References for TeX and Friends

Peter Karp

tex-refs@karpfenteich.net

Michael Wiedmann

mw@miwie.in-berlin.de

2003-04-12

Revision History

Revision	Date	Remark
0.2.3	2003-04-12	Provide (experimental) PDF output using **ConTeXt** and **DocBook In ConTeXt**; reedited (beautified) sections LaTeX / Commands / Counters | Cross References | Definitions | Layout | Environments | Footnotes
0.0.1	2002-06-21	This version was adapted from the edition 1.6 of the LaTeX2e documentation, converted to DocBook XML using **texi2db**, and further edited manually.

Abstract

LaTeX2e is a document preparation system implemented as a macro package for Donald E. Knuth's TeX typesetting program.

LaTeX was originally conceived by Leslie Lamport.

This updated LaTeX reference is by no means complete, but it's a first step towards a more complete LaTeX reference! We welcome if we'll get pointed to missing commands or even better when you make additions and further improvements and send them to the authors.

TUG (TeX User Group) and **DANTE (German TeX User Group)** both have excellent FAQ's which answer many questions or give great pointers to additional resources both on the internet or in books. Most if not all missing packages can also be found their on the CTAN servers.

This reference is not meant to replace the package documentation or the standard literature like the LaTeX handbook and the LaTeX companion. Look there for detailed descriptions of the commands and even mostly more options and examples.

The authors intend to extend this documentation to other TeX related packages (e.g. ConTeXt, MetaPost, Metafont, etc.). Contributions are very welcome!

Contents

Tables

1 TeX

1.1 PlainTeX

1.1.1 Overview

For a really excellent reference of all TeX primitive control sequences see David Bausum's **TeX Primitive Control Sequences**.

1.2 LaTeX

1.2.1 Overview of LaTeX and Local Guide

The LaTeX command typesets a file of text using the TeX program and the LaTeX Macro package for TeX. To be more specific, it processes an input file containing the text of a document with interspersed commands that describe how the text should be formatted. It produces at least three files as output:

1. A *Device Independent*, or `.dvi` file. This contains commands that can be translated into commands for a variety of output devices. You can view the output of LaTeX by using a program such as `xdvi`, which actually uses the `.dvi` file.

2. A *transcript* or `.log` file that contains summary information and diagnostic messages for any errors discovered in the input file.

3. An *auxiliary* or `.aux` file. This is used by LaTeX itself, for things such as sectioning.

For a description of what goes on inside TeX, you should consult The TeXbook by Donald E. Knuth, ISBN 0-201-13448-9, published jointly by the American Mathematical Society and Addison-Wesley Publishing Company.

For a description of LaTeX, you should consult:

LaTeX: A Document Preparation System, by Leslie Lamport, Addison-Wesley Publishing Company, 2nd edition, 1994.

The LaTeX Companion, by Michel Goossens, Frank Mittelbach, and Alexander Samarin, Addison-Wesley, 1994.

1.2.2 Commands

A LaTeX command begins with the command name, which consists of a \ followed by either (a) a string of letters or (b) a single non-letter. Arguments contained in square brackets, `[]`, are optional while arguments contained in braces, `{}`, are required.

NOTE: LaTeX is case sensitive. Enter all commands in lower case unless explicitly directed to do otherwise.

1.2.2.1 Counters

Everything LaTeX numbers for you has a counter associated with it. The name of the counter is the same as the name of the environment or command that produces the number, except with no \ (`enumi` - `enumiv` are used for the nested enumerate environment). Below is a list of the counters used in LaTeX's standard document classes to control numbering.

```
part            paragraph       figure          enumi
chapter         subparagraph    table           enumii
section         page            footnote        enumiii
subsection      equation        mpfootnote      enumiv
subsubsection
```

\addtocounter

The `\addtocounter` command increments the `counter` by the amount specified by the `value` argument. The `value` argument can be negative.

`\alph{counter}`

This command causes the value of the `counter` to be printed in alphabetic characters. The `\alph` command uses lower case alphabetic alphabetic characters, i.e., `a, b, c...` while the `\Alph` command uses upper case alphabetic characters, i.e., `A, B, C...`.

`\arabic{counter}`

The `\arabic` command causes the value of the `counter` to be printed in Arabic numbers, i.e., `3`.

`\fnsymbol{counter}`

The `\fnsymbol` command causes the value of the `counter` to be printed in a specific sequence of nine symbols that can be used for numbering footnotes.

NB. `counter` must have a value between 1 and 9 inclusive.

`\newcounter{foo}[counter]`

The `\newcounter` command defines a new counter named `foo`. The counter is initialized to zero.

The optional argument `[counter]` causes the counter `foo` to be reset whenever the counter named in the optional argument is incremented.

`\refstepcounter{counter}`

The `\refstepcounter` command works like `\stepcounter` See \stepcounter, except it also defines the current `\ref` value to be the result of `\thecounter`.

`\roman{counter}`

This command causes the value of the `counter` to be printed in Roman numerals. The `\roman` command uses lower case Roman numerals, i.e., `i, ii, iii...`, while the `\Roman` command uses upper case Roman numerals, i.e., `I, II, III...`.

`\stepcounter{counter}`

The `\stepcounter` command adds one to the `counter` and resets all subsidiary counters.

`\setcounter{counter}{value}`

The `\setcounter` command sets the value of the `counter` to that specified by the `value` argument.

`\usecounter{counter}`

The `\usecounter` command is used in the second argument of the `list` environment to allow the counter specified to be used to number the list items.

`\value{counter}`

The `\value` command produces the value of the `counter` named in the mandatory argument. It can be used where LaTeX expects an integer or number, such as the second argument of a `\setcounter` or `\addtocounter` command, or in:

```
\hspace{\value{foo}\parindent}
```

It is useful for doing arithmetic with counters.

1.2.2.2 Cross References

One reason for numbering things like figures and equations is to refer the reader to them, as in *See Figure 3 for more details.* A smarter version of the cross reference commands is available in the package `varioref`.

`\label{key}`

A `\label` command appearing in ordinary text assigns to the `key` the number of the current sectional unit; one appearing inside a numbered environment assigns that number to the `key`.

A `key` can consist of any sequence of letters, digits, or punctuation characters. Upper and lowercase letters are different.

To avoid accidentally creating two labels with the same name, it is common to use labels consisting of a prefix and a suffix separated by a colon. The prefixes conventionally used are

- `cha` for chapters
- `sec` for lower-level sectioning commands
- `fig` for figures
- `tab` for tables
- `eq` for equations

Thus, a label for a figure would look like `fig:bandersnatch`.

`\pageref{key}`

The `\pageref` command produces the page number of the place in the text where the corresponding `\label` command appears. ie. where `\label{key}` appears. For a smart version see `\vpageref`.

`\ref{key}`

The `\ref` command produces the number of the sectional unit, equation number, ... of the corresponding `\label` command. For a smart version see \vref.

1.2.2.3 Definitions

`\newcommand`

```
\newcommand{cmd}[args]{definition}
\newcommand{cmd}[args][default]{definition}
\renewcommand{cmd}[args]{definition}
\renewcommand{cmd}[args][default]{definition}
```

These commands define (or redefine) a command.

`\newenvironment`

```
\newenvironment{nam}[args]{begdef}{enddef}
\newenvironment{nam}[args][default]{begdef}{enddef}
\renewenvironment{nam}[args]{begdef}{enddef}
```

These commands define or redefine an environment.

`\newtheorem`

```
\newtheorem{env_name}{caption}[within]
\newtheorem{env_name}[numbered_like]{caption}
```

cmd	A command name beginning with a \. For `\newcommand` it must not be already defined and must not begin with `\end`; for `\renewcommand` it must already be defined.
args	An integer from 1 to 9 denoting the number of arguments of the command being defined. The default is for the command to have no arguments.
def	If this optional parameter is present, it means that the command's first argument is optional. The default value of the optional argument is `def`.
definition	The text to be substituted for every occurrence of `cmd`; a parameter of the form `#n` in `cmd` is replaced by the text of the nth argument when this substitution takes place.

Table 1 Options for `\newcommand`

nam	The name of the environment. For `\newenvironment` there must be no currently defined environment by that name, and the command `\nam` must be undefined. For `\renewenvironment` the environment must already be defined.
args	An integer from 1 to 9 denoting the number of arguments of the newly-defined environment. The default is no arguments. Arguments can only be used in te `{begdef}` part.
default	If this is specified, the first argument is optional, and `default` gives the default value for that argument.
begdef	The text substituted for every occurrence of `\begin{nam}`; a parameter of the form `#n` in `cmd` is replaced by the text of the nth argument when this substitution takes place.
enddef	The text substituted for every occurrence of `\end{nam}`. It may not contain any argument parameters.

Table 2 Options for `\newenvironment`

This command defines a theorem-like environment.

The `\newtheorem` command may have at most one optional argument.

`\newfont`

`\newfont{cmd}{font_name}`

Defines the command name `cmd`, which must not be currently defined, to be a declaration that selects the font named `font_name` to be the current font.

env_name	The name of the environment to be defined. A string of letters. It must not be the name of an existing environment or counter.
caption	The text printed at the beginning of the environment, right before the number. This may simply say *Theorem*, for example.
within	The name of an already defined counter, usually of a sectional unit. Provides a means of resetting the new theorem counter *within* the sectional unit.
numbered_like	The name of an already defined theorem-like environment.

Table 3 Options for `\newtheorem`

1.2.2.4 Document Classes

Valid LaTeX document classes include:

- article
- report
- letter
- book
- slides

Other document classes are often available. See Overview, for details. They are selected with the following command:

```
\documentclass [options] {class}
```

All the standard classes (except slides) accept the following options for selecting the typeface size (10 pt is default):

10pt, 11pt, 12pt

All classes accept these options for selecting the paper size (default is letter):

a4paper, a5paper, b5paper, letterpaper, legalpaper, executivepaper

Miscellaneous options:

- landscape — selects landscape format. Default is portrait.
- titlepage, notitlepage — selects if there should be a separate title page.
- leqno — equation number on left side of equations. Default is right side.
- fleqn — displayed formulas flush left. Default is centered.
- openbib — use *open* bibliography format.
- draft, final — mark/do not mark overfull boxes with a rule. Default is final.

These options are not available with the slides class:

- oneside, twoside — selects one- or twosided layout. Default is oneside, except for the book class.
- openright, openany — determines if a chapter should start on a right-hand page. Default is openright for book.
- onecolumn, twocolumn — one or two columns. Defaults to one column.

The slides class offers the option `clock` for printing the time at the bottom of each note.

If you specify more than one option, they must be separated by a comma.

Additional packages are loaded by a

`\usepackage[options]{pkg}`

command. If you specify more than one package, they must be separated by a comma.

Any options given in the `\documentclass` command that are unknown by the selected document class are passed on to the packages loaded with `\usepackage`.

1.2.2.5 Layout

Miscellaneous commands for controlling the general layout of the page.

`\flushbottom`

The `\flushbottom` declaration makes all text pages the same height, adding extra vertical space when necessary to fill out the page.

This is the standard if twocolumn mode is selected.

`\onecolumn`

The `\onecolumn` declaration starts a new page and produces single-column output.

`\raggedbottom`

The `\raggedbottom` declaration makes all pages the height of the text on that page. No extra vertical space is added.

`\twocolumn`

`\twocolumn[text]`

The `\twocolumn` declaration starts a new page and produces two-column output. If the optional `text` argument is present, it is typeset in one-column mode.

1.2.2.6 Environments

LaTeX provides a number of different paragraph-making environments. Each environment begins and ends in the same manner.

```
\begin{environment-name}
 .
 .
 .
\end{environment-name}
```

array

```
\begin{array}{col1col2...coln}
column 1 entry & column 2 entry ... & column n entry \\
 .
 .
 .
\end{array}
```

Math arrays are produced with the array environment. It has a single mandatory argument describing the number of columns and the alignment within them. Each column, `coln`, is specified by a single letter that tells how items in that row should be formatted.

- `c` — for centered
- `l` — for flush left
- `r` — for flush right

Column entries must be separated by an `&`. Column entries may include other LaTeX commands. Each row of the array must be terminated with the string `\\`.

Note that the `array` environment can only be used in math mode, so normally it is used inside an `equation` environment.

`center`

```
\begin{center}
Text on line 1 \\
Text on line 2 \\
 .
 .
 .
\end{center}
```

The `center` environment allows you to create a paragraph consisting of lines that are centered within the left and right margins on the current page. Each line must be terminated with the string `\\`.

`\centering`

This declaration corresponds to the `center` environment. This declaration can be used inside an environment such as `quote` or in a `parbox`. The text of a figure or table can be centered on the page by putting a `\centering` command at the beginning of the figure or table environment.

Unlike the `center` environment, the `\centering` command does not start a new paragraph; it simply changes how LaTeX formats paragraph units. To affect a paragraph unit's format, the scope of the declaration must contain the blank line or `\end` command (of an environment like quote) that ends the paragraph unit.

`description`

```
\begin{description}
\item [label] First item
\item [label] Second item
 .
 .
 .
\end{description}
```

The `description` environment is used to make labelled lists. The `label` is bold face and flushed right.

`enumerate`

```
\begin{enumerate}
\item First item
\item Second item
 .
 .
 .
\end{enumerate}
```

The `enumerate` environment produces a numbered list. Enumerations can be nested within one another, up to four levels deep. They can also be nested within other paragraph-making environments.

Each item of an enumerated list begins with an `\item` command. There must be at least one `\item` command within the environment.

The `enumerate` environment uses the `enumi` through `enumiv` counters (see Counters). The type of numbering can be changed by redefining `\theenumi` etc.

`eqnarray`

```
\begin{eqnarray}
math formula 1 \\
math formula 2 \\
```

```
 .
 .
 .
\end{eqnarray}
```

The `eqnarray` environment is used to display a sequence of equations or inequalities. It is very much like a three-column `array` environment, with consecutive rows separated by `\\` and consecutive items within a row separated by an `&`.

An equation number is placed on every line unless that line has a `\nonumber` command.

The command `\lefteqn` is used for splitting long formulas across lines. It typesets its argument in display style flush left in a box of zero width.

`equation`

```
\begin{equation}
 math formula
\end{equation}
```

The `equation` environment centers your equation on the page and places the equation number in the right margin.

`figure`

```
\begin{figure}[placement]

 body of the figure

\caption{figure title}
\end{figure}
```

Figures are objects that are not part of the normal text, and are usually *floated* to a convenient place, like the top of a page. Figures will not be split between two pages.

The optional argument `[placement]` determines where LaTeX will try to place your figure. There are four places where LaTeX can possibly put a float:

1. `h` (Here) - at the position in the text where the figure environment appears.
2. `t` (Top) - at the top of a text page.
3. `b` (Bottom) - at the bottom of a text page.
4. `p` (Page of floats) - on a separate float page, which is a page containing no text, only floats.

The standard report and article classes use the default placement `tbp`.

The body of the figure is made up of whatever text, LaTeX commands, etc. you wish. The `\caption` command allows you to title your figure.

`flushleft`

```
\begin{flushleft}
Text on line 1 \\
Text on line 2 \\
 .
 .
 .
\end{flushleft}
```

The `flushleft` environment allows you to create a paragraph consisting of lines that are flushed left, to the left-hand margin. Each line must be terminated with the string `\\`.

`\raggedright`

This declaration corresponds to the `flushleft` environment. This declaration can be used inside an environment such as `quote` or in a `parbox`.

Unlike the `flushleft` environment, the `\raggedright` command does not start a new paragraph; it simply changes how LaTeX formats paragraph units. To affect a paragraph unit's format, the scope of the declaration must contain the blank line or `\end` command (of an environment like quote) that ends the paragraph unit.

`flushright`

```
\begin{flushright}
Text on line 1 \\
Text on line 2 \\
 .
 .
 .
\end{flushright}
```

The `flushright` environment allows you to create a paragraph consisting of lines that are flushed right, to the right-hand margin. Each line must be terminated with the string `\\`.

`\raggedleft`

This declaration corresponds to the `flushright` environment. This declaration can be used inside an environment such as `quote` or in a `parbox`.

Unlike the `flushright` environment, the `\raggedleft` command does not start a new paragraph; it simply changes how LaTeX formats paragraph units. To affect a

paragraph unit's format, the scope of the declaration must contain the blank line or `\end` command (of an environment like quote) that ends the paragraph unit.

`itemize`

```
\begin{itemize}
\item First item
\item Second item
 .
 .
 .
\end{itemize}
```

The `itemize` environment produces a *bulleted* list. Itemizations can be nested within one another, up to four levels deep. They can also be nested within other paragraph-making environments.

Each item of an `itemized` list begins with an `\item` command. There must be at least one `\item` command within the environment.

The `itemize` environment uses the `itemi` through `itemiv` counters (see Counters). The type of numbering can be changed by redefining `\theitemi` etc.

`letter`

This environment is used for creating letters. See Letters.

`list`

The `list` environment is a generic environment which is used for defining many of the more specific environments. It is seldom used in documents, but often in macros.

```
\begin{list}{label}{spacing}
\item First item
\item Second item
 .
 .
 .
\end{list}
```

The `{label}` argument specifies how items should be labelled. This argument is a piece of text that is inserted in a box to form the label. This argument can and usually does contain other LaTeX commands.

The `{spacing}` argument contains commands to change the spacing parameters for the list. This argument will most often be null, i.e., `{}`. This will select all default spacing which should suffice for most cases.

`minipage`

```
\begin{minipage}[position][height][inner-pos]{width}
 text
\end{minipage}
```

The `minipage` environment is similar to a `\parbox` command. It takes the same optional `position` argument and mandatory `width` argument. You may use other paragraph-making environments inside a minipage.

Footnotes in a `minipage` environment are handled in a way that is particularly useful for putting footnotes in figures or tables. A `\footnote` or `\footnotetext` command puts the footnote at the bottom of the minipage instead of at the bottom of the page, and it uses the `mpfootnote` counter instead of the ordinary `footnote` counter See Counters.

NOTE: Don't put one minipage inside another if you are using footnotes; they may wind up at the bottom of the wrong minipage.

`picture`

```
\begin{picture}(width,height)(x offset,y offset)
 .
 .
 picture commands
 .
 .
\end{picture}
```

The `picture` environment allows you to create just about any kind of picture you want containing text, lines, arrows and circles. You tell LaTeX where to put things in the picture by specifying their coordinates. A coordinate is a number that may have a decimal point and a minus sign — a number like `5`, `2.3` or `-3.1416`. A coordinate specifies a length in multiples of the unit length `\unitlength`, so if `\unitlength` has been set to `1cm`, then the coordinate 2.54 specifies a length of 2.54 centimetres. You can change the value of `\unitlength` anywhere you want, using the `\setlength` command, but strange things will happen if you try changing it inside the picture environment.

A position is a pair of coordinates, such as `(2.4,-5)`, specifying the point with x-coordinate `2.4` and y-coordinate `-5`. Coordinates are specified in the usual way with respect to an origin, which is normally at the lower-left corner of the picture. Note that when a position appears as an argument, it is not enclosed in braces; the parentheses serve to delimit the argument.

The `picture` environment has one mandatory argument, which is a `position`. It specifies the size of the picture. The environment produces a rectangular box with width and height determined by this argument's x- and y-coordinates.

The `picture` environment also has an optional `position` argument, following the `size` argument, that can change the origin. (Unlike ordinary optional arguments, this argument is not contained in square brackets.) The optional argument gives the coordinates of the point at the lower-left corner of the picture (thereby determining the origin). For example, if `\unitlength` has been set to `1mm`, the command

```
\begin{picture}(100,200)(10,20)
```

produces a picture of width 100 millimetres and height 200 millimetres, whose lower-left corner is the point (10,20) and whose upper-right corner is therefore the point (110,220). When you first draw a picture, you will omit the optional argument, leaving the origin at the lower-left corner. If you then want to modify your picture by shifting everything, you just add the appropriate optional argument.

The environment's mandatory argument determines the nominal size of the picture. This need bear no relation to how large the picture really is; LaTeX will happily allow you to put things outside the picture, or even off the page. The picture's nominal size is used by LaTeX in determining how much room to leave for it.

Everything that appears in a picture is drawn by the `\put` command. The command

```
\put (11.3,-.3){...}
```

puts the object specified by `...` in the picture, with its reference point at coordinates (11.3,-.3). The reference points for various objects will be described below.

The `\put` command creates an *LR box*. You can put anything in the text argument of the `\put` command that you'd put into the argument of an `\mbox` and related commands. When you do this, the reference point will be the lower left corner of the box.

Picture commands:

`\circle{diameter}, \circle*{diameter}`

The `\circle` command produces a circle with a diameter as close to the specified one as possible. If the `*`-form of the command is used, LaTeX draws a solid circle.

Note that only circles up to 40 pt can be drawn.

`\dashbox`

Draws a box with a dashed line.

`\dashbox{dash_length}(width,height){...}`

The `\dashbox` has an extra argument which specifies the width of each dash. A dashed box looks best when the `width` and `height` are multiples of the `dash_length`.

`\frame{...}`

The `\frame` command puts a rectangular frame around the object specified in the argument. The reference point is the bottom left corner of the frame. No extra space is put between the frame and the object.

`\framebox(width,height)[position]{...}`

The `\framebox` command is exactly the same as the `\makebox` command, except that it puts a frame around the outside of the box that it creates.

The `framebox` command produces a rule of thickness `\fboxrule`, and leaves a space `\fboxsep` between the rule and the contents of the box.

`\line(x slope,y slope){length}`

The `\line` command draws a line of the specified `length` and `slope`.

Note that LaTeX can only draw lines with slope = x/y, where x and y have integer values from -6 through 6.

`\linethickness{dimension}`

Declares the thickness of horizontal and vertical lines in a picture environment to be `dimension`, which must be a positive length. It does not affect the thickness of slanted lines and circles, or the quarter circles drawn by `\oval` to form the corners of an oval.

`\makebox(width,height)[position]{...}`

The `\makebox` command for the picture environment is similar to the normal `\makebox` command except that you must specify a `width` and `height` in multiples of `\unitlength`.

The optional argument, `[position]`, specifies the quadrant that your text appears in. You may select up to two of the following:

- `t` - Moves the item to the top of the rectangle
- `b` - Moves the item to the bottom
- `l` - Moves the item to the left
- `r` - Moves the item to the right

See \makebox.

`\multiput(x coord,y coord)(delta x,delta y){number of copies}{object}`

The `\multiput` command can be used when you are putting the same object in a regular pattern across a picture.

`\oval(width,height)[portion]`

The `\oval` command produces a rectangle with rounded corners. The optional argument, `[portion]`, allows you to select part of the oval.

- `t` - Selects the top portion
- `b` - Selects the bottom portion

- `r` - Selects the right portion
- `l` - Selects the left portion

Note that the *corners* of the oval are made with quarter circles with a maximum radius of 20 pt, so large *ovals* will look more like boxes with rounded corners.

`\put(x coord,y coord){ ... }`

The `\put` command places the item specified by the mandatory argument at the given coordinates.

`\shortstack[position]{... \\ ... \\ ...}`

The `\shortstack` command produces a stack of objects. The valid positions are:

- `r` - Moves the objects to the right of the stack
- `l` - Moves the objects to the left of the stack
- `c` - Moves the objects to the centre of the stack (default)

`\vector(x slope,y slope){length}`

The `\vector` command draws a line with an arrow of the specified length and slope. The `x` and `y` values must lie between -4 and +4, inclusive.

