CHAPTER 3

Basic Formatting Tools

The way information is presented visually can influence, to a large extent, the message as it is understood by the reader. Therefore, it is important that you use the best possible tools available to convey the precise meaning of your words. It must, however, be emphasized that visual presentation forms should aid the reader in understanding the text, and should not distract his or her attention. For this reason, visual consistency and uniform conventions for the visual clues are a must, and the way given structural elements are highlighted should be the same throughout a document. This constraint is most easily implemented by defining a specific command or environment for each document element that has to be treated specially and by grouping these commands and environments in a package file or in the document preamble. By using exclusively these commands, you can be sure of a consistent presentation form.

This chapter explains various ways for highlighting parts of a document. The first part looks at how short text fragments or paragraphs can be made to stand out and describes tools to manipulate such elements.

The second part deals with the different kind of “notes”, such as footnotes, marginal notes, and endnotes, and explains how they can be customized to conform to different styles, if necessary.

Typesetting lists is the subject of the third part. First, the various parameters and commands controlling the standard \LaTeX\ lists, \texttt{enumerate}, \texttt{itemize}, and \texttt{description}, are discussed. Then, the extensions provided by the \texttt{paralist} package and the concept of “headed lists” exemplified by the \texttt{amsthm} package are presented. These will probably satisfy the structure and layout requirements of most readers. If not, then the remainder of this part introduces the generic \texttt{list}
Basic Formatting Tools

environment and explains how to build custom layouts by varying the values of
the parameters controlling it.

The fourth part explains how to simulate “verbatim” text. In particular, we
have a detailed look at the powerful packages fancyvrb and listings.

The final part presents packages that deal with line numbering, handling of
columns, such as parallel text in two columns, or solving the problem of producing
multiple columns.

3.1 Phrases and paragraphs

In this section we deal with small text fragments and explain how they can be
manipulated and highlighted in a consistent manner by giving them a visual ap-
pearance different from the one used for the main text.

We start by discussing how to define commands that take care of the space
after them, then show a way to produce professional-looking marks of omission.

For highlighting text you can customize the font shape, weight, or size (see
Section 7.3.1 on page 338). Text can also be underlined, or the spacing between
letters can be varied. Ways for performing such operations are offered by the four
packages relsize, textcase, ulem, and soul.

The remainder of this section then turns to paragraph-related issues, such as
producing large initial letters at the start of a paragraph, modifying paragraph
justification, altering the vertical spacing between lines of a paragraph, and in-
troducing rectangular holes into it, that can be filled with small pictures, among
other things.

3.1.1 \xspace—Gentle spacing after a macro

The small package xspace (by David Carlisle) defines the \xspace command, for
use at the end of macros that produce text. It adds a space unless the macro is
followed by certain punctuation characters.

The \xspace command saves you from having to type \_, or {} after most
occurrences of a macro name in text. However, if either of these constructs follows
\xspace, a space is not added by \xspace. This means that it is safe to add
\xspace to the end of an existing macro without making too many changes in
your document. Possible candidates for \xspace are commands for abbreviations
such as “e.g.,” and “i.e.,”.

\newcommand{\eg\{e.g.,\xspace}
\newcommand{\ie\{i.e.,\xspace}
\newcommand{\etc\{etc.\@\xspace}

Notice the use of the \@ command to generate the correct kind of space. If used to
the right of a punctuation character, it prevents extra space from being added: the
3.1 Phrases and paragraphs

.. \usepackage{xspace}
.. \newcommand\USA{United States of America\xspace}
.. \newcommand\GB {Great Britain\xspace}
\GB was unified in 1707. \GB, the \USA, and Canada have close cultural links.

3.1.2 ellipsis, \textipa{ips}—Marks of omission

Omission marks are universally represented by three consecutive periods (also known as an \textipa{ipsis}). Their spacing, however, depends on house style and typographic conventions, and significant differences are observed. In French, according to Hart [63] or \textit{The Chicago Manual of Style} [38], “points de suspension” are set close together and immediately follow the preceding word with a space on the right:

C’est une chose... bien difficile.

In German, according to the Duden [44], “Auslassungspunkte” have space on the left and right unless they mark missing letters within a word or a punctuation after them is kept:

Du E... du! Scher dich zum ...!

Elsewhere, such as in British and American typography, the dots are sometimes set with full word spaces between them and rather complex rules determine how to handle other punctuation marks at either end.

\LaTeX{} offers the commands \texttt{\dots} and \texttt{textellipsis} to produce closely spaced omission marks. Unfortunately, the standard definition (inherited from plain \TeX{}) produces uneven spacing at the left and right—unsuitable to typeset some of the above examples properly. The extra thin space at the right of the ellipsis is correct in certain situations (e.g., when a punctuation character follows). If the ellipsis is followed by space, however, it looks distinctly odd and is best canceled as shown in the example below (though removing the space in the second instance brings the exclamation mark a bit too close).

\begin{verbatim}
\usepackage{xspace}
\newcommand\USA{United States of America\xspace}
\newcommand\GB {Great Britain\xspace}
GB was unified in 1707. GB, the USA, and Canada have close cultural links.
\end{verbatim}

\begin{verbatim}
\newcommand\lips{\dots\unkern}
\end{verbatim}

Compare the following:

\begin{verbatim}
Du E\dots du! Scher dich zum ...!
\end{verbatim}

\begin{verbatim}
Du E\lips\ du! Scher dich zum \lips!
\end{verbatim}
This problem is addressed in the package \texttt{ellipsis} written by Peter Heslin, which redefines the \texttt{\dots} command to look at the following character to decide whether to add a final separation. An extra space is added if the following character is listed in the command \texttt{\ellipsispunctuation}, which defaults to “.,::!?”.

When using some of the language support packages that make certain characters active, this list may have to be redeclared afterwards to enable the package to still recognize the characters.

