The LaTeX Companion

Third Edition – Part I & Part II

This eBook is a compilation of Part I and Part II of *The LaTeX Companion*, Third Edition. To navigate to a specific page, click the links in the text or enter the part number, a hyphen, and the page number — e.g., `II-39` for page 39 in the second part.

Detailed information about the production of this eBook are in the Production Notes on page →II 983.

Cover illustration by Lonny Garris/Shutterstock
Photos of Sebastian Rahtz courtesy of the TEX Users Group

Book design by Frank Mittelbach
Typeset with LATEX in Lucida Bright at 8.47pt/11.72pt

Library of Congress Control Number: 2022947208

Part I: Print ISBN-13: 978-0-13-465894-0

Part II: Print ISBN-13: 978-0-201-36300-5

Part I+II (bundled):
Print ISBN-13: 978-0-13-816648-9

Part I+II (combined) digital:
ePub ISBN-13: 978-0-13-816652-6
uPDF ISBN-13: 978-0-13-816657-1

First digital release, 2023

A TeX Haiku

```
\expandafter\def
\csname def\endcsname
{\message{farewell}}\bye
```

SPQR
at the poetry competition

TUG conference,
Vancouver, 1999

I dedicate this edition to all my friends in the TEX world and in particular to the memory of my good friend Sebastian P. Q. Rahtz (1955–2016), with whom I spent many happy hours discussing parenting, literature, LATEX and other important aspects of life [146].

CHAPTER 1

Introduction

LaTeX is not just a system for typesetting mathematics. Its applications span one-page memoranda, business and personal letters, newsletters, articles, and books covering the whole range of the sciences and humanities ... right up to full-scale expository texts and reference works on all topics. Versions of LaTeX now exist for practically every type of computer and operating system. This book provides a wealth of information about its many present-day uses but first provides some background information.

The first section of this chapter looks back at the origins and subsequent development of LaTeX.[1] The second section gives an overview of the file types used by a typical current LaTeX system and the rôle played by each. Finally, the chapter offers some guidance on how to use the book.

1.1 A brief history (of nearly half a century)

In the Beginning ...

In May 1977, Donald Knuth of Stanford University [95] started work on the text-processing system that is now known as "TeX and METAFONT" [84–88]. In the foreword of *The TeXbook* [84], Knuth writes: "TeX [is] a new typesetting system intended for the creation of beautiful books — and especially for books that contain a lot of mathematics. By preparing a manuscript in TeX format, you are telling a computer exactly how the manuscript is to be transformed into pages whose typographic quality is comparable to that of the world's finest printers."

[1] A more personal account can be found in *The LaTeX legacy: 2.09 and all that* [176].

In 1979, Gordon Bell wrote in a foreword to an earlier book, *TEX and METAFONT, New Directions in Typesetting* [82]: “Don Knuth’s Tau Epsilon Chi (TEX) is potentially the most significant invention in typesetting in this century. It introduces a standard language in computer typography and in terms of importance could rank near the introduction of the Gutenberg press.”

In the early 1990s, Donald Knuth produced an updated version and also officially announced that TEX would not undergo any further development [96, 97] in the interest of stability. Perhaps unsurprisingly, the 1990s saw a flowering of experimental projects that extended TEX in various directions; many of these are coming to fruition in the early 21st century, making it an exciting time to be involved in automated typography.

The development of TEX from its birth as one of Don’s “personal productivity tools” (created simply to ensure the rapid completion and typographic quality of his then-current work on *The Art of Computer Programming*) [90] was largely influenced and nourished by the American Mathematical Society on behalf of U.S. research mathematicians.

… and Lamport saw that it was Good.

While Don was developing TEX, in the early 1980s, Leslie Lamport started work on the document preparation system now called LATEX, which used TEX’s typesetting engine and macro system to implement a declarative document description language based on that of a system called Scribe by Brian Reid [168]. The appeal of such a system is that a few high-level LATEX declarations, or commands, allow the user to easily compose a large range of documents without having to worry much about their typographical appearance. In principle at least, the details of the layout can be left for the document designer to specify elsewhere.

The second edition of *LATEX: A Document Preparation System* [106] begins as follows: “LATEX is a system for typesetting documents. Its first widely available version, mysteriously numbered 2.09, appeared in 1985.” This release of a stable and well-documented LATEX led directly to the rapid spread of TEX-based document processing beyond the community of North American mathematicians.

LATEX was the first widely used language for describing the logical structure of a large range of documents and hence introducing the philosophy of logical design, as used in Scribe. The central tenet of “logical design” is that the author should be concerned only with the logical content of his or her work and not its visual appearance. Back then, LATEX was described variously as “TEX for the masses” and “Scribe liberated from inflexible formatting control”. Its use spread very rapidly during the next decade. By 1994 Leslie could write, “LATEX is now extremely popular in the scientific and academic communities, and it is used extensively in industry”. But that level of ubiquity looks quite small when compared with the present day when it has become, for many professionals on every continent, a workhorse whose presence is as unremarkable and essential as the workstation on which it is used.

Going global

The worldwide availability of LATEX quickly increased international interest in TEX and in its use for typesetting a range of languages. LATEX 2.09 was (deliberately) not globalized, but it was globalizable; moreover, it came with documentation worth translating because of its clear structure and straightforward style. Two pivotal conferences (Exeter UK, 1988, and Karlsruhe Germany, 1989) established clearly the widespread adoption of LATEX in Europe and led directly to International LATEX [180] and to work led by Johannes Braams [23] on more general support for using a wide variety of languages and switching between them (see Chapter 13).

Note that in the context of typography, the word *language* does not refer exclusively to the variety of natural languages and dialects across the universe; it also has a wider meaning. For typography, “language” covers a lot more than just the choice of “characters that make up words”, as many important distinctions derive from other cultural differences that affect traditions of written communication. Thus, important typographic differences are not necessarily in line with national groupings but rather arise from different types of documents and distinct publishing communities.

The Next Generation

Another important contribution to the reach of LaTeX was the pioneering work of Frank Mittelbach and Rainer Schöpf on a complete replacement for LaTeX’s interface to font resources, the New Font Selection Scheme (NFSS) (see Chapter 9). They were also heavily involved in the production of the $\mathcal{A}\!\mathcal{M}\!\mathcal{S}$-LaTeX system that added advanced mathematical typesetting capabilities to LaTeX (see Chapter 11).

As a reward[1] for all their efforts, which included a steady stream of bug reports (and fixes) for Leslie, by 1989 Frank and Rainer “were allowed” to take over the maintenance and further development of LaTeX. One of their first acts was to consolidate International LaTeX as part of the kernel[2] of the system, “according to the standard developed in Europe”. Very soon version 2.09 was formally frozen, and although the change-log entries continued for a few months into 1992, plans for its demise as a supported system were already far advanced as something new was badly needed.

Too much of a Good Thing^{TM}

The worldwide success of LaTeX had by the early 1990s led in a sense to too much development activity: under the hood of Leslie’s “family sedan” many TeXnicians had been laboring to add such goodies as super-charged, turbo-injection, multivalved engines and much “look-no-thought” automation. Thus, the announcement in 1994 of the new standard LaTeX, christened LaTeX 2_{ε}, explains its existence in the following way:

> Over the years many extensions have been developed for LaTeX. This is, of course, a sure sign of its continuing popularity but it has had one unfortunate result: incompatible LaTeX formats came into use at different sites. Thus, to process documents from various places, a site maintainer was forced to keep LaTeX (with and without NFSS), SLITeX, $\mathcal{A}\!\mathcal{M}\!\mathcal{S}$-LaTeX, and so on. In addition, when looking at a source file it was not always clear for which format the document was written.
>
> To put an end to this unsatisfactory situation a new release of LaTeX was produced. It brings all such extensions back under a single format and thus prevents the proliferation of mutually incompatible dialects of LaTeX 2.09.

Standard LaTeX (LaTeX 2_{ε})

The development of this “New Standard LaTeX” and its maintenance system was started in 1993 by the LaTeX Project Team [148], which soon comprised the author of this book, Rainer Schöpf, Chris Rowley, Johannes Braams, Michael Downes (1958–2003), David Carlisle, Alan Jeffrey, and Denys Duchier, with some encouragement and gentle bullying from Leslie. Although the major changes to the basic LaTeX system (the kernel) and the standard document classes (styles in 2.09) were completed by

[1] Pronounced “punishment”.

[2] *Kernel* here means the core, or center, of the system.

1994, substantial extra support for colored typography, generic graphics, and fine positioning control were added later, largely by David Carlisle. Access to fonts for the new system incorporated work by Mark Purtill on extensions of NFSS to better support variable font encodings and scalable fonts [30–32].

1994 — The first edition of the LaTeX Companion

At this point in the story the first edition of the *LaTeX Companion* was written, which helped a lot in making many important packages known to a wide audience and as a side effect helped shape a standard corpus of LaTeX packages expected to be available on any installation across the world.

Towards the 21st century

Although the original goal for this LaTeX 2_ε was consolidation of the wide range of incompatible models carrying the LaTeX marquee, what emerged was a substantially more powerful system with both a robust mechanism (via LaTeX packages) for extension and, importantly, a solid technical support and maintenance system. This provides robustness via standardization and maintainability of both the code base and the support systems. The core of this system remains the current standard LaTeX system that is described in this book. It has fulfilled most of the goals for “a new LaTeX for the 21st Century”, as they were envisaged back in 1989 [151, 153].

The specific claims of the current system are “… better support for fonts, graphics and color; actively maintained by the LaTeX Project Team”. The details of how these goals were achieved, and the resulting subsystems that enabled the claims to be substantially attained, form a revealing study in distributed software support: the core work was done in at least five countries and, as is illustrated by the bugs database [108], the total number of active contributors to the technical support effort remains high.

The package system

Although the LaTeX kernel suffered a little from feature creep in the late 1990s, the package system together with the clear development guidelines and the legal framework of the LaTeX Project Public License (LPPL) [111, 132] have enabled LaTeX to remain almost completely stable while supporting a wide range of extensions. These have largely been provided by a similarly wide range of people who have, as the project team are happy to acknowledge and the online catalogue [197] bears witness, enhanced the available functionality in a vast panoply of areas.

Development work

All major developments of the base system have been listed in the regular issues of *LaTeX News* [107]. At the turn of the century, development work by the LaTeX Project Team focused on the following areas: supporting multi-language documents [130]; a “Designer Interface for LaTeX” [141]; major enhancements to the output routine [131]; improved handling of inter-paragraph formatting; and the complex front-matter requirements of journal articles. Back then prototype code had been made available (see [140]), but the work has otherwise been kept separate from LaTeX — partly because it was executing simply too slowly on the available hardware.

No new features at the kernel level …

One thing the project team steadfastly refused to do at that time was to unnecessarily “enhance” the kernel by providing additional features as part of it, thereby avoiding the trap into which LaTeX 2.09 fell in the early 1990s: the disintegration into incompatible dialects where documents written at one site could not be successfully processed at another site. In this discussion it should not be forgotten that LaTeX serves not only to produce high-quality documents but also to enable collaboration and exchange by providing a lingua franca for various research communities.

With LaTeX 2ε, documents written in 1996[1] can still be run with today's LaTeX. In the opposite direction, new documents run on older kernel releases if the additional packages used are brought up-to-date — a task that, in contrast to updating the LaTeX kernel software, is easily manageable even for users working in a multiuser environment (e.g., in a university or company setting).

... but no standstill

But a stable kernel is not identical to a standstill in software development; of equally crucial importance to the continuing relevance and popularity of LaTeX is the diverse collection of contributed packages building on this stable base. The success of the package system for nonkernel extensions is demonstrated by the enthusiasm of these contributors — many thanks to all of them! As can be easily appreciated by visiting the highly accessible and stable Comprehensive TeX Archive Network (see Appendix C) or by reading this book (where more than 250 of these "Good Guys"[2] are listed on page →II 967), this has supported the existence of an enormous treasure trove of LaTeX packages and related software.

The back office

The provision of services, tools, and systems-level support for such a highly distributed maintenance and development system was itself a major intellectual challenge, because many standard working methods and software tools for these tasks assume that your colleagues are in the next room, not the next continent (and in the early days of the development, e-mail and FTP were the only reliable means of communication). The technical inventiveness and the personalities of everyone involved were both essential to creating this example of the friendly face of open software maintenance, but Alan Jeffrey and Rainer Schöpf deserve special mention for "fixing everything".

A vital part of this system that is barely visible to most people is the regression testing system with its vast suite of test files [129]. It was initially devised and set up by Frank and Rainer with Daniel Flipo; it has proved its worth countless times in the never-ending battle of the bugs. Over the years it has seen many refinements, cumulating in a complete rewrite as part of l3build [147], which we describe in Section 17.3 on page →II 606.

2004 — The second edition of the LaTeX Companion

In 2004, i.e., roughly a decade after its first edition, the second edition of the *LaTeX Companion* was published. Due to the popularity of LaTeX 2ε and its extended features for developers, new important packages had emerged, and LaTeX had reached out into new domains. While the advice given in the first edition remained largely valid (last not least because of the long-term backward compatibility paradigm of LaTeX), we ended up rewriting 90% of the original content and added about 600 pages to account for new developments. As before, the second edition helped a lot in standardizing the use, and this way the interoperability, of LaTeX across the world.

Research

Some members of the LaTeX Project Team have built on the team's experience to extend their individual research work in document science beyond the current LaTeX structures and paradigms. Some examples of their work up to now can be found

[1]The time between 1994 and 1996 was a consolidation time for LaTeX 2ε, with major fixes and enhancements being made until the system was thoroughly stable. In fact, with some minor alterations in pagination or font usage, it is usually possible to reprocess even documents from the eighties (i.e., written for LaTeX 2.09) or make them reusable with little effort.

[2]Unfortunately, this is nearly the literal truth: you need a keen eye to spot the few ladies listed.

in the following references: [33, 35–37, 133–135, 138, 149, 175, 177]. An important spin-off from the research work was the provision of some interfaces and extensions that are immediately usable with standard LaTeX.

... and into the future

The decision to keep the core of the standard LaTeX system stable and essentially unchanging had two major advantages over any other approach to support fully automated document processing. First, the system already efficiently provided high-quality formatting of a large range of elements in very complex documents of arbitrary size. Second, it was robust in both use and maintenance and hence offered the potential to remain in widespread use for at least a further 15 years.[1] In the second edition of this book we wrote on this topic:

> As more such functionality is added, it will become necessary to assess the likelihood that merely extending LaTeX in this way will provide a more powerful, yet still robust and maintainable, system. This is not the place to speculate further about the future of LaTeX but we can be sure that it will continue to develop and to expand its areas of influence whether in traditional publishing or in electronic systems for education and commerce.

Reassessment time

This reassessment became necessary in the second decade of the new century, when it became obvious that this position was gradually getting unsustainable, because more and more areas in which people were looking for solutions could not be adequately addressed with a model of a fixed kernel and all developments outsourced to the package level. Examples are the move to Unicode in basically all operating systems and the growing pressure to produce “accessible” documents that conform to standards such as PDF/UA (Portable Document Format/Universal Accessibility).

An important policy change

Thus, in 2015, the LaTeX Project Team changed its policy and restarted kernel development. To retain the best of both worlds this was accompanied by developing a rollback/roll-forward functionality for the kernel and packages (that care to implement it). This allows a current LaTeX format to roll back to an earlier point in time in order to process old documents that rely on interfaces that have been changed since then or to process documents that explicitly worked around bugs (and so expect them to be there) that have been fixed in the meantime.

The first action of the team was to retire the fixltx2e package and instead include the accumulated fixes it contained directly in the format and to officially support LaTeX when using the Unicode engines XeTeX and LuaTeX. A big step forward happened in 2018 when LaTeX switched its default input encoding to UTF-8. This change proved that the policy change was the right thing to do and that the preparatory work (e.g., providing rollback) allows executing even major changes without disruption in its user base in order to keep LaTeX relevant and useful. A good indicator for the renewed and increased activity are the regular LaTeX newsletters [107] accompanying each release, which grew bulkier and again appeared semi-annually.