`quotation`

```
\begin{quotation}
 text
\end{quotation}
```

The margins of the `quotation` environment are indented on the left and the right. The text is justified at both margins and there is paragraph indentation. Leaving a blank line between text produces a new paragraph.

`quote`

```
\begin{quote}
 text
\end{quote}
```

The margins of the `quote` environment are indented on the left and the right. The text is justified at both margins. Leaving a blank line between text produces a new paragraph.

`tabbing`

```
\begin{tabbing}
text \= more text \= still more text \= last text \\
second row \>  \> more \\
 .
```

```
 .
 .
\end{tabbing}
```

The `tabbing` environment provides a way to align text in columns. It works by setting tab stops and tabbing to them much the way you do with an ordinary typewriter.

It is best suited for cases where the width of each column is constant and known in advance.

This environment can be broken across pages, unlike the `tabular` environment.

The following commands can be used inside a `tabbing` environment:

This example typesets a Pascal function in a traditional format:

```
\begin{tabbing}
function \= fact(n : integer) : integer;\\
    \> begin \= \+ \\
        \> if \= n $>$ 1 then \+ \\
                  fact := n * fact(n-1) \- \\
              else \+ \\
                  fact := 1; \-\- \\
              end;\\
\end{tabbing}
```

`table`

```
\begin{table}[placement]

 body of the table

\caption{table title}
\end{table}
```

Tables are objects that are not part of the normal text, and are usually *floated* to a convenient place, like the top of a page. Tables will not be split between two pages.

The optional argument `[placement]` determines where LaTeX will try to place your table. There are four places where LaTeX can possibly put a float:

- `h` : Here - at the position in the text where the table environment appears.
- `t` : Top - at the top of a text page.
- `b` : Bottom - at the bottom of a text page.
- `p` : Page of floats - on a separate float page, which is a page containing no text, only floats.

\=	Sets a tab stop at the current position.
\>	Advances to the next tab stop.
\<	This command allows you to put something to the left of the local margin without changing the margin. Can only be used at the start of the line.
\+	Moves the left margin of the next and all the following commands one tab stop to the right.
\-	Moves the left margin of the next and all the following commands one tab stop to the left.
\'	Moves everything that you have typed so far in the current column, i.e. everything from the most recent `\>`, `\<`, `\'`, `\\`, or `\kill` command, to the right of the previous column, flush against the current column's tab stop.
\`	Allows you to put text flush right against any tab stop, including tab stop 0. However, it can't move text to the right of the last column because there's no tab stop there. The `\`` command moves all the text that follows it, up to the `\\` or `\end{tabbing}` command that ends the line, to the right margin of the tabbing environment. There must be no `\>` or `\'` command between the `\`` and the command that ends the line.
\kill	Sets tab stops without producing text. Works just like `\\` except that it throws away the current line instead of producing output for it. The effect of any `\=`, `\+` or `\-` commands in that line remain in effect.
\pushtabs	Saves all current tab stop positions. Useful for temporarily changing tab stop positions in the middle of a `tabbing` environment.
\pushtabs	Restores the tab stop positions saved by the last `\pushtabs`.
\a	In a `tabbing` environment, the commands `\=`, `\'` and `\`` do not produce accents as normal. Instead, the commands `\a=`, `\a'` and `\a`` are used.

Table 4 Commands in a `tabbing` environment

The standard `report` and `article` classes use the default placement `[tbp]`.

The body of the table is made up of whatever text, LaTeX commands, etc., you wish. The `\caption` command allows you to title your table.

`tabular`

```
\begin{tabular}[pos]{cols}
column 1 entry & column 2 entry ... & column n entry \\
```

```
 .
 .
 .
\end{tabular}
```

or

```
\begin{tabular*}{width}[pos]{cols}
column 1 entry & column 2 entry ... & column n entry \\
 .
 .
 .
\end{tabular*}
```

These environments produce a box consisting of a sequence of rows of items, aligned vertically in columns. The mandatory and optional arguments consist of:

These commands can be used inside a `tabular` environment:

`\cline{i-j}`

The `\cline` command draws horizontal lines across the columns specified, beginning in column `i` and ending in column `j`, which are identified in the mandatory argument.

`\hline`

The `\hline` command will draw a horizontal line the width of the table. It's most commonly used to draw a line at the top, bottom, and between the rows of the table.

`\multicolumn`

`\multicolumn{cols}{pos}{text}`

The `\multicolumn` is used to make an entry that spans several columns. The first mandatory argument, `cols`, specifies the number of columns to span. The second mandatory argument, `pos`, specifies the formatting of the entry; `c` for centred, `l` for flushleft, `r` for flushright. The third mandatory argument, `text`, specifies what text is to make up the entry.

`\vline`

The `\vline` command will draw a vertical line extending the full height and depth of its row. An `\hfill` command can be used to move the line to the edge of the column. It can also be used in an @-expression.

`thebibliography`

```
\begin{thebibliography}{widest-label}
\bibitem[label]{cite_key}
 .
```

width Specifies the width of the `tabular*` environment. There must be rubber space between columns that can stretch to fill out the specified width.

pos Specifies the vertical position; default is alignment on the centre of the environment.

- `t` - align on top row
- `b` - align on bottom row

cols Specifies the column formatting. It consists of a sequence of the following specifiers, corresponding to the sequence of columns and intercolumn material.

- `l` - A column of left-aligned items.
- `r` - A column of right-aligned items.
- `c` - A column of centred items.
- `|` - A vertical line the full height and depth of the environment.
- `@{text}` - This inserts `text` in every row. An @-expression suppresses the intercolumn space normally inserted between columns; any desired space between the inserted text and the adjacent items must be included in text. An `\extracolsep{wd}` command in an @-expression causes an extra space of width `wd` to appear to the left of all subsequent columns, until countermanded by another `\extracolsep` command. Unlike ordinary intercolumn space, this extra space is not suppressed by an @-expression. An `\extracolsep` command can be used only in an @-expression in the `cols` argument.
- `p{wd}` - Produces a column with each item typeset in a parbox of width `wd`, as if it were the argument of a `\parbox[t]{wd}` command. However, a `\\` may not appear in the item, except in the following situations:
  1. inside an environment like `minipage`, `array`, or `tabular`.
  2. inside an explicit `\parbox`.
  3. in the scope of a `\centering`, `\raggedright`, or `\raggedleft` declaration. The latter declarations must appear inside braces or an environment when used in a `p`-column element.
- `*{num}{cols}` - Equivalent to `num` copies of `cols`, where `num` is any positive integer and `cols` is any list of column-specifiers, which may contain another `*-expression`.

Table 5 Commands used inside a `tabular` environment

```
  .
  .
\end{thebibliography}
```

The `thebibliography` environment produces a bibliography or reference list. In the `article` class, this reference list is labelled *References*; in the `report` class, it is labelled *Bibliography*.

- `widest-label`: Text that, when printed, is approximately as wide as the widest item label produces by the `\bibitem` commands.

`\bibitem[label]{cite_key}`

The `\bibitem` command generates an entry labelled by `label`. If the `label` argument is missing, a number is generated as the `label`, using the `enumi` counter. The `cite_key` is any sequence of letters, numbers, and punctuation symbols not containing a comma. This command writes an entry on the `.aux` file containing `cite_key` and the item's `label`. When this `.aux` file is read by the `\begin{document}` command, the item's `label` is associated with `cite_key`, causing the reference to `cite_key` by a `\cite` command to produce the associated `label`.

`\cite[text]{key_list}`

The `key_list` argument is a list of citation keys. This command generates an in-text citation to the references associated with the keys in `key_list` by entries on the `.aux` file read by the `\begin{document}` command.

The optional `text` argument will appear after the citation, i.e. `\cite[p.~2f.]{knuth}` might produce '[Knuth, p. 2]'.

The package `cite.sty` allows a line break in the `\cite` reference and can sort numerical entries.

`overcite.sty` makes citations like footnotes.

`\nocite{key_list}`

The `\nocite` command produces no text, but writes `key_list`, which is a list of one or more citation keys, on the `.aux` file. `\nocite{*}` uses all entries from the Bibtex database.

Using BibTeX

If you use the BibTeX program by Oren Patashnik (highly recommended if you need a bibliography of more than a couple of titles) to maintain your bibliography, you don't use the `thebibliography` environment. Instead, you include the lines

```
\bibliographystyle{style}
\bibliography{bibfile}
```

where `style` refers to a file `style.bst`, which defines how your citations will look.

alpha	Sorted alphabetically. Labels are formed from name of author and year of publication.
plain	Sorted alphabetically. Labels are numeric.
unsrt	Like `plain`, but entries are in order of citation.
abbrv	Like `plain`, but more compact labels.

Table 6 Standard styles distributed with BibTeX

In addition, numerous other BibTeX style files exist tailored to the demands of various publications.

The argument to `\bibliography` refers to the file `bibfile.bib`, which should contain your database in BibTeX format. Only the entries referred to via `\cite` and `\nocite` will be listed in the bibliography.

`theorem`

```
\begin{theorem}
 theorem text
\end{theorem}
```

The `theorem` environment produces *Theorem x* in boldface followed by your theorem text.

`titlepage`

```
\begin{titlepage}
 text
\end{titlepage}
```

The `titlepage` environment creates a title page, i.e. a page with no printed page number or heading. It also causes the following page to be numbered page one. Formatting the title page is left to you. The `\today` command comes in handy for title pages.

Note that you can use the `\maketitle` (see \maketitle) command to produce a standard title page.

`verbatim`

```
\begin{verbatim}
 text
\end{verbatim}
```

The `verbatim` environment is a paragraph-making environment that gets LaTeX to print exactly what you type in. It turns LaTeX into a typewriter with carriage returns and blanks having the same effect that they would on a typewriter.

`\verb char literal_text char` `\verb*char literal_text char`

Typesets `literal_text` exactly as typed, including special characters and spaces, using a typewriter (`\tt`) type style. There may be no space between `\verb` or `\verb*` and `char` (space is shown here only for clarity). The `*-form` differs only in that spaces are printed

`verse`

```
\begin{verse}
 text
\end{verse}
```

The `verse` environment is designed for poetry, though you may find other uses for it.

The margins are indented on the left and the right. Separate the lines of each stanza with \\, and use one or more blank lines to separate the stanzas.

1.2.2.7 Footnotes

Footnotes can be produced in one of two ways. They can be produced with one command, the `\footnote` command. They can also be produced with two commands, the `\footnotemark` and the `\footnotetext` commands. See the specific command for information on why you would use one over the other.

`\footnote[number]{text}`

The `\footnote` command places the numbered footnote `text` at the bottom of the current page. The optional argument, `number`, is used to change the default footnote number. This command can only be used in outer paragraph mode; i.e., you cannot use it in sectioning commands like `\chapter`, in figures, tables or in a `tabular` environment.

`\footnotemark`

The `\footnotemark` command puts the footnote `number` in the text. This command can be used in inner paragraph mode. The text of the footnote is supplied by the `\footnotetext` command.

This command can be used to produce several consecutive footnote markers referring to the same footnote by using

`\footnotemark[\value{footnote}]`

after the first `\footnote` command.

`\footnotetext[number]{text}`

The `\footnotetext` command produces the `text` to be placed at the bottom of the page. This command can come anywhere after the `\footnotemark` command. The `\footnotetext` command must appear in outer paragraph mode.

The optional argument, `number`, is used to change the default footnote number.

1.2.2.8 Lengths

A `length` is a measure of distance. Many LaTeX commands take a length as an argument.

\newlength

`\newlength{\gnat}`

The `\newlength` command defines the mandatory argument, `\gnat`, as a `length` command with a value of `0in`. An error occurs if a `\gnat` command already exists.

\setlength

`\setlength{\gnat}{length}`

The `\setlength` command is used to set the value of a `length` command. The `length` argument can be expressed in any terms of length LaTeX understands, i.e., inches (`in`), millimetres (`mm`), points (`pt`), etc.

\addtolength

`\addtolength{\gnat}{length}`

The `\addtolength` command increments *length command* by the amount specified in the `length` argument. It can be a negative amount.

\settodepth

`\settodepth{\gnat}{text}`

The `\settodepth` command sets the value of a `length` command equal to the depth of the `text` argument.

\settoheight

`\settoheight{\gnat}{text}`

The `\settoheight` command sets the value of a `length` command equal to the height of the `text` argument.

\settowidth

`\settowidth{\gnat}{text}`

The `\settowidth` command sets the value of a `length` command equal to the width of the `text` argument.

Predefined lengths

`\width`

`\height`

`\depth`

`\totalheight`

These length parameters can be used in the arguments of the box-making commands See Spaces & Boxes. They specify the natural width etc. of the text in the box. `\totalheight` equals `\height` + `\depth`. To make a box with the text stretched to double the natural size, e.g., say

```
\makebox[2\width]{Get a stretcher}
```

1.2.2.9 Letters

You can use LaTeX to typeset letters, both personal and business. The `letter` document class is designed to make a number of letters at once, although you can make just one if you so desire.

Your `.tex` source file has the same minimum commands as the other document classes, i.e., you must have the following commands as a minimum:

```
\documentclass{letter}
\begin{document}
 ... letters ...
\end{document}
```

Each letter is a `letter` environment, whose argument is the name and address of the recipient. For example, you might have:

```
\begin{letter}{Mr. Joe Smith\\ 2345 Princess St.\\ Edinburgh, EH1 1AA}
  ...
\end{letter}
```

The letter itself begins with the `\opening` command. The text of the letter follows. It is typed as ordinary LaTeX input. Commands that make no sense in a letter, like `\chapter`, do not work. The letter closes with a `\closing` command.

After the `closing`, you can have additional material. The `\cc` command produces the usual *cc:* .... There's also a similar `\encl` command for a list of enclosures. With both these commands, use `\\` to separate the items.

These commands are used with the `letter` class:

\address

```
\address{Return address}
```

The return address, as it should appear on the letter and the envelope. Separate lines of the address should be separated by `\\` commands. If you do not make an `\address` declaration, then the letter will be formatted for copying onto your organisation's standard letterhead. (See Overview, for details on your local implementation). If you give an `\address` declaration, then the letter will be formatted as a personal letter.

\cc

`\cc{Kate Schechter\\Rob McKenna}`

Generate a list of other persons the letter was sent to. Each name is printed on a separate line.

\closing

`\closing{text}`

The letter closes with a `\closing` command, i.e.,

```
\closing{Best Regards,}
```

\encl

`\encl{CV\\Certificates}`

Generate a list of enclosed material.

\location

`\location{address}`

This modifies your organisation's standard address. This only appears if the `firstpage` pagestyle is selected.

\makelabels

`\makelabels{number}`

If you issue this command in the preamble, LaTeX will create a sheet of address labels. This sheet will be output before the letters.

\name

`\name{June Davenport}`

Your name, used for printing on the envelope together with the return address.

\opening

`\opening{text}`

The letter begins with the `\opening` command. The mandatory argument, `text`, is whatever text you wish to start your letter, i.e.,

```
\opening{Dear Joe,}
```

\ps

`\ps`

Use this command before a postscript.

\signature

`\signature{Harvey Swick}`

Your name, as it should appear at the end of the letter underneath the space for your signature. Items that should go on separate lines should be separated by \\ commands.

\startbreaks

`\startbreaks`

Used after a `\stopbreaks` command to allow page breaks again.

\stopbreaks

`\stopbreaks`

Inhibit page breaks until a `\startbreaks` command occurs.

\telephone

`\telephone{number}`

This is your telephone number. This only appears if the `firstpage` pagestyle is selected.

1.2.2.10 Line & Page Breaking

The first thing LaTeX does when processing ordinary text is to translate your input file into a string of glyphs and spaces. To produce a printed document, this string must be broken into lines, and these lines must be broken into pages. In some environments, you do the line breaking yourself with the \\ command, but LaTeX usually does it for you.

\\

`\\[extra-space], \\*[extra-space]`

The \\ command tells LaTeX to start a new line. It has an optional argument, `extra-space`, that specifies how much extra vertical space is to be inserted before the next line. This can be a negative amount.

The `\\*` command is the same as the ordinary \\ command except that it tells LaTeX not to start a new page after the line.

\-

The `\-` command tells LaTeX that it may hyphenate the word at that point. LaTeX is very good at hyphenating, and it will usually find all correct hyphenation points. The `\-` command is used for the exceptional cases.

Note that when you insert `\-` commands in a word, the word will only be hyphenated at those points and not at any of the hyphenation points that LaTeX might otherwise have chosen.

\cleardoublepage

The `\cleardoublepage` command ends the current page and causes all figures and tables that have so far appeared in the input to be printed. In a two-sided printing

style, it also makes the next page a right-hand (odd-numbered) page, producing a blank page if necessary.

\clearpage

The `\clearpage` command ends the current page and causes all figures and tables that have so far appeared in the input to be printed.

\enlargethispage

`\enlargethispage{size}`

`\enlargethispage*{size}`

Enlarge the `\textheight` for the current page by the specified amount; e.g. `\enlargethispage{\baselineskip}` will allow one additional line.

The starred form tries to squeeze the material together on the page as much as possible. This is normally used together with an explicit `\pagebreak`.

\samepage

`\samepage`

The object in the argument `{}` should be on the current page.

\fussy

`\fussy`

This declaration (which is the default) makes TeX more fussy about line breaking. This can avoids too much space between words, but may produce overfull boxes.

This command cancels the effect of a previous `\sloppy` command. \sloppy

\hyphenation

`\hyphenation{words}`

The `\hyphenation` command declares allowed hyphenation points, where `words` is a list of words, separated by spaces, in which each hyphenation point is indicated by a `-` character.

\linebreak

`\linebreak[number]`

The `\linebreak` command tells LaTeX to break the current line at the point of the command. With the optional argument, `number`, you can convert the `\linebreak` command from a demand to a request. The number must be a number from 0 to 4. The higher the number, the more insistent the request is.

The `\linebreak` command causes LaTeX to stretch the line so it extends to the right margin.

\newline

The `\newline` command breaks the line right where it is. It can only be used in paragraph mode.

\newpage

The `\newpage` command ends the current page. In contrast to \pagebreak it can produce a partly empty page, even when \flushbottom is active.

\nolinebreak

`\nolinebreak[number]`

The `\nolinebreak` command prevents LaTeX from breaking the current line at the point of the command. With the optional argument, `number`, you can convert the `\nolinebreak` command from a demand to a request. The number must be a number from 0 to 4. The higher the number, the more insistent the request is.

\nopagebreak

`\nopagebreak[number]`

The `\nopagebreak` command prevents LaTeX from breaking the current page at the point of the command. With the optional argument, `number`, you can convert the `\nopagebreak` command from a demand to a request. The number must be a number from 0 to 4. The higher the number, the more insistent the request is.

\pagebreak

`\pagebreak[number]`

The `\pagebreak` command tells LaTeX to break the current page at the point of the command. With the optional argument, `number`, you can convert the `\pagebreak` command from a demand to a request. The number must be a number from 0 to 4. The higher the number, the more insistent the request is.

\sloppy

`\sloppy`

This declaration makes TeX less fussy about line breaking. This can prevent overfull boxes, but may leave too much space between words.

Lasts until a `\fussy` command is issued. \fussy.

1.2.2.11 Making Paragraphs

A paragraph is ended by one or more completely blank lines — lines not containing even a `%`. A blank line should not appear where a new paragraph cannot be started, such as in math mode or in the argument of a sectioning command.

\indent

`\indent`

This produces a horizontal space whose width equals the width of the paragraph indentation. It is used to add paragraph indentation where it would otherwise be suppressed.

\noindent

`\noindent`
When used at the beginning of the paragraph, it suppresses the paragraph indentation. It has no effect when used in the middle of a paragraph.

\par
Equivalent to a blank line; often used to make command or environment definitions easier to read.

1.2.2.12 Margin Notes

The command `\marginpar[left]{right}` creates a note in the margin. The first line will be at the same height as the line in the text where the `\marginpar` occurs.

When you only specify the mandatory argument `right`, the text will be placed

- in the right margin for one-sided layout
- in the outside margin for two-sided layout
- in the nearest margin for two-column layout.

By issuing the command `\reversemarginpar`, you can force the marginal notes to go into the opposite (inside) margin.

When you specify both arguments, `left` is used for the left margin, and `right` is used for the right margin.

The first word will normally not be hyphenated; you can enable hyphenation by prefixing the first word with a `\hspace{0pt}` command.

1.2.2.13 Math Formulae

The `math` environment can be used in both paragraph and LR mode, but the `displaymath` and `equation` environments can be used only in paragraph mode. The `math` and `displaymath` environments are used so often that they have the following short forms:

```
\(...\)     instead of     \begin{math}...\end{math}
```

math	For Formulae that appear right in the text.
displaymath	For Formulae that appear on their own line.
equation	The same as the displaymath environment except that it adds an equation number in the right margin.

Table 7 Three environments that put LaTeX in math mode

```
\[...\]     instead of     \begin{displaymath}...\end{displaymath}
```

In fact, the `math` environment is so common that it has an even shorter form:

```
$ ... $     instead of     \(...\)
```

Subscripts & Superscripts

To get an expression *exp* to appear as a subscript, you just type `_{`*exp*`}` . To get *exp* to appear as a superscript, you type `^{`*exp* `}`. LaTeX handles superscripted superscripts and all of that stuff in the natural way. It even does the right thing when something has both a subscript and a superscript.

Math Symbols

Spacing in Math Mode

In a `math` environment, LaTeX ignores the spaces you type and puts in the spacing that it thinks is best. LaTeX formats mathematics the way it's done in mathematics texts. If you want different spacing, LaTeX provides the following four commands for use in math mode:

1. `\;` - a thick space
2. `\:` - a medium space
3. `\,` - a thin space
4. `\!` - a negative thin space

Math Miscellany

1.2.2.14 Modes

When LaTeX is processing your input text, it is always in one of three modes:

- paragraph mode
- Math mode
- Left-to-right mode, called LR mode for short

LaTeX changes mode only when it goes up or down a staircase to a different level, though not all level changes produce mode changes. Mode changes occur only when entering or leaving an environment, or when LaTeX is processing the argument of certain text-producing commands.

Paragraph mode is the most common; it's the one LaTeX is in when processing ordinary text. In that mode, LaTeX breaks your text into lines and breaks the lines into pages. LaTeX is in *math mode* when it's generating a mathematical formula.

\cdots	Produces a horizontal ellipsis where the dots are raised to the centre of the line.
\ddots	Produces a diagonal ellipsis.
\frac{num}{den}	Produces the fraction `num` divided by `den`.
\ldots	Produces an ellipsis. This command works in any mode, not just math mode.
\overbrace{text}	Generates a brace over text.
\overline{text}	Causes the argument text to be overlined.
\sqrt[root]{arg}	Produces the square root of its argument. The optional argument, `root`, determines what root to produce, i.e., the cube root of `x+y` would be typed as `$\sqrt[3]{x+y}$`.
\underbrace{text}	Generates text with a brace underneath.
\underline{text}	Causes the argument text to be underlined. This command can also be used in paragraph and LR modes.
\vdots	Produces a vertical ellipsis.
\ensuremath{}	It's argument is set in math mode. This is needed for own definitions.

