Chapter 1

Things You Need to Know

The first part of this chapter presents a short overview of the philosophy and history of LaTeX 2ε. The second part focuses on the basic structures of a LaTeX document. After reading this chapter, you should have a rough knowledge of how LaTeX works, which you will need to understand the rest of this book.

1.1 A Bit of History

1.1.1 TeX

TeX is a computer program created by Donald E. Knuth [2]. It is aimed at typesetting text and mathematical formulae. Knuth started writing the TeX typesetting engine in 1977 to explore the potential of the digital printing equipment that was beginning to infiltrate the publishing industry at that time, especially in the hope that he could reverse the trend of deteriorating typographical quality that he saw affecting his own books and articles. TeX as we use it today was released in 1982, with some slight enhancements added in 1989 to better support 8-bit characters and multiple languages. TeX is renowned for being extremely stable, for running on many different kinds of computers, and for being virtually bug free. The version number of TeX is converging to π and is now at 3.14159265.

TeX is pronounced "Tech," with a "ch" as in the German word "Ach"[1] or in the Scottish "Loch." The "ch" originates from the Greek alphabet where X is the letter "ch" or "chi". TeX is also the first syllable of the Greek word technique. In an ASCII environment, TeX becomes `TeX`.

[1] In German there are actually two pronunciations for "ch" and one might assume that the soft "ch" sound from "Pech" would be a more appropriate. Asked about this, Knuth wrote in the German Wikipedia: *I do not get angry when people pronounce TeX in their favorite way ... and in Germany many use a soft ch because the X follows the vowel e, not the harder ch that follows the vowel a. In Russia, 'tex' is a very common word, pronounced 'tyekh'. But I believe the most proper pronunciation is heard in Greece, where you have the harsher ch of ach and Loch.*

1.1.2 LaTeX

LaTeX enables authors to typeset and print their work at the highest typographical quality, using a predefined, professional layout. LaTeX was originally written by Leslie Lamport [1]. It uses the TeX formatter as its typesetting engine. These days LaTeX is maintained by the LaTeX Project.

LaTeX is pronounced "Lay-tech" or "Lah-tech." If you refer to LaTeX in an ASCII environment, you type `LaTeX`. LaTeX 2_ε is pronounced "Lay-tech two e" and typed `LaTeX2e`.

1.2 Basics

1.2.1 Author, Book Designer, and Typesetter

To publish something, authors give their typed manuscript to a publishing company. One of their book designers then decides the layout of the document (column width, fonts, space before and after headings, ...). The book designer writes his instructions into the manuscript and then gives it to a typesetter, who typesets the book according to these instructions.

A human book designer tries to find out what the author had in mind while writing the manuscript. He decides on chapter headings, citations, examples, formulae, etc. based on his professional knowledge and from the contents of the manuscript.

In a LaTeX environment, LaTeX takes the role of the book designer and uses TeX as its typesetter. But LaTeX is "only" a program and therefore needs more guidance. The author has to provide additional information to describe the logical structure of his work. This information is written into the text as "LaTeX commands."

This is quite different from the WYSIWYG[2] approach that most modern word processors, such as *MS Word* or *LibreOffice*, take. With these applications, authors specify the document layout interactively while typing text into the computer. They can see on the screen how the final work will look when it is printed.

When using LaTeX it is not normally possible to see the final output while typing the text, but the final output can be previewed on the screen after processing the file with LaTeX. Then corrections can be made before actually sending the document to the printer.

1.2.2 Layout Design

Typographical design is a craft. Unskilled authors often commit serious formatting errors by assuming that book design is mostly a question of aesthetics—"If a document looks good artistically, it is well designed." But

[2] What you see is what you get.

as a document has to be read and not hung up in a picture gallery, the readability and understandability is much more important than the beautiful look of it. Examples:

- The font size and the numbering of headings have to be chosen to make the structure of chapters and sections clear to the reader.
- The line length has to be short enough not to strain the eyes of the reader, while long enough to fill the page beautifully.

With WYSIWYG systems, authors often generate aesthetically pleasing documents with very little or inconsistent structure. LaTeX prevents such formatting errors by forcing the author to declare the *logical* structure of his document. LaTeX then chooses the most suitable layout.