The spacing between the periods and the one possibly added after the ellipsis can be controlled through the command \texttt{\ellipsisgap}. To allow for automatic adjustments depending on the font size use a font-dependent unit like \texttt{em} or a fraction of a \texttt{\fontdimen} (see page 428).

\begin{verbatim}
\usepackage{ellipsis}
\end{verbatim}

\texttt{\usepackage{ellipsis}}

Compare the following:\
\begin{verbatim}
Du E\texttt{\dots}\ du! Scher dich zum \texttt{\dots}!
\end{verbatim}
\begin{verbatim}
\renewcommand{\ellipsisgap}{1.5\texttt{\fontdimen3\font}}
Du E\texttt{\dots}\ du! Scher dich zum \texttt{\dots}!
\end{verbatim}
\begin{verbatim}
\renewcommand{\ellipsisgap}{0.3\texttt{em}}
Du E\texttt{\dots}\ du! Scher dich zum \texttt{\dots}!
\end{verbatim}

\begin{verbatim}
\renewcommand{\ellipsisgap}{0.3\texttt{em}}
\end{verbatim}

For the special case when you need an ellipsis in the middle of a word (or for other reasons want a small space at either side), the package offers the command \texttt{\midwordellipsis}. If the package is loaded with the option \texttt{mla} (Modern Language Association style), the ellipsis is automatically bracketed without any extra space after the final period.

If one follows The Chicago Manual of Style [38], then an ellipsis is set with full word spaces between the dots. For this, one can deploy the \texttt{lips} package\footnote{\texttt{lips} is actually part of a larger suite of packages. If used on a stand-alone basis, you also have to load the package \texttt{moredefs} by the same author.} by Matt Swift. It implements the command \texttt{lips}, which follows the recommendations in this reference book. For example, an ellipsis denoting an omission at the end of a sentence should, according to [38, §10.48–63], consist of four dots with the first dot being the sentence period.\footnote{Not that the authors of this book can see any logic in this.} The \texttt{lips} command implements this by interpreting “\texttt{lips}.” like “\texttt{.\n\texttt{lips}}”, as can be seen in the next example.

\begin{verbatim}
\usepackage{moredefs,lips}
\end{verbatim}

\texttt{\usepackage{moredefs,lips}}

Elsewhere \texttt{\dots} the dots are normally set with full word spaces between them. \texttt{\dots} An example would be this paragraph.

\begin{verbatim}
Elsewhere \texttt{\dots} the dots are normally set with full word spaces between them \texttt{\dots}. An example would be this paragraph.
\end{verbatim}

The \texttt{lips} command looks for punctuation characters following it and ensures that in case of \texttt{,::?!?)}/ the ellipsis and the punctuation are not separated by a line break. In other cases (e.g., an opening parenthesis), a line break would be possible. The above list is stored in \texttt{\LPNobreakList} and can be adjusted if
3.1 Phrases and paragraphs

necessary. To force an unbreakable space following \lips, follow the command with a tie (~).

When applying the mla option the ellipsis generated will be automatically bracketed and a period after the \lips command will not be moved to the front. If necessary, \olips will produce the original unbracketed version.

Elsewhere . . . the dots are normally set with full word spaces between them [. . .]. An example would be this paragraph.

3.1.3 amsmath—Nonbreaking dashes

The amsmath package, extensively discussed in Chapter 8, also offers one command for use within paragraphs. The command \nobreakdash suppresses any possibility of a line break after the following hyphen or dash. A very common use of \nobreakdash is to prevent undesirable line breaks in usages such as "p-adic" but here is another example: if you code "Pages 3–9" as Pages 3\nobreakdash--9 then a line break will never occur between the dash and the 9.

This command must be used immediately before a hyphen or dash (~, --, or ---). The following example shows how to prohibit a line break after the hyphen but allow normal hyphenation in the following word (it suffices to add a zero-width space after the hyphen). For frequent use, it's advisable to make abbreviations, such as p. As a result "dimension" is broken across the line, while a break after "p-" is prevented (resulting in an overfull box in the example) and "3–9" is moved to the next line.

The generalization to the \n-dimensional case (using the standard p-adic topology) can be found on Pages 3–9 of Volume IV.

3.1.4 relsize—Relative changes to the font size

Standard \LaTeX offers 10 predefined commands that change the overall font size (see Table 7.1 on page 342). The selected sizes depend on the document class but are otherwise absolute in value. That is, \texttt{\small} will always select the same size within a document regardless of surrounding conditions.
However, in many situations it is desirable to change the font size relative to the current size. This can be achieved with the relsize package, originally developed by Bernie Cosell and later updated and extended for \LaTeX by Donald Arseneau and Matt Swift.

The package provides the declarative command `\relsize`, which takes a number as its argument denoting the number of steps by which to change the size. For example, if the current size is `\large` then `\relsize{-2}` would change to `\small`. If the requested number of steps is not available then the smallest (i.e., `\tiny`) or largest (i.e., `\Huge`) size command is selected. This means that undoing a relative size change by negating the argument of `\relsize` is not guaranteed to bring you back to the original size—it is better to delimit such changes by a brace group and let \LaTeX undo the modification.

The package further defines `\smaller` and `\larger`, which are simply abbreviations for `\relsize` with the arguments `-1` and `1`, respectively. Convenient variants are `\textsmaller` and `\textlarger`, whose argument is the text to reduce or enlarge in size. These four commands take as an optional argument the number of steps to change if something different from `1` (the default) is needed.

\begin{verbatim}
\usepackage{relsize}
\Large Some large text with a few small words inside.
\par\medskip
\normalsize
S\textsmaller[2]{MALL} C\textsmaller[2]{APS} (faked)\textsc{Small Caps} (real; compare the running length and stem thickness to previous line).
\end{verbatim}

In fact, the above description for `\relsize` is not absolutely accurate: it tries to increase or decrease the size by 20\% for each step and selects the \LaTeX font size command that is closest to the resulting target size. It then compares the selected size and target size. If they differ by more than the current value of `\RSpercentTolerance` (interpreted as a percentage), the package calls `\fontsize` with the target size as one of the arguments. If this happens it is up to \LaTeX's font selection scheme to find a font matching this request as closely as possible. By default, `\RSpercentTolerance` is an empty macro, which is interpreted as 30 (percent) when the current font shape group is composed of only discrete sizes (see Section 7.10.3), and as 5 when the font shape definition covers ranges of sizes.

Using a fixed factor of 1.2 for every step may be too limiting in certain cases. For this reason the package additionally offers the more general declarative command `\relscale{factor}` and its variant `\textscale{factor}{text}`, to select the size based on the given `factor`, such as 1.3 (enlarge by 30\%).