Table 8 Math Miscellany

In *LR mode*, as in paragraph mode, LaTeX considers the output that it produces to be a string of words with spaces between them. However, unlike paragraph mode, LaTeX keeps going from left to right; it never starts a new line in LR mode. Even if you put a hundred words into an `\mbox`, LaTeX would keep typesetting them from left to right inside a single box, and then complain because the resulting box was too wide to fit on the line.

LaTeX is in LR mode when it starts making a box with an `\mbox` command. You can get it to enter a different mode inside the box - for example, you can make it enter math mode to put a formula in the box. There are also several text-producing commands and environments for making a box that put LaTeX in paragraph mode. The box make by one of these commands or environments will be called a `parbox`. When LaTeX is in paragraph mode while making a box, it is said to be in *inner paragraph mode*. Its normal paragraph mode, which it starts out in, is called *outer paragraph mode*.

1.2.2.15 Page Styles

The `\documentclass` command determines the size and position of the page's head and foot. The page style determines what goes in them.

\maketitle

`\maketitle`

The `\maketitle` command generates a title on a separate title page - except in the `article` class, where the title normally goes at the top of the first page. Information used to produce the title is obtained from the following declarations:

See Page Styles for the commands to give the information.

\author

`\author{names}`

The `\author` command declares the author(s), where `names` is a list of authors separated by `\and` commands. Use `\\` to separate lines within a single author's entry – for example, to give the author's institution or address.

\date

`\date{text}`

The `\date` command declares *text* to be the document's date. With no `\date` command, the current date is used.

\thanks

`\thanks{text}`

The `\thanks` command produces a `\footnote` to the title.

\title

`\title{text}`

The `\title` command declares `text` to be the title. Use `\\` to tell LaTeX where to start a new line in a long title.

\pagenumbering

`\pagenumbering{num_style}`

Specifies the style of page numbers. Possible values of `num_style` are:

- `arabic` - Arabic numerals
- `roman` - Lowercase Roman numerals
- `Roman` - Uppercase Roman numerals
- `alph` - Lowercase letters
- `Alph` - Uppercase letters

\pagestyle

`\pagestyle{option}`

The `\pagestyle` command changes the style from the current page on throughout the remainder of your document.

The valid options are:

- `plain` - Just a plain page number.
- `empty` - Produces empty heads and feet - no page numbers.
- `headings` - Puts running headings on each page. The document style specifies what goes in the headings.
- `myheadings` - You specify what is to go in the heading with the `\markboth` or the `\markright` commands.

\markboth

```
\markboth{left head}{right head}
```

The `\markboth` command is used in conjunction with the page style `myheadings` for setting both the left and the right heading. You should note that a *left-hand heading* is generated by the last `\markboth` command before the end of the page, while a *right-hand heading* is generated by the first `\markboth` or `\markright` that comes on the page if there is one, otherwise by the last one before the page.

\markright

```
\markright{right head}
```

The `\markright` command is used in conjunction with the page style `myheadings` for setting the right heading, leaving the left heading unchanged. You should note that a *left-hand heading* is generated by the last `\markboth` command before the end of the page, while a *right-hand heading* is generated by the first `\markboth` or `\markright` that comes on the page if there is one, otherwise by the last one before the page.

\thispagestyle

`\thispagestyle{option}`

The `\thispagestyle` command works in the same manner as the `\pagestyle` command except that it changes the style for the current page only.

1.2.2.16 Sectioning

Sectioning commands provide the means to structure your text into units.

- `\part`
- `\chapter` (report and book class only)
- `\section`
- `\subsection`
- `\subsubsection`
- `\paragraph`
- `\subparagraph`

All sectioning commands take the same general form, i.e.,

`\chapter[optional]{title}`

In addition to providing the heading in the text, the mandatory argument of the sectioning command can appear in two other places:

1. The table of contents

2. The running head at the top of the page

You may not want the same thing to appear in these other two places as appears in the text heading. To handle this situation, the sectioning commands have an `optional` argument that provides the text for these other two purposes.

All sectioning commands have `*`-forms that print a *title*, but do not include a number and do not make an entry in the table of contents.

`\appendix`

The `\appendix` command changes the way sectional units are numbered. The `\appendix` command generates no text and does not affect the numbering of parts. The normal use of this command is something like

```
\chapter{The First Chapter}
 ...
\appendix
\chapter{The First Appendix}
```

1.2.2.17 Spaces & Boxes

All the predefined length parameters See Predefined lengths can be used in the arguments of the box-making commands.

\dotfill

The `\dotfill` command produces a *rubber length* that produces dots instead of just spaces.

\hfill

The `\hfill` fill command produces a *rubber length* which can stretch or shrink horizontally. It will be filled with spaces.

\hrulefill

The `\hrulefill` fill command produces a *rubber length* which can stretch or shrink horizontally. It will be filled with a horizontal rule.

\hspace

`\hspace{length}, \hspace*{length}`

The `\hspace` command adds horizontal space. The length of the space can be expressed in any terms that LaTeX understands, i.e., points, inches, etc. You can add negative as well as positive space with an `\hspace` command. Adding negative space is like backspacing.

LaTeX removes horizontal space that comes at the end of a line. If you don't want LaTeX to remove this space, include the optional `*` argument. Then the space is never removed.

\addvspace

`\addvspace{length}`

The `\addvspace` command normally adds a vertical space of height length. However, if vertical space has already been added to the same point in the output by a previous `\addvspace` command, then this command will not add more space than needed to make the natural length of the total vertical space equal to `length`.

\bigskip

The `\bigskip` command is equivalent to `\vspace{bigskipamount}` where `bigskipamount` is determined by the document class.

\medskip

The `\medskip` command is equivalent to `\vspace{medskipamount}` where `medskipamount` is determined by the document class.

\smallskip

`\smallskip`

The `\smallskip` command is equivalent to `\vspace{smallskipamount}` where `smallskipamount` is determined by the document class.

\vfill

The `\vfill` fill command produces a rubber length which can stretch or shrink vertically.

\vspace

`\vspace{length}, \vspace*{length}`

The `\vspace` command adds vertical space. The length of the space can be expressed in any terms that LaTeX understands, i.e., points, inches, etc. You can add negative as well as positive space with an `\vspace` command.

LaTeX removes vertical space that comes at the end of a page. If you don't want LaTeX to remove this space, include the optional `*` argument. Then the space is never removed.

\fbox

`\fbox{text}`

The `\fbox` command is exactly the same as the `\mbox` command, except that it puts a frame around the outside of the box that it creates.

\framebox

`\framebox[width][position]{text}`

The `\framebox` command is exactly the same as the `\makebox` command, except that it puts a frame around the outside of the box that it creates.

The framebox command produces a rule of thickness `\fboxrule`, and leaves a space `\fboxsep` between the rule and the contents of the box.

lrbox

`\begin{lrbox}{cmd} text \end{lrbox}`

This is the environment form of `\sbox`.

The text inside the environment is saved in the box `cmd`, which must have been declared with `\newsavebox`.

\makebox

`\makebox[width][position]{text}`

The `\makebox` command creates a box just wide enough to contain the `text` specified. The width of the box is specified by the optional `width` argument. The position of the text within the box is determined by the optional `position` argument.

- `c` — centred (default)
- `l` — flushleft
- `r` — flushright
- `s` — stretch from left to right margin. The text must contain stretchable space for this to work.

See \makebox (picture).

\mbox

`\mbox{text}`

The `\mbox` command creates a box just wide enough to hold the text created by its argument.

Use this command to prevent text from being split across lines.

\newsavebox

`\newsavebox{cmd}`

Declares `cmd`, which must be a command name that is not already defined, to be a bin for saving boxes.

\parbox

`\parbox[position][height][inner-pos]{width}{text}`

A `parbox` is a box whose contents are created in `paragraph` mode. The `\parbox` has two mandatory arguments:

- `width` - specifies the width of the parbox, and
- `text` - the text that goes inside the parbox.

LaTeX will position a `parbox` so its centre lines up with the centre of the text line. The optional *position* argument allows you to line up either the top or bottom line in the parbox (default is top).

If the *height* argument is not given, the box will have the natural height of the text.

The *inner-pos* argument controls the placement of the text inside the box. If it is not specified, *position* is used.

- `t` — text is placed at the top of the box.
- `c` — text is centred in the box.
- `b` — text is placed at the bottom of the box.
- `s` — stretch vertically. The text must contain vertically stretchable space for this to work.

A `\parbox` command is used for a parbox containing a small piece of text, with nothing fancy inside. In particular, you shouldn't use any of the paragraph-making environments inside a `\parbox` argument. For larger pieces of text, including ones containing a paragraph-making environment, you should use a `minipage` environment See minipage.

\raisebox

`\raisebox{distance}[extend-above][extend-below]{text}`

The `\raisebox` command is used to raise or lower text. The first mandatory argument specifies how high the text is to be raised (or lowered if it is a negative amount). The text itself is processed in `LR mode`.

Sometimes it's useful to make LaTeX think something has a different size than it really does - or a different size than LaTeX would normally think it has. The `\raisebox` command lets you tell LaTeX how tall it is.

The first optional argument, `extend-above`, makes LaTeX think that the text extends above the line by the amount specified. The second optional argument, `extend-below`, makes LaTeX think that the text extends below the line by the amount specified.

\rule

`\rule[raise-height]{width}{thickness}`

The `\rule` command is used to produce horizontal lines. The arguments are defined as follows:

- `raise-height` - specifies how high to raise the rule (optional)
- `width` - specifies the length of the rule (mandatory)
- `thickness` - specifies the thickness of the rule (mandatory)

\savebox

`\savebox{cmd}[width][pos]{text}`

This command typeset `text` in a box just as for `\makebox`. However, instead of printing the resulting box, it saves it in bin `cmd`, which must have been declared with `\newsavebox`.

\sbox

`\sbox{text}`

This commands typeset `text` in a box just as for `\mbox`. However, instead of printing the resulting box, it saves it in bin `cmd`, which must have been declared with `\newsavebox`.

\usebox

`\usebox{cmd}`

Prints the box most recently saved in bin `cmd` by a `\savebox` command.

1.2.2.18 Special Characters

The following characters play a special role in LaTeX and are called *special printing characters*, or simply *special characters*.

```
# $ % & ~ _ ^ \ { }
```

Whenever you put one of these special characters into your file, you are doing something special. If you simply want the character to be printed just as any other letter, include a \ in front of the character. For example, `\$` will produce $ in your output.

One exception to this rule is the \ itself because `\\` has its own special meaning. A \ is produced by typing `$\backslash$` in your file.

Also, `\~` means 'place a tilde accent over the following letter', so you will probably want to use `\verb` instead.

In addition, you can access any character of a font once you know its number by using the `\symbol` command. For example, the character used for displaying spaces in the `\verb*` command has the code decimal 32, so it can be typed as `\symbol{32}`.

You can also specify octal numbers with `'` or hexadecimal numbers with `"`, so the previous example could also be written as `\symbol{'40}` or `\symbol{"20}`.

1.2.2.19 Special Symbols

Accents on Characters

The rules differ somewhat depending whether you are in text mode, math modes, or the tabbing environment.

Text Mode

The following accents may be placed on letters. Although *o* is used in most of the example, the accents may be placed on any letter. Accents may even be placed above a *missing* letter; for example, `\~{}` produces a tilde over a blank space.

The following commands may be used only in paragraph or LR mode:

- `\‘{o}` produces a grave accent, ograve
- `\’{o}` produces an acute accent, oacute
- `\^{o}` produces a circumflex, ocirc
- `\"{o}` produces an umlaut or dieresis, *ö*
- `\H{o}` produces a long Hungarian umlaut
- `\~{o}` produces a tilde, otilde
- `\c{c}` produces a cedilla, ccedil
- `\={o}` produces a macron accent (a bar over the letter)
- `\b{o}` produces a bar under the letter
- `\.{o}` produces a dot over the letter
- `\d{o}` produces a dot under the letter
- `\u{o}` produces a breve over the letter
- `\v{o}` produces a *v* over the letter
- `\t{oo}` produces a *tie* (inverted u) over the two letters

Note that the letters *i* and *j* require special treatment when they are given accents because it is often desirable to replace the dot with the accent. For this purpose, the commands `\i` and `\j` can be used to produce dotless letters.

For example,

- `\^{\i}` should be used for i, circumflex, icirc
- `\"{\i}` should be used for i, umlaut, iuml

Math Mode

Several of the above and some similar accents can also be produced in math mode. The following commands may be used *only* in math mode:

- `\hat{o}` is similar to the circumflex (cf. `\^`)
- `\widehat{oo}` is a wide version of `\hat` over several letters
- `\check{o}` is a vee or check (cf. `\v`)
- `\tilde{o}` is a tilde (cf. `\~`)
- `\widetilde{oo}` is a wide version of `\tilde` over several letters
- `\acute{o}` is an acute accent (cf. `\‘`)
- `\grave{o}` is a grave accent (cf. `>\’`)
- `\dot{o}` is a dot over the letter (cf. `\.`)
- `\ddot{o}` is a double dot over the letter

- `\breve{o}` is a breve (cf. `\u`)
- `\bar{o}` is a macron (cf. `\=`)
- `\vec{o}` is a vector (arrow) over the letter

Tabbing Environment

Some of the accent marks used in running text have other uses in the Tabbing Environment. In that case they can be created with the following command:

- `>\a'` for an acute accent
- `\a`` for a grave accent
- `\a=` for a macron accent

Arrows

LaTeX provides commands for all sorts of arrows. The following commands are used only in math mode.

In general the command names are created from `left`, `right`, `up`, or `down`. `left-right` gives a double headed arrow. Prefacing with `long` gives a longer arrow. Capitalizing the first letter gives a double-shanked arrow.

Examples are:

- `\leftarrow`
- `\Leftarrow`
- `\longleftarrow`
- `\Longleftarrow`
- `\rightarrow`
- `\Rightarrow`
- `\longrightarrow`
- `\Longrightarrow`
- `\leftrightarrow`
- `\Leftrightarrow`
- `\longleftrightarrow`
- `\uparrow`
- `\downarrow`
- `\Uparrow`
- `>\nearrow` points from southwest to northeast
- `\searrow` points from northwest to southeast
- `\swarrow` points from northeast to southwest
- `\nwarrow` points from southeast to northwest

The `\stackrel` command is useful for putting things over or under your arrow.

See also:

- Math Symbols
- Binary and Relational Operators
- Greek Letters
- Miscellaneous Symbols
- Variable Size Math Symbols

Binary and Relational Operators

Some *math* symbols are obtained by typing the corresponding keyboard character. Examples include

```
+ - = < >
```

> plus, minus, and equal sign may be used in either text or math mode, but < and > are math mode only (they produce inverted exclamation and question marks, respectively, in text mode).

The following commands may be used only in math mode:

- `\pm` plus or minus sign
- `\mp` minus or plus sign
- `\times` times (an "x")
- `\div` divided by sign
- `\ast` an asterisk (centered)
- `\star` a five-point star (centered)
- `\bullet` a bullet
- `\circ` an open bullet
- `\cdot` a centered dot
- `\leq` less than or equal to
- `\ll` much less than
- `\subset` is a subset of
- `\geq` greater than or equal to
- `\gg` much greater than
- `\equiv` equivalence symbol
- `\sim` similar to
- `\simeq` similar or equal to
- `\approx` approximately equal to
- `\neq` not equal to
- `\per` "perpendicular to" symbols
- `\propto` proportional to

Note that *greater than* and *less than* are obtained simply by entering `>` and `<`.

A slash, indicating *not* can be placed through a symbol (or a letter) with the `\not` command. For example, not less than is `\not<` and not less than or equal to is `\not\leq`. If the slash isn't properly positioned it can be moved by putting a math mode spacing command between the `\not`and the symbol.

Delimiters

Delimiters are objects which act logically like parentheses. These can be used only in math mode.

The delimiters recognized by LaTeX include:

- `(` (left parenthesis)
- `)` (right parenthesis)
- `[` (left bracket)
- `]` (right bracket)
- `\{` (left brace)
- `\}` (right brace)
- `|` (vertical line)
- `\vert` (vertical line)
- `\|` (double vertical lines)
- `\Vert` (double vertical lines)
- `/` (slash)
- `\backslash` (backslash)
- `\langle` (left angle bracket)
- `\rangle` (right angle bracket)
- `\uparrow` (uparrow)
- `\downarrow` (down arrow)
- `\updownarrow` (up/down arrow)

Making Delimiters the Right Size

Delimiters in formulas should be big enough to *fit* around the formulas they delimit (for example arround arrays). To obtain *stretchable* delimiters (LaTeX makes them the appropriate size) type a `\left` or `\right` command before the delimiter. `\left` and `\right` commands must come in matching pairs, although the delimiters themselves need not be the same. Thus, `\left \{ ... \right \[` produces a legal pair. In cases where only one delimiter is desired, it is possible to make the matching delimiter *invisible* by typing a period (`.`) after the command, i.e., `\left.` or `\right.`

In an eqnarray environment the matching `\left` and `\right` cannot be split between lines and it may be necessary to use an *invisible* `\right.` and `\left.` to terminate and begin parts on different lines. In this case a second problem may arise, since the size of the delimiters will be chosen only for the *local part*, so that the size of the *visible left"* and *right* delimiters might not match. The solution is to trick LaTeX

into thinking that both parts have the same vertical height. This can be done by placing a strut, that is a zero-width `\rule`. It can also be accomplished with the `\vphantom` command, which I have not found documented, but which appears to work.

`\vphantom{construct}` creates a zero-width object with the height of `construct`. The argument can contain things such as `\frac` or the *variable size math symbols* and should be chosen according to what is in the section with the delimiter you want to match.

Some Examples

A six-j symbol

```
\[ \left\{
\begin{array}{ccc}
a & b & c \\
d & e & f \end{array}
\right>\} \]
```

This should be displayed something like (insofar as it can be rendered in "ascii art"):

```
( a b c )
-       -
( d e f )
```

Note that the `\[ ... \]` set this off as *Display Math*, and that the *Array Environment* is used to generate the three centered columns inside the braces.

A "multiple choice" equation

```
\[ f(x) =
   \left\{ \begin{array}{l}
       0, x < 0 \\ 1, x = 0 \\
       2, x > 0 \end{array} \right. \]
```

will be displayed as

```
        ( 0, x < 0
f(x) =  - 1, x = 0
        ( 2, x > 0
```

Note that the *invisible* `\right` delimiter is specified using a *period*.

Ellipsis

Ellipses (three dots) can be produced by the following commands

- `\ldots` horizontally at bottom of line
- `\cdots` horizontally center of line (math mode only)
- `\ddots` diagonal (math mode only)
- `\vdots` vertical (math mode only)

Greek Letters

These commands may be used only in math mode:

- `\alpha`
- `\beta`
- `\gamma`
- `\delta`
- `\epsilon`
- `\varepsilon` (variation, script-like)
- `\zeta`
- `\eta`
- `\theta`
- `\vartheta` (variation, script-like)
- `\iota`
- `\kappa`
- `\lambda`
- `\mu`
- `\nu`
- `\xi`
- `\pi`
- `\varpi` (variation)
- `\rho`
- `\varrho` (variation, with the tail)
- `\sigma`
- `\varsigma` (variation, script-like)
- `\tau`
- `\upsilon`
- `\phi`
- `\varphi` (variation, script-like)
- `\chi`
- `\psi`
- `\omega`

Capital letters:

- `\Gamma`
- `\Delta`

- `\Theta`
- `\Lambda`
- `\Xi`
- `\Pi`
- `\Sigma`
- `\Upsilon`
- `\Phi`
- `\Psi`
- `\Omega`

See also:

- Math Symbols
- Accents
- Miscellaneous Symbols

Miscellaneous Symbols

Some symbols for math

The following symbols are also used only in math mode

- `\aleph` Hebrew aleph
- `\hbar` h-bar, Planck's constant
- `\imath` variation on *i*; no dot
- `\jmath` variation on *j*; no dot
- `\ell` script (loop) *l*
- `\wp` fancy script lowercase *P*
- `\Re` script capital *R* (*Real*)
- `\Im` script capital *I* (*Imaginary*)
- `\prime` prime (also obtained by typing ')
- `\nabla` inverted capital Delta
- `\surd` radical (square root) symbol
- `\angle` angle symbol
- `\forall` *for all* (inverted A)
- `\exists` *exists* (left-facing E)
- `\partial` partial derivative symbol
- `\infty` infinity symbol
- `\triangle` open triangle symbol
- `\Box` open square
- `\Diamond` open diamond
- `\flat` music: flat symbol
- `\natural` music: natural symbol
- `\clubsuit` playing cards: club suit symbol

- `\diamondsuit` playing cards: diamond suit symbol
- `\heartsuit` playing cards: heart suit symbol
- `\spadesuit` playing cards: space suit symbol

Some Other Symbols

The following symbols can be used in any mode:

- `\dag` dagger
- `\ddag` double dagger
- `\S` section symbol
- `\P` paragraph symbol
- `\copyright` copyright symbol
- `\pounds` British pound sterling symbol

Calligraphic Style Letters

Twenty-six calligraphic letters are provided (the upper case alphabet). These can only be used in math mode.

In LaTeX 2.09 they are produced with the `\cal` declaration:

```
${\cal A}$
```

In LaTeX2e they are obtained with the `\mathcal` command:

```
$\mathcal{CAL}$
```

Math Functions

Functions like *log* or *cos* are normally used in math mode. However, inmath mode strings of letters are treated as a product of variables, which would normally be displayed in math italics rather than a text font which would be appropriate for these functions. To get proper display of such functions they are generated with LaTeX commands.

Some of these commands are:

- `\arccos`
- `\arcsin`
- `\arctan`
- `\cos`
- `\cosh`
- `\cot`
- `\coth`
- `\csc`
- `\deg`
- `\det`

- `\dim`
- `\exp`
- `\gcd`
- `\hom`
- `\inf`
- `\ker`
- `\lg`
- `\lim`
- `>\liminf`
- `\limsup`
- `\ln`
- `\log`
- `\max`
- `\min`
- `\sec`
- `\sin`
- `\sinh`
- `\sup`
- `\tan`
- `\tanh`

Two commands are provided for the "modulus" function

- `\bmod`
- `\pmod{}`

The former would be used to write "a mod b" as `a \bmod b` and the latter, which requires an argument that is displayed in parentheses, would be used to write "n = i (mod j)" as `n = i \pmod{j}`

Variable Size Math Symbols

The size of some mathematical symbols, notably summation signs, product signs, and integral signs, depends on the environment in which they appear (i.e., `displaymath` as opposed to `math` environments.

These include:

- `\sum` a summation sign (capital sigma)
- `\prod` a product (capital pi)
- `\coprod` a coproduct (inverted capital pi)
- `\int` an integral sign
- `\oint` a surface (circular) integral sign
- `\bigcup` big "U"
- `\bigcap` big inverted "U"

- `\bigvee` big "V"
- `\bigwedge` big inverted "V"
- `\bigodot` big "O" with dot at center
- `\bigotimes` big "O" with cross inside
- `\bigoplus` big "O" with a + inside
- `\biguplus` big "U" with a + inside

The `\sqrt` command also produces a variable size symbol appropriate for the size of hte radicand argument.

The "limits" associated with these symbols are entered as *subscripts* for entries appearing below the symbol and as *superscripts* for entries appearing above the symbol. For example the sum from n=0 to infinity of x n would be entered as

```
\sum_{n=0}^{\infty} x_{n}
```

The actual placement of the limits depends on whether this is in `displaymath` mode in which case they are placed below/above or in *math* mode in running text in which case they are placed as regular subscripts and superscripts.