[1] One of the authors of the second edition had publicly staked a modest amount of beer on TeX remaining in general use (at least by mathematicians) until at least 2010. He should have made a larger bet, given that this is now 2022 and LaTeX is healthy and in fact growing its user base due to its many unsurpassed qualities.

And where is the mythical LaTeX3?

The event of providing the mythical LaTeX3 had long become a standing joke as "two years from 'now' — with 'now' a moving target". The reason was that the concepts and ideas for LaTeX3 have been simply a decade or more too early, and while the team implemented a fully working version already in 1990, it was simply too slow to be usable with the then available computing power. Thus, we gave up pursuing it and instead concentrated on offering LaTeX 2$_\varepsilon$, which then went public in 1994.

But ideas and concepts were never forgotten by the team, and especially its newer members (who joined in this century) pushed them back to the forefront and improved them dramatically. As a result, the code was eventually publicly made available as the expl3 package. It was then picked up by a number of enthusiastic package developers and used as the basis for their new packages. For example, if you use acro, breqn, fontspec, siunitx, unicode-math, or xparse, to name a few, you use "LaTeX3" under the hood; a recent count shows more than 200 such packages or classes as part of TeX Live.

. . . well it got merged into the kernel in 2020

So in 2019 the LaTeX Project Team made two wide-ranging decisions: there will not be a separate LaTeX3 that is being developed alongside LaTeX 2$_\varepsilon$ (as was originally planned). Instead, we will modernize the current LaTeX gradually from the inside, using the new rollback mechanism and "development" formats as a safety net to ensure that there is no disruption of service for our user base. As a first step on this journey, the L3 programming layer and the LaTeX3 document-level command declarations (formerly known as expl3 and xparse) were made an integral part of LaTeX on February 2, 2020. Thus, more or less exactly 30 years after its conception, LaTeX3 became a reality for every LaTeX user — even though few will have immediately noticed.

The foundation layer for modernization

The importance of this step is that it allows the team to modernize other parts of the kernel and develop new functionality entirely based on the L3 programming layer, which offers many features not available with legacy LaTeX programming constructs. For example, the new Hook Management System for LaTeX, which is a cornerstone for modernizing and transforming the existing LaTeX, is entirely written using the new L3 programming layer, and other parts will follow suit.

Today's challenge: structured and accessible output is needed

As already mentioned, there is a steadily increasing interest in the production of "tagged" PDF documents that are "accessible", in the sense that they contain information to assist screen reading software, etc., and, more formally, that they adhere to the PDF/UA (Portable Document Format/Universal Accessibility) standard [190], explained further in [47]. In many disciplines this is starting to become a requirement when applying for grants or when publishing results.

At the moment, all methods of producing such "accessible PDFs", including the use of LaTeX, require extensive manual labor in preparing the source or in post-processing the PDF (maybe even at both stages); and these labors often have to be repeated after making even minimal changes to the (LaTeX or other) source. This is a huge pity, because LaTeX should in theory be well positioned to do this work automatically, given that its source is already well-structured.

The production of tagged (i.e., structured) PDF documents is not only important in order to comply to accessibility standards. It also opens possibilities to reuse data from such PDFs, because it allows other applications to correctly identify the structure inside the output document and this way extract or manipulate parts of the content — workflows that become increasingly important in the digital world.

The LaTeX Project Team has for some years been well aware that these new usages are not adequately supported by the current system architecture of LaTeX 2ε and that major work in this area is therefore urgently needed to ensure that LaTeX remains an important and relevant document source format. However, the amount of work required to make such major changes to the LaTeX system architecture is enormous and definitely way beyond the limited resources of a small team of volunteers working in their spare time (or maybe just about possible, but only given a very long — and most likely too long — period of time).

A multi-year project to shape the future of LaTeX

At the TeX Users Group conference 2019 in Palo Alto the team's previously pessimistic outlook on this subject became cautiously optimistic, because of discussions with senior executives from Adobe about the possibility of producing structured PDF from LaTeX source without the need for the usual requirement of considerable manual post-processing. As a result of these discussions, towards the end of 2019 the team produced an extended feasibility study for the project, aimed primarily at Adobe engineers and decision-makers. This study [144] describes in some detail the various tasks that constitute the project and their interdependencies. It also contains a project plan covering how, and in what order, these tasks should be tackled both to achieve the final goal and, at the same time, to provide intermediate concrete results that are relevant to user communities (both LaTeX and PDF); these intermediate results will help in obtaining feedback that is essential to the successful completion of later tasks.

This multi-year project found the approval of Adobe, which then committed to financially and otherwise supporting this endeavor [150]. Unfortunately — thanks to the COVID-19 pandemic — the start got delayed, but since the end of 2020, this exciting project is now well under way. First results from this project that are already in existence (such as the new hook management system and the alignment of the hyperref package with the LaTeX kernel) are already described in this book. Other parts are obviously still vaporware at this point. Fortunately, none is expected to render any documentation or suggestion made in this book obsolete — after all, the project goal is to enable tagging of existing documents, simply by reprocessing with minor configuration changes as outlined in the "Spoiler alert" Section 2.1.1 on page 23.

1.2 Today's systems

When we wrote the second edition of *The LaTeX Companion* (i.e., 2003–2004), standard LaTeX was (officially) supported only on 8-bit engines, e.g., pdfTeX. Around the same time, the first version of the Unicode engine XeTeX and (somewhat later, in 2007) the first beta version of LuaTeX appeared, and there were soon unofficial support files that helped people running LaTeX on these Unicode engines as well.

When LuaTeX reached version 1.0, the LaTeX Project Team used the opportunity and officially took on LaTeX support for all three engines that included, for example, running the release regression test suite with its roughly 1000 tests against all three engines. Besides these three engines (which are covered in this book), there are further ones, such as pTeX and upTeX for Japanese, where the LaTeX adjustments for the engine are maintained by the respective user groups.

CHAPTER 2

The Structure of a LaTeX Document

One of the ideas behind LaTeX is the separation between layout and structure (as far as possible), which allows the user to concentrate on content rather than having to worry about layout issues [106]. This chapter explains how this general principle is implemented in LaTeX.

The first section of this chapter shows how document class files, packages, options, and preamble commands can affect the structure and layout of a document.

The logical subdivisions of a document are then discussed in general, before explaining in more detail how sectioning commands and their arguments define a hierarchical structure, how they generate numbers for titles, and how they produce running heads and feet. This is followed by discussing a few useful packages that allow you to customize different aspects of the layout of sectional units or to provide your own definitions.

In Section 2.3 we take a closer look at the design of table of contents structures and how it can be influenced or extended.

This is followed by a section discussing important packages that support you in providing cross-references that remain correct, even if you change parts of your document. These packages can automatically insert appropriate phrases (varioref, cleveref, nameref), can help you manage your label keys (showkeys and refcheck), or support you in providing references to external documents (xr) or hyperlinks in general (hyperref).

The final section introduces packages and programs that support you in archiving documents or managing them when you work jointly with others on some document.

2.1 The overall structure of a source file

You can use LaTeX for several purposes, such as writing an article or a book or producing presentations. Clearly, documents for different purposes may need different logical structures, i.e., different commands and environments. We say that a document belongs to a *class* of documents having the same general structure (but not necessarily the same typographical appearance). You specify the class to which your document belongs by starting your LaTeX file with a `\documentclass` command, where the mandatory parameter specifies the *name* of the *document class*. The document class defines the available logical commands and environments (for example, `\chapter` in the report class) as well as a default formatting for those elements. An optional argument allows you to modify the formatting of those elements by supplying a list of *class options*. For example, `11pt` is an option recognized by most document classes that instructs LaTeX to choose eleven point as the basic document type size.

Many LaTeX commands described in this book are not specific to a single class but can be used with several classes. A collection of such commands is called a *package*, and you inform LaTeX about your use of certain packages in the document by placing one or more `\usepackage` commands after `\documentclass`.

Just like the `\documentclass` declaration, `\usepackage` has a mandatory argument consisting of the *name* of the package and an optional argument that can contain a list of *package options* that modify the behavior of the package.[1]

The document classes and the packages reside in external files with the extensions `.cls` and `.sty`, respectively. Code for options is sometimes stored in external files (in the case of class files with the extension `.clo`) but is normally directly specified in the class or package file (see Appendix A for information on declaring options in classes and packages). However, in the case of options, the file name can differ from the option name. For example, the option `11pt` is related to `size11.clo` when used in the article class and to `bk11.clo` inside the book class.

The document preamble

Commands placed between `\documentclass` and `\begin{document}` are in the so-called *document preamble*. All style parameters must be defined in this preamble, either in package or class files or directly in the document *before* the `\begin{document}` command, which sets the values for some of the global parameters. A typical document preamble could look similar to the following:

```
\documentclass[twocolumn,a4paper]{article}
\usepackage{multicol}
\usepackage[ngerman,french]{babel}
\addtolength\textheight{3\baselineskip}
\begin{document}
```

This document preamble defines that the class of the document is article and that the layout is influenced by the formatting request `twocolumn` (typeset in two columns) and the option `a4paper` (print on A4 paper). The first `\usepackage` declaration

[1] These commands also have a second optional argument that is intended for cases where a specific release of a package or a document class is required. This is discussed in Section 2.5.5 on page 114.

informs LaTeX that this document contains commands and structures provided by the package multicol. In addition, the babel package with the options `ngerman` (support for German language) and `french` (support for French language) is loaded. Finally, the default height of the text body was enlarged by three lines for this document.

Generally, nonstandard LaTeX package files contain modifications, extensions, or improvements[1] with respect to standard LaTeX, while commands in the preamble define changes for the current document. Thus, to modify the layout of a document, you have several possibilities:

- Change the standard settings for parameters in a class file with options defined for that class.
- Add one or more packages to your document and make use of them.
- Change the standard settings for parameters in a package file with options defined for that package.
- Write your own local packages containing special parameter settings and load them with `\usepackage` after the package or class they are supposed to modify (as explained in the next section).
- Make final adjustments inside the preamble.

If you want to get deeper into LaTeX's internals, you can, of course, define your own general-purpose packages that can be manipulated with options. You find additional information on this topic in Appendix A.

2.1.1 Spoiler alert — The `\DocumentMetadata` command

When LaTeX changed from LaTeX 2.09 to LaTeX 2_ε around 1994, the overall document structure was slightly changed to automatically distinguish old from new documents (to switch to compatibility mode, if necessary). LaTeX 2_ε documents start with `\documentclass` as described above, while LaTeX 2.09 documents started with the command `\documentstyle`, and `\usepackage` was unavailable.

Now, roughly a quarter century later, there is another major shift under way during which LaTeX is being modernized to support accessible PDF/UA (Portable Document Format/Universal Accessibility) and other functionality that is important for it to remain useful; see the discussion in Section 1.1 on page 7. This time around, the functionality change is essentially upward compatible, and old documents can be easily reprocessed using the new features. Thus, instead of dividing documents into two classes (old and new) by changing the first command, you can now indicate that you want to use the new functionality by adding a `\DocumentMetadata` declaration in front of `\documentclass` while leaving the rest of the document unchanged.

[1]Many of these packages have become de facto standards and are described in this book. This does not mean, however, that packages that are not described here are necessarily less important or useful, of inferior quality, or should not be used. We merely concentrated on a few of the more established ones; for others, we chose to explain what functionality is possible in a given area.

```
\DocumentMetadata{key/value list}
```

This declaration should be the first command in a document; i.e., if present, it should come before `\documentclass`. It expects a *key/value list* as its argument in which you specify "metadata" about the document that guides the production of the final output, e.g., should it adhere to a certain standard, should it be a tagged PDF, what is its author, title, and keywords that are shown in the metadata of the resulting PDF, etc. All these "metadata" are stored so that packages and users can access the data in a consistent way.

For example, the key `pdfversion` allows you to set the PDF version. With the key `pdfstandard` it is possible to require a standard such as `A-2b`. If that is specified, it directs LaTeX to embed an appropriate color profile and set up verification tests that packages like hyperref can use to suppress actions not allowed in this standard. A further example is the `backend` key that allows you to specify a backend, e.g., `dvipdfmx` or `dvisvg`, which is useful in cases where the correct backend cannot be detected automatically.

At the time of writing this book the details about which other keys are going to be supported are still open (the whole exercise is a multi-year project [150] after all), but what we can say is that already now you can use this future interface to enable some new functionality. For example, just adding

```
\DocumentMetadata{}
\documentclass{article}         % (or any other class)
...                             % with preamble as previously
\begin{document}
```

is enough to load the new support code for managing PDF output, and this enables packages, such as hyperref, to provide features otherwise not available; see Section 2.4.6 on page 96 for details.

2.1.2 Processing of options of the document class and packages

You can think of options to the document class or to packages as a simple way to adjust some of the properties of the whole document (when used in `\documentclass`) or of properties of individual packages (if specified in `\usepackage`). More fine-grain control is usually also possible through declarations and setup commands that are defined by a class or package file and are available for use once that file is loaded.

You can specify options in a `\usepackage` command only if these options are explicitly declared by the package. Otherwise, you receive an error message, informing you that your specified option is unknown to the package in question. Options to the `\documentclass` are handled slightly differently. If a specified option is not declared by the class, it is assumed to be a "global option".

All options given to `\documentclass` (whether declared or global) are automatically passed as class options to all `\usepackage` declarations. Thus, if a package file loaded with a `\usepackage` declaration recognizes (i.e., declares) some of the class options, it can take appropriate actions. If not, the class options are ignored while processing that package. Because all options have to be defined inside the class or package file, their actions are under the control of the class or package (an action

CHAPTER 3

Basic Formatting Tools — Paragraph Level

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

In this chapter we look at such tools, starting at the micro level; larger structures are covered in Chapter 4. The first section covers different aspects of paragraph formatting, such as producing large initial letters at the start of a paragraph, modifying paragraph justification, altering the vertical spacing between lines of a paragraph, and similar topics. This is followed by a look at handling special characters such as ellipses, dashes, underscores, or spaces.

In the third section we discuss generated or specially formatted text, i.e., counter values represented as ordinals or cardinals, fractions formatted for use in running text, and in particular the acro package for consistently managing acronyms and abbreviations. A special focus is given to scientific notation provided by the siunitx package, which forms the last and rather lengthy topic of this section.

The fourth section then covers various way of highlighting and quoting text. This includes a number of generally useful packages as well as some more specialized ones that are occasionally useful.

Section 3.5 deals with the different kind of "notes", such as footnotes, marginal notes, and endnotes, and explains how they can be customized to conform to different styles, if necessary. In the final section we take a quick look at different helper packages for document development, e.g., how to add different kind of notes, copy-editing marks, or change bars to your documents.

3.1 Shaping your paragraphs

Paragraph justification in TEX and LATEX

For formatting paragraphs LATEX deploys the algorithms already built into the TEX program, which by default produce justified paragraphs. In other words, spaces between words are slightly stretched or shortened to produce lines of equal length. TEX achieves this outcome with an algorithm that attempts to find an optimal solution for a whole paragraph, using the current settings of about 20 internal parameters. They include aspects such as trying to produce visually compatible lines, such that a tight line is not followed by one very loosely typeset, or considering several hyphens in a row as a sign of bad quality. The interactions between these parameters are very subtle, and even experts find it difficult to predict the results when tweaking them. Because the standard settings are suitable for nearly all applications, we describe only some of the parameters in this book. Appendix B.4.3 discusses how to trace the algorithm. If you are interested in delving further into the matter of automatic paragraph breaking, refer to *The TEXbook* [84, chap. 14], which describes the algorithm in great detail, or to the very interesting article by Michael Plass and Donald Knuth on the subject, which is reprinted in *Digital Typography* [99].