There are also two commands, `\mathsmaller` and `\mathlarger`, for use in math mode. \LaTeX recognizes only four different math sizes, of which two (`\displaystyle` and `\textstyle`) are nearly identical for most symbols, so the application domain of these commands is somewhat limited. With `exscale` addi-
3.1 Phrases and paragraphs

tionally loaded the situation is slightly improved: the \texttt{\textbackslash mathlarger} command, when used in \texttt{\textbackslash displaystyle}, will then internally switch to a larger text font size and afterwards select the \texttt{\textbackslash displaystyle} corresponding to that size.

\[ \sum \neq \mathlarger{\sum} \]

and \[ \frac{1}{2} \neq \frac{1}{2} \] but \( N = N \)

These commands will attempt to correctly attach superscripts and subscripts to large operators. For example,

\[ \sum_{i=1}^{n} \neq \mathlarger{\sum}_{i=1}^{n} \]
\[ \int_{0}^{\infty} \neq \mathlarger{\int}_{0}^{\infty} \]

Be aware that the use of these commands inside formulas will hide the true nature of the math atoms inside the argument, so that the spacing in the formula, without further help, might be wrong. As shown in following example, you may have to explicitly use \texttt{\textbackslash mathrel}, \texttt{\textbackslash mathbin}, or \texttt{\textbackslash mathop} to get the correct spacing.

\[ a \times b \neq a \mathlarger{x} b \neq a \times b \]

Due to these oddities, the \texttt{\textbackslash mathlarger} and \texttt{\textbackslash mathsmler} commands should not be trusted blindly, and they will not be useful in every instance.

3.1.5 \texttt{\textbackslash textcase}—Change case of text intelligently

The standard \LaTeX\ commands \texttt{\textbackslash MakeUppercase} and \texttt{\textbackslash MakeLowercase} change the characters in their arguments to uppercase or lowercase, respectively, thereby expanding macros as needed. For example,

\texttt{\textbackslash MakeUppercase{On \today}}

will result in “ON 28TH OF JULY 2003”. Sometimes this will change more characters than desirable. For example, if the text contains a math formula, then upper-casing this formula is normally a bad idea because it changes its meaning. Similarly, arguments to the commands \texttt{\textbackslash label}, \texttt{\textbackslash ref}, and \texttt{\textbackslash cite} represent semantic information, which, if modified, will result in incorrect or missing references, because \LaTeX\ will look for the wrong labels.
The package textcase by David Carlisle overcomes these defects by providing two alternative commands, \texttt{\MakeTextUppercase} and \texttt{\MakeTextLowercase}, which recognize math formulas and cross-referencing commands and leave them alone.

1 Textcase example

\begin{verbatim}
\usepackage{textcase}
\section{Textcase example} \label{exa}
\MakeTextUppercase{Text in section \ref{exa}, about $a=b$ and $\alpha \neq a$}
\end{verbatim}

Sometimes portions of text should be left unchanged for one reason or another. With \texttt{\NoCaseChange} the package provides a generic way to mark such parts. For instance:

\begin{verbatim}
\usepackage{textcase}
\MakeTextUppercase{Some text \NoCaseChange{Some More} text}
\end{verbatim}

If necessary, this method can be used to hide syntactic information, such as

\begin{verbatim}
\NoCaseChange{\begin{tabular}{ll}} ... \NoCaseChange{\end{tabular}}
\end{verbatim}

thereby preventing \texttt{tabular} and \texttt{ll} from incorrectly being uppercased.

All this works only as long as the material is on the top level. Anything that is inside a group of braces (other than the argument braces to \texttt{\label}, \texttt{\ref}, \texttt{\cite}, or \texttt{\NoCaseChange}) will be uppercased or lowercased regardless of its nature.

\begin{verbatim}
\usepackage{textcase}
\MakeTextUppercase{Both of these will fail $a+b=c$}
\emph{\NoCaseChange{unfortunately}}
\end{verbatim}

In the above case you could avoid this pitfall by taking the formula out of the argument to \texttt{\textbf} and moving \texttt{\emph} inside the argument to \texttt{\NoCaseChange}.

In other situations this kind of correction might be impossible. In such a case the (somewhat cumbersome) solution is to hide the problem part inside a private macro and protect it from expansion during the case change; this method works for the standard \TeX{} commands as well, as shown in the next example.

\begin{verbatim}
\newcommand\mymath{$a+b=c$}
\MakeUppercase{But this will}
\textbf{\protect\mymath{unfortunately}}
\end{verbatim}

Some classes and packages employ \texttt{\MakeTextUppercase} internally—for example, in running headings. If you wish to use \texttt{\MakeTextUppercase} instead, you should
3.1 Phrases and paragraphs

load the textcase package with the option overload. This option will replace the standard \LaTeX commands with the variants defined by the package.

3.1.6 ulem—Emphasize via underline

\LaTeX encourages the use of the \emph command and the \em declaration for marking emphasis, rather than explicit font-changing declarations, such as \textbf and \textit. The ulem package (by Donald Arseneau) redefines the command \emph to use underlining, rather than italics. It is possible to have line breaks and even primitive hyphenation in the underlined text. Every word is typeset in an underlined box, so automatic hyphenation is normally disabled, but explicit discretionary hyphens (\texttt{-}) can still be used. The underlines continue between words and stretch just like ordinary spaces do. As spaces delimit words, some difficulty may arise with syntactical spaces (e.g., "2.3 pt"). Some effort is made to handle such spaces. If problems occur you might try enclosing the offending command in braces, since everything inside braces is put inside an \texttt{\textbackslash mbox}. Thus, braces suppress stretching and line breaks in the text they enclose. Note that nested emphasis constructs are not always treated correctly by this package (see the gymnastics performed below to get the interword spaces correct in which each nested word is put separately inside an \emph expression).

\begin{verbatim}
\usepackage{ulem}
\end{verbatim}

No, I did \emph{not} act in the movie The Persecution and Assassination of Jean-Paul Marat, as performed by the Inmates of the Asylum of Charenton under the direction of the Marquis de Sade! But I \emph{did} see it.

