibility to change the alignment of the signature by re-defining the command `\raggedsignature`. This is recommended if the width of the real signature is greater than the signature-definition of the command `\setkomavar{signature}{...}`.

6.10 Authors

The authors listed below are responsible for this chapter or have contributed to this chapter in different ways.

- **Markus Kohm** <Markus.Kohm@gmx.de>
- *Harald H.-J. Bongartz*
- *Georg Grandke*

- *Raimund Kohl*
- *Jens-Uwe Morawski*

7 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 of the kind described in the previous chapter.

`\InputAddressFile{`*file name*`}`

The command `\InputAddressFile` 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. First a short reminder of how an address entry is structured.

`\adrentry{`*Lastname*`}{`*Firstname*`}{`*Address*`}{`*Telephone*`}{`*F1*`}{`*F2*`}{`*Comment*`}{`*Key*`}`
or
`\addrentry{`*Lastname*`}{`*Firstname*`}{`*Address*`}{`*Telephone*`}{`*F1*`}{`*F2*`}{`*F3*`}{`*F4*`}{`*Key*`}`

The last parameter is the identifier of an entry, thus the *Key* has to be unique and not blank. If the file contains more than one entry with the same *Key* value, the last occurrence will be used. The *Key* should only be composed of letters.

```
\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 parameter *Key* is the same as in the `\adrentry` or `\addrentry` command. In the address of letters often both, first-name and last-name, are required. The command `\Name{Key}` is an abridgement for `\FirstName{Key} \LastName{Key}`.

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 4$^{\text{th}}$ 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. 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 the 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 pasions were very wordly.
    \FirstName{FRODO} enjoyed to smoke \FreeII{FRODO},
    his attendant has appreciate 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}
```

The 5^{th} and 6^{th} parameter of the `\adrentry` or `\adrentry` command are for free use. They are accessible with the commands `\FreeI` and `\FreeII`. In this example the 5^{th} parameter contains the name of a person who is the most important in the life of the entry's person, the 6^{th} contains the person's passion. The 7^{th} parameter is a comment or in general also a free parameter. The commands `\Comment` or `\FreeIII` give access to the data. Using `\FreeIV` is only valid for `\addrentry` entries; for `\adrentry` entries it results in a warning.

7.3 Package Warning Options

As mentioned above the command `\FreeIV` leads to a fault if it is used for `\adrentry` entries.

Therefore scraddr supports package options in order to give the user the possibility to choose how the package should react in such situation. It is possible to choose different settings between *ignore* and *rupture* of the LaTeX run.

```
adrFreeIVempty
adrFreeIVshow
adrFreeIVwarn
adrFreeIVstop
```

One of these options can be given in the optional argument of the `\usepackage` command. The default setting is `adrFreeIVshow`.

`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 log-file

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

7.4 Authors

The authors listed below are responsible for this chapter or have contributed to this chapter in different ways.

- **Jens-Uwe Morawski**

8 Creating address files from a address database

In former versions of KOMA-Script the package addrconv was a permanent part of the KOMA-Script sytem.

The chief involvment with KOMA-Script was that with the help of addrconv it was possible to create address files from a address database in BibTeX format. Next the address file could be used for 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, as can be seen above, address files can be generated. For this addrconv employs BibTeX and some BibTeX styles. Additionally there are some LaTeX files which help to create various telephon and address lists for printing.

The package addrconv was indeed a separate package, since besides what is required for KOMA-Script it includes some interessting features more. Therefore the package addrconv, as announced in the previous KOMA-Script release, is removed from the KOMA-Script system.

The package adrconv, only one *d*, replaces addrconv entirely. If it is not included in your TeX distribution then it can be downloaded from [Kie99] and you can install it separately.

8.1 Authors

The authors listed below are responsible for this chapter or have contributed to this chapter in different ways.

- **Jens-Uwe Morawski**

9 Control Package Dependencies with scrlfile

9.1 About Package Dependencies

The introduction of LaTeX 2ε in 1994 brought many changes in the handling with LaTeX-extensions. Today the package author has many macros in order 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 the package is loaded later. This has led to the expectation that the order of package-loading will be not important. But this hope has not been fulfilled, since often different packages define or redefine one macro again and again. In such a case the order of package-loading becomes very important. For the user this is sometimes difficult to understand the behaviour and in some cases the user wants only react on the loading of a package. This is also not really simple.