Note that it is possible to treat several of these symbols (a common example would be a double sum) as a single symbol for placing limits above and/or below by using the `\mathop` command.

Hats and *tildes* over symbols which stretch (as best they can) to the correct size for their arguments are produced by `\widehat` and `\widetilde`

1.2.2.20 Splitting the Input

A large document requires a lot of input. Rather than putting the whole input in a single large file, it's more efficient to split it into several smaller ones. Regardless of how many separate files you use, there is one that is the root file; it is the one whose name you type when you run LaTeX.

\include

`\include{file}`

The `\include` command is used in conjunction with the `\includeonly` command for selective inclusion of files. The `file` argument is the first name of a file, denoting `file.tex`. If `file` is one the file names in the file list of the `\includeonly` command or if there is no `\includeonly` command, the `\include` command is equivalent to

```
\clearpage \input{file} \clearpage
```

except that if the file `file.tex` does not exist, then a warning message rather than an error is produced. If the file is not in the file list, the `\include` command is equivalent to `\clearpage`.

The `\include` command may not appear in the preamble or in a file read by another `\include` command.

\includeonly

`\includeonly{`*file_list*`}`

The `\includeonly` command controls which files will be read in by an `\include` command. *file_list* should be a comma-separated list of filenames. Each filename must match exactly a filename specified in a `\include` command. This command can only appear in the preamble.

\input

`\input{file}`

The `\input` command causes the indicated `file` to be read and processed, exactly as if its contents had been inserted in the current file at that point. The file name may be a complete file name with extension or just a first name, in which case the file `file.tex` is used.

1.2.2.21 Starting & Ending

Your input file must contain the following commands as a minimum:

```
\documentclass{class}
\begin{document}
  ... your text goes here ...
\end{document}
```

where the `class` selected is one of the valid classes for LaTeX. See Document Classes (and see Overview), for details of the various document classes available locally.

You may include other LaTeX commands between the `\documentclass` and the `\begin{document}` commands (i.e., in the 'preamble').

1.2.2.22 Table of Contents

A table of contents is produced with the `\tableofcontents` command. You put the command right where you want the table of contents to go; LaTeX does the rest for you. It produces a heading, but it does not automatically start a new page. If you want a new page after the table of contents, include a `\newpage` command after the `\tableofcontents` command.

There are similar commands `\listoffigures` and `\listoftables` for producing a list of figures and a list of tables, respectively. Everything works exactly the same as for the table of contents.

NOTE: If you want any of these items to be generated, you cannot have the `\nofiles` command in your document.

\addcontentsline

`\addcontentsline{file}{sec_unit}{entry}`

The `\addcontentsline` command adds an entry to the specified list or table where:

- `file` is the extension of the file on which information is to be written: `toc` (table of contents), `lof` (list of figures), or `lot` (list of tables).
- `sec_unit` controls the formatting of the entry. It should be one of the following, depending upon the value of the file argument:
  1. `toc` — the name of the sectional unit, such as part or subsection.
  2. `lof` — figure
  3. `lot` — table
- `entry` is the text of the entry.

\addtocontents

`\addtocontents{file}{text}`

The `\addtocontents` command adds text (or formatting commands) directly to the file that generates the table of contents or list of figures or tables.

- `file` is the extension of the file on which information is to be written: `toc` (table of contents), `lof` (list of figures), or `lot` (list of tables).
- `text` is the information to be written.

1.2.2.23 Terminal Input/Output

\typein

`\typein[cmd]{msg}`

Prints `msg` on the terminal and causes LaTeX to stop and wait for you to type a line of input, ending with return. If the `cmd` argument is missing, the typed input is processed as if it had been included in the input file in place of the `\typein` command. If the `cmd` argument is present, it must be a command name. This command name is then defined or redefined to be the typed input.

\typeout

`\typeout{msg}`

Prints `msg` on the terminal and in the `log` file. Commands in `msg` that are defined with `\newcommand` or `\renewcommand` are replaced by their definitions before being printed.

LaTeX's usual rules for treating multiple spaces as a single space and ignoring spaces after a command name apply to `msg`. A `\space` command in `msg` causes a single space to be printed. A `^^J` in `msg` prints a newline.

1.2.2.24 Typefaces

The `typeface` is specified by giving the *size* and *style*. A typeface is also called a *font*.

\Styles

The following type style commands are supported by LaTeX.

These commands are used like `\textit{italics text}`. The corresponding command in parenthesis is the *declaration form*, which takes no arguments. The scope of the declaration form lasts until the next type style command or the end of the current group.

The declaration forms are cumulative; i.e., you can say `\sffamily\bfseries` to get sans serif boldface.

You can also use the environment form of the declaration forms; e.g. `\begin{ttfamily}...\end{ttfami`

In addition, the command `\mathversion{bold}` can be used for switching to bold letters and symbols in formulas. `\mathversion{normal}` restores the default.

Sizes

The following standard type size commands are supported by LaTeX.

The commands as listed here are "declaration forms". The scope of the declaration form lasts until the next type style command or the end of the current group.

You can also use the environment form of these commands; e.g. `\begin{tiny}...\end{tiny}`.

\tiny
\scriptsize
\footnotesize
\small
\normalsize (default)
\large
\Large
\LARGE
\huge
\Huge

Table 9 Standard type size commands

\textrm (\rmfamily)	Roman.
\textit (\itshape) , \emph	Emphasis (toggles between \textit and \textrm).
\textmd (\mdseries)	Medium weight (default). The opposite of boldface.
\textbf (\bfseries)	Boldface.
\textup (\upshape)	Upright (default). The opposite of slanted.
\textsl (\slshape)	Slanted.
\textsf (\sffamily)	Sans serif.
\textsc (\scshape)	Small caps.
\texttt (\ttfamily)	Typewriter.
\textnormal (\normalfont)	Main document font.
\mathrm	Roman, for use in math mode.
\mathbf	Boldface, for use in math mode. For bold symbols or complete equations take a look at `\boldsymbol` and `\bm`.
\mathsf	Sans serif, for use in math mode.
\mathtt	Typewriter, for use in math mode.
\mathit	Italics, for use in math mode, e.g. variable names with several letters.
\mathnormal	For use in math mode, e.g. inside another type style declaration.
\mathcal	'Calligraphic' letters, for use in math mode.

Table 10 Typefaces

Low-level font commands

These commands are primarily intended for writers of macros and packages. The commands listed here are only a subset of the available ones. For full details, you should consult Chapter 7 of The LaTeX Companion.

1.2.3 Parameters

The input file specification indicates the file to be formatted; TeX uses `.tex` as a default file extension. If you omit the input file entirely, TeX accepts input from the terminal. You specify command options by supplying a string as a parameter to the command; e.g.

```
latex \scrollmode\input foo.tex
```

will process `foo.tex` without pausing after every error.

\fontencoding{enc}	Select font encoding. Valid encodings include `OT1` and `T1`.
\fontfamily{family}	Select font family. Valid families include: • `cmr` for Computer Modern Roman • `cmss` for Computer Modern Sans Serif • `cmtt` for Computer Modern Typewriter and numerous others.
\fontseries{series}	Select font series. Valid series include: • `m` Medium (normal) • `b` Bold • `c` Condensed • `bc` Bold condensed • `bx` Bold extended and various other combinations.
\fontshape{shape}	Select font shape. Valid shapes are: • `n` Upright (normal) • `it` Italic • `sl` Slanted (oblique) • `sc` Small caps • `ui` Upright italics • `ol` Outline The two last shapes are not available for most font families.
\fontsize{size}{skip}	Set font size. The first parameter is the font size to switch to; the second is the `\baselineskip` to use. The unit of both parameters defaults to pt. A rule of thumb is that the baselineskip should be 1.2 times the font size.
\selectfont	The changes made by calling the four font commands described above do not come into effect until `\selectfont` is called.
\usefont{enc}{family}{series}{shape}	Equivalent to calling `\fontencoding`, `\fontfamily`, `\fontseries` and `\fontshape` with the given parameters, followed by `\selectfont`.

Table 11 Low-level font commands

Output files are always created in the current directory. When you fail to specify an input file name, TeX bases the output names on the file specification associated with the logical name TEX_OUTPUT, typically `texput.log`.

1.3 LaTeX Packages

1.3.1 `amsmath`

`amsmath` consists of several packages to aid in typesetting math.

`\boldsymbol` Produces a bold math symbol. When not only a single symbol, but a complete equation has to be bold you might consider using the superior `\bm` command which takes better care of spacing.

`\pmb` Can be used for mathematic symbols, when no bold version is available. It prints the standard symbol several times slightly shifted and is inferior to a real bold font!

1.3.2 `fontenc`

`\usepackage[T1]{fontenc}`
wählt T1-Fonts und erlaubt damit korrekte automatische Trennung in Wörtern mit Umlauten.

1.3.3 `german`

Das `german` Paket muß geladen sein:

`"a` erzeugt ein *ä*

entsprechend `"o`, `"u`, `"A`, `"O`, `"U`, `"s (ß)`

`"ck` wird bei Trennung zu `k-k`

`"ff` wird bei Trennung zu `ff-f`

entsprechend `"ll`, `"mm`, `"pp`, `"rr`, `"tt`

`\glqq` oder `"‘` erzeugt untere doppelte Anführungszeichen

`\grqq` oder `"’` erzeugt obere doppelte Anführungszeichen

`\glq` erzeugt untere einfache Anführungszeichen

`\flqq` oder `"<` doppelte linke französische Anführungszeichen

`\frqq` oder `">` doppelte rechte französische Anführungszeichen

`\flq` einfache linke französische Anführungszeichen

`\frq` einfache rechte französische Anführungszeichen
`"|` verhindert eine Ligatur an der Stelle
`\-` markiert mögliche Trennstelle, im restlichen Wort ist keine Trennung mehr möglich
`"-` Trennvorschlag, der Rest des Wortes kann noch automatisch getrennt werden
`""` Trennvorschlag, bei dem bei einer Trennung kein Bindestrich eingefügt wird
`"~` Bindestrich, an dem nicht getrennt werden darf
`"=` Bindestrich, an dem getrennt werden darf
`\dq` erzeugt ein nicht-typographisches gerades Anführungszeichen

```
\texttt{\dq}
```

`<` oder `\dq` ergibt das Zoll-Zeichen

1.3.4 `graphics`

Look up details in `grfguide`.

1.3.4.1 \includegraphics

`\includegraphics[]{picture-name}`
File extensions of the graphics should be omitted, so it's easier to change the format (for example for pdf or ps output).

1.3.4.2 \graphicspath

`\graphicspath{dir-list}`
`dir-list` is a list of absolute or relative paths, where the command `\include-graphics{}` searches for the graphic files:
`\graphicspath{{eps/}{tiff/}}` would cause the system to look in the subdirectories `eps` and `tiff` of the current directory. This is unix syntax, on a Mac it would be:
`\graphicspath{{:eps:}{:tiff:}}`
Under Windows/DOS use also the slash `/` to separate directories. Not all tex installations can handle filenames with spaces. So it's better to avoid them.

1.3.5 `hyperref`

For more infos read in the book *LaTeX Web Companion* or *Mit LaTeX ins Web*.

1.3.5.1 Options

4 use Acrobat 4 features (default: `false`).

a4paper, a5paper, b5paper, legalpaper, letterpaper, executivepaper use the specified paper size.

anchorcolor set color of anchors (default: `black`).

backref do bibliographical back references (default: `false`).

baseurl set base URL for document (default: empty).

bookmarks make bookmarks (default: `true`).

bookmarksnumbered put section numbers in bookmarks (default: `false`).

bookmarksopen open up bookmark tree (default: `false`).

bookmarksopenlevel level to which bookmarks are open (default: `\maxdimen`).

bookmarkstype to specify which 'toc' file to mimic toc.

breaklinks allow links to break over lines (default: `false`).

citebordercolor color of border around cites (default: `0 1 0`).

citecolor color of citation links (default: `green`).

colorlinks color links (default: `false`).

debug provide details of anchors defined (same as verbose, default: `false`).

draft do not do any hyper linking (default: `false`).

dvipdf, dvipdfm, dvips, dvipsone, dviwindo, hypertex, latex2html, pdftex, tex4ht, textures, ps2pdf, vtex use the specified backend.

extension suffix of linked files (default: `dvi`).

filebordercolor color of border around file links (default: `0 .5 .5`).

filecolor color of file links (default: `cyan`).

frenchlinks use small caps instead of color for links (default: `false`).

hyperfigures make figures hyper links (default: `false`).

hyperindex set up hyperlinked indices (default: `true`).

hypertexnames use guessable names for links (default: `true`).

implicit redefine LaTeX internals (default: `true`).

linkbordercolor color of border around links (default: `1 0 0`).

linkcolor color of links (default: `red`).

linktocpage make page number, not text, be link on TOC, LOF and LOT (default: `false`).

menubordercolor color of border around menu links (default: `1 0 0`).

menucolor color for menu links (default: `red`).
naturalnames use LaTeX-computed names for links (default: `false`).
nesting allow nesting of links (default: `false`).
pageanchor put an anchor on every page (default: `true`).
pagebackref backreference by page number (default: `false`).
pagebordercolor color of border around page links (default: `1 1 0`).
pagecolor color of page links (default: `red`).
pdfauthor text for PDF Author field (default: empty).
pdfborder width of PDF link border (default: `0 0 1`; `0 0 0` for `colorlinks`)
pdfcenterwindow position the document window center of the screen (default: `false`).
pdfcreator text for PDF Creator field (default: LaTeX with hyperref package).
pdffitwindow resize document window to fit document size (default: `false`).
pdfhighlight set highlighting of PDF links (default: `/I`).
pdfkeywords text for PDF Keywords field (default: empty).
pdfmenubar make PDF viewer's menu bar visible (default: `true`).
pdfnewwindow make links that open another PDF file start a new window (default: `false`).
pdfpagelayout set layout of PDF pages (default: empty).
pdfpagemode set default mode of PDF display (default: empty).
pdfpagescrop set crop size of PDF document (default: empty).
pdfpagetransition set PDF page transition style (default: empty).
pdfproducer text for PDF Producer field (default: empty).
pdfstartpage page at which PDF document opens (default: `1`).
pdfstartview starting view of PDF document (default: `/Fit`).
pdfsubject text for PDF Subject field (default: empty).
pdftitle text for PDF Title field(default: empty).
pdftoolbar make PDF toolbar visible (default: `true`).
pdfview PDF 'view' when on link traversal (default: empty).
pdfwindowui make PDF user interface elements visible (default: `true`).
plainpages do page number anchors as plain arabic (default: `true`).
raiselinks raise up links (for HyperTeX backend; default: `false`).
runbordercolor color of border around run links (default: `0 .7 .7`).
unicode Unicode encoded pdf strings (default: `true`).
urlbordercolor color of border around URL links (default: `0 1 1`).

urlcolor color of URL links (default: `magenta`).
verbose be chatty (default: `false`).

1.3.5.2 Commands

`\texorpdfstring{TeX-string}{pdf-string}` allows different texts for PDF or TeX creation.

```
\section{The role of \texorpdfstring{H$_2$O}{water}}
```

`\href{URL}{Text}` the text will be resolved into a URL. The URL must be complete, according to a base-url which might be defined in a `\hyperbaseurl`.

```
\href{http://www.karpfenteich.net/pit/}{Peter's input tips}
```

`\hyperbaseurl{URL}` defines a base URL, which is preset to a given URL. This helps to make portable documents.

`\hyperimage{image-URL}` includes the picture of the URL reference.

`\hyperref{URL}{category}{name}{text}`, `\hyperref[mark]{text}` `text` is activated as a hyperlink to the point which is defined with a `\label` command with the symbolic name `mark`.

`\hyperlink{name}{text}`, `\hyertarget{name}{text}` `\hyperlink` creates an internal link to an object which is somewhere defined with a `\hypertarget` command.

For HTML the command `\hyperlink` inserts a `#` character in front of each link. Thus it'll refer to the topical document, while `\href` will expect a complete URL.

`\autoref{marker}` `\autoref` is a substitution for the standard `\ref` command. It inserts a context sensitive phrase.

```
see \autoref{foo}
```

for a `\label` in a section will output *see section 3* for an example.

There are macros to change the default output with the help of the `\renewcommand`:

```
\figurename             *\figurename*
\tablename              *\tablename*
\partname               *\partname*
\appendixname           *\appendixname*
\equationname           *\equationname*
\Itemname               *\Itemname*
\chaptername            *\chaptername*
\sectionname            *\sectionname*
```

```
\subsectionname          *\subsesctionname*
\subsubsectionname       *\subsubsectionname*
\paragraphname           *\paragraphname*
\Hfootnotename           *\Hfootnotename*
\AMSname                 *\AMSname*
\theoremname             *\theoremname*
```

`\nolinkurl` Allows line breaks in a verbatim like environment like `\url`, but without the hyperlink function. This is useful to define own commands like this example:

```
\newcommand*{\urlw}[1]{\href{http://www.#1}%
            {\nolinkurl{www.#1}}}
```

This new defined command `\urlw` allows the user to typeset a linkable (this function is offered by `\href`) `www.example.org` address in the latex source like this:

```
\urlw{example.org}
```

This will produce the string `www.example.org` which is hyperlinked and breakable (when the driver like pdftex allows line breaks in links).

1.3.6 inputenc

`\usepackage[encoding name]{inputenc}`

Allows to specify an input encoding for direct input of character codes > 127, e.g. accented characters.

Important encodings are:

`latin9, latin1` Linux, Unix, VMS

`ansinew` Windows (codepage 1252)

`cp850` OS/2, MSDOS (codpage 850)

`applemac` Apple Macintosh

1.3.7 KOMA-Script

The `KOMA-Script` package has a very detailed and well written documentation. Read the `scrguide` for further information! :-)

1.3.7.1 Page-Layout

`\typearea[BCOR]{DIV}`
`\areaset[BCOR]{Breite}{Höhe}`

1.3.7.2 Options

headinclude, headexclude, footinclude, footexclude will determine if headers and footers are used in the calculation of the page size. Default is exclude. Include will make the text area smaller.

DIV is a factor which determins how large the used page size is. This method takes care of good typography.

Try values between 10 and 15. Higher values will cause smaller margins. An example would be: `DIV12`.

BCOR Binding correction. Takes care of the loss of the visible paper size when the paper is cut and a book is bound.

1.3.7.3 Letter Class `scrlttr2`

This reference is based on the excellent `scrguien.pdf` from 2002-09-06.

Options

Defining Options Later

`\KOMAoptions{options list}` Use this to change options after loading the class. List single or multiple options, separated by commas.

Page Layout Options

`paper=format` Defines the paper format.

`BCOR=length, DIV=value, headlines=count` Divisor, binding correction, and number of headlines.

`enlargefirstpage` If *true* some more text would fit on the first page

Other Layout Options

`twoside` Activate possibilities of a double-sided document, but stay with with the one-sided layout as far as possible.

`cleardoublepage=style` Inserted pages by `\cleardoublepage` get one of these page styles:

- `empty`
- `plain`
- `standard`

`headsepline`, `footsepline` Insert a separator line below the head or above the foot.

`mpinclude`, `mpexclude` These two options of the `typearea` package should not be used with `scrlttr2`.

`pagenumber=position` This option defines if and where a page number will be placed. Possible values are:

`bot`, `foot` Page number in foot, horizontal position not changed

`botcenter`, `botcentered`, `botmiddle`, `footcenter`, `footcentered`, `footmiddle` Page number infoot,centered

`botleft`, `footleft` Page number in foot, left justified

`botrigth`, `footright` Page number in foot, right justified

`center`, `centered`, `middle` Page number centered horizontally, vertical position not changed

`false`, `no`, `off` No page number

`head`, `top` Page number in head, horizontal position not changed

`headcenter`, `headcentered`, `headmiddle`, `topcenter`, `topcentered`, `topmiddle` Page number in head, centered

`headleft`, `topleft` Page number in head, left justified

`headright`, `topright` Page number in head, right justified

`left` Page number left, vertical position not changed

`right` Page number right, vertical position not changed

Default is `botcenter`

`parskip=value` Mark paragraphs alternatively by vertical skip. Possible values are:

`false`, `off` Paragraph indentation instead of verticalk space

`full`, `on`, `true` One line vertical space between paragraphs; at least 1 em free space in the last line.

`full*` One line vertical space betwen paragraphs; at least a quarter of a line free space at the end of a paragraph.

`full+` One line vertical space betwen paragraphs; at least a third of aline free space at the end of a paragraph.

`full-` One line vertical space betwen paragraphs; last line of a paragraph may be arbitrarily filled.

`half` Half a line vertical space between paragraphs; at least 1 em free space in the last line.

`half*` Half a line vertical space between paragraphs; at least a quarter of a line free space at the end of a paragraph.

`half+` Half a line vertical space between paragraphs; at least a third of a line free space at the end of a paragraph.

`half-` One line vertical space between paragraphs; last line may be arbitrarily filled.

Default is `false`.

Font Options

`fontsize=size` Font size of the document. Defaults to `12pt`.

Options for Letterhead and Address

`fromalign` Defines the placement of the from address in the letterhead. Possible values are:

`center`, `centered`, `middle` Return address centered.

`false`, `no`, `off` Standard design for return address.

`left` Left justified return address.

`rigth` Right justified return address.

Default is `left`

`fromrule` Allows to place a horizontal rule within return address. Possible values are:

`afteraddress`, `below`, `true`, `yes` Line below return address

`aftername` Line right below sender's name

`false`, `no`, `off` No line.

Default is `false`.

`fromphone` Defines whether the phone number will be part of the return address.

`fromfax` Defines whether the facsimile number will be part of the return address.

`fromemail` Defines whether the email address will be part of the return address.

`fromurl` Defines whether the URL will be part of the return address.

`fromlogo` Defines whether the logo will be part of the return address.

`addrfield` Defines whether a return address field will be set.

`backaddress` Defines whether a return address for window envelopes will be set.

`subject` Choose of your subject should have a title and if the subject should be set before or after the opening. Possible values 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` Toggle between presets for the field width of additionbal sender attributes.

`narrow` Small sender supplement field.

`wide` Wide sender supplement field.

Default is `narrow`.

`foldmarks` Activates fold marks.

`numericaldate` Toggles between standard, language-dependent date presentation and a short, numerical one.

`refline` Defines that the reference line may extend beyond the normal type area.

Format Options

`draft` Toggles between the final and the draft version.

Letter Class Option Files

`\LoadLetterOption{name}` Load a `lco` file. `name` is the filename without suffix. Predefined `lco` files:

`DIN` For A4 size paper, complying with DIN 676.

`DINmtext` For A4 size paper, complying with DIN 676, but using an alternate layout with more text on the first page.

`KOMAold` Fo AQ4 paper size using a layout close to the obsolete `scrlettr` class.

`SN` For Swiss letters with address field on the right side, according to SN 010 130

`SNleft` For Swiss letters with address field on the left side.

`\LetterOptionNeedsPaperSize{option name}{paper size}` Useful to be abel to emit warnings if the user selects a different paper size from the one the `lco` file is based on.

General Document Properties

Font Selection

See also the section for *Changing Fonts* in KOMA-Script (still to be written).