Alternatively, underlining can be explicitly requested using the \texttt{\textbackslash uline} command. In addition, a number of variants are available that are common in editorial markup. These are shown in the next example.

\begin{verbatim}
\usepackage{ulem}
\end{verbatim}

Double underlining (\texttt{\textbackslash uline\{under-line\}}),
a wavy underline (\texttt{\textbackslash wave\{under-wave\}}),
a line through text (\texttt{\textbackslash xout\{strike out\}}),
crossing out text (\texttt{\textbackslash xout\{cross out, X out\}}),

The redefinition of \emph can be turned off and on by using \texttt{\textbackslash normalem} and \texttt{\textbackslash ULforem}. Alternatively, the package can be loaded with the option \texttt{\textbackslash normalem} to suppress this redefinition. Another package option is \texttt{\textbackslash UWforbf}, which replaces \texttt{\textbackslash textbf} and \texttt{\textbackslash textit} by \texttt{\textbackslash uwave} whenever possible.

The position of the line produced by \texttt{\textbackslash uline} can be set explicitly by specifying a value for the length \texttt{\textbackslash ULdepth}. The default value is font-dependent, denoted...
by the otherwise senile value \maxdimen. Similarly, the thickness of the line can be controlled via \ULthickness, which, for some historical reason, needs to be redefined using \renewcommand.

3.1.7 soul—Letterspacing or stealing sheep

Frederic Goudy supposedly said, "Anyone who would letterspace black letter would steal sheep". Whether true or a myth, the topic of letterspacing clearly provokes heated discussions among typographers and is considered bad practice in most situations because it changes the "grey" level of the text and thus disturbs the flow of reading. Nevertheless, there are legitimate reasons for undertaking letterspacing. For example, display type often needs a looser setting and in most fonts uppercased text is improved this way. You may also find letterspacing being used to indicate emphasis, although this exhibits the grey-level problem.

\TeX is ill equipped when it comes to supporting letterspacing. In theory, the best solution is to use specially designed fonts rather than trying to solve the problem with a macro package. But as this requires the availability of such fonts, it is not an option for most users. Thus, in practice, the use of a macro-based solution is usually easier to work with, even though it means dealing with a number of restrictions. Some information about the font approach can be found in the documentation for the fontinst package [74,75].

The soul package written by Melchior Franz provides facilities for letterspacing and underlining, but maintains \TeX's ability to automatically hyphenate words, a feature not available in ulem. The package works by parsing the text to be letterspaced or underlined, token by token, which results in a number of peculiarities and restrictions. Thus, users who only wish to underline a few words and do not need automatic hyphenation are probably better off with ulem, which is far less picky about its input.

\caps{text} \hl{text} \so{text} \st{text} \ul{text}

The use of the five main user commands of soul are shown in the next example. In cases where \TeX's hyphenation algorithm fails to find the appropriate hyphenation points, you can guide it as usual with the \- command. If the color package is loaded, \hl will work like a text marker, coloring the background using yellow as the default color; otherwise, it will behave like \ul and underline its argument.
3.1 Phrases and paragraphs

Normally, the soul package interprets one token after another in the argument of \so{}, \st{}, and so on. However, in case of characters that are represented by more than one token (e.g., accented characters) this might fail with some low-level \TeX error messages. Fortunately, the package already knows about all common accent commands, so these are handled correctly. For others, such as those provided by the textcomp package, you can announce them to soul with the help of a \soulaccent declaration. The alternative is to surround the tokens by braces.

\usepackage{soul} \usepackage{textcomp} \soulaccent{\capitalgrave} \Huge \st{"a} \'u \~O \capitalgrave X \{\capitalbreve Y\}

The soul package already knows that quotation characters, en dash, and em dash consist of several tokens and handles them correctly. In case of other syntactical ligatures, such as the Spanish exclamation mark, you have to help it along with a brace group.

“So there,” he said. \usepackage{soul} \soul{{‘So there,’}} he said. \caps{{‘}Hola---my \textbf{friend}!{’}}

The soul package also knows about math formulas as long as they are surrounded by $ signs (the form \(...\) is not supported) and it knows about all standard font-changing commands, such as \textbf. If you have defined your own font-switching command or use a package that provides additional font commands, you have to register them with soul using \soulregister. This declaration expects the font command to be registered as its first argument and the number of arguments (i.e., 0 or 1) for that command to appear as its second argument. Within the soul commands none of the font commands inserts any (necessary) italic correction. If needed, one has to provide it manually using /.

\newcommand\textsfbf[1]{\textsf{\bfseries#1}} \usepackage{soul} \soulregister{\textsfbf}{1} \so{Here we see soul in \emph{action}: $x \neq y$ OK?} \caps{{‘}Hola---my \textbf{friend}!{’}}

If you look carefully, you will see that the font commands suppress letterspacing directly preceding and following them, such as between “action” and the colon. This can be corrected by adding \\bour, which forces a space.

\usepackage{soul} \so{bloody viz. bloody} \soul{bloody viz. bloody}

Text inside a brace group is regarded as a single object during parsing and is therefore not spaced out. This is handy if certain ligatures are to be kept intact inside spaced-out text. However, this method works only if the text inside the brace group contains no hyphenation points. If it does, you will receive the package error message “Reconstruction failed”. To hide such hyphenation points
you need to put the text inside an \textit{mbox}, as shown in the second text line of the
next example (\LaTeX{} would hyphenate this as "Es-cher"—that is, between the "sch"
that we try to keep together). You can also use \soulomit{to achieve this effect,
but then your text will work only when the soul package is loaded.

\begin{verbatim}
\usepackage{soul,yfonts} \usepackage[latin1]{inputenc}
\textfrak{\soul{S\{ch\}u\{tz\}vorri\{ch\}tung}} \par
\soul{Gödel, E\{sch\}er, Bach} \par
\ul{Temporarily disabling the scanner}
\end{verbatim}

One of the most important restrictions of the above commands is that they
cannot be nested; any attempt to nest soul commands will result in low-level \LaTeX{}
errors. If you really need nesting you will have to place the inner material in a box,
which means you lose the possibility to break the material at a line ending.

\begin{verbatim}
\usepackage{soul} \newsavebox{soulbox}
\sbox{soulbox}{\soul{ is hell }}
\ul{This\mbox{\usebox{soulbox}}for all of us!}
\end{verbatim}

A few other commands are special within the argument of \soul{} and friends.
Spacing out at certain points can be canceled using \textless{} or forced with \textgreater{} as we
saw above. As usual with \LaTeX{} a ~ will produce an unbreakable space. The \textbackslash{} command is supported, though only in its basic form—no star, no optional argument.
You can also use \linebreak to break a line at a certain point, but again the optional argument is not supported. Other \LaTeX{} commands are likely to break the package—some experimentation will tell you what is safe and what produces havoc. The next example shows applications of these odds and ends.

\begin{verbatim}
\usepackage{soul}
\soul{''\textless{}So there\textgreater{}, he said. Let's\linebreak production a spaced out line, OK?}
\end{verbatim}

The \texttt{\sodef} declaration allows you to define your own letterspacing commands. It
can also be used to overwrite the defaults for \soul{}.