Downside of global optimization

The downside of the global optimizing approach of TEX, which you will encounter sooner or later, is that making small changes, like correcting a typo near the end of a paragraph, can have drastic and surprising effects, as it might affect the line breaking of the whole paragraph. It is possible, and not even unlikely, that, for example, the *removal* of a word might actually result in making a paragraph one line *longer*.

This behavior can be very annoying if you are near the end of an important project (like the third edition of this book) and a correction wreaks havoc on your already manually adjusted page breaks. In such a situation it is best to place `\linebreak` or `\pagebreak` commands into strategic places to force TEX to choose a solution that it would normally consider inferior. To be able to later get rid of such manual corrections you can easily define your own commands, such as

```
\newcommand\CElinebreak{\linebreak}
```

rather than using the standard LATEX commands directly. This helps you to distinguish the layout adjustments for a particular version from other usages of the original commands — a method successfully used in the preparation of this book.

Interword spacing

The interword spacing in a justified paragraph (the white space between individual words) is controlled by several TEX parameters — the most important ones are `\tolerance` and `\emergencystretch`. By setting them suitably for your document you can prevent most or all of the "Overfull box" messages without any manual line breaks. The `\tolerance` command is a means for setting how much the interword space in a paragraph is allowed to diverge from its optimum value.[1] This command is a TEX (not LATEX) counter, and therefore it has an uncommon assignment syntax — for example, `\tolerance=500`. Lower values make TEX try harder to stay near the optimum; higher values allow for loose typesetting. The default value is often 200. When TEX is unable to stay in the given tolerance, you will find overfull boxes in your output (i.e., lines sticking out into the margin like this). Enlarging the value of `\tolerance` means that TEX also considers poorer but hopefully still acceptable line breaks, instead of turning the problem over to you for manual intervention. Sensible values are between 50 and 9999. Do not use 10000 or higher, because that allows TEX to produce a single arbitrarily bad line (like   this   one) to keep the rest of the paragraph perfect. If you really need fully automated line breaking, it is better to set the length parameter `\emergencystretch` to a positive value. If TEX cannot break a paragraph without producing overfull boxes (due to the setting of `\tolerance`) and `\emergencystretch` is positive, it adds this length as stretchable space to every line, thereby accepting line-breaking solutions that have been rejected before. You may get some underfull box messages because all the lines are now set in a loose measure, but this result will still look better than a single horrible line in the middle of an otherwise perfectly typeset paragraph.

Careful with TEX's idea about infinitely bad

LATEX has two predefined commands influencing the above parameters: `\fussy`, which is the default, and `\sloppy`, which allows for relatively bad lines. The `\sloppy` command is automatically applied by LATEX in some situations (e.g., when typesetting `\marginpar` arguments or `p` columns in a `tabular` environment) where perfect line breaking is seldom possible due to the narrow measure. It uses a `\tolerance` of 9999 together with an `\emergencystretch` of 3 em.

Unjustified text

While the theory on producing high-quality justified text is well understood (even though surprisingly few typesetting systems other than TEX use algorithms that can produce high quality other than by chance), the same cannot be said for the situation when unjustified text is being requested. This may sound strange at first hearing. After all, why should it be difficult to break a paragraph into lines of different length? The answer lies in the fact that we do not have quantifiable quality measures that allow us to easily determine whether a certain breaking is good or bad. In comparison to its

[1] The optimum is font defined; see Section 9.8.1 on page 745.

work with justified text, TeX does a very poor job when asked to produce unjustified paragraphs. Thus, to obtain the highest quality we have to be prepared to help TeX far more often by adding explicit line breaks in strategic places. A good introduction to the problems in this area is given in an article by Paul Stiff (1949–2011) [183].

The main type of unjustified text is the one in which lines are set flush left but are unjustified at the right. For this arrangement LaTeX offers the environment `flushleft`. It typesets all text in its scope "flush left" by adding very stretchable white space at the right of each line; that is, it sets the internal parameter `\rightskip` to `0pt plus 1fil`. This setting often produces very ragged-looking paragraphs because it makes all lines equally good independent of the amount of text they contain. In addition, hyphenation is essentially disabled because a hyphen adds to the "badness" of a line and, because there is nothing to counteract it, TeX's paragraph-breaking algorithm normally chooses line breaks that avoid hyphenated words.

"The LaTeX document preparation system is
a special version of Donald Knuth's TeX
program. TeX is a sophisticated program
designed to produce high-quality
typesetting, especially for mathematical
text."

```
\begin{flushleft}
  ‘‘The \LaTeX{} document preparation system is
  a special version of Donald Knuth’s \TeX{}
  program. \TeX{} is a sophisticated program
  designed to produce high-quality typesetting,
  especially for mathematical text.’’
\end{flushleft}
```

3-1-1

In summary, LaTeX's `flushleft` environment is not particularly well suited to continuous unjustified text, which should vary at the right-hand boundary only to a certain extent and where appropriate should use hyphenation (see `ragged2e` in the next section for alternatives). Nevertheless, it can be useful to place individual objects, like a graphic, flush left to the margin, especially because this environment adds space above and below itself in the same way as list environments do.

Another important restriction is the fact that the settings chosen by this environment have no universal effect, because some environments (e.g., `minipage` or `tabular`) and commands (e.g., `\parbox`, `\footnote`, and `\caption`) restore the alignment of paragraphs to full justification. That is, they set the `\rightskip` length parameter to `0pt` and thus cancel the stretchable space at the right line endings. A way to automatically deal with this problem is provided by the package `ragged2e`.

Other ways of typesetting paragraphs are flush right and centered, with the `flushright` and `center` environments, respectively. In these cases the line breaks are usually indicated with the `\\` command, whereas for ragged-right text (the `flushleft` environment discussed above) you can let LaTeX do the line breaking itself (if you are happy with the resulting quality).

The three environments discussed in this section work by changing declarations that control how TeX typesets paragraphs. These declarations are also available as LaTeX commands, as shown in the following table of correspondence:

environment:	`center`	`flushleft`	`flushright`
command:	`\centering`	`\raggedright`	`\raggedleft`

CHAPTER 4

Basic Formatting Tools — Larger Structures

While the previous chapter was concerned with micro-typography, this chapter now looks at commands and environments for formatting larger chunks of text.

Typesetting lists is the subject of the first part. We start with a discussion of the various parameters and commands controlling the standard LaTeX lists, `enumerate`, `itemize`, and `description`, followed by a brief look at LaTeX's generic list capabilities. Then, the important enumitem package is discussed, which we recommend as a basis for many documents. The production of horizontally oriented lists is covered by the tasks package, the concept of "headed lists" is exemplified with the amsthm and thmtools packages and typed-checklist, helps you write and maintain check lists of various kinds. Together these should satisfy the structure and layout requirements of most readers.

The second part then explains how to simulate "verbatim" text. In particular, we take a detailed look at the powerful packages fancyvrb and listings.

The third part presents packages that deal with line numbering (lineno); handling of columns, such as parallel text in two columns (paracol); or solving the problem of producing multiple columns with multicol.

At the end we take a brief look at packages for generating sample texts. They are useful for testing layouts or for reporting bugs when you are asked to produce a so-called Minimal Working Example (MWE) to show your problem.

4.1 Lists

Lists are very important LaTeX constructs and are used to build many of LaTeX's display-like environments. LaTeX's three standard list environments and the generic list environment are discussed in the first two sections, where we also show how they can be customized.

Section 4.1.3 starting on page 261 provides an in-depth discussion of the package enumitem, which introduces a number of new list structures and offers comprehensive methods to customize them, as well as the standard lists.

It is followed by a discussion of "headed lists", such as theorems and exercises. Finally, Sections 4.1.6 and 4.1.7 cover two packages for tasks and check lists.

4.1.1 Using and modifying the standard lists

It is relatively easy to customize the three standard LaTeX list environments `itemize`, `enumerate`, and `description`, and the next three sections look at each of these environments in turn. Changes to the default definitions of these environments either can be made globally by redefining certain list-defining parameters in the document preamble or can be kept local.

Customizing the `itemize` list environment

For a simple unnumbered `itemize` list, the labels are defined by the commands shown in Table 4.1 on the facing page. To create a list with different-looking labels, you can redefine the label-generating command(s). You can make that change local for one list, as in the example below, or you can make it global by putting the redefinition in the document preamble. The following simple list is a standard `itemize` list with a marker from the PostScript Zapf Dingbats font (see Section 10.13.1 on page →II 113) for the first-level labels:

```
\usepackage{pifont}
\newenvironment{myitemize}
   {\renewcommand\labelitemi{\ding{43}}\begin{itemize}}
   {\end{itemize}}
\begin{myitemize}
\item Text of the first item in the list.
\item Text of the first paragraph in the second
      item of the list.

      The second paragraph of the item.
\item Text of the third item.
\end{myitemize}
```

☞ Text of the first item in the list.

☞ Text of the first paragraph in the second item of the list.

The second paragraph of the item.

☞ Text of the third item.

4-1-1

The `\labelitemfont` command (which defaults to `\normalfont`) is intended to alter the font for all labels in one go. This is especially useful if the body font for the document is altered and its symbols are less suitable to be used as labels; see also the discussion on page 697.

	Command	Default Definition	Representation
First Level	`\labelitemi`	`\labelitemfont\textbullet`	•
Second Level	`\labelitemii`	`\labelitemfont\bfseries \textendash`	–
Third Level	`\labelitemiii`	`\labelitemfont\textasteriskcentered`	∗
Fourth Level	`\labelitemiv`	`\labelitemfont\textperiodcentered`	·

Table 4.1: Commands controlling an `itemize` list environment

Customizing the `enumerate` list environment

LaTeX's enumerated (numbered) list environment `enumerate` is characterized by the commands and representation forms shown in Table 4.2 on the next page. The first row shows the names of the counter used for numbering the four possible levels of the list. The second and third rows are the commands giving the representation of the counters and their default definition in the standard LaTeX class files. Rows four, five, and six contain the commands, the default definition, and an example of the actual enumeration string printed by the list.

A reference to a numbered list element is constructed using the `\theenumi`, `\theenumii`, and similar commands, prefixed by the internal commands `\p@enumi`, `\p@enumii`, etc., respectively. The last three rows in Table 4.2 on the following page show these commands, their default definition, and an example of the representation of such references. It is important to consider the definitions of both the representation and reference-building commands to get the references correct.

We can now create several kinds of numbered description lists simply by applying what we have just learned.

Our first example redefines the first- and second-level counters to use capital roman digits and Latin characters. The visual representation should be the value of the counter followed by a dot, so we can use the default value from Table 4.2 on the next page for `\labelenumi`.

I. **Introduction**

A. **Applications**
Motivation for research

B. **Organization**
Structure of the report

II. **Literature Survey**

4-1-2 q1=I q2=IA q3=IB q4=II

```
\renewcommand\theenumi    {\Roman{enumi}}
\renewcommand\theenumii   {\Alph{enumii}}
\renewcommand\labelenumii {\theenumii.}
\begin{enumerate}
  \item \textbf{Introduction}      \label{q1}
  \begin{enumerate}
    \item \textbf{Applications}    \label{q2} \\
      Motivation for research
    \item \textbf{Organization}    \label{q3} \\
      Structure of the report
  \end{enumerate}
  \item \textbf{Literature Survey} \label{q4}
\end{enumerate}
q1=\ref{q1} q2=\ref{q2} q3=\ref{q3} q4=\ref{q4}
```

	First Level	*Second Level*	*Third Level*	*Fourth Level*
Counter	`enumi`	`enumii`	`enumiii`	`enumiv`
Representation	`\theenumi`	`\theenumii`	`\theenumiii`	`\theenumiv`
Default Definition	`\arabic{enumi}`	`\alph{enumii}`	`\roman{enumiii}`	`\Alph{enumiv}`
Label Field	`\labelenumi`	`\labelenumii`	`\labelenumiii`	`\labelenumiv`
Default Form	`\theenumi.`	`(\theenumii)`	`\theenumiii.`	`\theenumiv.`
Numbering Example	1., 2.	(a), (b)	i., ii.	A., B.
		Reference representation		
Prefix	`\p@enumi`	`\p@enumii`	`\p@enumiii`	`\p@enumiv`
Default Definition	`{}`	`\theenumi`	`\theenumi(\theenumii)`	`\p@enumiii\theenumiii`
Reference Example	1, 2	1a, 2b	1(a)i, 2(b)ii	1(a)iA, 2(b)iiB

Table 4.2: Commands controlling an `enumerate` list environment

After these redefinitions we get funny-looking references; to correct this we have to adjust the definition of the prefix command `\p@enumii`. For example, to get a reference like "I–A" instead of "IA" as in the previous example, we could do

```
\makeatletter \renewcommand\p@enumii{\theenumi--} \makeatother
```

because the reference is typeset by executing `\p@enumii` followed by `\theenumii`. To simplify this LaTeX offers the command `\labelformat` (see page 77) to alter the representation, i.e.,

```
\labelformat{enumii}{\theenumi--#1}
```

to achieve the same effect. You can also decorate an `enumerate` field by adding something to the label field and its reference representation. In the example below, we have chosen for the first-level list elements the section sign (§) as a prefix and a period as a suffix (omitted in references).

§1. item of list

§2. item of list

w1=§1 w2=§2

```
\renewcommand\labelenumi{\S\theenumi.} \labelformat{enumi}{\S#1}
\begin{enumerate}
 \item \label{w1} item of list \item \label{w2} item of list
\end{enumerate}
w1=\ref{w1}  w2=\ref{w2}
```

4-1-3

You might even want to select different markers for consecutive labels. For instance, in the following example, characters from the PostScript font ZapfDingbats are used. In this case there is no straightforward way to automatically make the `\ref` commands produce the correct references. Instead of `\theenumi` simply producing the representation of the `enumi` counter, we define it to calculate from the counter value which symbol to select.

CHAPTER 5

The Layout of the Page

In this chapter we see how to specify different page layouts. Often a single document requires several different page layouts. For instance, the layout of the first page of a chapter, which carries the chapter title, is generally different from that of the other pages in that chapter.

We first introduce LaTeX's dimensional parameters that influence the page layout and describe ways to change them and visualize their values. This is followed by an in-depth discussion of the packages typearea and geometry, both of which provide sophisticated ways to implement page layout specifications.

The third section deals with the LaTeX concepts used to provide data for running headers and footers. This section includes a description of the new mark mechanism introduced in LaTeX in 2022.

This is followed by a section that explains how to format such elements, including many examples deploying the fancyhdr package and others.

The fifth section then introduces commands that help in situations when the text does not fit into the layout and manual intervention is required. This includes advice and tools for manual pagination, which is sometimes necessary to avoid widows and orphans, avoid excessive white space, etc.

The chapter concludes with a brief look at two generic classes that go a long way toward providing almost full control over the page layout specification process.

5.1 Geometrical dimensions of the layout

The text of a document usually occupies a rectangular area on the paper — the so-called *type area* or *body*. Above the text there might be a *running header* and below it a *running footer*. They can consist of one or more lines containing the page number; information about the current chapter, section, time, and date; and possibly other markers. If they are visually heavy and closely tied to the text, then these elements are considered part of the type area; this is often the case for running headers, especially when underlined. Otherwise, they are considered to belong to the top or bottom *margins*. This distinction is important when interpreting size specifications.

The fields to the left and the right of the body are also called *margins*. Usually they are left blank, but small pieces of text such as remarks or annotations — so-called *marginal notes* — can appear there.

In general one talks about the *inner* and *outer* margins. For two-sided printing, inner refers to the middle margins — that is, the left margin on recto (odd-numbered) pages and the right margin on verso (even-numbered) ones. For one-sided printing, inner always indicates the left margin. In a book spread, odd-numbered pages are those on the right-hand side.[1]

The size, shape, and position of these fields and margins on the output medium (paper or screen) and the contents of the running headers and footers are collectively called a *page layout*.