1.2.3 Advantages and Disadvantages

When people from the WYSIWYG world meet people who use LaTeX, they often discuss "the advantages of LaTeX over a normal word processor" or the opposite. The best thing to do when such a discussion starts is to keep a low profile, since such discussions often get out of hand. But sometimes there is no escaping ...

So here is some ammunition. The main advantages of LaTeX over normal word processors are the following:

- Professionally crafted layouts are available, which make a document really look as if "printed."
- The typesetting of mathematical formulae is supported in a convenient way.
- Users only need to learn a few easy-to-understand commands that specify the logical structure of a document. They almost never need to tinker with the actual layout of the document.
- Even complex structures such as footnotes, references, table of contents, and bibliographies can be generated easily.
- Free add-on packages exist for many typographical tasks not directly supported by basic LaTeX. For example, packages are available to include PostScript graphics or to typeset bibliographies conforming to exact standards. Many of these add-on packages are described in *The LaTeX Companion* [3].
- LaTeX encourages authors to write well-structured texts, because this is how LaTeX works—by specifying structure.

- TEX, the formatting engine of LATEX 2ε, is highly portable and free. Therefore the system runs on almost any hardware platform available.

LATEX also has some disadvantages, and I guess it's a bit difficult for me to find any sensible ones, though I am sure other people can tell you hundreds ;-)

- LATEX does not work well for people who have sold their souls ...
- Although some parameters can be adjusted within a predefined document layout, the design of a whole new layout is difficult and takes a lot of time.[3]
- It is very hard to write unstructured and disorganized documents.
- Your hamster might, despite some encouraging first steps, never be able to fully grasp the concept of Logical Markup.

1.3 LATEX Input Files

The input for LATEX is a plain text file. On Unix/Linux text files are pretty common. On windows, one would use Notepad to create a text file. It contains the text of the document, as well as the commands that tell LATEX how to typeset the text. If you are working with a LATEX IDE, it will contain a program for creating LATEX input files in text format.

1.3.1 Spaces

"Whitespace" characters, such as blank or tab, are treated uniformly as "space" by LATEX. *Several consecutive* whitespace characters are treated as *one* "space". Whitespace at the start of a line is generally ignored, and a single line break is treated as "whitespace".

An empty line between two lines of text defines the end of a paragraph. *Several* empty lines are treated the same as *one* empty line. The text below is an example. On the left hand side is the text from the input file, and on the right hand side is the formatted output.

```
It does not matter whether you
enter one or several     spaces
after a word.

An empty line starts a new
paragraph.
```

It does not matter whether you enter one or several spaces after a word.

An empty line starts a new paragraph.

[3] Rumour says that this is one of the key elements that will be addressed in the upcoming LATEX3 system.

1.3.2 Special Characters

The following symbols are reserved characters that either have a special meaning under LaTeX or are not available in all the fonts. If you enter them directly in your text, they will normally not print, but rather coerce LaTeX to do things you did not intend.

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

As you will see, these characters can be used in your documents all the same by using a prefix backslash:

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

$ % ^ & _ { } ~ \

The other symbols and many more can be printed with special commands in mathematical formulae or as accents. The backslash character \ can *not* be entered by adding another backslash in front of it (\\); this sequence is used for line breaking. Use the `\textbackslash` command instead.

1.3.3 LaTeX Commands

LaTeX commands are case sensitive, and take one of the following two formats:

- They start with a backslash \ and then have a name consisting of letters only. Command names are terminated by a space, a number or any other 'non-letter.'
- They consist of a backslash and exactly one non-letter.
- Many commands exist in a 'starred variant' where a star is appended to the command name.

LaTeX ignores whitespace after commands. If you want to get a space after a command, you have to put either an empty parameter `{}` and a blank or a special spacing command after the command name. The empty parameter `{}` stops LaTeX from eating up all the white space after the command name.

```
New \TeX users may miss whitespaces
after a command. % renders wrong
Experienced \TeX{} users are
\TeX perts, and know how to use
whitespaces. % renders correct
```

New TeXusers may miss whitespaces after a command. Experienced TeX users are TeXperts, and know how to use whitespaces.