The letterspacing algorithm works by putting a certain \textit{inter-letter space} between characters of a word, a certain \textit{word space} between words, and a certain \textit{outer space} at the beginning and end of the letterspaced text section. The latter space is added only if it is appropriate at that point. The default values for these spaces are adjusted for typesetting texts in Fraktur fonts but with the help of the \texttt{\sodef} declaration it is easy to adjust them for your own needs. The \textit{font} argument allows you to specify font attributes; in most cases it will be empty. Rather than using explicit dimensions in the other arguments it is advisable to resort to.
3.1 Phrases and paragraphs

\em{values}, thereby making your definition depend on the current font and its size.

\begin{verbatim}
\usepackage{soul}
\sodef\sobf{\bfseries}{.3em}{1em plus .1em minus .2em}
\end{verbatim}

Here we \textit{emphasize words} a lot. Here we \textbf{emphasize words} a lot.

While \texttt{so} or any new command defined via \texttt{sodef} simply retrieves and executes its stored definition, the \texttt{caps} command works somewhat differently. It examines the current font and tries to find it (or a close match) in an internal database. It then uses the letterspacing values stored there. You can extend this database using the \texttt{capsdef} declaration by providing values for individual fonts or groups of fonts. In this way you can fine-tune the letterspacing—for example, for text in headings. It is even possible to keep several such databases and change them on the fly within a document.

\begin{verbatim}
capsdef{match spec}{font}{inter-letter space}{word space}{outer space}
\end{verbatim}

Apart from the first argument, which is totally different, the other arguments to \texttt{capsdef} are identical to those of \texttt{sodef}. The first argument, \textit{match spec}, defines the font (or fonts) to which the current declaration applies.

Its syntax is \textit{encoding}, \textit{family}, \textit{series}, \textit{shape}, and \textit{size} separated by slashes using the naming conventions of NFSS. Empty values match anything, so /// matches any font, /ptm///10 matches all Times fonts in 10 points, and OT1/cm\textit{r}/m/n/ matches Computer Modern (cm\textit{r}) medium series (m) normal shape (n) encoded in OT1 in any size. It is also possible to specify size ranges. For example, 5-14 means \texttt{size < 14pt} and 14- matches all sizes equal or greater 14pt. Refer to the tables in Chapter 7 for details on the NFSS font naming conventions.

As with \texttt{sodef}, in most declarations the \textit{font} argument will be empty. On some occasions it may make sense to use \texttt{scshape} in this place, such as to change the font shape to small caps before applying letterspacing.

Because \texttt{caps} uses the first matching entry in its database, the order of \texttt{capsdef} declarations is important. Later declarations are examined first so that it is possible to overwrite or extend existing declarations.

\begin{verbatim}
\usepackage{titlesec,soul}
\newcommand{\allcaps}[1]{\MakeUppercase{\caps{#1}}}
\titleformat{\section}{[block]}{\thesection.}{.5em}{\allcaps}
\titlespacing*{\section}{0pt}{8pt}{3pt}
capsdef{/phv///}{\scshape}{.17em}{.55em}{.4em}
\section*{A Sample Heading}
The \texttt{capsdef} declaration applies here, because the heading definition specifies sans serif and our examples are typeset with Times and Helvetica (\texttt{phv}).
\end{verbatim}

A SAMPLE HEADING

The \texttt{capsdef} declaration applies here, because the heading definition specifies sans serif and our examples are typeset with Times and Helvetica (\texttt{phv}).
Basic Formatting Tools

The previous example also contained an interesting combination of \caps and \MakeUppercase: the command \allcaps changes its argument to uppercase and then uses \caps to letterspace the result.

\capssave{name} \capsselect{name} \capsreset

With \capsreset the database is restored to its initial state containing only a generic default. You can then add new entries using \capsdef. The current state of the \caps database can be stored away under a name by using \capssave. You can later retrieve this state by recalling it with \capsselect. If you use the \capsdefault option when loading the package, then all uses of \caps that have no matching declaration are flagged by underlining the text.

AS AMPLE H EADING
Notice the different letterspacing in the heading and RUNNING TEXT. For Times we have no definition above so that the DEFAULT will match.

\usepackage{titlesec} \usepackage[capsdefault]{soul}
\capsdef{/phv///}{\scshape}{.17em}{.55em}{.4em}
\capssave{display} \capsreset
\capsdef{/phv///}{\scshape}{.04em}{.35em}{.35em}
\titlespacing*{\section}{0pt}{8pt}{3pt}
\titleformat{\section}[block]{\centering\sffamily}{\thesection.}{.5em}{\capsselect{display}\caps}
\section*{A Sample Heading}
Notice the different letterspacing in the heading and \textsf{\caps{Running Text}}. For Times we have no definition above so that the \caps{default} will match.

The position and the width of the line produced by the \ul command can be customized using either \setul or \setuldepth. The command \setul takes two dimensions as arguments: the position of the line in relation to the baseline and the width of the line. Alternatively, \setuldepth can be used to specify that the line should be positioned below the text provided as an argument. Finally, \resetul will restore the default package settings.

\usepackage{titlesec} \usepackage[capsdefault]{soul}
\capsdef{/phv///}{\scshape}{.17em}{.55em}{.4em}
\capssave{display} \capsreset
\capsdef{/phv///}{\scshape}{.04em}{.35em}{.35em}
\titlespacing*{\section}{0pt}{8pt}{3pt}
\titleformat{\section}[block]{\centering\sffamily}{\thesection.}{.5em}{\capsselect{display}\caps}
\section*{A Sample Heading}
Notice the different letterspacing in the heading and \textsf{\caps{Running Text}}. For Times we have no definition above so that the \caps{default} will match.