The standard LaTeX document classes allow document formatting for recto–verso (*two-sided*) printing. Two-sided layouts can be either asymmetrical or symmetrical (the LaTeX default). In the latter case the type areas of recto and verso pages are positioned in such a way that they overlap if one holds a sheet to the light. Also, marginal notes are usually swapped between left/right pages.

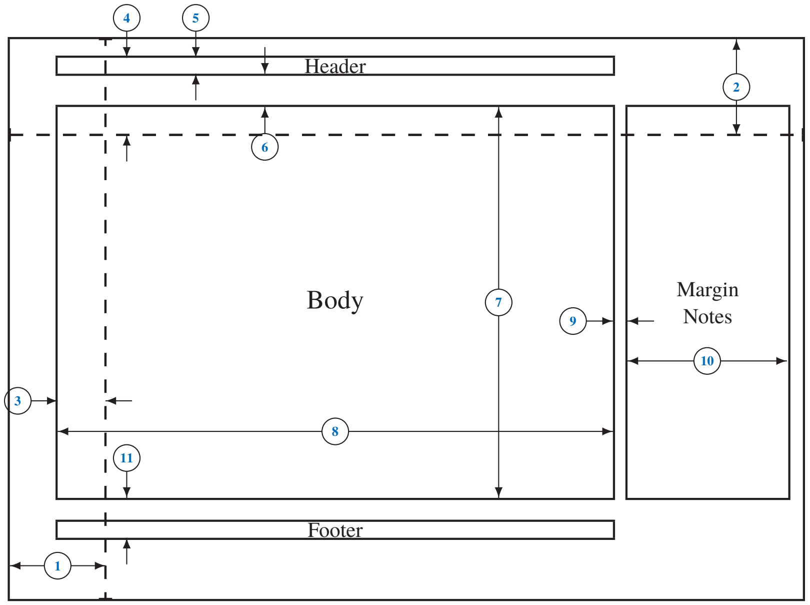
The dimensional parameters controlling the page layout are described and shown schematically in Figure 5.1 on the facing page.[2] The default values of these parameters depend on the paper size. To ease the adjustments necessary to print on different paper sizes, the LaTeX class files support a number of options that set those parameters to the physical size of the requested paper as well as adjust the other parameters (e.g., `\textheight`) that depend on them.

Table 5.1 on page 368 shows the paper size options known to standard LaTeX classes together with the corresponding page dimensions. Table 5.2 on page 369 presents the page layout parameter values for the `letterpaper` paper size option, the default when no explicit option is selected. They are identical for the three standard LaTeX document classes (article, book, and report). If a different paper size option is selected, the values may change. Thus, to print on A4 paper, you can simply specify `\documentclass[a4paper]{article}`.

Additional or different options may be available for other classes. Nevertheless, there seems to be little point in providing, say, an `a0paper` option for the book class that would produce incredibly wide text lines.

[1]This assumes left-to-right typesetting, e.g., for Arabic or Hebrew the situation is reversed.

[2]The graphical presentation was produced with the layout package, described in Section 5.2.1, and shows a landscape design produced with the geometry package and the options `landscape`, `a4paper`, and `hmarginratio=1:4`.



1	`one inch + \hoffset`	2	`one inch + \voffset`
3	`\oddsidemargin = -36pt`	4	`\topmargin = -58pt`
5	`\headheight = 12pt`	6	`\headsep = 25pt`
7	`\textheight = 294pt`	8	`\textwidth = 418pt`
9	`\marginparsep = 11pt`	10	`\marginparwidth = 121pt`
11	`\footskip = 30pt`		`\marginparpush = 5pt (not shown)`
	`\hoffset = 0pt`		`\voffset = 0pt`
	`\paperwidth = 597pt`		`\paperheight = 421pt`

5-1-1

The dashed lines represent the reference point for TEX (`\hoffset` + 1 inch) and (`\voffset` + 1 inch) from the top and left of the page.

`\paperheight` Height of the paper to print on.

`\paperwidth` Width of the paper to print on.

`\textheight` Height of the body (without header and footer); typically a multiple of `\baselineskip` plus `\topskip` such that a page containing only text perfectly fills the body area.

`\textwidth` Width of the body.

`\columnsep` Width of space between columns of text in multicolumn mode.

`\columnseprule` Width of a vertical line separating the two adjacent columns in multicolumn output (default 0 pt, i.e., no visible rule).

`\columnwidth` Width of a single column in multicolumn mode. Calculated by LATEX from `\textwidth` and `\columnsep` as appropriate.

`\linewidth` Width of the current text line. Usually equals `\columnwidth` but might get different values in environments that change the margins.

`\evensidemargin` For two-sided printing, the extra space added at the left of even-numbered pages.

`\oddsidemargin` For two-sided printing, the extra space added at the left of odd-numbered pages; otherwise the extra space added at the left of all pages.

`\footskip` Vertical distance separating the baseline of the last line of text and the baseline of the footer.

`\headheight` Height of the header.

`\headsep` Vertical separation between header and body.

`\topmargin` Extra vertical space added at the top of the header.

`\marginparpush` Minimal vertical space between two successive marginal notes (not shown in the figure).

`\marginparsep` Horizontal space between body and marginal notes.

`\marginparwidth` Width of marginal notes.

Figure 5.1: Page layout parameters and visualization

`letterpaper`	$8\,^1\!/_2 \times 11$	inches		
`legalpaper`	$8\,^1\!/_2 \times 14$	inches		
`executivepaper`	$7\,^1\!/_4 \times 10\,^1\!/_2$	inches		
`a4paper`	$\approx 8\,^1\!/_4 \times 11\,^3\!/_4$	inches	210×297	mm
`a5paper`	$\approx 5\,^7\!/_8 \times 8\,^1\!/_4$	inches	148×210	mm
`b5paper`	$\approx 7 \times 9\,^7\!/_8$	inches	176×250	mm

Table 5.1: Standard paper size options in LaTeX

Most of the layout parameters in LaTeX class files are specified in terms of the physical page size. Thus, they automatically change when `\paperwidth` or `\paperheight` is modified via one of the paper size options. Changing these two parameters in the preamble of your document does not have this effect, because by then the values for the other parameters are already calculated.

One-inch default margins

Standard-conforming Device Independent File Format (DVI) drivers place the reference point for TeX one inch down and to the right of the upper-left corner of the paper. These one-inch offsets are called *driver margins*.[1] The reference point can be shifted by redefining the lengths `\hoffset` and `\voffset`. By default, their values are zero. In general, the values of these parameters should never be changed. They provide, however, a convenient way to shift the complete page image (body, header, footer, and marginal notes) on the output plane without disturbing the layout. The driver margins are inherited from TeX and are not needed in LaTeX's parameterization of the page layout. A change to `\topmargin` shifts the complete text vertically, while changes to `\oddsidemargin` and `\evensidemargin` shift it horizontally.

To make sure that the reference point is properly positioned, you can run the test file `testpage.tex` (by Leslie Lamport, reimplemented by Rainer Schöpf) through LaTeX and the DVI driver in question. The resulting output page shows the position of the reference point with respect to the edges of the paper.

5.2 Changing the layout

Change parameters only in the preamble

When you want to redefine the value of one or more page layout parameters, the `\setlength` and `\addtolength` commands should be used. It is important to keep in mind that changes to the geometrical page layout parameters should be made only in class or package files and/or in the preamble (i.e., before the `\begin{document}` command). Although changing them mid-document is not absolutely impossible, it most likely produces havoc, due to the inner workings of TeX, which involve a number of subtle dependencies and timing problems. For example, if you change the `\textwidth`, you might find that the running header of the previous page is changed.

[1]These one-inch offsets are the reason why some of the values in Figure 5.1 are negative.

CHAPTER 6

Tabular Material

Data is often most efficiently presented in tabular form. TeX uses powerful primitives for arranging material in rows and columns. Because they implement only a low-level, formatting-oriented functionality, several macro packages have been developed that build on those primitives to provide a higher-level command language and a more user-friendly interface. In Standard LaTeX, two types of environments for constructing tables are provided. Most commonly the `tabular` environment or its math-mode equivalent, the `array` environment, is used. However, in some circumstances the `tabbing` environment might prove useful.

Tables contained within floating environments

Tables typically form large units of the document that must be allowed to "float" so that the document may be paginated correctly. The environments described in this chapter are principally concerned with the table layout. To achieve correct pagination they are often used within the `table` environment described in Chapter 7. Exceptions are the environments for multipage tables described in Section 6.4, which should never be used in conjunction with the LaTeX float mechanism. Be careful, however, not to confuse the `tabular` environment with the `table` environment. The former allows material to be aligned in columns, while the latter is a logical document element identifying its contents as belonging together and allowing the material to be floated jointly. In particular, one `table` environment can contain several `tabular` environments.

After taking a quick look at the `tabbing` environment, this chapter describes the extensions to LaTeX's basic `tabular` and `array` environments provided by the array package. This package offers increased functionality, especially in terms of a more flexible positioning of paragraph material, a better control of inter-column and inter-row spacing, and the possibility of defining new preamble specifiers. Several packages build on the primitives provided by the array package to provide specific extra functionality. By combining the features in these packages, you are able to construct complex tables in a simple way. For example, the tabularx and tabulary packages provide extra column types that allow table column widths to be calculated automatically.

Standard LaTeX tabular environments do not produce tables that may be broken over a page. We give several examples of multipage tables using the `supertabular`, `longtable`, and `xltabular` environments provided by the similarly named packages. We then briefly look at the use of color in tables and at several packages that give finer control over rules, and the spacing around rules, in tables. Next, we discuss table entries spanning multiple rows, created via the multirow package, and packages that provide new column specifiers for special occasions, such as dcolumn and fcolumn, which provide mechanisms for aligning columns of figures on a decimal point.

We also discuss the use of footnotes in tables. The threeparttable package provides a convenient mechanism to have table notes and captions combined with a tabular layout.

The final section discusses the very interesting keyvaltable package that offers a completely different approach to inputting table data. In many cases it is superior to the traditional methods. It makes use of the other packages, e.g., xltabular, as its backend and so understanding their concepts and mechanisms is nevertheless useful.

Mathematically oriented readers should consult the chapter on advanced mathematics, especially Section 11.2 on page →II 131, which discusses the alignment structures for equations.

6.1 Standard LaTeX environments

Standard LaTeX has two families of environments that allow material to be lined up in columns — namely, the `tabbing` environment and the `tabular` and `array` environments. The main differences between the two kinds of environments are:

- The `tabbing` environment is not as general as the `tabular` environment. It can be typeset only as a separate paragraph, whereas a `tabular` environment can be placed anywhere in the text or inside mathematics.
- The `tabbing` environment can be broken between pages, whereas the standard `tabular` environment cannot.
- With the `tabbing` environment the user must specify the position of each tab

stop explicitly. With the `tabular` environment LaTeX can automatically determine the width of the columns.

- Multiple `tabbing` environments cannot be nested, whereas `tabular` environments can, thus allowing complex alignments to be realized.

6.1.1 Using the `tabbing` environment

This section deals with some of the lesser-known features of the `tabbing` environment. First, it must be realized that formatting is under the complete control of the user. Somewhat unexpectedly, when moving to a given tab stop, you always end up at the exact horizontal position where it was defined, independently of where the current point is. As a consequence, the current point can move backward and overwrite previous text. The scope of declarations within rows is usually limited to the region between tab stops, e.g., `\bfseries` and `\itshape` in the next example stop at the next `\>` or `\\`, respectively.

Alternative names for accent commands

Be aware that the usual LaTeX commands for making accents, `\'`, `` \` ``, and `\=`, are redefined inside the `tabbing` environment. The accents are available by typing `\a'`, `` \a` ``, and `\a=` instead (or by using accented characters directly). The `\-` command, which normally signals a possible hyphenation point, is also redefined, but this consideration is not so important because the lines in a `tabbing` environment are never broken.

If the command `\'` is used between two tab stops, then all text to the left of it is placed into the *previous* tab region and typeset flush right against the previous tab stop (only separated by a distance of `\tabbingsep`). The default value for `\tabbingsep` is set equal to `\labelsep`, which in turn is usually 5 pt. To set text flush right to the right margin you can use `` \` ``. The effect of both commands is shown below in the next example.

There exist a few common ways to define tab stops — that is, using a line to be typeset or explicitly specifying a skip to the next tab stop. The `\kill` command may be used to terminate a line that is only used to set tab stops: the line itself is not typeset. The following example demonstrates this and shows the redefinition of tab stops on the fourth line.

```
one                    two      three  four
one                    two

one        again one   éè       three      flushed right

new tab pos.  two               same pos./////// overprint
one           two               three  four
```

6-1-1

```
\begin{tabbing}
First Real Tab Stop \= Second \= Third \=\kill
one \> two \> three \> four            \\
\bfseries one \> \itshape two          \\[5mm]
one \> again one \' \a'{e}\a`{e}
          \> three   \` flushed right \\[5mm]
new tab pos. \= two  \> same pos.
               \> //////   overprint \\
one \> two \> three \> four            \\
\end{tabbing}
```

If you use the above accent commands within the definition of a command that may be used inside a tabbing environment, you must use the `\a...` forms because the standard accent commands such as `\'` are interpreted as `tabbing` commands, as shown below. Fortunately, LaTeX now accepts UTF-8 input so that you can use the accented letter directly, instead of entering them as 7-bit input.

```
Tab one Tab two Tab three
7-bitcaf e (bad)  café
UTF-8   café      or café
```

```
\newcommand\badcafe{caf\'e (bad)}
\newcommand\goodcafe{caf\a'e}        \newcommand\greatcafe{café}
\begin{tabbing} Tab one \= Tab two \= Tab three   \\
                7-bit \> \badcafe   \> \goodcafe  \\
                UTF-8 \> \greatcafe \> or café    \end{tabbing}
```

6-1-2

The `tabbing` environment is most useful for aligning information into columns whose widths are constant and known. The following example is from Table A.1 on page →II 652:

```
pc    Pica = 12pt          └─┘
cc    Cicero = 12dd        └─┘
cm    Centimeter = 10mm    └─────┘
```

```
\newcommand\lenrule[1]{\makebox[#1]{%
  \rule{.4pt}{4pt}\hrulefill\rule{.4pt}{4pt}}}
\begin{tabbing}
  dd\quad \= \hspace{.55\linewidth} \= \kill
  pc   \> Pica = 12pt        \> \lenrule{1pc} \\
  cc   \> Cicero   = 12dd    \> \lenrule{1cc} \\
  cm   \> Centimeter = 10mm  \> \lenrule{1cm} \\
\end{tabbing}
```

6-1-3

6.1.2 tabto — An alternative way to tab stops

As an alternative to the standard `tabbing` environment Donald Arseneau developed the **tabto** package that offers a set of commands for moving to tab positions that can be used within normal text, list environments, etc., without the need to be confined in a special environment.

```
\tabto{length}      \tabto*{length}
```

The `\tabto` command moves to a position *length* away from the left margin (where list indentations count as part of the margin). If the text on the current line is already past the desired position, the command starts a new line and then moves to the right point. In contrast, `\tabto*` always stays in the current line and performs backspacing if we are past the desired target point; that is, it may result in overprinting of text.