`backaddress` Back address for a window envelope

`descriptionlabel` Label in a `description` environment

`fromaddress` Sender's address in letterhead

`fromname` Sender's address in letterhead if different from `fromaddress`

`pagefoot` Footer, sometimes header of a page

`pagehead` Header, sometimes footer of a page

`pagenumber` Page number in footer or header

`subject` Subject in the opening

`title` Headline in the opening

Page Style

See also the section for *Page Style* in KOMA-Script (still to be written).

`empty` Entirely empty headers and footers

`plain` Empty head and and only page number in header or footer

`headings` Automatic page headings

`myheadings` Manual page headings

Variables

The main difference between a *command* and a *variable* is that a command usually triggers an action whereas a variable only consists of plain text. Furthermore a variable can have an additional description.

`\newkomavar[description]{name}`, `\newkomavar*[description]{name}`, `\addtoreffields{name}` `\newkomavar` defines a new variable, addressed with `name`. The command `\addtoreffields` adds the variable `name` to the reference fields. See section Business Line. The command `\newkomavar*` works like `\newkomavar` with an additional call of the command `\addtoreffields`.

Overview of all variables:

`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 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` 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

`\setkomavar{name}[description]{content}`, `\setkomavar*{name}{description}` `\setkomavar` sets the content of the variable `name`. The optional argument sets the `description` of the variable. `\setkomavar*` only sets the `description`.

`\usekomavar[command]{name}`, `\usekomavar*[command]{name}` `\usekomavar` gives you access to the content of the variable `name`. `\usekomavar*` gives you access to the `description` of the variable `name`.

`\ifkomavarempty{name}{true}{false}`, `\ifkomavarempty*{name}{true}{false}` The `true` argument will be executed if the content or description is empty. Otherwise the `false` argument will be executed. The starred version handles the description of a variable, the variant without star the content.

Pseudo Lengths

`\@newplength{name}` Defines a new pseudo length

`\useplength{name}` Access value of pseudo length `name`

`\setlengthtoplength[factor]{length}{pseudo length}`, `\addtolengthplength[factor]{length}{pseudo length}` Assign a multiple of `pseudo length` to a real `length`. Adds a multiple of `pseudo length` to `length`.

`\@setplength[factor]{pseudo length}{value}`, `\@addtoplength[factor]{pseudo length}{value}` Assign a multiple of a `value` to a `pseudo length`. Adds `value` to `pseudo length`.

Letter Declaration

Letterhead

`firstheadvpos` Distance between the top of the sheet and the letterhead.

`firstheadwidth` Width of the letterhead.

`fromname`, `fromaddress`, `fromphone`, `fromfax`, `fromemail`, `fromurl`, `fromlogo` This variables give all statements concerning the sender necessary to create the letterhead.

`phoneseparator`, `faxseparator`, `emailseparator`, `urlseparator` Separators for variables in letterhead.

`\firsthead{construction}` Create letterhead freely.

Footer

`firstfootvpos` Distance between footer and upper border of the sheet.

`firstfootwidth` Width of letter's firstpage footer.

`\firstfoot{Construction}` Set definition for first page's footer

Address

`toaddrvpos`, `toaddrhpos` Distance between address window of a window envelope and the upper and left border of the sheet.

`toaddrwidth` Width of the address window.

`toaddrindent` Value of indentation for the address from the left border.

`backaddress`, `backaddresssseparator`, `backaddrheight` Double backslashes within return address will be exchanged with `backaddresssseparator`. Height of the return address field.

`specialmail`, `specialmailindent`, `specialmailrightindent` Optionally set between return address and addressee. `specialmailindent` and `specialmail-rightindent` determine left and right indentation.

`toname`, `toaddress` Usually not set directly by the user.

`\begin{letter}[options]{address}` Major point of the letter class.

`\AtBeginLetter{command}` Hook for commands run at begin of a letter.

Sender's Extension

`locwidth` Declares width of sender's extensions.

`location` Content of sender's extension.

Business Line

`refvpos` Distance between upper border of sheet and business line.

`refwidth` Available width for the business line.

`refaftervskip` Vertical space to be inserted beneath business line.

`place`, `placeseparator` If all variables for business line are empty, the content of `place` and `placeseparator` will be set, followed by content of `date`.

`yourref`, `yourmail`, `myref`, `customer`, `invoice`, `date` Typical fields for business lines.

Title and Subject Line

`title` Additional tile. Set centered beneath the business line.

`subject`, `subjectseparator` Set the subject of the letter, optional prefixed by a label.

Further Issues

`tfoldmarkvpos`, `bfoldmarkvpos` Position of upper and lower fold mark.

`foldmarkhpos` Distance between fold marks and the sheet's left border.

`frombank` Internally not used yet.

`\nexthead{construction}`, `\nextfoot{construction}` Create letter head or foot for the following pages freely.

The Text

Opening

`\opening{opening}` Set the opening of the letter.

Footnotes

See section *Footnotes* in KOMA-Script (still to be written).

Lists

See section *Lists* in KOMA-Script (still to be written).

Margin Notes

See section *Margin Notes* in KOMA-Script (still to be written).

Text Emphasis

See section *Logical Markup of Text* in KOMA-Script (still to be written).

Closing Part

Closing

`signature` Inlcudes an explanation for the inscription. Defaults to `\usekomavar{fromname}`.

`\closing{closing phrase}` Typesets the colsing phrase follwoed by a vertical space and the contents fo variable `signature`.

`sigindent`, `sigbeforevskip`, `\raggedsignature` Indentation of the box for closing phrase, inscription and signature (defaults to 0mm). Vertical space between closing phrase and signature (defaults to two lines). Alignment inside the box.

Postscript, Carbon Copy, and Enclosures

`\ps` Set a postscript.

`\cc{distribution list}`, `ccseparator` Set a ddistribution list. If the contents of `ccseparator` isn't empty then the name and the content of the variable is inserted prior to distribution list.

`\encl{enclosures}`, `enclseparator` Same structur as the distribution list.

Language Support

Language Selection

`\captionenglish`, `\captionUSenglish`, `\captionamerican`, `\captionbritish`, `\captionUKenglish`, `\captiongerman`, `\captionngerman`, `\captionaustrian`, `\captionfrench`, `\captionitalian`, `\captionspanish`, `\captiondutch`, `\captioncroatian` If the used language selection scheme does not support switching the language the above commands can be used directly.

`\dateenglish`, `\dateUSenglish`, `\dateamerican`, `\datebritish`, `\dateUKenglish`, `\dategerman`, `\datengerman`, `\dateaustrian`, `\datefrench`, `\dateitalian`, `\datespanish`, `\datedutch`, `\datecrotian` Numerical date representation.

Language-Dependent Terms

`\yourrefname`, `\youremailname`, `\myrefname`, `\customername`, `\invoicename`, `\subjectname`, `\ccname`, `\enclname`, `\headtoname`, `\headfromname`, `\datename`, `\pagename`, `\phonename`, `\faxname`, `\emailname`, `\wwwname`, `\bankname` These defintions can be modified to support a new language of for private customization.

Defining Language Terms

`\providecaptioname{language}{term}{definition}`, `\newcaptioname{language}{term}{definti`
`\renewcaptionname{language}{term}{definition}` Assign a *definition* for a *language* to a *term*.

Address Files and Circular Letters

`\adrentry{Lastname}{Firstname}{Address}{Telephone}{F1}{F2}{Comment}{Key}`
Address entry for address files. The file extension has to be `.adr`!

`\addrentry{Lastname}{Firstname}{Address}{Telephone}{F1}{F2}{F3}{F4}{Key}`
This command supports four freely definable parameters for an address entry.

`\adrchar{initial letter}`, `\addrchar{initial letter}` Separator for address entries. Ignored by `scrlttr2`.

1.3.7.4 Address Files with `scraddr`

`\InputAddressFile{file name}` Read the contents of the given address file. Filename extension has to be `.adr` and has to be omitted in the argument.

`\Name{Key}`, `\FirstName{Key}`, `\LastName{Key}`, `\Address{Key}`, `\Telephone{Key}`, `\FreeI{Key}`, `\FreeII{Key}`, `\Comment{Key}`, `\FreeIII{Key}`, `\FreeIV{Key}`
Access to the data in address file. `\Name` is a concatenation of `\FirstName` and `\LastName`.

`AdrFreeIVempty`, `AdrFreeIVshow`, `AdrFreeIVwarn`, `AdrFreeIVstop` Package warning options. Defaults to `AdrFreeIVshow`.

1.3.8 `mathpazo`

Math fonts for the use with the Palatino font.

1.3.8.1 \upDelta / \upOmega

Math fonts for the use with the Palatino font.

Package options are:

[**slantedGreek**] Uppercase Greek will be typeset slanted.

[**noBBpl**] Do not use PazoMathBlackboardBold as (partial) blackboard bold font (e.g with `\mathbb{R}`).

[**osf**] Make the font family `pplj` (Palatino with old style numerals) the default roman font, and use an alternative version of the virtual math italic fonts (`zplmrj7m` and `zplmbj7m`) with upright Palatino old style numerals for use with the `\oldstylenums` command.

[**osfeqnnum**] Use old style numerals for equation numbering.

1.3.9 `varioref`

Smarter version of the original latex2e cross reference commands. Generated strings are customizable, Babel options are recognized (further info in the package documentation).

`\vref, \vpageref` `\vref` is similar to `\ref` but adds an additional page reference, like *on the facing page* or *on page 27* whenever the corresponding `\label` is not on the same page.

`\vpageref` is a variation of `\pageref` with similar functionality.

`\vrefrange, \vpagerefrange` The `\v...range` commands take two labels as arguments and produce strings which depend on wether or not these labels fall onto a single page or on different pages.

`\vref*, \vpageref*, \vpagerefrange*` Star * variants do not add any space before the generated text for situations like:

```
(\vref{foo} ...)
```

1.3.10 Several Small Packages

Must be loaded with the usual `\usepackage` command. Further info for a package to be found in the documentation (dvi, pdf, ps) or in the `sty`-file itself.

1.3.10.1 `bm`

Bold math symbols or equations with better spacing then the `\boldsymbold` command. If possible load after other packages which redefine the fonts. When no bold font is available for a certain char `bm` will use *poor man's bold* `\pmb`.

`\bm` Produces bold math symbol or equation.

`\unboldmath` Can be used to set parts of an equation unbold.

1.3.10.2 `url`

Defines line breakable hyperlinked (uses hyperref package) verbatim input for urls and e-mail addresses.

Example: `\url{http://example.org}` outputs `http://example.org`.

Related commands are `\href` and `\nolinkurl`.

1.4 Generic Packages

1.4.1 PiCTeX (tbd.)

tbd.

1.4.2 PSTricks

1.4.2.1 Overview

The idea behind the PSTricks package of Timothy van Zandt is to provide most of the PostScript language in LaTeX and TeX syntax.

1.4.2.2 Commands and Parameters

Color

```
\newgray{color}{num} , \newrgbcolor{color}{num1 num2 num3} , \newhsb-
color{color}{num1 num2 num3} , \newcmykcolor{color}{num1 num2 num3 num4}
```

Note that these commands are obsolete for LaTeX. Load PSTricks with `\usepackage{pstcol}` and use the standard LaTeX color commands instead.

Setting graphics parameters

```
\psset{par1=value1,par2=value2,...}
```

Dimensions, coordinates and angles

```
\pssetlength{cmd}{dim}
unit=dim, xunit=dim, yunit=dim, yunit=dim                Default: 1cm
```

`\degrees[num]`
`\radians`

Basic graphics parameters

`linewidth=dim` Default: `.8pt`
`linecolor=color` Default: `black`
`showpoints=true/false` Default: `false`

Lines and polygons

`linearc=dim` Default: `0pt`
`framearc=num` Default: `0`
`cornersize=relative/absolute` Default: `relative`
`\psline*[par]{arrows}(x0,y0)(x1,y1)...(xn,yn)`
`\qline(coor0)(coor1)`
`\pspolygon*[par](x0,y0)(x1,y1)(x2,y2)...(xn,yn)`
`\psframe*[par](x0,y0)(x1,y1)`

Arcs, circles and ellipses

`\pscircle*[par](x0,y0){radius}`
`\qdisk(coor){radius}`
`\pswedge*[par](x0,y0){radius}{angle1}{angle2}`
`\psellipse*[par](x0,y0)(x1,y1)`
`\psarc*[par]{arrows}(x,y){radius}{angleA}{angleB}`
`\psarcn*[par]{arrows}(x,y){radius}{angleA}{angleB}`
`arcsep=dim, arcsepA=dim, arcsepB=dim` Default: `0pt`

Curves

`\psbezier*[par]{arrows}(x0,y0)(x1,y1)(x2,y2)(x3,y3)`
`\parabola*[par]{arrows}(x0,y0)(x1,y1)`
`\pscurve*[par]{arrows}(x1,y1)...(xn,yn)`
`\psecurve*[par]{arrows}(x1,y1)...(xn,yn)`
`\psccurve*[par]{arrows}(x1,y1)...(xn,yn)`
`curvature=num1 num2 num3` Default: `1 .1 0`

Dots

`\psdots*[par](x1,y1)(x2,y2)...(xn,yn)`
`dotstyle=style` Default: `*` Dots: `*, o, +, triangle, triangle*, square, square*, pentagon, pentagon*, |`
`dotscale=num1 num2` Default: `1`
`dotangle=angle` Default: `0`

Grids

`\psgrid(x0,y0)(x1,y1)(x2,y2)`

`gridwidth=dim` Default: `.8pt`

`gridcolor=color` Default: `black`

`griddots=num` Default: `0`

`gridlabels=dim` Default: `10pt`

`gridlabelcolor=color` Default: `black`

`subgriddiv=int` Default: `5`

`subgridwidth=dim` Default: `.4pt`

`subgridcolor=color` Default: `gray`

`subgriddots=num` Default: `0`

Plots

`\fileplot*[par]{file}`

`\dataplot*[par]{commands}`

`\savedata{command}[data]`

`\readdata{command}{file}`

`\listplot*[par]{list}`

`\psplot*[par]{xmin}{xmax}{function}`

`\parametricplot*[par]{tmin}{tmax}{function}`

`plotstyle=style` Default: `line`

`plotpoints=int` Default: `50`

Coordinate systems

`origin={coor}` Default: `0pt,0pt`

`swapaxes=true` Default: `false`

Line styles

`linestyle=style` Default: `solid`

`dash=dim1 dim2` Default: `5pt 3pt`

`dotsep=dim` Default: `3pt`

`border=dim` Default: `0pt`

`bordercolor=color` Default: `white`

`doubleline=true/false` Default: `false`

`doublesep=dim` Default: `1.25\pslinewidth`

`doublecolor=color` Default: `white`

`shadow=true/false` Default: `false`

`shadowsize=dim` Default: `3pt`

`shadowangle=angle` Default: `-45`
`shadowcolor=color` Default: `darkgray`
`dimen=outer/inner/middle` Default: `outer`

Fill styles

`fillstyle=style` Default: `none`
`fillcolor=color` Default: `white`
`hatchwidth=dim` Default: `.8pt`
`hatchsep=dim` Default: `4pt`
`hatchcolor=color` Default: `black`
`hatchangle=rot` Default: `45`

Arrowheads and such

`arrows=style` Default: `-` Arrows: `-, <->, >-<, <<->>, >>-<<, |-|, |*-|*, [-], (-), o-o, *-*, oo-oo, **-**, c-c, cc-cc, C-C`
`arrowsize=dim num` Default: `2pt 3`
`arrowlength=num` Default: `1.4`
`arrowinset=num` Default: `.4`
`tbarsize=dim num` Default: `2pt 5`
`bracketlength=num, rbracketlength=num` Default: `.15`
`dotsize=dim num` Default: `.5pt 2.5`
`arrowscale=num1 num2` Default: `1`

Custom styles

```
\newpsobject{name}{object}{par1=value1,...}
\newpsstyle{name}{par1=value1,...}
```

The basics

```
\pscustom*[par]{commands}
```

Parameters

`linetype=int` Default: `0`

Graphics objects

`liftpen=0/1/2` Default: `0`

Safe tricks

```
\newpath
\moveto(coor)
\closepath
\stroke[par]
\fill[par]
```

```
\gsave
\grestore
\translate(coor)
\scale{num1 num2}
\rotate{angle}
\swapaxes
\msave
\mrestore
\openshadow[par]
\closedshadow[par]
\movepath(coor)
```

Pretty safe tricks

```
\lineto(coor)
\rlineto(coor)
\curveto(x1,y1)(x2,y2)(x3,y3)
\rcurveto(x1,y1)(x2,y2)(x3,y3)
```

For hackers only

```
\code{code}
\dim{dim}
\coor(x1,y1)(x2,y2)...(xn,yn)
\rcoor(x1,y1)(x2,y2)...(xn,yn)
\file{file}
\arrows{arrows}
\setcolor{color}
```

Pictures

```
\pspicture*[baseline](x0,y0)(x1,y1)
\endpspicture
```

Placing and rotating whatever

```
\rput*[refpoint]{rotation}(x,y){stuff}
\uput*{labelsep}[refangle]{rotation}(x,y){stuff}
\pslabelsep
```

`labelsep=dim` Default: `5pt`

Repetition

```
\multirput*[refpoint]{angle}(x0,y0)(x1,y1){int}{stuff}
\multips{angle}(x0,y0)(x1,y1){int}{graphics}
```

Horizontal	*Vertical*	*Dflt*	*Description*
Ox=num	Oy=num	0	Label at origin.
Dx=num	Dy=num	1	Label increment.
dx=dim	oy=dim	0pt	Dist btwn labels.

Table 12 Axes label parameters

Axes

`\psaxes*[par]{arrows}(x0,y0)(x1,y1)(x2,y2)`

`labels=all/x/y/none` Default: `all`

`showorigin=true/false` Default: `true`

`ticks=all/x/y/none` Default: `all`

`tickstyle=full/top/bottom` Default: `full`

`ticksize=dim` Default: `3pt`

`\psxlabel, \psylabel`

`axesstyle=axes/frame/none` Default: `axes`

Framed boxes

`framesep=dim` Default: `3pt`

`boxsep=true/false` Default: `true`

`\psframebox*[par]{stuff}`

`\psdblframebox*[par]{stuff}`

`\psshadowbox*[par]{stuff}`

`\pscirclebox*[par]{stuff}`

`\cput*[par]{angle}(x,y){stuff}`

`\psovalbox*[par]{stuff}`

Clipping

`\clipbox[dim]{stuff}`

`\psclip{graphics} ... \endpsclip`

Rotation and scaling boxes

`\rotateleft{stuff}`

`\rotateright{stuff}`

`\rotatedown{stuff}`

`\scalebox{num1 num2}{stuff}`

`\scaleboxto(x,y){stuff}`

Nodes

`\rnode[refpoint]{name}{stuff}`

```
\Rnode(x,y){name}{stuff}
\RnodeRef
\pnode(x,y){name}
\cnode*[par](x,y){radius}{name}
\circlenode*[par]{name}{stuff}
\cnodeput*[par]{angle}(x,y){name}{stuff}
\ovalnode*[par]{name}{stuff}
```

Node connections

```
nodesep=dim        Default: 0
offset=dim         Default: 0
arm=dim        Default: 10pt
angle=angle        Default: 0
arcangle=angle         Default: 8
ncurv=num        Default: .67
loopsize=dim         Default: 1cm
\ncline*[par]{arrows}{nodeA}{nodeB}
\ncLine*[par]{arrows}{nodeA}{nodeB}
\nccurve*[par]{arrows}{nodeA}{nodeB}
\ncarc*[par]{arrows}{nodeA}{nodeB}
\ncbar*[par]{arrows}{nodeA}{nodeB}
\ncdiag*[par]{arrows}{nodeA}{nodeB}
\ncdiagg*[par]{arrows}{nodeA}{nodeB}
\ncangle*[par]{arrows}{nodeA}{nodeB}
\ncangles*[par]{arrows}{nodeA}{nodeB}
\ncloop*[par]{arrows}{nodeA}{nodeB}
\nccircle*[par]{arrows}{node}{radius}
\pcline*[par]{arrows}(x1,y1)(x2,y2)
\pccurve*[par]{arrows}(x1,y1)(x2,y2)
\pcarc*[par]{arrows}(x1,y1)(x2,y2)
\pcbar*[par]{arrows}(x1,y1)(x2,y2)
\pcdiag*[par]{arrows}(x1,y1)(x2,y2)
\pcangle*[par]{arrows}(x1,y1)(x2,y2)
\pcloop*[par]{arrows}(x1,y1)(x2,y2)
```

Attaching labels to node connections

```
\lput*[refpoint]{rotation}(pos){stuff}
```

`\aput*[labelsep]{angle}(pos){stuff}`
`\bput*[labelsep]{angle}(pos){stuff}`
`\mput*[refpoint]{stuff}`
`\Aput*[labelsep]{stuff}`
`\Bput*[labelsep]{stuff}`

Coils and zigzags

`\pscoil*[par]{arrows}(x0,y0)(x1,y1)`
`\psCoil*[par]{angle1}{angle2}`
`\pszigzag*[par]{arrows}(x0,y0)(x1,y1)`
`coilwidth=dim` Default: `1cm`
`coilheight=num` Default: `1`
`coilarm=dim` Default: `.5cm`
`coilaspect=angle` Default: `45`
`coilinc=angle` Default: `10`
`\nccoil*[par]{arrows}{nodeA}{nodeB}`
`\nczigzag*[par]{arrows}{nodeA}{nodeB}`
`\pccoil*[par]{arrows}(x1,y1)(x2,y2)`
`\pczigzag*[par]{arrows}(x1,y1)(x2,y2)`

Special coordinates

`\SpecialCoor`
`\NormalCoor`

Coordinate	*Example*	*Description*
(x,y)	(3,4)	Cartesian coordinate.
(r;a)	(3;110)	Polar coordinate.
(node)	(A)	Center of node.
([par]node)	([angle=45]A)	Relative to node.
(!ps)	(!5 3.3 2 exp)	Raw PostScript.
(coor1|coor2)	(A|1in;30)	Combination.

Angle	*Example*	*Description*
num	45	Angle.
(coor)	(-1,1)	Coordinate (vector).
!ps	!33 sqrt	Raw PostScript.

Table 13 Special coordinates and angles

Overlays

`\overlaybox stuff\endoverlaybox`

`\psoverlay{string}`

`\putoverlaybox{string}`

`gradbegin=color` Default: `gradbegin`

`gradend=color` Default: `gradend`

`gradlines=int` Default: `500`

`gradmidpoint=num` Default: `.9`

`gradangle=angle` Default: `0`

Typesetting text along a path

`\pstextpath[pos](x,y){graphics object}{text}`

Stroking and filling character paths

`\pscharpath*[par]{text}`

`\pscharclip*[par]{text} ... \endpscharclip`

Exporting EPS files

`\TeXtoEPS ... \endTeXtoEPS`

`\PSTtoEPS[par]{file}{graphics objects}`

`bbllx=dim` Default: `-1pt`

`bblly=dim` Default: `-1pt`

`bburx=dim` Default: `1pt`

`bbury=dim` Default: `1pt`

`headerfile=file` Default: `s`

`headers=none/all/user` Default: `none`

Boxes

`\psmathboxtrue, \psmathboxfalse`

`\everypsbox{commands}`

`\pslongbox{name}{cmd}`

`\psverbboxtrue, \psverbboxfalse`

Tips and More Tricks

`\pslbrace, \psrbrace`

1.4.2.3 List of Additional PSTricks Packages

`psgo` Draw Go diagrams

`pst-blur` PSTricks package for "blurred" shadows

`pst-euc` Géométrie en LaTeX et PSTricks
`pst-fr3d` Three dimensional framed Boxes
`pst-ghsb` PSTricks package for HSB Gradients
`pst-gr3d` PSTricks package for 3D grids
`pst-lens` Optique géométrique
`pst-node` PSTricks package for nodes
`pst-ob3d` A PSTricks package for three dimensional basic objects
`pst-osci` Oscilloscopes with PSTricks
`pst-poly` Polygons with PSTricks
`pst-tree` PSTricks package for trees
`pst-uml` Draw easily diagrams with UML notation
`vaucansom.sty` Drawing automata

1.5 `fontinst` (tbd.)

tbd.