The position and the width of the line produced by the \ul command can be customized using either \setul or \setuldepth. The command \setul takes two dimensions as arguments: the position of the line in relation to the baseline and the width of the line. Alternatively, \setuldepth can be used to specify that the line should be positioned below the text provided as an argument. Finally, \resetul will restore the default package settings.

\usepackage{titlesec} \usepackage[capsdefault]{soul}
\capsdef{/phv///}{\scshape}{.17em}{.55em}{.4em}
\capssave{display} \capsreset
\capsdef{/phv///}{\scshape}{.04em}{.35em}{.35em}
\titlespacing*{\section}{0pt}{8pt}{3pt}
\titleformat{\section}[block]{\centering\sffamily}{\thesection.}{.5em}{\capsselect{display}\caps}
\section*{A Sample Heading}
Notice the different letterspacing in the heading and \textsf{\caps{Running Text}}. For Times we have no definition above so that the \caps{default} will match.

The position and the width of the line produced by the \ul command can be customized using either \setul or \setuldepth. The command \setul takes two dimensions as arguments: the position of the line in relation to the baseline and the width of the line. Alternatively, \setuldepth can be used to specify that the line should be positioned below the text provided as an argument. Finally, \resetul will restore the default package settings.

\usepackage{titlesec} \usepackage[capsdefault]{soul}
\capsdef{/phv///}{\scshape}{.17em}{.55em}{.4em}
\capssave{display} \capsreset
\capsdef{/phv///}{\scshape}{.04em}{.35em}{.35em}
\titlespacing*{\section}{0pt}{8pt}{3pt}
\titleformat{\section}[block]{\centering\sffamily}{\thesection.}{.5em}{\capsselect{display}\caps}
\section*{A Sample Heading}
Notice the different letterspacing in the heading and \textsf{\caps{Running Text}}. For Times we have no definition above so that the \caps{default} will match.

The position and the width of the line produced by the \ul command can be customized using either \setul or \setuldepth. The command \setul takes two dimensions as arguments: the position of the line in relation to the baseline and the width of the line. Alternatively, \setuldepth can be used to specify that the line should be positioned below the text provided as an argument. Finally, \resetul will restore the default package settings.

\usepackage{titlesec} \usepackage[capsdefault]{soul}
\capsdef{/phv///}{\scshape}{.17em}{.55em}{.4em}
\capssave{display} \capsreset
\capsdef{/phv///}{\scshape}{.04em}{.35em}{.35em}
\titlespacing*{\section}{0pt}{8pt}{3pt}
\titleformat{\section}[block]{\centering\sffamily}{\thesection.}{.5em}{\capsselect{display}\caps}
\section*{A Sample Heading}
Notice the different letterspacing in the heading and \textsf{\caps{Running Text}}. For Times we have no definition above so that the \caps{default} will match.

The position and the width of the line produced by the \ul command can be customized using either \setul or \setuldepth. The command \setul takes two dimensions as arguments: the position of the line in relation to the baseline and the width of the line. Alternatively, \setuldepth can be used to specify that the line should be positioned below the text provided as an argument. Finally, \resetul will restore the default package settings.

\usepackage{titlesec} \usepackage[capsdefault]{soul}
\capsdef{/phv///}{\scshape}{.17em}{.55em}{.4em}
\capssave{display} \capsreset
\capsdef{/phv///}{\scshape}{.04em}{.35em}{.35em}
\titlespacing*{\section}{0pt}{8pt}{3pt}
\titleformat{\section}[block]{\centering\sffamily}{\thesection.}{.5em}{\capsselect{display}\caps}
\section*{A Sample Heading}
Notice the different letterspacing in the heading and \textsf{\caps{Running Text}}. For Times we have no definition above so that the \caps{default} will match.

The position and the width of the line produced by the \ul command can be customized using either \setul or \setuldepth. The command \setul takes two dimensions as arguments: the position of the line in relation to the baseline and the width of the line. Alternatively, \setuldepth can be used to specify that the line should be positioned below the text provided as an argument. Finally, \resetul will restore the default package settings.

\usepackage{titlesec} \usepackage[capsdefault]{soul}
\capsdef{/phv///}{\scshape}{.17em}{.55em}{.4em}
\capssave{display} \capsreset
\capsdef{/phv///}{\scshape}{.04em}{.35em}{.35em}
\titlespacing*{\section}{0pt}{8pt}{3pt}
\titleformat{\section}[block]{\centering\sffamily}{\thesection.}{.5em}{\capsselect{display}\caps}
\section*{A Sample Heading}
Notice the different letterspacing in the heading and \textsf{\caps{Running Text}}. For Times we have no definition above so that the \caps{default} will match.

The position and the width of the line produced by the \ul command can be customized using either \setul or \setuldepth. The command \setul takes two dimensions as arguments: the position of the line in relation to the baseline and the width of the line. Alternatively, \setuldepth can be used to specify that the line should be positioned below the text provided as an argument. Finally, \resetul will restore the default package settings.

Both \ul and \at use a black rule by default. If you additionally load the color package, you can use colored rules instead and, if desired, modify the highlighting color:

\usepackage{soul,color}
\sethlcolor{green} \setulcolor{blue} \setstcolor{red}

Rules can be in black blue. Rules \hl{can} be in \st{black} \ul{blue}.
3.1 Phrases and paragraphs

3.1.8 url—Typesetting URLs, path names, and the like

E-mail addresses, URLs, path or directory names, and similar objects usually require some attention to detail when typeset. For one thing, they often contain characters with special significance to \LaTeX, such as ~, #, &, {, or }. In addition, breaking them across lines should be avoided or at least done with special care. For example, it is usually not wise to break at a hyphen, because then it is not clear whether the hyphen was inserted because of the break (as it would be the case with normal words) or was already present. Similar reasons make breaks at a space undesirable. To help with these issues, Donald Arseneau wrote the url package, which attempts to solve most of these problems.

\[ \url{text} \quad \url!{text!} \quad \path{text} \quad \path=text=text \]

The base command provided by the package is \url, which is offered in two syntax variants: the text argument either can be surrounded by braces (in which case the text must not contain unbalanced braces) or, like \verb, can be delimited by using an arbitrary character on both sides that is not used inside text. (The syntax box above uses ! and = but these are really only examples.) In that second form one can have unbalanced braces in the argument.