Assuming the simple example that loads the package longtable with a KOMA-Script document-class employed. The longtable package defines table captions suitable for the standard classes, but the captions are totally unsuitable for documents using KOMA-Script and thus the provided configuration commands have no influence. In order to solve this problem the commands which are responsible for the table captions of the longtable package have to be re-defined. But at the moment when the longtable is loaded the KOMA-Script class is already processed.

The only chance for KOMA-Script was to delay the re-definition until the begin 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. But this is impossible since later at `\begin{document}` KOMA-Scriptwill again overwrite the user-definition with its own. Therefore the user has to delay his definition with `\AtBeginDocument` as well.

However, KOMA-Script shouldn't delay the re-definition until `\begin{document}`. It would be enough to delay until the package longtable has been loaded. But unfortunately the basic LaTeX 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 with any other file can be attended. For example the user can be warned if the not uncritical file `french.ldf` has been loaded.

Though the package is particularly interesting 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 and After Loading

`\BeforeFile{`*file*`}{`*instructions*`}`
`\AfterFile{`*file*`}{`*instructions*`}`

The macro `\BeforeFile` enables to execute *instructions* the next time the *file* is loaded. In the same way works `\AfterFile`, but here the *instructions* will be executed after the *file* has been loaded. If *file* will never be loaded then the *instructions* will never be executed.

In order to implement those features scrlfile re-defines the well known LaTeX command `\InputIfFileExists`. If this macro has not the expected definition scrlfile gives a warning. This is for the case that in future LaTeX versions the macro can have a different definition or an other package has already re-defined it.

The command `\InputIfFileExists` is used everytime LaTeX has to load a file. This is independent from whether the actual command was `\LoadClass`, `\documentclass`, `\usepackage`, `\RequiresPackage`, or `\include`. Exclusively the command

```
\input foo
```

loads the file `foo` without to utilize `\InputIfFileExists`. Therefore one should always use

```
\input{foo}
```

instead. Notice the parentheses surrounding the file name!

`\BeforeClass{`*class*`}{`*instructions*`}`
`\BeforePackage{`*package*`}{`*instructions*`}`

These both commands work 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 the file extensions `.cls` and `.sty` can be omitted.

```
\AfterClass{class}{instructions}
\AfterClass*{class}{instructions}
\AfterPackage{package}{instructions}
\AfterPackage*{package}{instructions}
```

The commands `\AfterClass` und `\AfterPackage` work 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 the file extensions `.cls` and `.sty` can be omitted. The starred versions execute the *instructions* not only next time 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 re-define a macro of the scrbook class in a way that does not work with newer KOMA-Script versions. New versions of hyperref neglect these changes if a new KOMA-Script version is detected. For the case that hyperref is loaded the code in scrbook verifies that a appropriate hyperref version is used. If not the command gives a warning.

At other places in three KOMA-Script classes following can be found:

```
\AfterPackage{caption2}{%
  \renewcommand*{\setcapindent}{%
```

Next the package caption2 has been loaded, and only if the package has been loaded really, KOMA-Script re-defines its own command `\setcapindent`. The exact code of the definition is not important. It should only be noted that caption2 claims the control over the `\caption` macro. Thus the normal definition of the `\setcapindent` macro would not work as expected. The redefinition improves the collaboration with caption2.

There are of course useful examples for normal LaTeX user too. Suppose a document that should be availbale as PS file, using LaTeX and dvips, and as PDF file, using pdfLaTeX. The document should contain hyper-links. In the List of Tables there are entries longer than one line. This is not a problem for the pdfLaTeX way, since here hyper-links can run across multiple lines. But if a hyperref driver for dvips or hyperTeX is used then this is not possible. In this case one wants that for the hyperref setup the `linktocpage` is used.

The decision what hyperref driver has to be used happens automatically via `hyperref.cfg`. The file has for example the content below.

```
\ProvidesFile{hyperref.cfg}
\@ifundefined{pdfoutput}{\ExecuteOptions{dvips}}
                        {\ExecuteOptions{pdftex}}
\endinput
```

All following is now the task of `\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 long figure caption,
    but even though it does not employ the optional
       caption argument that would allow to write
```