1.6 ConTeXt (tbd.)

1.6.1 Overview

ConTeXt is based on PlainTeX and MetaPost and gets controlled with some Perl scripts. It's mainly targeted towards layout oriented users and especially useful for presentations. It's extensible by the use of modules and has in it's base distribution already a large amount of functionality. The preferred output format is PDF, but DVI is possible too.

Some of the unique features of ConTeXt are:

- Inline XML (including MathML, ChemML, and PhysML)
- really good XML support in general
- inline MetaPost

1.7 Texinfo (tbd.)

tbd.

2 Metafont (tbd.)

tbd.

3 Bibtex

> The contents of this section was taken from the HTML helppages for Bibtex of Norman Walsh (Version 1.0, 12 Apr 94).

Invokes the BibTeX utility to compile a bibliography file for LaTeX. Full details can be found in "LaTeX: A Document Preparation System" by Leslie Lamport.

3.1 Parameters

`bibliography-file-spec`

Specifies the name of the bibliography database file to be compiled by BibTeX. If the file specification does not include a file type, BibTeX assumes a default type of BIB.

3.2 Command Qualifiers

`/BIBINPUTS /BIBINPUTS=(name,...)`

Specify directories containing input files, and the order in which they will be searched to locate each input file. A null value in the list indicates the current directory. The search procedure TeX uses to locate input files is to first search your default directory and then search each of the directories specified by the `/BIBINPUTS` option.

Default is `/BIBINPUTS=(TEX_BIB:)`; TeX looks in the directory associated with the logical name `TEX_BIB. /STATS /STATS /NOSTATS [D]`

This qualifier is used while debugging `.BST` files to determine BIBTEX memory usage. `/TEXINPUTS /TEXINPUTS=(name,...)`

Specify directories containing input files, and the order in which they will be searched to locate each input file. A null value in the list indicates the current directory. The search procedure TeX uses to locate input files is to first search your default directory and then search each of the directories specified by the `/TEXINPUTS` option.

Default is `/TEXINPUTS=(TEX_INPUTS)`; TeX looks in the directory associated with the logical name `TEX_INPUTS. /TRACE /TRACE /NOTRACE [D]`

This qualifier is used while debugging `.BST` files to follow program flow.

3.3 bib files

This help entry contains the same information as Appendix B of the LaTeX manual. It describes the format of a bibliography database (`.BIB`) file.

A bibliography database file may contains two types of entry - an abbreviation definition or a reference entry for citation.

3.3.1 `@STRING` command

The `@STRING` command is used to define abbreviations for use by BibTeX within the bibliography database file. The command

```
@string{jgg1 = "Journal of Gnats and Gnus, Series~1"}
```

defines 'jgg1' to be the abbreviation for the string "Journal of Gnats and Gnus, Series~1". Parentheses can be used in place of the outermost braces in the @string command, and braces can be used instead of the quotation marks. The text must have matching braces.

The case of letters is ignored in an abbreviation as well as in the command name @string, so the command above could have been written:

```
@STRING{JgG1 = "Journal of Gnats and Gnus, Series~1"}
```

A `@string` command can appear anywhere before or between entries in a bibliography database file. However, it must come before any use of the abbreviation, so a sensible place for `@string` commands is at the beginning of the file. A `@string` command in the bibliography database file takes precedence over a definition made by the bibliography style, so it can be used to change the definition of an abbreviation such as 'Feb'.

3.3.2 Entry Format

A bibliography database file contains a series of reference entries like the following:

```
@BOOK{kn:gnus,
      AUTHOR = "Donald E. Knudson",
      TITLE  = "1966 World Gnus Almanac",
      PUBLISHER = {Permafrost Press},
      ADDRESS = {Novisibirsk}             }
```

The `@BOOK` states that this is an entry of type book. various entry types are described below. The 'kn:gnus' is the citation key, as it appears in the argument of a `\cite` command referring to the entry.

This entry has four fields, named `AUTHOR`, `TITLE`, `PUBLISHER` and `ADDRESS`. The meanings of these and other fields are described below. A field consists of the name, an '=' character with optional space around it, followed by its text. The text of a field is a string of characters, with no unmatched braces, surrounded by either a pair of braces or a pair of '"' characters. Entry fields are separated from one another, and from the citation key, by commas. A comma may have optional space around it.

The outermost braces that surround the entire entry may be replaced by parentheses. As in TeX input files, an end-of-line character counts as a space and one space is equivalent to many spaces. Unlike TeX, BibTeX ignores the case of letters in the entry type, citation key and field names. The above entry could have been typed as follows:

```
@BOOK(kn:gnus, author = {Donald E. Knudson},
      TITLE  = "1966 World Gnus Almanac",
      PUBLISHER = {Permafrost Press},
      ADDRESS = {Novisibirsk}             )
```

However, the case of letters does matter to LaTeX, so the citation key ("kn:gnus" in the example above) should appear exactly the same in all `\cite` commands in the LaTeX input file.

The quotes or braces can be omitted around text consisting entirely of numerals. The following two fields are equivalent:

```
Volume = "27"           Volume = 27
```

3.3.3 Entry Types

When entering a reference in the bibliography database, the first thing to decide is what type of entry it is. No fixed classification scheme can be complete, but BibTeX provides enough entry types to handle almost any reference reasonably well.

References to different types of publications contain different information; a reference to a journal might include the volume and number of the journal, which is usually not meaningful for a book. Therefore, database entries of different types have different fields for each entry type, the fields are divided into three classes:

Required omitting the field will produce an error message and may result in a badly formatted bibliography entry. If the required information is not meaningful, you are using the wrong entry type.

Optional the field's information will be used if present, but can be omitted without causing any formatting problems. A reference should contain any available

information that might help the reader, so you should include the optional field if it is applicable.

Ignored the field is ignored. BibTeX ignores any field that is not required or optional, so you can include any fields you want in a bibliography entry. It's often a good idea to put all relevant information about a reference in its bibliography entry - even information that may never appear in the bibliography. For example, if you want to keep an abstract of a paper in a computer file, put it in an 'abstract' field in the paper's bibliography entry. The bibliography database file is likely to be as good a place as any for the abstract, and it is possible to design a bibliography style for printing selected abstracts.

BibTeX ignores the case of letters in the entry type.

3.3.3.1 article entry

An article from a journal or magazine.

Format:

```
@ARTICLE{citation_key,
         required_fields [, optional_fields] }
```

Required fields: author, title, journal, year

Optional fields: volume, number, pages, month, note, key

3.3.3.2 book entry

A book with an explicit publisher.

Format:

```
@BOOK{citation_key,
      required_fields [, optional_fields] }
```

Required fields: author or editor, title, publisher, year

Optional fields: volume, series, address, edition, month, note, key

3.3.3.3 booklet entry

A work that is printed and bound, but without a named publisher or sponsoring institution.

Format:

```
@BOOKLET{citation_key,
         required_fields [, optional_fields] }
```

Required fields: title

Optional fields: author, howpublished, address, month, year, note, key

3.3.3.4 conference entry

An article in the proceedings of a conference. This entry is identical to the 'inproceedings' entry and is included for compatibility with another text formatting system.

Format:

```
@CONFERENCE{citation_key,
            required_fields [, optional_fields] }
```

Required fields: author, title, booktitle, year

Optional fields: editor, pages, organization, publisher, address, month, note, key

3.3.3.5 inbook entry

A part of a book, which may be a chapter and/or a range of pages.

Format:

```
@INBOOK{citation_key,
        required_fields [, optional_fields] }
```

Required fields: author or editor, title, chapter and/or pages, publisher, year

Optional fields: volume, series, address, edition, month, note, key

3.3.3.6 incollection entry

A part of a book with its own title.

Format:

```
@INCOLLECTION{citation_key,
              required_fields [, optional_fields] }
```

Required fields: author, title, booktitle, year

Optional fields: editor, pages, organization, publisher, address, month, note, key

3.3.3.7 inproceedings entry

An article in the proceedings of a conference.
Format:

```
@INPROCEEDINGS{citation_key,
               required_fields [, optional_fields] }
```

Required fields: author, title, booktitle, year
Optional fields: editor, pages, organization, publisher, address, month, note, key

3.3.3.8 manual entry

Technical documentation.
Format:

```
@MANUAL{citation_key,
        required_fields [, optional_fields] }
```

Required fields: title
Optional fields: author, organization, address, edition, month, year, note, key

3.3.3.9 mastersthesis entry

A Master's thesis.
Format:

```
@MASTERSTHESIS{citation_key,
               required_fields [, optional_fields] }
```

Required fields: author, title, school, year
Optional fields: address, month, note, key

3.3.3.10 misc entry

Use this type when nothing else seems appropriate.
Format:

```
@MISC{citation_key,
      required_fields [, optional_fields] }
```

Required fields: none

Optional fields: author, title, howpublished, month, year, note, key

3.3.3.11 phdthesis entry

A PhD thesis.
Format:

```
@PHDTHESIS{citation_key,
           required_fields [, optional_fields] }
```

Required fields: author, title, school, year
Optional fields: address, month, note, key

3.3.3.12 proceedings entry

The proceedings of a conference.
Format:

```
@PROCEEDINGS{citation_key,
             required_fields [, optional_fields] }
```

Required fields: title, year
Optional fields: editor, publisher, organization, address, month, note, key

3.3.3.13 techreport entry

A report published by a school or other institution, usually numbered within a series.
Format:

```
@TECHREPORT{citation_key,
            required_fields [, optional_fields] }
```

Required fields: author, title, institution, year
Optional fields: type, number, address, month, note, key

3.3.3.14 unpublished entry

A document with an author and title, but not formally published.
Format:

```
@UNPUBLISHED{citation_key,
             required_fields [, optional_fields] }
```

Required fields: author, title, note

Optional fields: month, year, key

3.3.4 Field Text

The text of the field is enclosed in braces or double quote characters. A part of the text is said to be enclosed in braces if it lies inside a matching pair of braces other than the ones enclosing the entire entry or the entire field text.

BibTeX manipulates the case of letters in the field text as described in the subtopics below.

3.3.4.1 Names

The text of an author or editor field represents a list of names. The bibliography style determines the format in which the name is printed: whether the first name or last name appears first, if the full first name or just the first initial is used, etc. The bibliography file entry simply tells BibTeX what the name is.

You should type an author's complete name and let the bibliography style decide what to abbreviate. (But an author's complete name may be "Donald~E. Knuth" or even "J.~P.~Morgan"; you should type it the way the author would like it to appear, if that's known.)

Most names can be entered in the obvious way, either with or without a comma, as in the following examples.

"John Paul Jones" "Jones, John Paul" "Ludwig von Beethoven" "von Beethoven, Ludwig"

Some people have multiple last names - for example, Per Brinch Hansen's last name is Brinch~Hansen. His name should be typed with a comma:

"Brinch Hansen, Per"

To understand why, you must understand how BibTeX handles names (for what follows, a "name" corresponds to a person).

Each name consists of four parts: First, von, Last, and~Jr; each part consists of a (possibly empty) list of name-tokens. For example the First part of Per Brinch~Hansen's name has the single token "Per"; the Last part has two, "Brinch" and "Hansen"; and the von and Jr parts are empty. If you had typed

"Per Brinch Hansen"

instead, BibTeX would erroneously think "Brinch" were a First-part token, just as "Paul" is a First-part token in "John˜Paul Jones".

Here's another example:

"Charles Louis Xavier Joseph de la Vallee Poussin"

This name has four tokens in the First part, two in the von, and two in the Last. Here BibTeX knows where one part ends and the other begins because the tokens in the von part begin with lower-case letters.

If you want BibTeX to consider something a single token, enclose it in braces. You should do this, for example, if a comma is part of a name:

"{Barnes and Noble, Inc.}" "{Barnes and} {Noble, Inc.}" "{Barnes} {and} {Noble,} {Inc.}"

The braces surrounding the comma keep "Inc." from being interpreted as a First token; this name has only a Last part, with either one, two, or four tokens (there must be a space separating the tokens in the second and third forms). Probably the second form is slightly more meaningful, but don't lose sleep over this since only rarely will an institution be an author or editor.

So the two names

"von Beethoven, Ludwig" "{von Beethoven}, Ludwig"

are considered by BibTeX to be different names. In the first, "Beethoven" is the Last part and "von" is the von part; in the second, which in this case happens to be incorrect, the Last part has a single token and there's no von part. The bibliography style will probably print both the same, but it may alphabetize and label them differently.

"Juniors" pose a special problem. Most people having "Jr." in their name precede it with a comma. Such a name should be entered as follows:

"Ford, Jr., Henry"

However, a few people do not use a comma. They are handled by considering the "Jr." to be part of the last Last token:

"{Steele Jr.}, Guy L." "Guy L. {Steele Jr.}"

This name has no Jr part.

To summarize, you may type a name in one of three forms:

"First von Last" "von Last, First" "von Last, Jr, First"

You may almost always use the first form; you shouldn't if either there's a Jr part or the Last part has multiple tokens but there's no von part.

If there are multiple authors or editors, their names must be separated by the word "and", surrounded by spaces, not enclosed in braces:

AUTHOR = "Ralph Alpher and Bethe, Hans and George Gamow"

Since BibTeX interprets commas as separating parts of a name and "and" as separating names themselves, this example has three coauthors: Ralph Alpher, Hans Bethe, and George Gamow. If the word "and" appears as part of a name, it must be enclosed in braces, as in the example of "Barnes and Noble,˜Inc." given above. If you have too many names to list in a field, you can end the list with "and others"; the standard styles appropriately append an "et al."

BibTeX's rules are actually a bit more complicated than indicated here, but this description will suffice for most names.

3.3.4.2 Titles

The bibliography style determines whether or not a title is capitalized; the titles of books usually are, the title of articles usually are not. You type a title the way it should appear if it is capitalized (you should capitalize everything but articles and unstressed conjunctions and prepositions, and even these should be capitalized as the first word or the first after a colon):

TITLE = "The Agony and the Ecstasy"

BibTeX will change uppercase letters to lowercase if appropriate. Uppercase letters that should not be changed are enclosed in braces. The following two titles are equivalent; the "A" of "Africa" will not be made lowercase.

"The Gnats and Gnus of {Africa}" "The Gnats and Gnus of {A}frica"

3.3.4.3 Abbreviations

Instead of an ordinary text string, the text of a field can be replaced by an abbreviation for it. An abbreviation is a string of characters that starts with a letter and does not contain a space or any of the following ten characters:

```
" # % ' ( ) , = { }
```

The abbreviation is typed in place of the text field, with no braces or quotation marks. If 'jgg1' is an abbreviation for

Journal of Gnats and Gnus, Series˜1

then the following are equivalent:

JOURNAL = jgg1 JOURNAL = "Journal of Gnats and Gnus, Series˜1"

Some abbreviations are predefined by the bibliography style. These always include the usual 3 letter abbreviations for the month: jan, feb, mar etc.

Bibliography styles usually contain abbreviations for the names of commonly referenced journals. Consult the Local Guide for a list of the predefined abbreviations for the bibliography styles available.

You can define your own abbreviations by using BibTeX's `@STRING` command.

3.3.5 Field Types

Below is a list of all fields recognized by the standard bibliography styles. An entry can also contain other fields, which are ignored by those styles.
BibTeX ignores the case of letters in the field names.

3.3.5.1 address field

Publisher's address. For major publishing houses, just the city is given. For small publishers, you can help the reader by giving the complete address.
Format:

```
ADDRESS = field_text
```

3.3.5.2 annote field

An annotation, used only for annotated bibliography styles (which are not among the standard ones).
Format:

```
ANNOTE = field_text
```

3.3.5.3 author field

The name(s) of the author(s).
Format:

```
AUTHOR = field_text
```

3.3.5.4 booktitle field

Title of a book, part of which is being cited.
Format:

```
BOOKTITLE = field_text
```

3.3.5.5 chapter field

A chapter number.
Format:

```
CHAPTER = field_text
```

3.3.5.6 edition field

The edition of a book - for example, "second".
Format:

```
EDITION = field_text
```

3.3.5.7 editor field

Name(s) of editor(s). If there is also an "author" field, then the "editor" field gives the editor of the book or collection in which the reference appears.
Format:

```
EDITOR = field_text
```

3.3.5.8 howpublished field

How something strange has been published.
Format:

```
HOWPUBLISHED = field_text
```

3.3.5.9 institution field

The institution that published the work.
Format:

```
INSTITUTION = field_text
```

3.3.5.10 journal field

A journal name. Abbreviations are provided for many journals; see the Local Guide.
Format:

```
JOURNAL = field_text
```

3.3.5.11 key field

Used for alphabetizing and creating a label when the "author" and "editor" fields are missing. This field should not be confused with the citation key that appears in the `\cite` command and at the beginning of the entry.
Format:

```
KEY = field_text
```

3.3.5.12 month field

The month in which the work was published or, for an unpublished work, in which it was written.
Format:

```
MONTH = field_text
```

3.3.5.13 note field

Any additional information that can help the reader.
Format:

```
NOTE = field_text
```

3.3.5.14 number field

The number of a journal, magazine, or technical report. An issue of a journal or magazine is usually identified by its volume and number; the organization that issues a technical report usually gives it a number.
Format:

```
NUMBER = field_text
```

3.3.5.15 organization field

The organization sponsoring a conference.
Format:

```
ORGANIZATION = field_text
```

3.3.5.16 pages field

A page number or range of numbers such as ”42–111”; you may also have several of these, separating them with commas: ”7,41,73–97”. The standard styles convert a single dash to a double. ¿

Format:

```
PAGES = field_text
```

3.3.5.17 publisher field

The publisher’s name.

Format:

```
PUBLISHER = field_text
```

3.3.5.18 school field

The name of the school where a thesis was written.

Format:

```
SCHOOL = field_text
```

3.3.5.19 series field

The name of a series or set of books. When citing an entire book, the the ”title” field gives its title and an optional ”series” field gives the name of a series in which the book is published.

Format:

```
SERIES = field_text
```

3.3.5.20 title field

The work’s title.

Format:

```
TITLE = field_text
```

3.3.5.21 type field

The type of a technical report - for example, "Research Note".
Format:

```
TYPE = field_text
```

3.3.5.22 volume field

The volume of a journal or multivolume book work.
Format:

```
VOLUME = field_text
```

3.3.5.23 year field

The year of publication or, for an unpublished work, the year it was written. This field's text should contain only numerals.
Format:

```
YEAR = field_text
```

3.4 bst files

Bibliography style files define the style of a bibliography source list.
The standard bibliography style files are PLAIN, UNSRT, ALPHA and ABBRV.
If you want to make a bibliography style of your own, look at SAMPLE.BST.

3.4.1 ABBRV.BST

This style is the same as the style defined in `PLAIN.BST`, except that entries are more compact because first names, month names and journal names are abbreviated.

3.4.2 ALPHA.BST

This style is the same as the style defined in `PLAIN.BST` except that entry labels like "Knu66", formed from the author's name and the year of publication, are used.

3.4.3 PLAIN.BST

This style is formatted more or less as suggested by Mary-Claire van Leunen in "A Handbook for Scholars" (Alfred A. Knopf, New York, 1979). Entries are sorted alphabetically and are labelled with numbers.

3.4.4 SAMPLE.BST

This is a sample bibliography style file meant to help you construct a new style. It creates a bibliography in which entries appear as follows:

[Jones79] Jones, R. L. and Richards, P. Q. The Birds and the Bees. {\it Journal of Irreproducible Results 4}, 2 (Jan. 1979), 27-33.

[Jones82a] Jones, P. G. The Bees and the Trees ... (1982).

[Jones82b] Jones, R. L. The Trees and the Peas ... (1982).

[Krist74] Kristofferson, R. P. Peopl and Places ... (1974)

It should illustrate how you write a style file. The functions are described in an informal Pascal-like style in comments. Because of the way while loops and if-then-else statements must use functions, the following convention is used. If a while loop is labeled 'foo' in the informal description, then its test and body are the functions named 'foo.test' and 'foo.body'. If an if statement is labeled 'foo', then its two clauses are the functions named 'foo.then' and 'foo.else'. (Null clauses just use the 'skip$' function.) Note that because functions have to be defined in terms of already-defined functions, the actual function definitions are given in a 'bottom-up' order.

3.4.5 UNSRT.BST

This style is that same as `PLAIN.BST` except that entries appear in the order of their first citation.

4 Makeindex

> The contents of this section was taken from the HTML helppages for Makeindex of Norman Walsh (Version 1.0, 12 Apr 94).

MakeIndex is a general purpose index processor. It takes one or more raw index files (normally generated by a formatter), sorts the entries, and produces the actual index file. It is not dependent on any particular format of raw index file, although the `.idx`

file generated by LaTeX is the default. Up to three levels (0, 1, and 2) of subitem nesting within the same entry is supported. The input format may be redefined in a style file so that raw index or glossary output from other formatters may be processed. The style file also defines the style of output index file. Unless specified otherwise, the file name base of the first input file (`idx0`) is used to determine other related input/output files. The default input file type is `.idx`.

4.1 Options

Makeindex is a Unix program, and therefore has a Unix-style command line. Instead of qualifiers delimited with a slash (`/`), Makeindex options are delimited with a hyphen.

4.1.1 `-i`

Use `stdin` as the input file. When this option is specified and the `-o` is not, output is written to `stdout`.

4.1.2 `-l`

Use letter ordering. Default is word ordering (explained in the Ordering section).

4.1.3 `-q`

Quiet mode, send no messages to `stderr`. By default progress and error messages are sent to `stderr` as well as the transcript file. The `-q` option disables the `stderr` messages.

4.1.4 `-r`

Disable implicit page range formation. By default three or more successive pages will be automatically abbreviated as a range (e.g. 1–5). The `-r` option disables it, making the explicit range operators the only way to create page ranges (see the Special Effects section below).

4.1.5 `-c`

Enable blank compression. By default every blank counts in the index key. The `-c` option ignores leading and trailing blanks and tabs and compresses intermediate ones to a single space.

4.1.6 `-s sty`

Take `sty` as the style file. There is no default for the style file name. The environment variable INDEXSTYLE defines the path where the style file should be found.

4.1.7 `-o ind`

Take `ind` as the output index file. By default the file name base of the first input file `idx0` concatenated with the extension `.ind` is used as the output file name.

4.1.8 `-t log`

Take `log` as the transcript file. By default the file name base of the first input file `idx0` concatenated with the extension `.ilg` is used as the transcript file name.