The \path command is the same except that it always uses typewriter fonts (\ttfamily), while \url can be customized as we will see below. The argument to both commands is typeset pretty much verbatim. For example, \url{~} produces a tilde. Spaces are ignored by default, as can be seen in the following example.

\begin{verbatim}
\url{http://www.latex-project.org} and my home directory is ~frank (sometimes).
\end{verbatim}

Line breaks can happen at certain symbols (by default, not between letters or hyphens) and in no case can the commands add a hyphen at the break point. Whenever the text contains either of the symbols % or #, or ends with \, it cannot be used in the argument to another command without producing errors (just like the \verb command). Another case that does not work properly inside the argument of another command is the use of two ^ characters in succession. However, the situation is worse in that case because one might not even get an error but simply incorrect output\footnote{It depends on the letter that is following. An uppercase F instead of the lowercase f would produce an error.} as the next example shows.

\begin{verbatim}
\url{^frank} and \url{^frank} (OK)
\end{verbatim}

\begin{verbatim}
\url{^^frank} but \url{^^frank} (bad)
\end{verbatim}
Even if the text does not contain any critical symbols, it is always forbidden to use such a command inside a moving argument—for instance, the argument of a \section. If used there, you will get the error message

```
! Undefined control sequence.
\Url Error ->\url used in a moving argument.
```
followed by many strange errors. Even the use of \protect will not help in that case. So what can be done if one needs to cite a path name or a URL in such a place? If you are prepared to be careful and only use “safe” characters inside text, then you can enable the commands for use in moving arguments by specifying the option allowmove when loading the package. But this does not help if you actually need a character like “#”. In that case the solution is to record the information first using \urldef and then reuse it later.

```
\urldef{cmd}{url-cmd}{text} \urldef{cmd}{url-cmd}=text=
```

The declaration \urldef defines a new command cmd to contain the url-cmd (which might be \url, \path, or a newly defined command—see below) and the text in a way such that they can be used in any place, including a moving argument. The url-cmd is not executed at this point, which means that style changes can still affect the typesetting (see Example 3-1-33 on the facing page). Technically, what happens is that the \catcodes of characters in text are frozen during the declaration, so that they cannot be misinterpreted in places like arguments.

\section{\texttt{\urlstyle{style}}} 

We have already mentioned style changes. For this task the url package offers the \urlstyle command, which takes one mandatory argument: a named style. Predefined styles are \texttt{rm}, \texttt{sf}, \texttt{tt}, and \texttt{same}. The first three select the font family of that name, while the \texttt{same} style uses the current font and changes only the line breaking.

The \url command uses whatever style is currently in force (the default is \texttt{tt}, i.e., typewriter), while \path internally always switches to the \texttt{tt} style. In the following example we typeset a URL saved in \texttt{lproject} several times using different styles. The particular example may look slightly horrifying, but imagine how
3.1 Phrases and paragraphs

If you studied the previous example closely you will have noticed that the option `hyphens` was used. This option allows breaking at explicit hyphens, something normally disabled for \url-like commands. Without this option breaks would have been allowed only at the periods, after the colon, or after "//".

As mentioned earlier spaces inside `text` are ignored by default. If this is not desired one can use the option `obeyspaces`. However, this option may introduce spurious spaces if the \url command is used inside the argument of another command and `text` contains any "$" character. In that case \urldef solves the problem. Line breaks at spaces are not allowed unless you also use the option `spaces`.

The package automatically detects which font encoding is currently in use. In case of T1 encoded fonts it will make use of the additional glyphs available in this encoding, which improves the overall result.

The package offers two hooks, \UrlLeft and \UrlRight, that by default do nothing but can be redefined to typeset material at the left or right of `text`. The material is typeset in the same fashion as the `text`. For example, spaces are ignored unless one uses `\␣` or specifies `obeyspaces` as an option. If the commands are redefined at the top level, they act on every \url-like command. See Example 3-1-34 on the next page for a possibility to restrict their scope.

\begin{verbatim}
\DeclareUrlCommand{cmd}{style-information}
\end{verbatim}

It is sometimes helpful to define your own commands that work similarly to \url or \path but use their own fonts, and so on. The command `\DeclareUrlCommand` can be used to define a new \url-like command or to modify an existing one. It takes two arguments: the command to define or change and the `style-information` (e.g., `\urlstyle`).

In the next example, we define `\email` to typeset e-mail addresses in `rm` style, prepending the string “e-mail: ” via `\UrlLeft`. The example clearly shows that the scope for this redefinition is limited to the `\email` command. If you look closely, it would have looked if the URL had not been allowed to split at all in this narrow measure.

\begin{verbatim}
\usepackage[hyphens]{url}
\urldef\lproject\url{http://www.latex-project.org}
\fontfamily{pzc}\selectfont Zapf Chancery!
\lproject (default setup) \quad \urlstyle{rm}\lproject (CM Roman) \quad \urlstyle{sf}\lproject (CM Sans Serif) \quad \urlstyle{tt}\lproject (CM Typewriter) \quad \urlstyle{same}\lproject (Zapf Chancery)
\end{verbatim}
you can see that a space inside \UrlLeft (as in the top-level definition) has no effect, while \␣ produces the desired result.

```latex
\usepackage[url]
\renewcommand\UrlLeft{<url: }
\renewcommand\UrlRight{>}
\DeclareUrlCommand\email{\urlstyle{rm}%
    \renewcommand\UrlLeft{e-mail:\ }%
    \renewcommand\UrlRight{}}\par
\url{http://www.latex-project.org} \par
\email{frank.mittelbach@latex-project.org} \par
<path{$HOME(figures)} oops, wrong!
```

The url package offers a number of other hooks that influence line breaking, among them \UrlBreaks, \UrlBigBreaks, and \UrlNoBreaks. These hooks can be redefined in the style-information argument of \DeclareUrlCommand to set up new or special conventions. For details consult the package documentation, which can be found at the end of the file url.sty.