In the next example "T1" and "T2" are placed at tab positions reachable in the current line, while both "T3" and "T4" jump to the next line because in each case we are already past the target position. Note that "T5" ends up before "T4" due to the fact that we used `\tabto*` here.

```
   Lorem ipsum dolor sit amet, consectetuer
adipiscing elit.         T1      T2
                         T3 Ut purus elit, vestibu-
lum ut, placerat ac, adipiscing vitae, felis.
                   T5            T4
```

```
\usepackage{lipsum,tabto}
\lipsum[1][1] \tabto{3cm}T1 \tabto{4cm}T2
\tabto{3cm}T3 \lipsum[1][2] \tabto{4cm}T4
\tabto*{2cm}T5
```

6-1-4

CHAPTER 7

Mastering Floats

Documents would be easier to read if all the material that belonged together was never split between pages. However, this is often technically impossible, and TeX, by default, splits textual material between two pages to avoid partially filled pages. Nevertheless, when this outcome is not desired (as with figures and tables), the material must be "floated" to a convenient place, such as the bottom or the top of the current or next page, to prevent half-empty pages.

This chapter shows how "large chunks" of material can be kept conveniently on the same or a nearby page by using a float object. We begin by introducing the general concepts through which LaTeX handles float objects and the parameters that define how LaTeX typesets its basic `figure` and `table` float environments and describe some of the packages that make it easy to control float placement (Section 7.2).

We then continue by explaining how you can define and use your own floating environments (Section 7.3.1) or, conversely, how captioning commands can be used to enter information into the list of figures and tables for nonfloating material (Section 7.3.2). Then methods for rotating the content of a float are described (Section 7.3.3). It is often visually pleasing to include a "picture" inside a paragraph, with the text wrapping around it. Some package authors have tried their hand on this difficult topic, and in Section 7.3.4 we look at one of them in some detail.

The fourth section addresses the problem of customizing captions. There is a recognized need to be able to typeset the description of the contents of figures and tables in many different ways. This includes specifying subfigures and subtables, each with its own caption and label, inside a larger float.

In the final section of this chapter we then look at two recent packages that offer a modern key/value approach to the float topic and attempt to bridge the gaps between the different packages introduced in the earlier sections. While both attempt to be fairly comprehensive, they have a different focus, which is why we describe both to enable you to make an educated decision based on the requirements of the job.

Many float-related packages have been developed over the years, unfortunately often with incompatible concepts and syntax, so there is no point discussing them all here. In fact, the packages that we describe often feature quite a few more commands than we are able to illustrate. Our aim is to enable you to make an educated choice and to show how a certain function can be obtained in a given framework. In each case consulting the original documentation will introduce you to the full possibilities of a given package.

7.1 An overview of LaTeX's float concepts

Floats are often problematic in the present version of LaTeX, because the system was developed at a time when documents contained considerably less graphical material than they do today. Placing floats (tables and figures) works relatively well as long as the space they occupy is not too large compared with the space taken up by the text. If a lot of floating material is present, however, then it is often the case that all material from a certain point onward floats to the end of the chapter or document unless you make some adjustments on the level of individual floats.

You can also try to fine-tune the float style parameters for a given document but often a combination of both configuration and local adjustments is necessary to achieve the desired typesetting results.

The present section explains the background and behavior of LaTeX's float algorithm and explains where and how you can manipulate it to serve your needs. We start with defining the terminology and then discuss the algorithm and its configuration possibilities. The section finishes with a look at the consequences resulting from the design choices made in the algorithm and how they affect the processing of your documents.

7.1.1 LaTeX float terminology

In this section we define the terminology used throughout the present chapter.

Float classes

Each float in LaTeX belongs to a class often referred to as its type. By default, LaTeX knows about two classes, viz., *figure* and *table*. Further classes can be added by a document class or by packages; this is discussed in Section 7.3.1 on page 529. The class a float belongs to influences certain aspects of the float positioning, such as its default placement specification (if not overridden on the float itself).

One important property of the float placement algorithm is that LaTeX never violates the order of placement within a class of floats. For example, if you have

Figure 1, Table 1, Figure 2 in a document, then Figure 1 is always placed before Figure 2. However, Table 1 (belonging to a different float class) is placed independently and hence can appear before, after, or between the figures.

Some of the extension packages discussed in Section 7.3 make no provisions to ensure this basic property, and one has to visually watch out for possible misorderings. Where this can happen, it is explicitly remarked upon in the text.

Float areas

LaTeX knows about two float areas within a column where it can place floats: the top area and the bottom area of the column. In two-column layout, it also knows about a top area spanning the two columns. There is no bottom area for page-wide floats in two-column mode.

In addition, LaTeX can make float columns and float pages, i.e., columns or pages that contain only floats. Finally, LaTeX can place floats in-line into the text (but only if so directed on the individual float).

Float placement specifiers

To direct a float to be placed into one of these areas, a float placement specifier can be provided as an optional argument to the float. If no such optional argument is given, then a default placement specifier is used (which depends on the float class as mentioned above but usually allows the float to be placed in all areas if not subject to other restrictions).

A float placement specifier can consist of the following characters in *any* order:

`!` indicates that some of the restrictions that normally apply should be ignored (discussed later)

`h` indicates that the float is allowed to be placed in-line ("here")

`t` indicates that the float is allowed to go into a top area

`b` indicates that the float is allowed to go into a bottom area

`p` indicates that the float is allowed to go on a float page or column area

The order in which these characters are put in the optional argument does *not* influence how the algorithm tries to place the float! The precise order is discussed on page 509. This is one of the common misunderstandings, for instance when people think that `bt` means that the bottom area should be tried first. Also note that if a letter is not present, then the corresponding area is not tried at all.

Float algorithm parameters

There are about 20 parameters that influence the placement; for a detailed discussion see page 510. They define

- how many floats can go into a certain area,
- how big an area can become,

- how much text there has to be on a page (in other words, how much the top and bottom areas can occupy) and
- how much space is inserted
  - between consecutive floats in an area, and
  - between the area and the text above or below.

Float reference points (aka call-outs)

A point in the document that references the float (e.g., “see Figure X”) is called a “call-out”, and the float body should be placed close to the (main) call-out, because its placement in the document affects the placement of the float in the output and determines when LaTeX sees the float for the first time. It is important to understand that if a float is placed in the middle of a paragraph, the reference point for the algorithm is the next line break, or page break, in the paragraph that follows the actual placement in the source.

For technical and practical reasons it is usually best to place all floats between paragraphs (i.e., after the paragraph with the call-out), even if that makes the call-out and reference point slightly disagree.

7.1.2 Basic behavioral rules of LaTeX’s float mechanism

With this knowledge, we are now ready to delve into the algorithm’s behavior. First we have to understand that all of LaTeX’s typesetting algorithms are designed to avoid any sort of backtracking. This means that LaTeX reads through the document source, formats what it finds, and (more or less) immediately typesets it. The reasons for this design choice were to limit complexity (which is still quite high) and also to maintain reasonable speed (remember that this is from the early eighties).

For floats, this means that the algorithm is greedy; i.e., the moment it encounters a float, it immediately tries to place it and, if it succeeds, it never changes its decision. This means that it may choose a solution that could be deemed inferior in light of data received later.

For example, if a figure is allowed to go to the top or bottom area, LaTeX may decide to place this figure in the top area. If this figure is followed by two tables that are allowed to go only to the top, these tables may not fit anymore. A solution that could have worked in this case (but was not tried) would have been to place the figure in the bottom area and the two tables in the top area.

The basic sequence when placing floats

Here is the basic sequence the algorithm runs through:

- If a float is encountered in the source document, LaTeX attempts to place it immediately according to its rules (detailed later);
- if this succeeds, the float is placed, and that decision is never changed;

CHAPTER 8

Graphics Generation and Manipulation

TEX probably has the best algorithms for formatting paragraphs and building pages from them. But in this era of ever-increasing information exchange, most publications do not limit themselves to text — the importance of graphical material has grown tremendously. TEX by itself does not address this area, as it deals only with positioning (glyph) boxes on pages. Knuth, however, provided a hook for implementing "features" that are not available in the basic language, via the `\special` command. The latter command does not affect the output page being formatted,[1] but TEX will put the material, specified as an argument to the `\special` command, literally at the current point in the output, e.g., the `.dvi` file. The program that displays or prints that output then has to interpret the received information and load the graphic or execute the appropriate graphic operation. Engines that can directly produce Portable Document Format (PDF) offer other primitives to add raw data to the PDF output, but conceptually the process is the same: it is the backend that has to deal with the graphics.

The main problem with this approach was that different backends provide different methods and interfaces so that any TEX source file that used `\special` directly became nonportable, working only with specific output devices or printers.

[1] In certain situations the `\special` command may change the formatting because it can produce an additional breakpoint or generate an extra space — thus not a perfect approach.

This problem was largely resolved with LaTeX 2ε, which provided a generalized driver-independent interface to include external graphic material and to scale and rotate LaTeX boxes.[1] This interface is the subject of Section 8.1. It exists in the form of two implementations: the graphics package offers a simple interface, while the graphicx package provides a convenient key/value interface with additional features. Free-standing scaling and rotation and other manipulations are the subjects of Section 8.2.

A similar abstraction for line graphics was announced at the same time, but it took more than a decade until it was finally implemented. This and some applications thereof are discussed in Section 8.3. This section also talks about special graphic languages and as one useful application thereof generating QR codes with LaTeX.

Section 8.4 is then devoted to tcolorbox, a very comprehensive package for producing boxed material.

We conclude the chapter by taking a brief look at the powerful tikz package. There is no way we can do justice to this package, which builds a universe of its own — one could easily devote a whole book to it. Thus, all we try here is to give you an introduction to its basics and give a few teasers to show what it is capable of, but direct you to its documentation (more than a thousand pages) if you get hooked.

8.1 LaTeX's image loading support

Since the introduction of LaTeX 2ε in 1994, LaTeX has offered a uniform syntax for including every kind of graphics file that can be handled by the different drivers. In addition, all kinds of graphic operations (such as resizing and rotating) as well as color support are available.

These features are not part of the LaTeX kernel but rather are provided by the standard, fully supported color, graphics, and graphicx extension packages. As the TeX program does not have any direct methods for graphic manipulation, the packages have to rely on features supplied by the "driver" used to print the Device Independent File Format (DVI) file or by the features of the extended engines pdfTeX, XeTeX, or LuaTeX that can produce PDF output directly. Unfortunately, not all drivers or PDF engines support the same features, and even the internal method of accessing these extensions varies among them. Consequently, all of these packages take options such as `dvips` or `pdftex` to specify which external driver or engine is being used. Through this method, any unavoidable device-dependent information is localized in a single place, the preamble of the document. However, each TeX distribution when installed sets up suitable detection methods and defaults (e.g., if the program pdflatex is used, then most likely the correct driver option is `pdftex`), so in practice most documents compile correctly without the need to specify any such option.

The packages graphics and graphicx can both be used to scale, rotate, and reflect LaTeX material or to include graphics files prepared with other programs. The difference between the two is that graphics uses a combination of macros with a "standard" or TeX-like syntax, while the "extended" or "enhanced" graphicx package presents a key/

[1] A generalized package for color is also available; see the *LaTeX Manual* [106] for more details.

value type of interface for specifying optional parameters to the `\includegraphics` and `\rotatebox` commands.

8.1.1 Options for graphics and graphicx

When using LaTeX's graphics packages, the necessary space for the typeset material after performing a file inclusion or applying some geometric transformation is reserved on the output page. It is, however, the task of the *device driver* (e.g., dvips, dvipdfmx, pdftex, etc.) to perform the actual inclusion or transformation in question and to show the correct result. Because different drivers require different code to carry out an action like rotation, one has to specify the target driver as an option to the graphics packages — for example, option `dvips` if you use one of the graphics packages with Tom Rokicki's dvips program, or option `pdftex` if the graphics packages are used with the pdfTeX engine.

Fortunately, however, the package is usually capable of figuring out the correct driver automatically, in particular when generating a PDF with pdfTeX, XeTeX, or LuaTeX. In all other cases it assumes dvips, which is often, but not always, correct. Only in the latter case does one need to explicitly provide a driver option. The full list of supported drivers (and their restrictions, if any) is listed in the graphics package documentation if you ever run into a case where the default does not work.

In addition to the driver options, the packages support some options controlling which features are enabled (or disabled):

`draft` Suppress all "special" features, such as including external graphics files in the final output. The layout of the page is not affected, because LaTeX still reads the size information concerning the bounding box of the external material. This option is of particular interest when a document is under development and you do not want to download the (often huge) graphics files each time you work on it. When `draft` mode is activated, the picture is replaced by a box of the correct size containing the name of the external file.

`final` The opposite of `draft`. This option can be useful when, for instance, "draft" mode was specified as a global option with the `\documentclass` command (e.g., for showing overfull boxes), but you do not want to suppress the graphics as well.

`hiresbb` In PostScript graphics look for bounding box comments that are of the form `%%HiResBoundingBox` (which typically have real values) instead of the standard `%%BoundingBox` (which must have integer values).

Of lesser importance these days are options that disable features because of restrictions in the output driver. With `hiderotate` you prevent showing rotated material (for instance, when the previewer cannot rotate material and produces error messages), and `hidescale` does the same for scaled material.

With the graphicx package, these options are also available locally as keys for individual `\includegraphics` commands.

8.1.2 The `\includegraphics` syntax in the `graphics` package

With the **graphics** package, an image file can be included by using the following command:

```
\includegraphics*[llx,lly][urx,ury]{file}
```

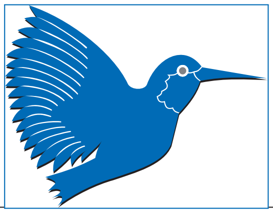
If the `[urx,ury]` argument is present, it specifies the coordinates of upper-right corner of the image as a pair of TEX dimensions. The default units are big (PostScript) points; thus, `[1in,1in]` and `[72,72]` are equivalent. If only one optional argument is given, the lower-left corner of the image is assumed to be located at `[0,0]`. Otherwise, `[llx,lly]` specifies the coordinates of that point. Without optional arguments, the size of the graphic is determined by reading the external *file* (containing the graphics itself or a description thereof; see below).

The starred form of the `\includegraphics` command "clips" the graphics image to the size of the specified bounding box. In the normal form (without the `*`), any part of the image that falls outside the specified bounding box overprints the surrounding text.

The examples in the current and next sections all use a small graphic showing the LATEX hummingbird logo. It is a small PDF graphic with a width of 100 pt and a height of 80 pt. Being a typical `.pdf` graphic, the lower-left coordinate of its bounding box is at `0 0`, and the top-right corner is at `[98,76]` in big points (bp), which means it is slightly wider than high.

In the examples we always embed the `\includegraphics` command in an `\fbox` (with a blue frame and zero `\fboxsep`) to show the space that LATEX reserves for the included image. In addition, the baseline is indicated by the horizontal rules produced by the `\HR` command, defined as an abbreviation for `\rule{1em}{0.4pt}`.

The first example shows the inclusion of the `latex-logo.pdf` graphic at its natural size. We have omitted the file extension, but the graphics is found nevertheless by searching for the file, adding supported extensions one after another until a match is made.[1]

LR

```
\usepackage{graphics,color}
\newcommand\HR{\rule{1em}{0.4pt}}
\newcommand\bluefbox[1]
     {\textcolor{blue}{\setlength\fboxsep{0pt}%
                         \fbox{\textcolor{black}{#1}}}}

L\HR\bluefbox{\includegraphics{latex-logo}}\HR R
```

8-1-1

Next, we specify a box that corresponds to a part of the picture (and an area outside it) so that some parts fall outside its boundaries, overlaying the material

[1] The advantage is that your document may work with different device drivers supporting distinct graphics formats, if you supply both formats together with your document. The disadvantage is that you make LATEX work harder in trying to find a suitable graphics file.

CHAPTER 9

Font Selection and Encodings

In this chapter we describe the font selection and encoding models of LaTeX, known as NFSS (New Font Selection Scheme) and the packages and commands to deal with encoding issues. Given that NFSS was first introduced about thirty years ago, it is, of course, no longer really new. The last decade, however, showed a number of exciting new developments, so this chapter, compared to the one in the previous edition of the book, nevertheless contains a lot of new and changed material.