4.1.9 `-p no`

Set the starting page number of the output index file to be `no`. This is useful when the index file is to be formatted separately. Other than pure numbers, three special cases are allowed for no: `any`, `odd`, and `even`. In these special cases, the starting page number is determined by retrieving the last page number from the source log file. The source log file name is determined by concatenating the file name base of the first raw index file (`idx0`) with the extension `.log`. The last source page is obtained by searching backward in the log file for the first instance of a number included in `[...]`. If a page number is missing or the log file is not found, no attempt will be made to set the starting page number. The meaning of each of these cases follows:

any The starting page is the last source page number plus 1.

odd The starting page is the first odd page following the last source page number.

even The starting page is the first even page following the last source page number.

4.2 Style File

The style file format is very simple. It is a list of `<specifier, attribute>` pairs. There are two types of specifiers (input and output). The pairs don't have to obey any particular order in the file. A line lead by '`%`' is a comment. The following is a list of all the specifiers and their respective arguments where `<string>` is an arbitrary string delimited by double quotes (`"..."`), `<char>` is a single letter embraced by single quotes (`'...'`), and `<number>` is a nonnegative integer. The maximum

length of a `<string>` is 144. Notice that a backslash must be escaped (by an extra backslash) in the string quotation. Anything not specified in the style file will be assigned a default value, which is shown at the rightmost column. This file can reside anywhere in the path defined by the environment variable INDEXSTYLE.

4.2.1 Input Style Specifiers

4.2.1.1 keyword <string>

`"\\indexentry"`
This is the command which tells MakeIndex that its argument is an index entry.

4.2.1.2 arg open <char>

`{`
This is the opening delimiter for the index entry argument.

4.2.1.3 arg close <char>

`}`
This is the closing delimiter for the index entry argument.

4.2.1.4 range open <char¿>

`(`
The opening delimiter indicating the beginning of an explicit page range.

4.2.1.5 range close <char>

`)`
The closing delimiter indicating the end of an explicit page range.

4.2.1.6 level <char>

`!`
The delimiter which denotes a new level of subitem.

4.2.1.7 actual <char>

`@`

The symbol which indicates that the next entry is to appear in the actual index file.

4.2.1.8 encap <char>

`|`

The symbol which indicates that the rest of the argument list is to be used as the encapsulating command for the page number.

4.2.1.9 quote <char>

`"`

4.2.1.10 escape <char>

`\\`

The symbol which escapes the next letter, unless its preceding letter is escape. In other words, quote is used to escape the letter which immediately follows it. But if it is preceded by escape, it does not escape anything.

Notice that the two symbols must be distinct.

4.2.2 Output Style Specifiers

4.2.2.1 preamble <string>

`"\\begin{theindex}\n"`

The preamble of actual index file.

4.2.2.2 postamble <string>

`"\n\n\\end{theindex}\n"`

The postamble of actual index file.

4.2.2.3 setpage prefix <string>

`"\n \\setcounter{page}{"`

The prefix of the command which sets the starting page number.

4.2.2.4 setpage suffix <string>

`"}\n"`
The suffix of the command which sets the starting page number.

4.2.2.5 group skip <string>

`"\n\n \\indexsapce\n"`
The vertical space to be inserted before a new group begins.

4.2.2.6 lethead prefix <string>

`""`
The header prefix to be inserted before a new letter begins.

4.2.2.7 lethead suffix <string>

`""`
The header suffix to be inserted before a new letter begins.

4.2.2.8 lethead flag <string>

`0`
The flag indicating the condition of inserting new letter header. Default is 0, which means no header. Positive means insert an uppercase letter between prefix and suffix. Negative means insert a lowercase letter.

4.2.2.9 item 0 <string>

`"\n \\item "`
The command to be inserted between two primary (level 0) items.

4.2.2.10 item 1 <string>

`"\n \\subitem "`
The command to be inserted between two secondary (level 1) items.

4.2.2.11 item 2 <string>

`"\n \\subsubitem "`

The command to be inserted between two level 2 items.

4.2.2.12 item 01 <string>

`"\n \\subitem "`

The command to be inserted between a level 0 item and a level 1 item.

4.2.2.13 item x1 <string>

`"\n \\subitem "`

The command to be inserted between a level 0 item and a level 1 item. The difference between this and previous is that in this case the level 0 item doesn't have any page numbers.

4.2.2.14 item 12 <string>

`"\n \\subsubitem "`

The command to be inserted between a level 1 item and a level 2 item.

4.2.2.15 item x2 <string>

`"\n \\subsubitem "`

The command to be inserted between a level 1 item and a level 2 item. The difference between this and previous is that in this case the level 1 item doesn't have any page numbers.

4.2.2.16 delim 0 <string>

`", "`

The delimiter to be inserted between a level 0 key and its first page number. Default is a comma followed by a blank.

4.2.2.17 delim 1 <string>

`", "`

The delimiter to be inserted between a level 1 key and its first page number. Default is a comma followed by a blank.

4.2.2.18 delim 2 <string>

`", "`

The delimiter to be inserted between a level 2 key and its first page number. Default is a comma followed by a blank.

4.2.2.19 delim n <string>

`", "`

The delimiter to be inserted between two page numbers for the same key in any level. Default is a comma followed by a blank.

4.2.2.20 delim r <string>

`"--"`

The delimiter to be inserted between the starting and ending page numbers of a range.

4.2.2.21 encap prefix <string>

`"\\"`

The prefix for the command which encapsulates the page number.

4.2.2.22 encap infix <string>

`"{"`

The prefix for the command which encapsulates the page number.

4.2.2.23 encap suffix <string>

`"}"`

The suffix for the command which encapsulates the page number.

4.2.2.24 line max <number>

72

The maximum length of a line in the output beyond which a line wraps around.

4.2.2.25 indent space <string>

`"\t\t"`

The space to be inserted in front of a wrapped line. Default is two tabs.

4.2.2.26 indent length <number>

`16`

The length of indent_space. In the default case this is 16 (for 2 tabs).

4.3 Example

The following example shows a style file called `book.isty` which defines a stand-alone index for a book. By stand-alone, we mean it can be formatted independent of the main source.

```
preamble
"\\documentstyle[12pt]{book}
\\begin{document}
\\begin{theindex}
{\\small\n"

postamble
"\n\n}
\\end{theindex}
\\end{document}\n"
```

Suppose a particular book style requires the index (as well as any chapters) to start from an odd page number. Given `foo.idx` as the raw index file, the following command line produces an index in file `FOO-.IND`.

```
makeindex  -s book.isty  -o foo-.ind  -p odd  foo
```

The reason to use a non-default output file name is to avoid clobbering the source output (presumably `foo.dvi`) because if the index is in file `foo.ind`, its output will also be in `foo.dvi` as a result of separate formatting using . In the example the index is in `foo-.ind`, its output will be in `foo-.dvi` and thus introduces no confusion.

4.4 Ordering

By default `makeindex` assumes word ordering. The `-l` option turns it into letter ordering. The only difference is whether a blank is treated as an effective letter or not. In word ordering, a blank precedes any letter in the alphabet, whereas in letter ordering, it doesn't count at all. This is best illustrated by the following example:

`word order letter order sea lion seal seal sea lion`

Numbers are sorted in numeric order. For instance,

`9 (nine), 123 10 (ten), see Derek, Bo`

Letters are first sorted with uppercase and lowercase considered identical; then, within identical words the uppercase letter precedes its lowercase counterpart.

Patterns lead by a special symbol precede numbers, which precede patterns lead by a letter. The symbol here refers to anything not in the union of digits and English alphabet. This includes those which follow 'z' in the ASCII chart. As a special case, anything started with a digit but mixed with non-digits is considered a symbol-leading pattern instead of a number.

4.5 Special Effects

In the normal case entries such as

```
\indexentry{alpha}{1}
\indexentry{alpha!beta}{3}
\indexentry{alpha!beta!gamma}{10}
```

in the raw index file will be converted to

```
\item alpha, 1 \subitem beta, 3 \subsubitem gamma, 10
```

in the output index file by makeindex. Notice that the level symbol (`!`) is used to delimit levels of nesting.

It is possible to make an item appear in a designated form by using the actual (`@`) operator. For instance,

```
\indexentry{alpha@{\it alpha\/}}{1}
```

will become

```
\item {\it alpha\/} 1
```

after the conversion. The idea is that the pattern preceding `@` is used as sort key, whereas the one following it is put in the actual result. However, the same key with and without the actual part are regarded as distinct entries.

It is also possible to encapsulate a page number with a designated command using the encap (`|`) operator. For example, in the default case,

```
\indexentry{alpha|bold}{1}
```

will be converted to

```
\item alpha \bold{1}
```

where `\bold{n}` will expand to `{\bf n}`. This allows the encap operator to be used to set pages in different fonts, thereby conveying more information about whatever being indexed. For instance, given the same key the page where its definition appears can be in one font while where its primary example is given can be in another, with other ordinary appearances in a third. Notice that in this example, the three output attributes associated with page encapsulation `encap_prefix`, `encap_infix`, and `encap_suffix` correspond respectively to backslash, left brace, and right brace. If this is to be formatted by languages other than , they would be defined differently.

By the same token, the `encap` operator can be used to make cross references in the index. For instance,

```
\indexentry{alpha|see{beta}}{1}
```

will become

```
\item alpha \see{beta}{1}
```

in the output index file after the conversion, where

```
\see{beta}{1}
```

will expand to

```
{\it see\/} beta
```

Notice that in a cross reference like this the page number disappears. Therefore, where to insert such a command in the source is immaterial.

A pair of encap concatenated with `range_open` (`|(`) and with `range_close` (`|)`) creates an explicit page range. That is,

```
\indexentry{alpha|(}{1}  \indexentry{alpha|)}{5}
```

will become

```
\item alpha, 1--5
```

Intermediate pages indexed by the same key will be merged into the range implicitly. This is especially useful when an entire section about a particular subject is to be indexed, in which case only the range opening and closing operators need to be inserted at the beginning and end of the section, respectively.

This explicit page range formation can also include an extra command to set the page range in a designated font. Thus

```
\indexentry{alpha|(bold}{1}  \indexentry{alpha|)}{5}
```

will become

```
\item alpha, \bold{1--5}
```

A couple of special cases are worth mentioning here. First, entries like

```
\indexentry{alpha|(}{1}  \indexentry{alpha|bold}{3}  \indexentry{alpha|)}{5}
```

will be interpreted as

```
\item alpha, \bold{3}, 1--5
```

but with a warning message in the transcript about the encounter of an inconsistent page encapsulator. Secondly, an explicit range beginning in a Roman page number and ending in Arabic is considered an error. In a case like this the range is broken into two subranges, if possible, one in Roman, the other in Arabic. For instance,

```
\indexentry{alpha|(}{i}  \indexentry{alpha}{iv}  \indexentry{alpha}{3}
 \indexentry{alpha|)}{7}
```

will be turned into

```
\item alpha, 1--iv, 3--7
```

with a warning message in the transcript complaining about the illegal range formation.

Finally, every special symbol mentioned in this section may be escaped by the quote operator ("). Thus

```
\indexentry{alpha"@beta}{1}
```

will actually become

```
\item alpha@beta, 1
```

as a result of executing makeindex. However, if quote is preceded by escape (\), its following letter is not escaped. That is,

```
\indexentry{f\"ur}{1}
```

means

```
\item f\"ur, 1
```

which represents umlaut accented `u` to the family of processors.

5 xindy

> The content of this section was taken from the original documentation of xindy V2.1 (`Doc/manual*.html`).

xindy means flexible *ind*exing system. It is an indexing system that can be used to generate book-like indexes for arbitrary document preparation systems. This term includes systems such as TeX and LaTeX, the Nroff-family or SGML-based systems (e.g. HTML) that process some kind of text and generate indexing information. It is not fixed to any specific system, but can be configured for a wide variety of purposes.

5.1 Command List

Here is the complete list of xindy's commands that may be used in the index style. The symbol `name` always refers to a string. We separate the commands into the processing and markup commands. The commands are listed in alphabetical order. The parenthesis `[` and `]'` denote optional parts of the syntax and `{` and `}` denote the grouping of elements. A vertical bar indicates alternatives. However, the enclosing round braces *are* part of the syntax and must be supplied.

5.1.1 Processing Commands

5.1.1.1 `define-alphabet`

```
(define-alphabet name string-list)
```

Defines `name` to be the alphabet consisting of all elements of the `string-list`. Examples:

```
(define-alphabet "example-alphabet" ("An" "Example" "Alphabet"))
```

defines an alphabet consisting of exactly three symbols. For the successor relationship holds: `succ("An")="Example"` and `succ("Example")="Alphabet"`. The built-in alphabet `digits` is defined as follows:

```
(define-alphabet "digits"
                 ("0" "1" "2" "3" "4" "5" "6" "7" "8" "9"))
```

5.1.1.2 `define-attributes`

```
(define-attributes attribute-list)
```

Defines all attributes the raw index may contain. Parameter `attribute-list` is a list of list of strings. The nesting level must not be more than 2. So `(..(..)..)` is allowed, whereas `(..(..(..)..)..)` is not.

The list has two kinds of elements: strings and list of strings. A single string is treated as if it were a single element list. So the lists `("definition")` and `( ("definition") )` are equivalent. All elements forming a list are a so-called *attribute group*. The members of a group are written to the output file before any member of the following groups are written.

Examples of valid attributes lists are:

`("definition" "usage")` defines two attribute groups. The first one contains all references with the attribute `definition` and the second one all with the attribute `usage`.

`(("definition" "important") "usage")` defines two attribute groups. The first one contains all references with the attributes `definition` or `important` and the second one all with the attribute `usage`. In the attribute group `("definition" "important")` the attribute `definition` overrides `important`.

5.1.1.3 `define-crossref-class`

```
(define-crossref-class name [:unverified])
```

Defines `name` to be a class of cross references. We distinguish two types of cross reference classes. *Verified* cross reference classes can be checked for dangling references. If for instance a cross reference points to the non-existent keyword ‘foo’ a warning is issued and the user is advised to correct the invalid cross reference. This is the default. If for some reasons this mechanism must be deactivated the switch `:unverified` can be used to suppress this behaviour.

5.1.1.4 `define-letter-group`

```
(define-letter-group name [:before lgname] [:after lgname]
                          [:prefixes list-of-prefixes])
```

```
(define-letter-groups list-of-letter-groups)
```

This command defines a letter group with name `name`, which must be a string value, grouping all index entries that have a *sort key* beginning with the prefix `name`. The command

```
(define-letter-group "a")
```

is equivalent to the command

```
(define-letter-group "a" :prefixes ("a"))
```

Using the latter form one can associate more than one prefix with a given letter group. Also further prefixes can be added to an already existing letter group by simply defining the same letter group again. This results not in a redefinition but in adding more prefixes to the currently defined prefixes.

Example:

```
(define-letter-group "a")
```

defines a letter group containing all index entries beginning with the string `"a"`.

```
(define-letter-group "c" :after "a")
```

defines a letter group containing all index entries beginning with the string `"c"`. The letter group appears behind the letter group `"a"`

```
(define-letter-group "b" :after "a" :before "c")
```

inserts letter group `"b"` between letter group `"a"` and `"c"`. This allows incremental definition of letter groups by extending already defined ones.

The arguments `:after` and `:before` define a partial order on the letter groups. xindy tries to convert this partial order into a total one. If this is impossible due to circular definitions, an error is reported. If more than one possible total ordering can result, it is left open which one is used, so one should always define a complete total order.

The command `define-letter-groups` (with an 's' at the end) is simply an abbreviation for a sequence of `define-letter-group` definitions where the elements are ordered in the ordering given by the list. Example:

```
(define-letter-groups ("a" "b" "c")
```

equals the definitions

```
(define-letter-group "a")
(define-letter-group "b" :after "a")
(define-letter-group "c" :after "b")
```

See also commands `markup-letter-group-list` and `markup-letter-group` for further information.

5.1.1.5 `define-location-class`

```
(define-location-class name layer-list
                       [:min-range-length num]
                       [:hierdepth depth]
                       [:var])
```

Defines `name` to be a location class consisting of the given list of layers. A list of layers consists of names of basetypes and/or strings representing separators. Separators must follow the keyword argument `:sep`. If the keyword `:min-range-length` is specified we define the *minimum range length* to be used when building ranges. The argument `num` must be a positive integer number or the keyword `none` in which case the building of ranges is disallowed. If the switch `:var` is specified the declared class is of type *variable*, i.e. it is a *var-location-class*. Since building of ranges is currently only allowed for standard classes `:var` and `:min-range-length` must not be used together. The keyword argument `:hierdepth` can be used to declare that the location references have to be tagged in a hierarchical form. Its argument `depth` must be an integer number indicating the number of layers the hierarchy does contain. See command `markup-locref-list` for more information. Examples:

```
(define-location-class "page-numbers" ("arabic-numbers")
                       :minimum-range-length 3)
```

Defines the location class `page-numbers` consisting of one layer which is the alphabet `arabic-numbers`. Since the minimum range length is set to 3 the location references 2, 3 and 4 don't form a range because the range length is only 2. But the references 6, 7, 8, and 9 are enough to form a range. Some example instances of this class are `0, 1, ... 2313`, etc.

```
(define-location-class "sections" :var
   ("arabic-numbers" :sep "."
    "arabic-numbers" :sep "."
```

```
    "arabic-numbers"))
```

defines a variable location class. Valid instances are `1, 1.1, 1.2, 2, 2.4.5`, but none of `2-3` (wrong separator), `1.2.3.4` (more than 3 layers), `2.3.iv` (roman number instead of arabic one).

5.1.1.6 `define-location-class-order`

```
(define-location-class-order list)
```

Defines the order in which the location classes are written to the output file. The parameter `list` is a list of names of location classes. Examples:

```
(define-location-class-order
    ("page-numbers" "sections" "xrefs"))
```

tells the system that the page numbers should appear before the section numbers and that the cross references should appear at the end. If this command is omitted, the declaration order of the location classes in the index style is implicitly used as the output order. In the case that a location class does not appear in the list, the output may behave unexpectedly, so one should always enumerate all used location classes when using this command.

5.1.1.7 `define-rule-set`

```
(define-rule-set name
        [ :inherit-from ("rule-set" "rule-set-2") ]
        :rules (<rule>...) )
```

A complete specification of a multi-phase sorting process for a language requires that some rules have to appear in several subsequent sorting phases. Rule sets can be used to define a set of rules that can be instantiated in an arbitrary sorting phase. Basically, they offer means to separate the definition of sorting rules from their instantiation, hence, acting as a wrapper for calls to `sort-rule`. They do not add new functionality that is not already present with `sort-rule`.
A rule can be of the form:

```
<rule> ::= ("pattern" "replacement"
            [:string|:bregexp|:egegexp] [:again])
```

The following incomplete example defines a new rule set of name `isolatin1-tolower` that inherits definitions from rule set `latin-tolower`, overriding or adding the sort rules in the list of `:rules`.

```
(define-rule-set "isolatin1-tolower"

  :inherit-from ("latin-tolower")

  :rules (("?" "?" :string :again)
          ("?" "?" :string :again)
          ("?" "?" :string :again)
          ("?" "?" :string :again)
          ("?" "?" :string :again)
          ("?" "?" :string :again)
          ("?" "?" :string :again)
        ...
          )
...)
```

Rule sets can be instantiated with the command `use-rule-set`. For further descriptions on the sorting model refer to the command `sort-rule`.

5.1.1.8 `define-sort-rule-orientations`

```
(define-sort-rule-orientations (orientations...))
```

Defines the order for the different sorting phases. The currently implemented *orientations* are `forward` and `backward`. This command must precede all `sort-rule` commands in an index style. It defines the orientations and implicitly sets the maximum number of sorting phases performed.

For further descriptions on the sorting model refer to the command `sort-rule`.

5.1.1.9 `merge-rule`

```
(merge-rule pattern replacement [:again]
                                [:bregexp | :eregexp | :string])
```

Defines a keyword mapping rule that can be used to generate the *merge key* from the *main key* of an index entry. This mapping is necessary to map all keywords that are differently written but belong to the same keyword to the same canonical keyword.

The parameter `pattern` can be a POSIX-compliant regular expression or an ordinary string. The implementation uses the GNU Rx regular expression library which

implements the POSIX regular expressions. Regular expressions (REs) can be specified as *basic regular expressions* (BREs) or *extended regular expressions* (EREs). You can use the switch `:bregexp` to force the interpretation of the pattern as a BRE, or `:eregexp` to interpret it as an ERE. If you want xindy to interpret the pattern literally, use the switch `:string`. If none of these switches is selected, xindy uses an auto-detection mechanism to decide, if the pattern is a regular expression or not. If it recognizes the pattern as a RE, it interprets it as an ERE by default.

The parameter `replacement` must be a string possibly containing the special characters `&` (substitutes for the complete match) and `\1 ,..., \9` (substituting for the *n*-th submatch. Examples:

```
(merge-rule "A" "a")
```

replaces each occurrence of the uppercase letter `‘A’` with its lowercase counterpart.

```
(merge-rule "\~"([AEOUaeou])" "\1")
```

transforms the TeX umlaut-letters into their stripped counterparts, such that `‘\"A ’` is treated as an `‘A’` afterwards.

The following sequences have a special meaning:

`‘ ~n ’` End of line symbol (*linefeed*).

`‘ ~b ’` The ISO-Latin character with the lowest ordinal number.

`‘ ~e ’` The ISO-Latin character with the highest ordinal number.

`‘ ~~ ’` The tilde character.

`‘ ~" ’` The double quote character.

Tilde characters and double quotes have to be quoted themselves with a tilde character. The special characters `‘ ~b ’` and `‘ ~e ’` allow the definition of arbitrary sorting orders by rules. In connection with an additional character every position in the alphabet can be described. E.g. `‘ m~e ’` is lexicographically placed between `‘m’` and `‘n’`.

Due to efficiency, rules that just exchange characters or substitute constant character sequences are not treated as regular expressions. Therefore, instead of using the rule

```
(merge-rule "[A-Z]" "&")
```

it is more efficient (though less comfortable) to use

```
(merge-rule "A" "Ax")
(merge-rule "B" "Bx")
...
(merge-rule "Z" "Zx")
```

Usually rules are applied in order of their definition. Rules with a special prefix precede those that begin with a class of characters, so that the search pattern `'alpha'` is checked before `'.*'`, but `'auto'` and `'a.*'` are checked in order of their definition.

The first rule from a style file that matches the input is applied and the process restarts behind the substituted text. If no rule could be applied, the actual character is copied from the input and the process continues with the next character.

Sometimes it is necessary to apply rules anew to the result of a transformation. By specifying the keyword argument `:again` in the merge rule the rule is marked as mutable, which means that after using this rule the transformation process shall restart at the same place. E.g. the rule

```
(merge-rule "\$(.*)\$" "\1" :again)
```

deletes *all* surrounding `'$ '` symbols from the input.

See also command `sort-rule`.

5.1.1.10 `merge-to`

```
(merge-to attr-from attr-to [:drop])
```

A merge rule says that the attribute `attr-from` can be used to build ranges in `attr-to`. Both attributes must name valid attribute names. The switch `:drop` indicates, that the original location reference with attribute `attr-from` has to be dropped (removed), if a successful range was built with location references in attribute `attr-to`. A detailed description is given in the section about processing phases.

5.1.1.11 `require`

```
(require filename)
```

This command allows to load more index style modules. The module is searched in the directories defined in the search path. The file is read in and processing of the current file continues. The argument `filename` must be a string. This allows to decompose the index style into several modules that can be included into the topmost index style file. Example:

```
(require "french/alphabet.xdy")
(require "french/sort-rules.xdy")
(require "tex/locations.xdy")
(require "tex/markup.xdy")
```

Submodules can load other submodules as well. If a file is required that was already loaded, the `require` command is simply ignored and processing continues without including this file twice. See also command searchpath.