### 3.1.9 euro—Converting and typesetting currencies

To ease the calculations needed to convert between national units and the euro, Melchior Franz developed the package euro. In fact, the package converts arbitrary currencies using the euro as the base unit. The calculations are done with high precision using the fp package written by Michael Mehlich. The formatting is highly customizable on a per-currency basis, so that this package can be used for all kind of applications involving currencies whether or not conversions are needed.

```
\EURO{from-currency} [to-currency] {amount}
```

The main command \EURO converts an amount in from-currency into to-currency or, if this optional argument is missing, into euros. The arguments from-currency and to-currency are denoted in ISO currency codes, as listed in Table 3.1 on the facing page. When inputting the amount a dot must separate the integer value from any fractional part, even if the formatted number uses a different convention.

With the default settings the amount is displayed in the from-currency with the converted value in the to-currency shown in parentheses.

```
\usepackage[euro]
\EURO{DEM}{FRF}{7}\quad \EURO{FRF}{DEM}{23.48}\quad \EURO{EUR}{DEM}{10.00}\quad \EURO{DEM}{20}
```
3.1 Phrases and paragraphs

<table>
<thead>
<tr>
<th>EUR</th>
<th>Europe</th>
<th>GRD</th>
<th>Greece</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATS</td>
<td>Austria</td>
<td>IEP</td>
<td>Ireland</td>
</tr>
<tr>
<td>BEF</td>
<td>Belgium</td>
<td>ITL</td>
<td>Italy</td>
</tr>
<tr>
<td>DEM</td>
<td>Germany</td>
<td>LUF</td>
<td>Luxembourg</td>
</tr>
<tr>
<td>ESP</td>
<td>Spain</td>
<td>NLG</td>
<td>The Netherlands</td>
</tr>
<tr>
<td>FIM</td>
<td>Finland</td>
<td>PTE</td>
<td>Portugal</td>
</tr>
<tr>
<td>FRF</td>
<td>France</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.1: ISO currency codes of the euro and the 12 euro-zone countries

The package offers a number of options to influence the general style of the output (unless overwritten by the more detailed formatting declarations discussed below). With \texttt{eco} the ISO codes precede the value and no customized symbols are used; with \texttt{dots} a period is inserted between every three-digit group (the default is to use a small space).

By default, integer amounts are printed as such, without adding a decimal separator and a (zero) fractional part. If the \texttt{table} option is specified this behavior is globally changed and either a — (option \texttt{emdash}, also the default), a – (option \texttt{endash}), or the right number of zeros (option \texttt{zeros}) is used.

\begin{verbatim}
3-1-36
\usepackage[eco,table,endash]{euro}
\EURO{DEM}[FRF]{7}\quad \EURO{FRF}[DEM]{23.48}
\EURO{EUR}[DEM]{10.00}\quad \EURO{DEM}{20}
\end{verbatim}

The more detailed output customizations, which we discuss below, can be placed anywhere in the document. It is, however, advisable to keep them together in the preamble, or even to put them into the file \texttt{euro.cfg}, which is consulted upon loading the package.

The monetary symbols typeset can be adjusted with a \texttt{EUROSYM} declaration; as defaults the package uses the ISO codes for most currencies. The example below changes the presentation for lira and euro using the currency symbols from the \texttt{textcomp} package. It also uses \texttt{dots} to help with huge lira amounts.

\begin{verbatim}
3-1-37
10.000 ₤ (5.16 €) 1.000 DM (989.999 ₤)
\end{verbatim}

\begin{verbatim}
\usepackage[textcomp]\usepackage[dots]{euro}
\EUROSYM{ITL}\textlira\texttt{\EUROSYM{EUR}{\textlira}}
\EURO{ITL}{10000}\quad \EURO{DEM}{ITL}{1000}
\end{verbatim}

The package is well prepared for new countries to join the euro-zone. In fact, it is well prepared to deal with conversions from and to any currency as long as the conversion rate to the euro is known. To add a new currency use the \texttt{EUROADD} declaration, which takes three arguments: the ISO currency code, the symbol or text to display for the currency, and the conversion rate to the euro. The next
example makes the British pound available. Note the abbreviation \GBP, which makes the input a bit easier.

\begin{verbatim}
\usepackage{eurosans,euro}
\EUROADD{GBP}{\textsterling}{0.6397} % 2002/12/21
\newcommand*\GBP{\EURO{GBP}} \EUROSYM{EUR}{\euro}
\noindent \GBP{14.9}\ 
\GBP[FRF]{10}\ 
\EURO{EUR}[GBP]{10} \hfill 3-1-38
\end{verbatim}

The conversion rates for the national currencies of the euro-zone countries are fixed (and predefined by the package). With other currencies the rates may change hourly, so you have to be prepared for frequent updates.

The package allows you to tailor the presentation via \EUROFORMAT declarations, either to provide new defaults or to adjust the typesetting of individual currencies. The first argument specifies which part of the formatting should be adjusted, and the second argument describes the formatting.

The main format specifies how the source and target currencies are to be arranged using the reserved keywords \in and \out to refer to the source and target currencies, respectively. In the example below the first line implements a format close to the default, the second line displays the result of the conversion, and the third line does not show the conversion at all (although it happens behind the scenes). The latter is useful if you want to make use of the currency formatting features of the package without being interested in any conversion.

\begin{verbatim}
\usepackage{euro}
\EUROFORMAT{main}{\in\ (=,\ out)} \EURO{DEM}[FRF]{1000}\par
\EUROFORMAT{main}{\out} \EURO{DEM}[FRF]{1000}\par
\EUROFORMAT{main}{\in} \EURO{DEM}{1000} \hfill 3-1-39
\end{verbatim}

The in and out formats specify how the source and target currencies should be formatted using the reserved keywords \val (monetary amount), \iso (currency code), and \sym (currency symbol if defined; ISO code otherwise).

\begin{verbatim}
\usepackage{euro}
\EUROFORMAT{in}{\sym\val} \EUROFORMAT{out}{\iso\val}
\EURO{DEM}[FRF]{1000} \hfill 3-1-40
\end{verbatim}

Perhaps more interesting are the possibilities to influence the formatting of monetary amounts, for which the package offers five declarations to be used in the second argument to \EUROFORMAT. The \round declaration specifies where to round the monetary amount: positive values round to the integer digits and negative values to the fractional digits. For example, \round{-3} means show and round to three fractional digits. The \form declaration takes three arguments: the integer group separator (default \,), the decimal separator (default a comma), and the fractional group separator (default \,).