We start with a short introduction to the history of font usage in different TeX engines. This is then followed by a section discussing font characteristics in general and introducing the major attributes used in LaTeX for orthogonal font switching. We then describe the use of the high-level interface — that is, the commands a user normally has to deal with. This includes commands used in normal text (Section 9.3), special features for use in mathematical formulae (Section 9.4), and an overview of basic support packages for NFSS — those being distributed together with LaTeX (Section 9.5).

Section 9.6 is solely devoted to the details of the fontspec package, the frontend to NFSS for use in the Unicode engines XeTeX and LuaTeX.

The final sections then describe the low-level interfaces that are useful when defining complex new commands and that are important when new fonts are to be made available in LaTeX. Here you find low-level commands for changing individual font attributes (Section 9.7), commands for setting up new fonts with LaTeX (Section 9.8), and a discussion of LaTeX's encoding models for text and math (Section 9.9).

9.1 Introduction

9.1.1 The history of LaTeX's font selection scheme (NFSS)

When TeX was developed in 1979, only a dozen fonts were set up for use with the program. This situation had not greatly changed five years later, when LaTeX was first released. It was thus quite natural that LaTeX's font selection scheme followed the plain TeX concept with the addition of size-changing commands that allowed typesetting in ten predefined sizes.

As a result, LaTeX's font selection was far from general because the font commands were not changing font attributes but exchanged one font with another one. For instance, when, say, the now obsolete command `\bf` was used inside emphasized text, the result was not a bold italic font, as normally desired, but rather a plain roman bold font. Furthermore, the original release had another major drawback: the correspondence tables between font commands and fonts were hardwired into LaTeX so that replacing the default fonts was a difficult, if not impossible, task.

Since that time low-priced laser printers became available, and simultaneously a large number of font families from PostScript and other font formats appeared. But, unfortunately, because of the LaTeX font selection model, there was no easy and standard method for integrating these new fonts into LaTeX — typesetting with LaTeX meant typesetting in Computer Modern on almost all installations. Of course, individual fonts could be loaded, but they would not work with the size commands nor were they otherwise integrated, so it was extremely complicated to typeset a whole document in a different font family.

This unsatisfactory situation was finally resolved in 1989 with the release of the New Font Selection Scheme (NFSS) [152] written by Frank Mittelbach and Rainer Schöpf, which became widely known after it was successfully used in $\mathcal{A}\mathcal{M}\mathcal{S}$-LaTeX (see Chapter 11). This system contains a generic concept for varying font attributes individually and for integrating new font families easily into an existing LaTeX system. The concept is based on five attributes that can be defined independently to access different fonts, font characteristics, or font families. To implement it, some of the LaTeX commands were redefined, and some new commands were added.

In 1994 NFSS became the official standard in the then newly released LaTeX 2_ε. It has now been in worldwide use for more than thirty years, proving the code to be stable, successful, and most importantly flexible enough to support further advances in font technology and use.

However, in the last decade new requirements appeared that were not easily supported with the original NFSS design and so some extensions in the form of additional support packages appeared.

For pdfTeX two important ones have been mweights by Bob Tennent and fontaxes by Andreas Bühmann and Michael Ummels, both extending the attribute handling of NFSS. Neither package is intended for direct usage but normally used within font support packages, such as those discussed in Chapter 10. In 2020 a large portion of their functionality has been integrated into NFSS so that it is now generally available.

The situation for Unicode engines is different. With the fontspec package by Will Robertson a new frontend for font loading is offered for use in XeTeX and LuaTeX. This frontend allows one to access all kind of font features available with OpenType or TrueType fonts. It then generates the necessary information for NFSS on the fly — thus under the hood NFSS is still the font selection machinery. The package is discussed in detail in Section 9.6.

9.1.2 Input and output handling in TeX systems over the years

When TeX was originally developed in 1979, computers used 7-bits to encode input characters, which allows for a direct representation of a maximum of 128 characters, i.e., Latin upper and lowercase base characters, digits, and a few symbols. Any other character had to be represented in the form of a command or a sequence of commands; e.g., `\ss` for ß or `\^\j` for ȷ̂, etc.

TeX79 largely restricted to English

This certainly allowed inputting text in most Latin-based languages (though already a bit inconvenient) where the need for using commands in the input is still infrequent. Already then there were problems; e.g., hyphenation would not work for words containing diacritics. However, that approach clearly failed for languages with a totally different set of characters, where essentially every character would be represented by a command if we assume that the input can represent only ASCII characters directly. For example, the Russian text for "on the next page" (`на следующей странице`) would have to be written as

```
\cyrn\cyra\ \cyrs\cyrl\cyre\cyrd\cyru\cyryu\cyrshch\cyre\cyrishrt
\ \cyrs\cyrt\cyrr\cyra\cyrn\cyri\cyrc\cyre
```

Clearly, no one wants to type text like this on a regular basis.

Fonts in TeX79 used 7-bits as well, i.e., contained a maximum of 128 glyphs and for text fonts the glyph positions matched the ASCII code used at input. In other words, there was a one-to-one correspondence between the number encoding a character on input and the number representing the glyph position in the fonts. For example, the seven bits 1000001 (decimal 65) in a file were interpreted by an editor as the character `A` and when processed by TeX the program fetched the glyph in position 65 in the current font, which then happened to be "A". For the class of documents that just need basic Latin characters this worked well and was very fast.

The 8-bit days

The 8-bit operating systems

Computers then evolved and started to use 8-bits for encoding data. With it TeX evolved too and supported 8-bit input and fonts (TeX82 and later TeX3.0). However, there was now a problem: different computers used different "code pages"; that is, they interpreted the bytes in input files in different ways depending on the operating system's regional settings or other criteria. Thus, while the lower 128 bytes were usually identical, in many code pages the upper half varied greatly, making data exchange difficult, if not outright impossible.

There was nothing inside a file that would explicitly tell how to interpret the 8-bit numbers inside. Thus, suddenly files written on one computer got partially scrambled when processed on a different one unless you knew (and could tell the processing software) under which assumption — that is, under which code page — the file was written.

In LaTeX 2_ε this problem was resolved by introducing the package `inputenc` that allows one to explicitly state under which input encoding the remainder of the file should be interpreted by LaTeX. It effectively added the missing information about the file encoding to the source so that LaTeX could use it even when running under an operating system that would normally use a different code page.

The input encoding concept

The `inputenc` package works by interpreting the 8-bit numbers present in the file (representing the characters) and mapping them to an "internal LaTeX representation", which uniquely (albeit on a somewhat ad hoc basis) covers all characters representable in LaTeX. For further processing, such as writing to some auxiliary file, LaTeX exclusively uses this internal representation, thereby avoiding any possible misinterpretation.

And there was (and unfortunately is) a second problem: TeX still assumes by default that input characters can be directly mapped to corresponding font slots. As a consequence, developing 8-bit fonts that use all possible slots would require different fonts for different code pages. For example, if your keyboard contained a key for the Euro symbol (€) and your computer used the Windows code page `cp-1252`, then pressing this key generated the code `80` while it generated `164` if your computer happened to use the code page `ISO-889-15`. Thus, if such codes define the glyph position in the fonts, then we need different fonts in each case, one with the Euro symbol in slot 80 and another where it is in 164.

So the `inputenc` package solves half of the problem, because once applied it correctly interprets the bytes in the input file and LaTeX then sees the command `\texteuro` in both cases. But then LaTeX needs to know which slot `\texteuro` has to access to fetch the glyph (and possibly even from which font).

The output encoding concept

All this is achieved in LaTeX through the concept of output encodings, which map the LaTeX internal character representations to appropriate glyph positions or to glyph-building actions depending on the actual glyphs available in the font used for typesetting. This is where the `fontenc` package comes into play that was introduced at the same time as `inputenc`. Each font usable with LaTeX is supposed to implement a specific "named" font encoding, and with `fontenc` you tell LaTeX where to find glyphs in such font encodings.

The Unicode days

The next level of evolution was the introduction of Unicode-based operating systems. They use UTF-8, which is a variable-length encoding that represents Unicode characters in one to four octets. In this case the TeX ecosystem only partially followed. The original program, as maintained by Donald Knuth (TeX3), as well as the variant pdfTeX

The LaTeX Companion

Third Edition – Part II

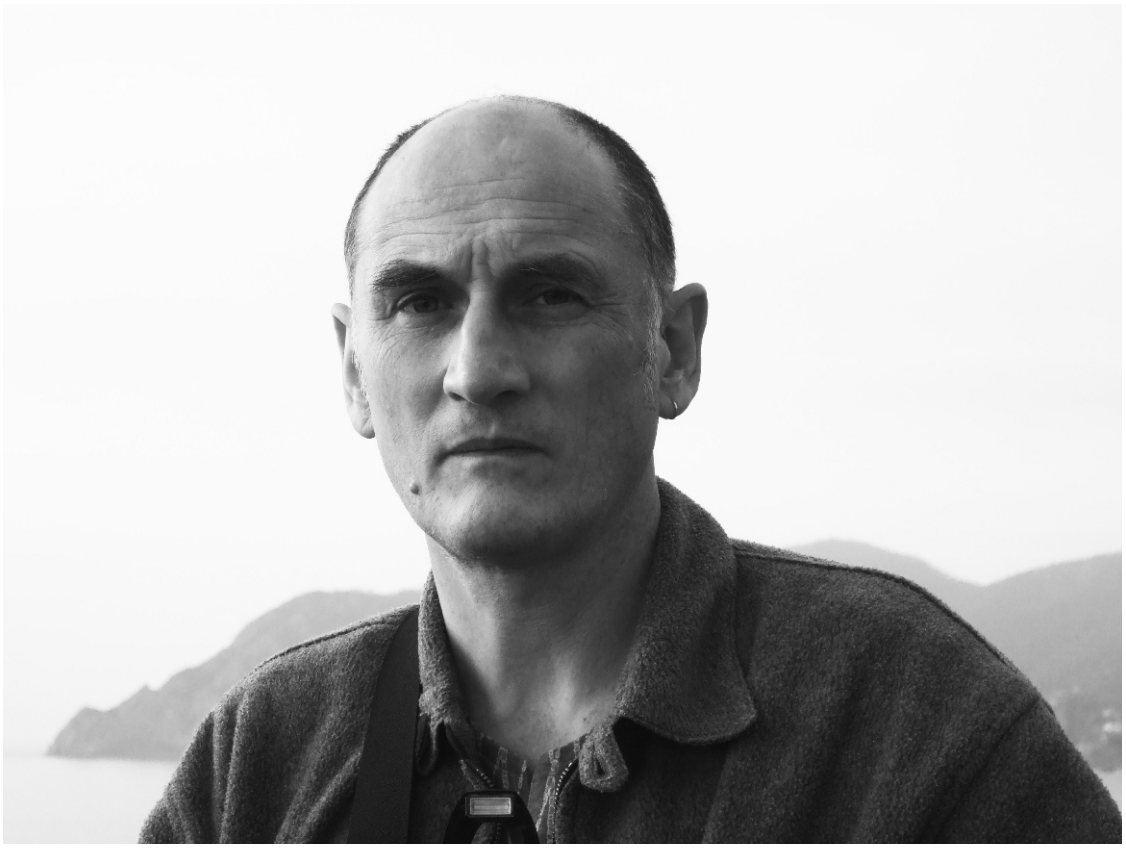
Frank Mittelbach
LaTeX Project, Mainz, Germany

Ulrike Fischer
LaTeX Project, Bonn, Germany

With contributions by
Javier Bezos, Johannes Braams, and Joseph Wright

Addison-Wesley

Boston • Columbus • New York • San Francisco • Amsterdam • Cape Town
Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico City
São Paulo • Sydney • Hong Kong • Seoul • Singapore • Taipei • Tokyo


This picture of Sebastian was taken 2007 in Cinque Terre.

I dedicate this edition to all my friends in the TEX world and in particular
to the memory of my good friend Sebastian Rahtz (1955–2016),
with whom I spent many happy hours discussing parenting, literature,
LATEX, and other important aspects of life [146].

CHAPTER 10

Text and Symbol Fonts

When we wrote the first edition of the *LaTeX Companion* in 1994, the section on available text fonts for LaTeX was a few pages long and listed a handful of font families. Not because they were best in class, but because that was all that was available including those of a somewhat dubious quality. Basically when typesetting with TeX in those days, one could use any font as long as it was called Computer Modern.

A decade later the situation finally started to change, and it was in theory possible to use any font available in Type 1 format [1]—but only after somewhat extensive work preparing the necessary support files needed by TeX, in particular the font metric files (`.tfm`) and usually virtual font files (`.vf`) that reencoded the fonts to put the glyphs into the positions expected by the TeX engine. Providing these was not magic, especially after the appearance of the fontinst program[1] by Alan Jeffrey, Rowland

[1]This is actually a TeX file that, when processed and given the right configuration data, produced the necessary helper files in human-readable format. In a further step those had to be converted by external programs to the binary format used by TeX.

McDonnell, and Lars Hellström, but it took time and effort (even if necessary only once per font) and a good understanding of the underlying mechanisms. Thus, the number of available text fonts for use with TEX compared to other programs remained severely restricted. The second edition of the book again covered all that was freely available for LATEX users, which amounted to fewer than two dozen font families. Of course, the CTAN archive already then contained some further support packages for a few commercial fonts. They were not included because the packages were useless unless you owned the particular family.

But with the third edition I faced a problem. With Unicode engines you can use essentially any freely or commercially available font by simply specifying it in the fontspec setup of your document. But also for the pdfTEX engine, font support exploded due to two factors: first the number of freely available fonts on the Web in either Type 1 [1], TrueType [8], or OpenType [127] format grew enormously, and second a few individuals like Marc Penninga, the author of autoinst, and Bob Tennent, Michael Sharpe, and a few others[1] took the time to prepare configuration files for Marc's autoinst program and with the necessary further adjustments and documentation produced packages that made a huge number of those freely usable fonts available to the LATEX community at large.

My initial thought was to select only a few of the high-quality free fonts and largely ignore the rest — with just a mention that you find more possibilities in your TEX installation and on CTAN. But while working through all the font packages on CTAN to make a selection, I realized how difficult it is to hunt for them and that most likely any family not described in an overview remains unnoticed by the majority of users. Furthermore, while in the early days most free fonts were of a somewhat dubious quality, that too has now changed quite impressively for the better.

So in the end I decided to continue the tradition of offering a fairly comprehensive overview[2] about today's freely available fonts for LATEX so that you can make an informed selection by just skimming this chapter and comparing the different possibilities and only then look further at the package documentation of the fonts you have chosen. You may also want to take a look at the online font catalogue maintained by Palle Jørgensen [77].

10.1 Overview

General font availability

All fonts described in this chapter are freely available and with a few exceptions are included in the major TEX distributions such as TEX Live or MiKTEX so that you can start using them directly. We made two exceptions and also included the commercially available Cambria fonts as well as the Lucida font families that we use in this book. The Cambria fonts, while under a proprietary license, ship as part of Windows and

[1]All package authors are acknowledged next to their work and listed in the index. There are too many to enumerate here. However, Bob and Michael are the ones who produced the major part of the work, which I think deserves an explicit acknowledgment.

[2]For the first time restricted to only high-quality or otherwise (for some reason) interesting fonts. A heartfelt thanks to Adam Twardoch who helped me a lot with his knowledge and expertise to select the high-quality fonts from the huge number of freely available fonts today.