Some commands require a parameter, which has to be given between curly braces `{ }` after the command name. Some commands take optional

parameters, which are inserted after the command name in square brackets `[ ]`.

command`[`*optional parameter*`]{`*parameter*`}`

The next examples use some LaTeX commands. Don't worry about them; they will be explained later.

```
You can \textsl{lean} on me!
```

You can *lean* on me!

```
Please, start a new line
right here!\newline
Thank you!
```

Please, start a new line right here!
Thank you!

1.3.4 Comments

When LaTeX encounters a `%` character while processing an input file, it ignores the rest of the present line, the line break, and all whitespace at the beginning of the next line.

This can be used to write notes into the input file, which will not show up in the printed version.

```
This is an % stupid
% Better: instructive <----
example: Supercal%
              ifragilist%
    icexpialidocious
```

This is an example: Supercalifragilisticexpialidocious

The `%` character can also be used to split long input lines where no whitespace or line breaks are allowed.

For longer comments you could use the `comment` environment provided by the verbatim package. Add the line `\usepackage{verbatim}` to the preamble of your document as explained below to use this command.

```
This is another
\begin{comment}
rather stupid,
but helpful
\end{comment}
example for embedding
comments in your document.
```

This is another example for embedding comments in your document.

Note that this won't work inside complex environments, like math for example.

1.4 Input File Structure

When LaTeX 2$_\varepsilon$ processes an input file, it expects it to follow a certain structure. Thus every input file must start with the command

```
\documentclass{...}
```

This specifies what sort of document you intend to write. After that, add commands to influence the style of the whole document, or load packages that add new features to the LaTeX system. To load such a package you use the command

```
\usepackage{...}
```

When all the setup work is done,[4] you start the body of the text with the command

```
\begin{document}
```

Now you enter the text mixed with some useful LaTeX commands. At the end of the document you add the

```
\end{document}
```

command, which tells LaTeX to call it a day. Anything that follows this command will be ignored by LaTeX.

Figure 1.1 shows the contents of a minimal LaTeX 2$_\varepsilon$ file. A slightly more complicated input file is given in Figure 1.2.

1.5 A Typical Command Line Session

I bet you must be dying to try out the neat small LaTeX input file shown on page 7. Here is some help: LaTeX itself comes without a GUI or fancy buttons to press. It is just a program that crunches away at your input file. Some LaTeX installations feature a graphical front-end where there is a LaTeX button to start compiling your input file. On other systems there

[4] The area between `\documentclass` and `\begin{document}` is called the *preamble*.

```
\documentclass{article}
\begin{document}
Small is beautiful.
\end{document}
```

Figure 1.1: A Minimal LaTeX File.

might be some typing involved, so here is how to coax LaTeX into compiling your input file on a text based system. Please note: this description assumes that a working LaTeX installation already sits on your computer.[5]

1. Edit/Create your LaTeX input file. This file must be plain ASCII text. On Unix all the editors will create just that. On Windows you might want to make sure that you save the file in ASCII or *Plain Text* format. When picking a name for your file, make sure it bears the extension `.tex`.

2. Open a shell or cmd window, `cd` to the directory where your input file is located and run LaTeX on your input file. If successful you will end up with a `.pdf` file. It may be necessary to run LaTeX several times to get the table of contents and all internal references right. When your input file has a bug LaTeX will tell you about it and stop processing your input file. Type `ctrl-D` to get back to the command line.

   ```
   xelatex foo.tex
   ```

[5] This is the case with most well groomed Unix Systems, and ... Real Men use Unix, so ... ;-)

```
\documentclass[a4paper,11pt]{article}
% define the title
\author{H.~Partl}
\title{Minimalism}
\begin{document}
% generates the title
\maketitle
% insert the table of contents
\tableofcontents
\section{Some Interesting Words}
Well, and here begins my lovely article.
\section{Good Bye World}
\ldots{} and here it ends.
\end{document}
```

Figure 1.2: Example of a Realistic Journal Article. Note that all the commands you see in this example will be explained later in the introduction.

1.6 The Layout of the Document

1.6.1 Document Classes

The first information LaTeX needs to know when processing an input file is the type of document the author wants to create. This is specified with the `\documentclass` command.