```
        a short caption in the List of Figures.}
\end{figure}
\end{document}
```

If now 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 an other 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 the private `hyperref.cfg`. But then instead of `\usepackage` the macro `\RequiresPackage` ought be used, see [Tea99b]. The new lines have to be inserted directly after `\ProvidesFile` line, thus prior to the execution of the options `dvips` or `pdftex`.

9.3 Autoren

The authors listed below are responsible for this chapter or have contributed to this chapter in different ways.

- Markus Kohm <Markus.Kohm@gmx.de>
- *Jens-Uwe Morawski*

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 substitude the prefix with `ftp://ftp.tex.ac.uk/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*, February 2001.
`CTAN://macros/latex/required/babel/`

[Car98] David Carlise: *The longtable package*, May 1998.
`CTAN://macros/latex/required/tools/`

[Car99a] David Carlisle: *The ifthen package*, September 1999.
`CTAN://macros/latex/base/`

[Car99b] David P. Carlisle: *Packages in the 'graphics' bundle*, February 1999.
`CTAN://macros/latex/required/graphics/`

[Dal99] Patrick W. Daly: *Natural Sciences Citations an References*, May 1999.
`CTAN://macros/latex/contrib/natbib/`

[DUD96] DUDEN: *Die deutsche Rechtschreibung*, DUDENVERLAG, Mannheim, 21th edition, 1996.

[Fai99] Robin Fairbairns: *topcapt.sty*, March 1999.
`CTAN://macros/latex/contrib/misc/topcapt.sty`

[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*, July 2001.
`CTAN://macros/latex/contrib/float/`

[Mit00] Frank Mittelbach: *An environment for multicolumn output*, July 2000.
`CTAN://macros/latex/required/tools/`

[Oos00] Piet van Oostrum: *Page layout in LaTeX*, October 2000.
`CTAN://macros/latex/contrib/fancyhdr/`

[OPHS99] Tobias Oetker, Hubert Partl et al.: *The Not So Short Introduction to LaTeX 2ε*, April 1999.
`CTAN://info/lshort/`

[Rah01] Sebastian Rahtz: *Hypertext marks in LaTeX: the hyperref package*, February 2001.
`CTAN://macros/latex/contrib/hyperref/`

[RNH02] Bernd Raichle, Rolf Niepraschk et al.: *DE-TeX-/DANTE-FAQ*, May 2002.
`http://www.dante.de/faq/de-tex-faq/`

[Sch03] Martin Schröder: *The Ragged2e-package*, January 2003.
`CTAN://macros/latex/contrib/ms/`

[SKPH99] Walter Schmidt, Jörg Knappen et al.: *LaTeX 2ε-Kurzbeschreibung*, April 1999.
`CTAN://info/lshort/german/`

[Som95] Harald Axel Sommerfeldt: *caption package*, October 1995.
`CTAN://macros/latex/contrib/caption/`

[Tea99a] LaTeX3 Project Team: *LaTeX 2ε for authors*, September 1999.
`CTAN://macros/latex/doc/usrguide.pdf`

[Tea99b] LaTeX3 Project Team: *LaTeX 2ε for class and package writers*, September 1999.
`CTAN://macros/latex/doc/clsguide.pdf`

[Tea00] LaTeX3 Project Team: *LaTeX 2ε font selection*, January 2000.
`CTAN://macros/latex/doc/fntguide.pdf`

[Tob00] Geoffrey Tobin: *setspace LaTeX package*, December 2000.
`CTAN://macros/latex/contrib/setspace/`

[Tsc87] Jan Tschichold: *Ausgewählte Aufsätze über Fragen der Gestalt des Buches und der Typographie*, Birkhäuser Verlag, Basel, 2nd edition, 1987.

[UK:02] *TeX Frequently Asked Questions on the Web*, May 2002.
`http://www.tex.ac.uk/faq/`

[Ume00] Hideo Umeki: *The geometry package*, June 2000.
`CTAN://macros/latex/contrib/geometry/`

[WF00] Hans Peter Willberg and Friedrich Forssman: *Erste Hilfe in Typografie*, Verlag Hermann Schmidt, Mainz, 2000.

List of changes

At this list of changes you will find all significant changes of the user interface of the KOMA-Script bundle at the last few versions. The list was sorted about the names of the classes and packages and their version. The numbers behind the versions are the pages, where the changes are described. At the margins of these pages you will find corresponding version marks.

Index

There are two kinds of page numbers at this index. The italic 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

E

F

H

I

K

L

M

N

O

P

Index of Lengths and Counters

Index of Elements with Possibilty of Fontswitching

Index of Files, Classes and Packages

Index of Class- and Package-Options