Office products and are therefore widely available without the need for buying an additional license. They offer excellent math support and for that alone are worth a closer look. The Lucida families are sold by the TEX Users Group (TUG) at a special price for members of TUG and several other user groups.[1]

In the open source world, "free" is a widely debated word, and some fonts are just not "free enough" to be included in free software distributions, because their license is somewhat restrictive, typically either forbidding modifications of the fonts (which is most likely not any issue for the readers of this book) or disallowing commercial usage. The latter refers to actions such as selling them or distributing them as part of a commercial system — none of the licenses of the fonts we cover prohibit the free use of the fonts even if the result, say, a book like this, is then sold. Thus, this should not pose a problem either, but of course, you should be aware that fonts you need to install "manually" have a special license, and you better check what it says if you intend to do anything with it other than simply typesetting your texts.

Omitting high-quality fonts because of license restrictions (even if very minor) is a sensible approach, as it means that the users of the distribution can rely on the fact that everything contained is covered by one of the major free licenses and that there are no restrictions on use and only well-known ones on modification; e.g., most LATEX software uses the LATEX Project Public License (LPPL), GNU Public License (GPL), or any other of the few major open source licenses.

But of course, it puts an additional burden on the user, as they have to manually install the fonts (besides checking the license requirements). Fortunately, there is an easy way to integrate such font packages into your installation. At the TEX Users Group website a script by Reinhard Kotucha is provided. It is a simple matter of downloading the install-getnonfreefonts installer from https://www.tug.org/fonts/getnonfreefonts/ and processing it on the command line (in a Windows, macOS, or Linux terminal window) using

```
texlua install-getnonfreefonts
```

As this is a Lua program and texlua is part of the standard distributions, this works on any operating system, provided your user has write access to the distribution directories. After this you have the program getnonfreefonts available on your system, and you can invoke it with lines such as

```
getnonfreefonts --sys --help        # see the help info or
getnonfreefonts --sys --lsfonts     # list all available fonts or
getnonfreefonts --sys luximono      # (re)install a fonts package or
getnonfreefonts --sys --all         # (re)install all available fonts
```

The line first displays the program usage information and the available options; the second gives you an overview about the currently supported fonts and their

[1]The set includes matching serif, sans serif, and typewriter families; a full set of fonts for use in math; and several specialized fonts; see Table 10.11 on page 22 for an overview. They can be used with all TEX engines if both Type 1 and OpenType versions are ordered, which you can do at https://tug.org/store/lucida/order.html.

state on your installation; the third installs or reinstalls such a font (or more if you add additional font names); and the final one installs or updates all supported font packages (which are eleven in total at the moment). Instead of `--sys`, you can use `--user`, in which case the installation is done in the user's TEX tree instead of the system-wide one. However, except in a few special cases, this is *the wrong thing to do*, so please read `https://tug.org/texlive/scripts-sys-user.html` first if you are considering using `--user`, because of its possibly undesirable side effects.

10.1.1 Notes on the font samples

All font families in this chapter are exhibited using the same example text to allow for easy comparison and at the same time to show many details and possible limitations of the fonts. The standard setup uses the following text:

Palatino 10pt/12.4pt
(qpl) TeX Gyre Pagella

With a price of £148, **almost anything** can be found FLOATING IN **FIELDS**. — ¿But aren't Kafka's Schloß & Æsop's Œuvres often *naïve* vis-à-vis the dæmonic *phœnix's official rôle* in fluffy soufflés?

In the margin we show the common name of the font (*Palatino*), the font size and the leading used (*10/12.4pt*), the NFSS family name to refer to it in parentheses (`qpl`), and finally with gray background the OpenType or TrueType name to use with Unicode engines (*TeX Gyre Pagella*). Depending on the length of the font names, this may be split differently over up to four lines.

The text is not typeset justified but slightly ragged right (still allowing hyphenation as one can see in a few of the later samples; e.g., on page 34). This is done so that the word spaces are not (or only minimally) altered from their default width. Most examples are set in 10/12pt, which is the default in most document classes, but sometimes we use a larger leading and/or a smaller or larger font size to account for the look and feel of the family.

The example text attempts to show different aspects of the font while still being concise (otherwise this chapter would be even longer). It contains some **bold** text, some **BOLD AND** NONBOLD SMALL CAPITALS, a *slanted word*, and a *few words in italics*. Above, these words are all typeset in blue. In the real samples they would be in black, unless the corresponding shape or series either is not available or is faked.

For instance, most families do not have both an italic and an oblique/slanted shape. Sometimes it is missing, but quite often the oblique shape is really just an alias for italics (or vice versa), which is fairly easy to see if you look at the shape of the "a" in "*naïve*" and "*official*" and compare it to "can" in the first line. Regardless of whether the slanted or italic shape is missing or aliased, it is shown in blue.

The bold text is normally shown in blue only if the series is missing altogether; the exception is the Concrete family where it is faked (by aliasing it to the bold typeface of a different font family, which is a somewhat questionable approach).

When the SMALL CAPS TEXT is shown in blue, half of the time it is because there is simply no such shape in the family and in the other half because the small capitals are faked using a scaled-down version of the uppercase letters. As explained on

CHAPTER 11

Higher Mathematics

Basic LaTeX offers excellent mathematical typesetting capabilities for straightforward documents. However, when complex displayed equations or more advanced mathematical constructs are heavily used, something more is needed. Although it is possible to define new commands or environments to ease the burden of typing in formulas, this is not the best solution. The American Mathematical Society (AMS) provides a major package, amsmath, which makes the preparation of mathematical documents much less time-consuming and more consistent.[1] It forms the core of a collection of packages known as AMS-LaTeX [5] and is the major subject of this chapter. A useful book by George Grätzer [59] also covers these packages in detail.

This chapter describes briefly, and provides examples of, a substantial number of the many features of these packages as well as a few closely related packages; it also gives a few pointers to other relevant packages. In addition, it provides some essential background on mathematical typesetting with TeX. Thus, it covers some of standard LaTeX's features for mathematical typesetting and layout and contains some general hints on how to typeset mathematical formulas, though these are not the main aims of this chapter. It is also definitely not a comprehensive manual of good practice for typesetting mathematics with LaTeX. Indeed, many of the examples are

[1]This package has its foundations in the macro-level extensions to TeX known as AMS-TeX.

offered purely for illustration purposes and, therefore, present neither good design nor good mathematics nor necessarily good LaTeX coding.

Advice on how to typeset mathematics according to late 20th century U.S. practice can be found in Ellen Swanson's *Math into Type* [187]. Many details concerning how to implement this advice using TeX or, equally, standard LaTeX appear in Chapters 16–18 of Donald Knuth's *The TeXbook* [84].

To use the majority of the material described in this chapter, you need to load at least the amsmath or mathtools package in the preamble of your document. If other packages are needed, they are clearly marked in the examples. Detailed installation and usage documentation is included with the individual packages.

11.1 Introduction to amsmath and mathtools

The AMS-LaTeX project commenced in 1987, and three years later AMS-LaTeX version 1.0 was released. This was the original conversion to LaTeX of the mathematical capabilities in Michael Spivak's AMS-TeX by Frank Mittelbach and Rainer Schöpf, working as consultants to the American Mathematical Society, with assistance from Michael Downes (1958–2003) of the AMS technical staff. In 1994, further work was done with David Jones. This work was coordinated by Michael Downes, and the packages have throughout been supported and much enhanced under his direction and the patronage of the AMS.[1] Initially AMS-LaTeX was a separate format, which is the reason for the separate name, but at some point, when computers got faster and faster, the functionality was moved to the package amsmath, which could be loaded into standard LaTeX, and the support for a separate format was dropped. Thus, these days speaking of AMS-LaTeX simply refers to amsmath or the AMS document classes that evolved from that project.

In 2016 the AMS passed maintenance and control of amsmath and some of its accompanying packages back to the LaTeX Project Team, because its math capabilities are a core functionality of LaTeX.[2]

Thanks to a great guy!

Michael Downes (1958–2003) would have been the author of this chapter when we wrote the second edition of this book had he not died in spring 2003. Much of the chapter is based on the documentation he prepared for AMS-LaTeX; thus, what you are reading is a particular and heartfelt tribute by its current author to the life and work of a dear friend and colleague with whom we shared many coding adventures in the uncharted backwaters of TeX.

Available package options

A few options are recognized by the amsmath package. Most of these affect only detailed positioning of the "limits" on various types of mathematical operators (Section 11.4.4) or that of equation tags (Section 11.2.4).

[1]Some material in this chapter is reprinted from the documentation that was distributed with AMS-LaTeX (with permission from the American Mathematical Society).

[2]However, the American Mathematical Society retained maintenance and support of the AMS fonts and the document classes amsart, amsbook, and amsproc; thus, any issue or inquiries concerning these classes or font packages have to be taken up with their technical stuff and not the LaTeX Project Team.

The following three options are often supplied as global document options, set on the `\documentclass` command. They are, however, also recognized when the amsmath package is loaded with the `\usepackage` command.

`reqno`	(**default**) Place equation numbers (tags) on the right.
`leqno`	Place equation numbers (tags) on the left.[1]
`fleqn`	Position equations at a fixed indent from the left margin rather than centered in the text column.

Available subpackages

For historical reasons some components of the amsmath package exist as separate packages so that they can be loaded on their own — a feature that was essential when computer memory was tight and just loading a large package like amsmath was taking considerable time. These days the advice is to always load amsmath and not try to determine what is needed and what is not.

amsopn	Provides `\DeclareMathOperator` for defining new operator names such as `\Ker` and `\esssup`.
amstext	Provides the `\text` command for typesetting a fragment of text in the correct type size.

Extension packages

The following packages, providing functionality additional to that in amsmath, must be loaded explicitly; they are listed here for completeness.

amscd	Defines some commands for easing the generation of commutative diagrams by introducing the CD environment (see Section 11.3.5 on page 159). There is no support for diagonal arrows.
amsthm	Provides a method to declare theorem-like structures and offers a `proof` environment. It is discussed in Section 4.1.4 on page →I 281.
amsxtra	Provides certain odds and ends that are needed for historical compatibility, such as `\fracwithdelims`, `\accentedsymbol`, and commands for placing accents as superscripts.
upref	Makes `\ref` print cross-reference numbers in an upright/roman font regardless of context.

The principal documentation for these packages is the *User's Guide for the* amsmath *Package* (*Version 2.1*) [5], which is part of the LaTeX distribution.

mathtools — A drop-in replacement for amsmath

The amsmath package is very comprehensive and provides most of what is needed when attempting serious mathematical typesetting. It is, however, rather static; i.e., it has not seen a lot additions (or fixes) since its initial development in the nineties. Because of this, several smaller packages, some of which we discuss in this chapter, have been developed to add one or the other missing feature. In addition, there is

[1] When using one of the AMS document classes, the default is `leqno`.

also the mathtools package by Morten Høgholm now maintained by Lars Madsen that has been explicitly developed to offer a replacement for amsmath. It provides all features of amsmath (including the support for all of its package options), augments it with several new features, and fixes a number of known bugs in amsmath. We give examples of its extended functionality in the appropriate places, and if you intend to use any of this, you should load mathtools in place of amsmath.

The AMS-LaTeX document classes

As already mentioned, the three document classes amsart, amsproc, and amsbook from the original AMS-LaTeX collection are still maintained by the American Mathematical Society. They correspond to LaTeX's article, proc, and book, respectively, and are designed to be used in the preparation of manuscripts for submission to the AMS. However, given that they are part of every LaTeX distribution, nothing prohibits their use for other purposes. In fact, they are often used as the base class for other journals instead of the standard LaTeX classes.

With these class files the amsmath package is automatically loaded so that you can start your document simply with `\documentclass{amsart}`. These classes are not covered in this book because they provide an interface similar to that provided by the LaTeX standard classes; refer to [4] for details of their use.

The AMSfonts collection

Some of the material in this chapter refers to another collection of packages from the American Mathematical Society, namely, the AMSfonts distribution. These packages, listed below, set up various fonts and commands for use in mathematical formulas.

amsfonts Defines the `\mathfrak` and `\mathbb` commands and sets up the fonts `msam` (extra math symbols A), `msbm` (extra math symbols B and blackboard bold), `eufm` (Euler Fraktur), extra sizes of `cmmib` (bold math italic and bold lowercase Greek), and `cmbsy` (bold math symbols and bold script) for use in mathematics.

amssymb Defines the names of the mathematical symbols available with the AMSfonts collection. These commands are discussed in Section 11.8. The package automatically loads the amsfonts package.

eufrak Sets up the fonts for the Euler Fraktur letters (`\mathfrak`), as discussed in Section 12.3.3. This alphabet is also available from the amsfonts package.

eucal Makes `\mathcal` use the Euler script instead of the usual Computer Modern script letters; see Section 12.3.3 for details.

Documentation is somewhat dated, but still relevant

The above packages and fonts are maintained by the American Mathematical Society, and the principal piece of documentation for both is the *User's Guide to AMSFonts Version 2.2d* [2]. However, the distribution is now in version 3, while the documentation is still describing version 2.2d. Though most of it is still accurate and relevant, there is one area where the guide talks about METAFONT (bitmapped) versus Type 1 (outline) fonts that you need to ignore. These days only the Type 1 fonts are distributed, so this part is giving incorrect advice. In particular, the `psamsfonts` option described there should no longer be used, because it is no longer recognized by all of the packages and produces either a warning or an error.

CHAPTER 12

Fonts in formulas

For most symbols in a formula, the font used for a glyph cannot be changed by a font declaration as it can be in text. Indeed, there is no concept of, for example, an italic plus sign or a small capital less than sign.

One exception involves the letters of the Latin alphabet, whose appearance can be altered by the use of math alphabet identifier commands such as `\mathcal`. The commands provided by standard LaTeX for this purpose are discussed in Section 9.4; in the first section of this chapter we introduce a few more and discuss their use in some detail.

Another exception relates to the use of bold versions of arbitrary symbols to produce distinct symbols with new meanings. This potentially doubles the number of symbols available, as boldness can be a recognizable attribute of a glyph for nearly every shape: depending on the font family, even “**<**” is noticeably different from “<”. Although there is a `\mathbf` command, the concept of a math alphabet identifier cannot be extended to cover bold symbols — better solutions are the topic of the second section.

To change the overall appearance of the mathematics in a document, the best approach is to replace all the fonts used to typeset formulas. This is usually done in the preamble of a document by loading a (set of) suitable packages, such as those discussed in Section 12.3 starting on page 238.

In the world of text fonts (Chapter 10) there is a clear separation between fonts suitable only for use with pdfTeX and fonts for exclusive use in Unicode engines. Fonts for pdfTeX are available as 8-bit fonts, encoded in `OT1`, `T1`, or some other 8-bit font encoding from Table 9.18 on page →I 737 and typically available as Type-1 or METAFONT fonts. In contrast, OpenType or TrueType fonts are usable only with XeTeX or LuaTeX and are encoded in the TU font encoding.[1,2]

For symbol fonts this is not the case, at least in one direction: it is easily possible to use an 8-bit symbol font, e.g., MarVoSym, originally designed for use with pdfTeX, with one of the Unicode engines.

Given that formulas largely consist of fixed symbols together with a few alphabetical characters and that there have not been (many) Unicode fonts dedicated to mathematics (i.e., fonts that contain the mathematical symbols in their appropriate Unicode slots), it should come as no surprise that the Unicode engines have been designed to work with traditional TeX math fonts.

This has changed in recent years, and while developing OpenType Math fonts (compared to Text fonts) remains a niche market, there are now a number of fonts available that make it interesting to provide a LaTeX math setup that directly uses such fonts. Such a setup is available with the unicode-math package by Will Robertson, and we will describe it in Section 12.4 on page 253.

In the final section of this chapter, starting at page 261, we showcase the effects of extensive changes to documents on a sample page of mathematics, made with just a few keystrokes. It uses the same input material typeset with both Computer Modern Math fonts (the default in LaTeX) and nearly 50 other font family setups for text and mathematics. All of the fonts used are readily available and, except for Lucida fonts, provided free of charge.