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

Here *class* specifies the type of document to be created. Table 1.1 lists the document classes explained in this introduction. The LaTeX 2_ε distribution provides additional classes for other documents, including letters and slides. The *options* parameter customizes the behavior of the document class. The options have to be separated by commas. The most common options for the standard document classes are listed in Table 1.2.

Example: An input file for a LaTeX document could start with the line

```
\documentclass[11pt,twoside,a4paper]{article}
```

which instructs LaTeX to typeset the document as an *article* with a base font size of *eleven points*, and to produce a layout suitable for *double sided* printing on *A4 paper*.

Table 1.1: Document Classes.

`article`	for articles in scientific journals, presentations, short reports, program documentation, invitations, ...
`proc`	a class for proceedings based on the article class.
`minimal`	is as small as it can get. It only sets a page size and a base font. It is mainly used for debugging purposes.
`report`	for longer reports containing several chapters, small books, PhD theses, ...
`book`	for real books
`slides`	for slides. The class uses big sans serif letters. You might want to consider using the Beamer class instead.

Table 1.2: Document Class Options.

`10pt`, `11pt`, `12pt` Sets the size of the main font in the document. If no option is specified, `10pt` is assumed.

`a4paper`, `letterpaper`, ... Defines the paper size. The default size is `letterpaper`. Besides that, `a5paper`, `b5paper`, `executivepaper`, and `legalpaper` can be specified.

`fleqn` Typesets displayed formulae left-aligned instead of centred.

`leqno` Places the numbering of formulae on the left hand side instead of the right.

`titlepage`, `notitlepage` Specifies whether a new page should be started after the document title or not. The `article` class does not start a new page by default, while `report` and `book` do.

`onecolumn`, `twocolumn` Instructs LaTeX to typeset the document in one column or two columns.

`twoside, oneside` Specifies whether double or single sided output should be generated. The classes `article` and `report` are single sided and the `book` class is double sided by default. Note that this option concerns the style of the document only. The option `twoside` does *not* tell the printer you use that it should actually make a two-sided printout.

`landscape` Changes the layout of the document to print in landscape mode.

`openright, openany` Makes chapters begin either only on right hand pages or on the next page available. This does not work with the `article` class, as it does not know about chapters. The `report` class by default starts chapters on the next page available and the `book` class starts them on right hand pages.

1.6.2 Packages

While writing your document, you will probably find that there are some areas where basic LaTeX cannot solve your problem. If you want to include graphics, coloured text or source code from a file into your document, you need to enhance the capabilities of LaTeX. Such enhancements are called packages. Packages are activated with the

`\usepackage[`*options*`]{`*package*`}`

command, where *package* is the name of the package and *options* is a list of keywords that trigger special features in the package. The `\usepackage` command goes into the preamble of the document. See section 1.4 for details.

Some packages come with the LaTeX 2_ε base distribution (See Table 1.3). Others are provided separately. You may find more information on the packages installed at your site in your *Local Guide* [5]. The prime source for information about LaTeX packages is *The LaTeX Companion* [3]. It contains descriptions on hundreds of packages, along with information of how to write your own extensions to LaTeX 2_ε.

Modern TeX distributions come with a large number of packages preinstalled. If you are working on a Unix system, use the command `texdoc` for accessing package documentation.

1.6.3 Page Styles

LaTeX supports three predefined header/footer combinations—so-called page styles. The *style* parameter of the

`\pagestyle{`*style*`}`

command defines which one to use. Table 1.4 lists the predefined page styles.

It is possible to change the page style of the current page with the command

`\thispagestyle{`*style*`}`

A description how to create your own headers and footers can be found in *The LaTeX Companion* [3] and in section 4.3 on page 76.