5.1.1.12 `searchpath`

```
(searchpath {path-string | path-list})
```

This command adds the given paths to the list of paths, xindy searches for index style files. The argument `path-string` must be a colon-separated string of directory names. If this path ends with a colon the default search path is added to the end of the path list. Example:

```
(searchpath ".:/usr/local/lib/xindy:/usr/local/lib/xindy/english:")
```

adds the specified directories to the search path. Since the last path ends with a colon, the built-in search path is added at the end. Specifying

```
(searchpath ("."
             "/usr/local/lib/xindy"
             "/usr/local/lib/xindy/english"
             :default))
```

yields exactly the same result as the example above. Here `path-list` must be a list of strings and/or the keyword(s) `:default` and `:last`. The keyword `:default` signifies that the default pathnames are to be inserted at the specified position in the list. The keyword `:last` allows to insert the currently active paths at the indicated position. Since this allows to insert the built-in paths at any position and incrementally adding new paths to the search path, this version of the command ist more flexible than the first version.

5.1.1.13 `sort-rule`

```
(sort-rule pattern replacement [:run level] [:again])
```

Defines a keyword mapping rule that can be used to generate the *sort key* of an index entry from the *merge key*. This key is used to sort the index entries lexicographically after they have been merged using the merge key.

The argument `:run` indicates that this rule is only in effect a the specified *level* (default is level 0). For a detailed discussion on the definition of sort rules for

different layers refer to the documentation about the new sorting scheme (`new-sort-rules`) that comes with this distribution.

See command `merge-rule` for more information about keyword rules.

5.1.1.14 `use-rule-set`

```
(use-rule-set [:run phase]
              [:rule-set ( <rule-set>... ))
```

This command instantiates the gives rule sets to be in effect at sorting phase `phase`. The order of the rule sets given with argument `:rule-set` is significant. Rule set entries of rule set appearing at the beginning of the list override entries in rule sets at the end of the list.

The following example declares that in phase 0 the rule sets `din5007` and `isolatin1-tolower` should be active, whereas in phase 2 the other rule sets have to be applied.

```
(use-rule-set :run 0
              :rule-set ("din5007" "isolatin1-tolower"))

(use-rule-set :run 1
              :rule-set ("resolve-umlauts"
                         "resolve-sharp-s"
                         "isolatin1-tolower"
                        ))
```

For a discussion on rule sets refer to command `define-rule-set`.

5.1.2 Markup Commands

The following commands can be used to define the markup of the index. They don't have any influence on the indexing process. Since the markup scheme is characterized by the concept of *environments*, the syntax and naming scheme of all commands follows a simple structure.

The commands can be separated into *environment* and *list-environment* commands. All commands of the first group support the keyword arguments `:open` and `:close`, whereas the second group additionally supports the keyword argument `:sep`. If one of these keyword arguments is missing, the default markup tag is always the empty tag. The `:open` tag is always printed before the object itself and the `:close` tag is always printed after the object has been printed. If a list is printed the `:sep` tag is printed between two elements of the list but not before the first element, or after

the last one. All commands dealing with a list have the suffix `'-list'` as part of their command name.

Since the number of commands and the heavy usage of *default* and *specialized* tags makes the markup somehow complex (but very powerful) we have added a mechanism to trace the markup tags xindy omits during its markup phase with the command `markup-trace`.

Here follows the list of markup commands in alphabetical order with some of the commands grouped together.

5.1.2.1 `markup-attribute-group-list`

```
(markup-attribute-group-list [:open string] [:close string]
                             [:sep string])

(markup-attribute-group      [:open string] [:close string]
                             [:group group-num])
```

Location class groups consist of lists of attribute groups. The markup of this list can be defined with the command `markup-attribute-group-list`.

To allow different markup for different attribute groups the command `markup-attribute-group` can be specialized on the group number with the keyword argument `:group` which must be an integer number. E.g., given are the groups `("definition" "theorem")` and `("default")` with group numbers 0 and 1, then

```
(markup-attribute-group :open "<group0>" :close </group0>"
                        :group 0)

(markup-attribute-group :open "<group1>" :close "</group1>"
                        :group 1)
```

can be used to assign different markup for both groups in a SGML-based language.

5.1.2.2 `markup-crossref-list`

```
(markup-crossref-list       [:open string] [:close string]
                            [:sep string]
                            [:class crossref-class])

(markup-crossref-layer-list [:open string] [:close string]
                            [:sep string]
```

```
                              [:class crossref-class])

(markup-crossref-layer        [:open string] [:close string]
                              [:class crossref-class])
```

A crossref class group contains cross references of the same class. The separator between the classes is defined with the `(markup-locclass-list :sep)`-parameter. A list of cross references can be tagged with the `command markup-crossref-list` that specializes on the `:class` argument.

Each cross reference is determined by a list of layers indicating the target of the cross reference. To define a suitable markup for such a list the `command markup-crossref-layer-list` can be used.

Each layer of a cross reference can be assigned two tags that specialize on the class of the reference, like all other commands.

A suitable markup for a cross reference class `see` within LaTeX2e could look like that:

```
(markup-crossref-list :class "see" :open "\emph{see} "
                                   :sep  "; ")
(markup-crossref-layer-list :class "see" :sep ",")
(markup-crossref-layer :class "see"
                                   :open "\textbf{" :close "}")
```

An example output could look like ... `see house; garden,winter; greenhouse`

5.1.2.3 `markup-index`

```
(markup-index [:open string] [:close string]
              [ :flat | :tree | :hierdepth depth ])
```

Defines the markup tags that enclose the whole index via the `:open` and `:close` parameters. Examples:

```
(markup-index :open  "Here comes the index~n"
              :close "That's all folks!~n")
```

defines that the `:open` string is printed before the rest of the index and the `:close` string appears after the index is printed.

Additionally one can specify the form of the generated index. It is possible to produce flat indexes by specifying the switch `:flat`, to generate a tree with the `:tree` switch or any kind of mixture between both by specifying the depth up to which trees shall be built with the parameter `:hierdepth`. Its argument depth

is the number of layers that can be formed into a tree. Therefore `:flat` is an abbrevation of `:hierdepth 0` and `:tree` is an abbrevation of `:hierdepth max-depth`, with `max-depth` being the maximum number of layers a keyword has. An example: the keywords

```
("tree" "binary" "AVL")
("tree" "binary" "natural")
```

can be transformed in the following ways:
A flat index (`:flat` or `:hierdepth 0`)

```
tree binary AVL
tree binary natural
```

with `:hierdepth 1`

```
tree
   binary  AVL
   binary  natural
```

and a tree (`:tree` or `:hierdepth > 1`)

```
tree
   binary
      AVL
      natural
```

Most often one will create tree-like indexes or ones that are flat.

5.1.2.4 `markup-indexentry-list`

```
(markup-indexentry-list [:open string] [:close string]
                        [:sep string]  [:depth integer])

(markup-indexentry      [:open string] [:close string]
                        [:depth integer])
```

Letter groups consists of a list of index entries. The command `markup-indexentry-list` defines the markup of these lists. The markup can be specialized on the depth if the index is hierarchically organized. The command

```
(markup-indexentry-list :open  "\begin{IdxentList}"
                        :close "\end{IdxentList}"
```

```
                          :sep   "~n")
```

defines that the index entries of all layers are wrapped into the given markup tags. If additionally

```
(markup-indexentry-list :open  "\begin{IdxentListII}"
                        :close "\end{IdxentListII}"
                        :sep   "~n"
                        :depth 2)
```

is defined, all index entry lists of all layers (except layer 2) are tagged according to the first specification, and the index entry list within depth 2 are tagged according to the second rule.

The command `markup-indexentry` defines the markup of an index entry at a given depth. Since index entries may also contain subentries and the markup for subentries may be different in different layers, the optional keyword argument `:depth` can be used to assign different markup for different layers. If depth is ommited the default markup for all possible depths is defined. The top-most index entries have depth 0.

```
(markup-indexentry :open  "\begin{Indexentry}"
                   :close "\end{Indexentry}")
```

defines that the index entries of all layers are wrapped into the given markup tags. If additionally

```
(markup-indexentry :open  "\begin{IndexentryII}"
                   :close "\end{IndexentryII}"
                   :depth 2)
```

is defined, all index entries of all layers (except layer 2) are tagged according to the first specification, and the index entries with depth 2 are tagged according to the second rule.

5.1.2.5 `markup-keyword-list`

```
(markup-keyword-list [:open string] [:close string]
                     [:sep string] [:depth integer])

(markup-keyword      [:open string] [:close string]
                     [:depth integer])
```

The print key of an index entry consists of a list of strings. The markup of this list can be defined with the command `markup-keyword-list`. The keyword argument `:depth` may be specified to define the markup of the list at a particular depth.

The keyword of an index entry consists of a list of strings. Each of these components is tagged with the strings defined with the command `markup-keyword`. Since we maybe need different markup for different layers, the optional keyword argument can be used to specialize this markup for some depth.

5.1.2.6 `markup-letter-group-list`

```
(markup-letter-group-list [:open string] [:close string]
                          [:sep string])

(markup-letter-group  [:open string] [:close string] [:group group-name]
                      [:open-head string] [:close-head string]
                      [:upcase | :downcase | :capitalize])
```

The first command defines the markup of the letter group with name `group-name`. Since the markup of letter groups often contains the name of the letter group as a part of it, the other keyword arguments allow an additional markup for this group name. If one of the parameters `:open-head` and `:close-head` is specified additional markup is added as can be described as follows:

```
<OPEN>
   IF (:open-head OR :close-head)
     <OPEN-HEAD>
       transformer-of(<GROUP-NAME>)
     <CLOSE-HEAD>
   FI
   <INDEXENTRIES...>
<CLOSE>
```

Here, `transformer-of` is a function that possibly transforms the string representing the group name into another string. The transformers we currently support can be specified with the switches `:upcase`, `:downcase` and `:capitalize` which result in the corresponding string conversions. If none of them is specified no transformation is done at all.

The command `markup-letter-group` defines the markup of the list of letter groups.

5.1.2.7 `markup-locclass-list`

```
(markup-locclass-list [:open string] [:close string]
                      [:sep string])
```

Each index entry contains a list of location class groups. This markup command can be used to define the markup of this list.

5.1.2.8 `markup-locref`

```
(markup-locref [:open string] [:close string]
               [:class locref-class]
               [:attr  attribute]
               [:depth integer])
```

The markup tags of a location reference can be specialized on the three arguments `:class`, `:attr` and additionally, if sub-references are used, `:depth`. Most often one will only use a tag depending on the attribute. For example, all location references with the attribute `definition` should appear in a font series like bold, emphasizing the importance of this location reference; those with the attribute `default` in font shape italic. The markup in this case would not specialize on the depth or any particular class. A valid definition, suitable for a usage within HTML, could look like this.

```
(markup-locref :open "<B>" :close "</B>" :attr "definition")
(markup-locref :open "<I>" :close "</I>" :attr "default")
```

5.1.2.9 `markup-locref-class`

```
(markup-locref-class [:open string] [:close string]
                     [:class locref-class])
```

All location references of a particular location reference class can be wrapped into the tags defined by this command. It specializes on the keyword argument `:class`.

5.1.2.10 `markup-locref-layer`

```
(markup-locref-layer      [:open string] [:close string]
                          [:depth integer] [:layer integer]
                          [:class locref-class])

(markup-locref-layer-list [:open string] [:close string]
```

```
                          [:sep string]
                          [:depth integer]
                          [:class locref-class])
```

A location reference contains a list of location reference layers. The second markup command can be used to markup this list. It specializes on the class of the location references and the depth (if sub-references are used).

The first command allows to tag the elements of a layer list differently. The first element of this list can is specialisable with `:layer 0`, the next element with `:layer 1`, etc. See the next example for an example.

5.1.2.11 `markup-locref-list`

```
(markup-locref-list [:open string] [:close string] [:sep string]
                    [:depth integer] [:class locref-class])
```

An attribute group contains a list of location references and/or ranges. Additionally a layered location reference itself may contain sub-references that are stored as a list of location references. We specialize the markup for these lists on the location class they belong to with the keyword argument `:class`, and on `:depth` that specializes on the different subentry levels when using location references with sub-references.

Given is a list of location references that have the class description

```
(define-location-class "Appendix"
                       ("ALPHA" :sep "-" "arabic-numbers")
                       :hierdepth 2)
```

This location class has instances like `A-1`, `B-5`, etc. The keyword argument `:hierdepth 2` informs xindy to markup these location references in a hierarchical form. With the commands

```
(markup-locref-list            :sep "; "
                     :depth 0  :class "Appendix")
(markup-locref-list  :open " " :sep ","
                     :depth 1  :class "Appendix")
(markup-locref-layer :open "{\bf " :close "}" :layer 0
                     :depth 0  :class "Appendix")
```

we obtain a markup sequence for some example data that could look like

```
 \bf A} 1,2,5; {\bf B} 5,6,9; {\bf D} 1,5,8; ...
```

5.1.2.12 `markup-range`

```
(markup-range [:open string] [:close string] [:sep string]
              [:class locref-class]
              [:length num] [:ignore-end])
```

A range consists of two location references. Markup can be specified with the `:open` and `:close` arguments and one separator given by the argument `:sep`.

Since both location references are tagged with markup defined by the command `markup-locref` a specialization on attributes or depth is not necessary. Specialization is allowed on the class they belong to, because the separator between two location refences may be different for each location class. Argument `:length` can be used to define different markup for different lengths. In conjunction with `:length` is may be useful not to print the second location reference at all. For example, one wishes to markup ranges of length 1 in the form Xf. instead of X–Y. This can be accomplished with the switch `:ignore-end`.

The markup tags for a range (X,Y) can be described as follows:

```
<OPEN>
  Markup of location reference X
<SEP>
  IF (not :ignore-end)
     Markup of location reference Y
  FI
<CLOSE>
```

The following tags can be used to define a range of page numbers (given in a location class `page-numbers`) without considering the open and close parameters:

```
(markup-range :sep "-" :class "page-numbers")
```

Location ranges then appear separated by a hyphen in a form like this:

```
   ..., 5-8, 19-23, ...
```

5.1.2.13 (markup-trace [:on] [:open string] [:close string])

This command can be used to activate the tracing of all markup commands xindy executes. The switch `:on` activates the trace. If `:on` is omitted, the command line flag `-t` can be used as well. All tags which are emitted but not yet defined explicitly by the user are tagged with a symbolic notation indicating the commands that must be used to define this tag. The defaults for the keyword argument `:open` is `'<'` and for `:close` is `'>'`. The beginning of an example output could look like:

```
<INDEX:OPEN>
  <LETTER-GROUP-LIST:OPEN>
    <LETTER-GROUP:OPEN ["a"]>
      <INDEXENTRY-LIST:OPEN [0]>
        <INDEXENTRY:OPEN [0]>
          <KEYWORD-LIST:OPEN [0]>
            <KEYWORD:OPEN [0]>
 ...
```

We use a simple indentation scheme to make the structure of the tags visible. The symbolic tag `<LETTER-GROUP:OPEN ["a"]>` for example indicates that the tag that can be specified with the command

```
(markup-letter-group :open "XXX" :group "a" ... )
```

is emitted at this point in the markup process. By incrementally adding markup commands to the index, more and more tags can be defined until the whole markup is defined. This general mechanism should allow everyone understand the markup process. The best is to start with a small index, define the complete markup and afterwards process the whole index. Additionally one can enclose the symbolic tags into an environment that is neutral to the document preparation system, such as a comment. For TeX this could be

```
(markup-trace :open "%%" :close "~n")
```

or a definition in the TeX document like

```
\def\ignore#1{}
```

combined with the command

```
(markup-trace :open "\ignore{" :close "}")
```

5.1.3 Raw Index Interface

This section can be skipped if the reader is not interested in adapting xindy to a new document preparation system.

The raw index is the file that represents the index that is to be processed. Since many different document preparation systems may use different forms of index representations, their output must be transformed in a form readable by xindy. We also could have written an configurable parser performing this task, but usually a tool written with some text processing tools such as `perl`, `sed` or `awk` can achieve

the same task as well. Therefore, adapting xindy to a completely different system can mostly be done by writing an appropriate raw index filter.
The format of the raw index interface of xindy is defined as follows:

```
(indexentry { :key string-list [:print string-list]
            | :tkey list-of-layers }
            [:attr string]
            { :locref string  [:open-range | :close-range]
            | :xref string-list } )
```

The pseudo variable *string* is a sequence of characters surrounded by double quotes, e.g.

```
"Hi, it's me"  "one"  "a string with two \"double quotes\""
```

are three examples of valid strings. If you need to include a double quote as a literal character, you must quote it itself with a backslash as shown in the third example. A *string list* is simply a list of strings separated by whitespaces and surrounded by round braces. An example of a string list is

```
("This" "is" "a" "list" "of" "strings")
```

So far about the syntax. The semantics of the different elements are described here.

`:key` The argument *string list* defines the keyword of the index entry. It must be a list of strings, since the keyword may consist of different layers such as `("heap" "fibonacci")`.

`:print` The optional *print key* defines the way the keyword has to be printed in the markup phase.

`:tkey` Another possibility to define the keys of an index entry is with the `:tkey` keyword argument. It can be used instead of the `:key` and `:print` arguments. Instead of specifying separately the key and the corresponding print key, we define the keyword by its layers. Each layer consist of a list of one or two strings. The first string will be interpreted as the main key, whereas the second one will become the print key. If the print key is ommited, the main key is taken instead. So the definition

```
:tkey (("This") ("is") ("a") ("bang" "BANG !!!"))
```

is equivalent to

```
:key   ("This" "is" "a" "bang")
:print ("This" "is" "a" "BANG !!!")
```

`:locref` The reference an index entry describes can be a *location reference* or a *cross reference*. The switch `:locref` describes a location reference. Its optional arguments are `:open-range` and `:close-range`. The *string* that must be supplied must somehow encode the location reference. It might look like the string `"25"` representing the page number 25, or `"Appendix-I"` representing the first appendix numbered in uppercase roman numerals.

`:open-range,:close-range` These are switches that do not take any arguments. They describe the beginning and ending of a *range*, starting or ending from the location reference that is given by the argument `:locref`. If they are supplied, the location reference may have influence on the way ranges are build.

`:xref` These arguments choose the second alternative. The argument *string list* of parameter `:xref` describes where the index entry should point to.

`:attr` This parameter may be used to tag a location reference with a certain attribute or it names the class of a cross reference. It may also used to associate different markup for different attributes in the markup phase. If this parameter is omitted or is the empty string, the indexentry is declared to have the attribute `default`.

Some examples:

```
(indexentry :key ("airplane") :locref "25" :attr "default")
```

defines an index entry with the key *airplane'* indexed on page *25'*. This index entry has the attribute `default`.

```
(indexentry :key ("house") :xref("building") :attr "see")
```

defines a cross reference with the key `house` pointing to the term `building`. This cross reference belongs to the cross reference class `see`.

```
(indexentry :key ("house") :xref("building") :open-range)
```

is an invalid specification, since `:open-range` mustn't be used together with cross references.

5.2 Invoking xindy

5.2.1 Command Line Options

The following command line options are accepted:

```
xindy  [-h] [-t] [-v] [-l logfile] [-o outfile]
       [-L n] [-f filterprog]
       indexstyle raw-index
```

The argument `indexstyle` names a file, containing the index style description. The argument `raw-index` names a file, containing the raw index. Both arguments are mandatory.

-h Gives a short summary of all command line options.

-l Writes helpful information into the specified `logfile`. For example, the keyword mappings are written into this file, so one can check if the intended mappings were actually performed this way.

-o Explicitly defines the name of the `output` file. If not given, the name of the `raw-index` is used with its extension changed to `.ind` (or added, if it had no extension at all).

-t Enters tracing mode of the symbolic markup tags. The format of the emitted tags can be defined with the command `markup-trace`.

-L Set the xindy logging-level to `n`.

-f Run `filterprog` on `raw-index` before reading. The program must act as a filter reading from `stdin` and writing to `stdout`. The most obvious use of this option in conjunction with TeX is to run `-f tex2xindy` on the index file prior to reading the entries into xindy.

-v Shows the version number of xindy.

Errors and warnings are reported to `stdout` and additionally to the logfile if `-l` was specified.

5.2.2 Search Path

The system uses the concept of a search path for finding the index style files and modules. The searchpath can be set with the environment variable XINDY_SEARCHPATH which must contain a list of colon-separated directories. If it ends with a colon, the built-in searchpath is added to the entire searchpath. See the command FIXME:searchpath for further details.

A Appendix

A Known Issues/Bugs

PDF output We know that currently the PDF output is not as good as it could be and work on this.

B Credits

The following people have contributed substantial parts to this documentation project or helped in some other way:

- Rolf Niepraschk
- Simon Pepping
- Bob Stayton

C About this Document

The source format of this document is **DocBook XML V4.2**.
Generation of the various output formats uses:

- **DocBook XSL Stylesheets V1.60.1**
- **Saxon V6.5.2** as XSLT processor
- **ConTeXt** and **DocBook In ConTeXt** for PDF output (still experimental)

You'll always find the newest version of this document at **http://www.miwie.org/tex-refs/**

C.1 Release News

V0.2.3 03-04-12

- Provide PDF output using **ConTeXt** and **DocBook In ConTeXt** (still experimental)
- Reedited (beautified) sections LaTeX / Commands / Counters | Cross References | Definitions | Layout | Environments | Footnotes

V0.2.2 03-01-26

- Added bzip2 compressed version of source and outfiles tarball
- Using new XSL stylesheets V1.60.1
- Eliminated more spurious Â characters in HTML output

V0.2.1 03-01-18

- Using new XSL stylesheets V1.59.2
- Eliminated spurious Â characters

V0.2.0 03-01-11

- Using new XSL stylesheets V1.58.1
- Minor changes to CSS file
- License changed to **GNU Free Documentation License**
- Added subsection Commands and Parameters to section PSTricks
- Reworked inputenc section

V0.1.3 02-10-17

- Started rework of KOMA-Script section
- Added subsection Additional PSTricks Packages
- Using new XSL stylesheets V1.56.1

V0.1.2 02-10-01

- Reworked hyperref section

V0.1.1 02-09-19

- Added template sections for PiCTeX and Texinfo
- Added subsection Release News in appendix About this Document
- Using new XSL stylesheets V1.55.0
- Added missing CSS file to outfiles tarball
- Corrected wrong FPI

V0.1.0 02-08-20

- Eliminated trailing '.' in numbered sections
- Minor markup errors corrected
- Reworked CSS file

V0.0.5 02-08-10

- Switched to DocBook XML 4.2
- Sections 'Bibindex' and 'xindy' completed

V0.0.4 02-07-25

- Section 'Makeindex' completed

V0.0.3 02-07-13

- New XSL stylesheets solve bug in creating index (no other changes)

V0.0.2 02-07-05

- New (template) sections 'fontinst', 'Bibtex', 'Makeindex', and 'xindy'
- New section 'Special Symbols'
- Added information for 'german' package
- Registered as a new project (tex-refs) on freshmeat.net

V0.0.1 02-06-21

- First official announcement

Index

l

x