1.7 Files You Might Encounter

When you work with LaTeX you will soon find yourself in a maze of files with various extensions and probably no clue. The following list explains the various file types you might encounter when working with TeX. Please

Table 1.3: Some of the Packages Distributed with LaTeX.

doc Allows the documentation of LaTeX programs.
Described in `doc.dtx`[a] and in *The LaTeX Companion* [3].

exscale Provides scaled versions of the math extension font.
Described in `ltexscale.dtx`.

fontenc Specifies which font encoding LaTeX should use.
Described in `ltoutenc.dtx`.

ifthen Provides commands of the form
'if...then do...otherwise do....'
Described in `ifthen.dtx` and *The LaTeX Companion* [3].

latexsym To access the LaTeX symbol font, you should use the `latexsym` package. Described in `latexsym.dtx` and in *The LaTeX Companion* [3].

makeidx Provides commands for producing indexes. Described in section 4.2 and in *The LaTeX Companion* [3].

syntonly Processes a document without typesetting it.

inputenc Allows the specification of an input encoding such as ASCII, ISO Latin-1, ISO Latin-2, 437/850 IBM code pages, Apple Macintosh, Next, ANSI-Windows or user-defined one. Described in `inputenc.dtx`.

[a] This file should be installed on your system, and you should be able to get a `dvi` file by typing `latex doc.dtx` in any directory where you have write permission. The same is true for all the other files mentioned in this table.

Table 1.4: The Predefined Page Styles of LaTeX.

`plain` prints the page numbers on the bottom of the page, in the middle of the footer. This is the default page style.

`headings` prints the current chapter heading and the page number in the header on each page, while the footer remains empty. (This is the style used in this document)

`empty` sets both the header and the footer to be empty.

note that this table does not claim to be a complete list of extensions, but if you find one missing that you think is important, please drop me a line.

.tex LaTeX or TeX input file. Can be compiled with `latex`.

.sty LaTeX Macro package. Load this into your LaTeX document using the `\usepackage` command.

.dtx Documented TeX. This is the main distribution format for LaTeX style files. If you process a .dtx file you get documented macro code of the LaTeX package contained in the .dtx file.

.ins The installer for the files contained in the matching .dtx file. If you download a LaTeX package from the net, you will normally get a .dtx and a .ins file. Run LaTeX on the .ins file to unpack the .dtx file.

.cls Class files define what your document looks like. They are selected with the `\documentclass` command.

.fd Font description file telling LaTeX about new fonts.

The following files are generated when you run LaTeX on your input file:

.dvi Device Independent File. This is the main result of a classical LaTeX compile run. Look at its content with a DVI previewer program or send it to a printer with `dvips` or a similar application. If you are using pdfLaTeX then you should not see any of these files.

.log Gives a detailed account of what happened during the last compiler run.

.toc Stores all your section headers. It gets read in for the next compiler run and is used to produce the table of contents.

.lof This is like .toc but for the list of figures.

.lot And again the same for the list of tables.

.aux Another file that transports information from one compiler run to the next. Among other things, the .aux file is used to store information associated with cross-references.

.idx If your document contains an index. LaTeX stores all the words that go into the index in this file. Process this file with `makeindex`. Refer to section 4.2 on page 74 for more information on indexing.

.ind The processed .idx file, ready for inclusion into your document on the next compile cycle.

.ilg Logfile telling what `makeindex` did.

1.8 Big Projects

When working on big documents, you might want to split the input file into several parts. LaTeX has two commands that help you to do that.

`\include{`*filename*`}`

Use this command in the document body to insert the contents of another file named *filename.tex*. Note that LaTeX will start a new page before processing the material input from *filename.tex*.

The second command can be used in the preamble. It allows you to instruct LaTeX to only input some of the `\included` files.

`\includeonly{`*filename*`,`*filename*`,...}`

After this command is executed in the preamble of the document, only `\include` commands for the filenames that are listed in the argument of the `\includeonly` command will be executed.

The `\include` command starts typesetting the included text on a new page. This is helpful when you use `\includeonly`, because the page breaks will not move, even when some include files are omitted. Sometimes this might not be desirable. In this case, use the

`\input{`*filename*`}`

command. It simply includes the file specified. No flashy suits, no strings attached.

To make LaTeX quickly check your document use the syntonly package. This makes LaTeX skim through your document only checking for proper syntax and usage of the commands, but doesn't produce any (pdf) output. As LaTeX runs faster in this mode you may save yourself valuable time. Usage is very simple:

```
\usepackage{syntonly}
\syntaxonly
```

When you want to produce pages, just comment out the second line (by adding a percent sign).