12.1 The world of (Latin) math alphabets

It is easily possible to add additional new math alphabets for use in formulas. While this can be done using arbitrary command names, by convention `\mathcal` (a calligraphic alphabet), `\mathscr` (a script alphabet), `\mathfrak` (a fraktur alphabet), and `\mathbb` (a double-stroke also known as blackboard bold alphabet) are normally used as command names. Out of the box only `\mathcal` is available; for the others you have to add a suitable package (e.g., mathalpha) in the preamble or make a declaration manually using `\DeclareMathAlphabet`.

By loading the amsfonts (or the amssymb) package, both the Euler Fraktur alphabet by Hermann Zapf (`\mathfrak`) and a blackboard bold alphabet[3] (`\mathbb`) become available.

$\forall n \in \mathbb{N} : \mathfrak{M}_n \leq \mathfrak{A}$

```
\usepackage{amsfonts}
$ \forall n \in \mathbb{N} : \mathfrak{M}_n \leq \mathfrak{A} $
```

12-1-1

[1]While it is possible to load 8-bit text fonts in Unicode engines, this is not a good idea because correct hyphenation depends on using the appropriate font encoding, which means that words with diacritics come out wrong if a legacy font encoding such as T1 is used.

[2]If text fonts are usable with all engines (and many are), then this is because somebody generated Type-1 fonts from Unicode fonts (restricting them to 8-bit and the appropriate encoding) or that somebody built an OpenType font from existing 8-bit sources.

[3]Not much is known about the origin of this alphabet. Research in old archives (thanks to Nelson Beebe, Barbara Beeton, and Ulrik Vieth for digging) produced no conclusive results.

The Euler Script font by Hermann Zapf

If you also want to use upright Euler Script, you can load the eucal package that is also part of the amsfonts distribution. Loaded without options, it replaces the default `\mathcal`, but if loaded with the option `mathscr`, as done below, it becomes accessible as `\mathscr`, and `\mathcal` remains free for a different alphabet:

$$\mathcal{ABCDEFGHIJKLMNOPQRSTUVWXYZ}$$

$$\mathscr{ABCDEFGHIJKLMNOPQRSTUVWXYZ}$$

12-1-2

```
\usepackage[mathscr]{eucal}
\[ \mathcal{ABCDEFGHIJKLMNOPQRSTUVWXYZ} \]
\[ \mathscr{ABCDEFGHIJKLMNOPQRSTUVWXYZ} \]
```

A double-stroke font by Alan Jeffrey

If you prefer a geometric hollowed-out blackboard bold font in `\mathbb`, you can load, for example, the bboldx package that provides support for a font designed by Alan Jeffrey. One interesting aspect of this font is that it contains many more glyphs than the usual (Uppercase) alphabet?!, which even allows you to use it outside formulas by accessing it as `\usefont{U}{bboldx}{m}{n}` as we did here. Michael Sharpe added an additional thin and a bold face to the original font and made it available through the package bboldx.

$$\mathbb{N} = \{\mathbf{0}, 1, 2, 3, \ldots\}$$

$$\mathbb{ABCDEFGHIJKLMNOPQRSTUVWXYZ}$$

$$\mathbb{abcdefghijklmnopqrstuvwxyz}$$

12-1-3

```
\usepackage[light]{bboldx}
\[ \mathbb{N}=\{\mathbfbb{0},1,2,3,\ldots\} \]
\[ \mathbb  {ABCDEFGHIJKLMNOPQRSTUVWXYZ}    \]
\[ \mathbfbb{abcdefghijklmnopqrstuvwxyz}    \]
```

By default the medium series is offered as `\mathbb`, while `\mathbfbb` selects the bold series. If you load the package with the option `light`, you get the thin and medium series with the two commands instead. Alternatively, you can use `bfbb`, which forces the package to use the bold series for `\mathbf` as if bold were the regular weight. You can also easily scale the alphabet slightly to make it fit with other glyphs in your formulas by using the option `scale`.

By default the other available characters in the font, such as the Greek letters or the special braces or brackets, are not available in math. If you want those in a formula, you can load the package with the option `bbsymbols`. However, note that using that option means that the font is declared as a symbol font, and as discussed in Section 9.4.1 on page →I 681, this means you are reducing your options to use other math fonts in your document. Therefore, do this only if you really intend to use any of the extra symbols.

$$\mathbf{0} \neq \mathbb{1} \neq \mathbf{1}$$

$$\begin{aligned} \mathbb{G} = [\![(\!(\langle\!\langle \alpha, \beta, \gamma, \delta, \ldots, \omega \\ \Gamma, \Delta, \Theta, \Lambda, \ldots, \Omega \rangle\!\rangle)\!)]\!] < \mathbf{G} \end{aligned}$$

12-1-4

```
\usepackage[bbsymbols,scale=0.9]{bboldx} \usepackage{amsmath}
$ \mathbfbb{0} \neq \mathbb{1} \neq \mathbfbb{1} $
\begin{align*} \mathbb{G} = \bbLbrack \bbLparen \bbLangle
 & \bbalpha, \bbbeta, \bbgamma, \bbdelta, \dots,\bbomega  \\
 & \bbGamma, \bbDelta, \bbTheta, \bbLambda, \dots,\bbOmega
   \bbRangle \bbRparen \bbRbrack < \mathbfbb{G} \end{align*}
```

A double-stroke font by Michael Sharpe

A serifed double stroke alphabet based on the Courier clone of URW in regular and bold is provided through the dsserif package by Michael Sharpe. It offers lower and uppercase Latin characters as well as digits and makes them available as `\mathbb` and `\mathbfbb`. As usual, `scale` can be used to adjust the font size to other letters in the formulas.

$$\mathbb{N}=\{\mathbb{0},1,2,3,\ldots\}$$
$$\mathbb{ABCDEFGHIJKLMNOPQRSTUVWXYZ}$$
$$\mathbb{abcdefghijklmnopqrstuvwxyz}$$

```
\usepackage{dsserif}
\[ \mathbb{N}=\{\mathbfbb{0},1,2,3,\ldots\} \]
\[ \mathbb  {ABCDEFGHIJKLMNOPQRSTUVWXYZ}    \]
\[ \mathbfbb{abcdefghijklmnopqrstuvwxyz}    \]
```
12-1-5

A double-stroke font by Olaf Kummer

The DS font designed by Olaf Kummer is a double stroke font that is based on the Computer Modern font shapes and thus works well in formulas done with Computer Modern or Latin Modern math fonts. Besides the usual uppercase alphabet, the font supports four additional characters: a variant form for the uppercase A (accessed by typing a lowercase a), lowercase h and k, and the digit 1. Trying anything else results in missing glyph warnings.

The package makes the alphabet available through the command `\mathds` and not through `\mathbb`. If you prefer the latter, you can alternatively set it up with the help of the mathalpha package or by using `\DeclareCommandCopy`.

$$\mathbb{ABCDEFGHIJKLMNOPQRSTUVWXYZ}$$
$$\mathbb{A} \quad \mathbb{h} \quad \mathbb{k} \quad \mathbb{1}$$

```
\usepackage{dsfont}
\[ \mathds{ABCDEFGHIJKLMNOPQRSTUVWXYZ} \]
\[ \mathds{a \quad h \quad k \quad 1}  \]
```
12-1-6

A nice feature of the package is that it alternatively offers you a matching Sans double stroke font, which is useful if your formulas are typeset with sans serif fonts, e.g., when making slides.

$$\mathbb{ABCDEFGHIJKLMNOPQRSTUVWXYZ}$$
$$\mathbb{A} \quad \mathbb{h} \quad \mathbb{k} \quad \mathbb{1}$$

```
\usepackage[sans]{dsfont}
\[ \mathds{ABCDEFGHIJKLMNOPQRSTUVWXYZ} \]
\[ \mathds{a \quad h \quad k \quad 1}  \]
```
12-1-7

Setting up math alphabets with `\DeclareMathAlphabet`

As an example of small-scale changes to the mathematical typesetting, those who prefer a visually distinct blackboard bold alphabet can load a different one, for example, the one contained in the PX fonts. The necessary declaration is done with `\DeclareMathAlphabet`, which is described in Section 9.4.1. In the example we first load the amsfonts package and then overwrite its definition of `\mathbb`.

$\{n,m \in \mathbb{N} \mid \mathfrak{N}_{n,m}\}$

```
\usepackage{amsfonts}  \DeclareMathAlphabet\mathbb{U}{px-ds}{m}{n}
$ \lbrace n,m \in \mathbb{N} \mid  \mathfrak{N}_{n,m} \rbrace $
```
12-1-8

The previous example shows how to include arbitrary alphabets from your LaTeX distribution as math alphabets, with the crucial part being the arguments of the `\DeclareMathAlphabet` declaration. Although getting these right may appear to be a tricky matter, it is not so difficult once you know where to look. Fonts suitable for inclusion need to have an `.fd` file of the form ⟨*enc*⟩⟨*name*⟩`.fd`, where the ⟨*name*⟩ is the font family name, often abbreviated — see the discussion on NFSS font family naming conventions on page 6. Chapter 10 covers more than one hundred high-quality font families and the necessary information to use them.

CHAPTER 13

Localizing documents

This chapter starts with a short introduction to the technical problems that must be solved if you want to use LATEX with a non-English language. Most of the remaining part of the chapter discusses the babel system, which provides a convenient way to generate documents in different languages. We look in particular at how we can typeset documents in French, Russian, and Greek, as the typesetting of those languages illustrates various aspects of the things one has to deal with in a non-English environment. We also say a few words about how to handle other languages, such as Arabic, Japanese, or Hindi, which are written with the so-called "complex scripts". Section 13.6 explains how to tailor the language styles to fit your needs in some common cases.

13.1 TEX and non-English languages

Due to its popularity in the academic world, TEX spread rapidly throughout the world and is now used not only with the languages based on the Latin alphabet, but also with languages using non-Latin alphabetic scripts, such as Russian, Greek, Arabic, Persian, Hebrew, Thai, Vietnamese, Malayalam, or Marathi. Implementations also exist for Chinese and Japanese (which use Kanji-based ideographic scripts) and Korean (which uses syllable blocks).

With the introduction of 8-bit TEX and METAFONT, which were officially released by Donald Knuth in March 1990, the problems of multilingual support could be more easily addressed for the first time. Nevertheless, by themselves, these versions did not solve all the problems associated with providing a convenient environment for using LATEX with multiple and/or non–English languages, which were eventually addressed with the advent of Unicode engines like XƎTEX and LuaTEX.

To achieve this goal, TEX and its companion programs should be made truly international, and the following points should be addressed:

1. Adjust all programs to the particular language(s):
   - Handle properly fonts containing national symbols, which includes setting up the so-called Unicode fonts where necessary and appropriate [67],
   - Define line breaking and justification rules, which in Western languages means generating patterns for the hyphenation algorithm,
   - Support typesetting in different directions.
2. Provide a translation for the language-dependent strings, create national layouts for the standard documents, and provide TEX or Lua code to treat the language-dependent typesetting rules automatically [130].
3. Support processing of multilingual documents (more than one language in the same document) and work in international environments (one language per document, but a choice between several possibilities). For instance, the sorting of indexes and bibliographic references should be performed in accordance with a given language's alphabet and collating sequence; see the discussion on upmendex in Section 14.3 and biber in Section 15.2.2.

At the same time, you should be able to conveniently edit, view, and print your documents using any given character set, and LATEX should be able to successfully process files created in this way. Encoding problems were ultimately solved with Unicode, which can encode not only the alphabetic languages but also ideographic scripts like Chinese or Japanese.

LATEX uses by default the encoding known as UTF-8, which has been universally adopted in the computing and publishing worlds. There still exist other character encoding schemes, but their use is declining, because with them it is difficult for documents to be reproducible in different environments, issues of standardization become important, and it is necessary to know the encoding in which a document was produced if not UTF-8. For such legacy encodings LATEX offers the inputenc package, described in Section 9.9.3 on page →I 758, but it is recommended to avoid it and use the default (UTF-8) for new documents.[1]

To adapt LATEX to different languages, several approaches have been followed. The first to appear was to use dedicated packages, which very often were incompatible,

[1]A few packages, including some babel languages, have not been yet updated for the current default encoding and may display a warning like "No input encoding specified" or similar. It can be usually ignored.

thus making a multilingual document almost impossible. A second step was the introduction of babel, which eased the standardization and development of TEX-related software adapted to different languages. The aim was to produce for each language or group of languages a package that would facilitate typesetting, with details about fonts, input conventions, hyphenation patterns, a LATEX option file compatible with the babel concept (see Section 13.1.3), possibly a preprocessor, and, of course, documentation in English and the target language. Still, because each language takes its own solutions, interoperation remains problematic, and a new approach, based on a core providing the basic behavior with descriptive `.ini` files, is being developed at the time of this writing, although it has reached a very advanced stage.

In some languages, particularly East Asian ones, there is another approach, namely, extending the TEX program to suit the needs of a particular family of languages. For example, upTEX adds features for vertical writing and deals with other facets of typesetting Japanese.

Besides XƎTEX and LuaTEX, there have been other extensions to the TEX program attempting to improve the TEX multilingual support, like mlTEX, EncTEX, or Omega. The latter, by Yannis Haralambous and John Plaice, incorporated a sophisticated bidi algorithm, which has become part of LuaTEX, and provided a mechanism to perform text transformations, which has inspired what in babel are called *transforms* (transliterations is an example).

13.1.1 Language-related aspects of typesetting

When thinking about supporting typesetting documents in languages other than English, a number of aspects that need to be dealt with come to mind.

First and foremost is the fact that other languages have different rules for line breaking, something that TEX accommodates in Western languages through its support for multiple hyphenation patterns. In some languages, however, certain letter combinations change when they appear at a hyphenation point.

Unicode engines

TEX does not support this capability “out of the box”, but LuaTEX can be extended to apply nonstandard hyphenation rules — a feature exploited by babel when using this engine. Scripts like Thai, Japanese, or Amharic follow some specific rules, which require XƎTEX and LuaTEX.

Some languages need different sets of characters to be properly typeset. This issue can vary from the need for additional “accented letters” (as is the case with many European languages) to the need for a completely different alphabet (as is the case with languages using the Cyrillic or Greek alphabet).

Unicode engines

When non-European languages need to be supported, the typesetting direction might be different as well (such as right to left for Arabic and Hebrew texts), or so many characters might be needed (as is the case with the Kanji script, for instance) that traditional TEX’s standard mechanisms cannot deal with them — in that case XƎTEX or LuaTEX is required.

A more “subtle” problem turns up when we look at the standard document classes that each LaTeX distribution supplies, because they were designed for the Anglo-American situation. A specific example where this preference interferes with supporting other languages is the start of a chapter. For some languages it is not enough to just translate the word “Chapter”; the order of the word and the denomination of the chapter needs to be changed as well, solely on the basis of grammatical rules. Where the English reader expects to see “Chapter 1”, the Hungarian reader expects to see “1. fejezet”.

13.1.2 Culture-related aspects of typesetting

An even thornier problem when faced with the need to support typesetting of many languages is the fact that typesetting rules differ, even between countries that use the same language. For instance, hyphenation rules differ between British English and American English. Translations of English words might vary between countries, just as they do for the German spoken in Germany and the German spoken (and written) in Austria.

Typographic rules may differ between countries, too. No worldwide standard tells us how nested lists should be typeset; on the contrary, their appearance may differ for different languages or countries or even printing houses. With these aspects we enter the somewhat fuzzy area comprising the boundary between the language aspects of typesetting and the cultural aspects of typesetting. It is not clear where that boundary lies. When implementing support for typesetting documents written in a specific language, this difference needs to be taken into account. The language-related aspects can be supported on a general level, but the cultural aspects are more often than not better (or more easily) handled by creating specific document classes.

13.1.3 babel — LaTeX speaks multiple languages

The LaTeX distribution contains a few standard document classes that are used by most users. These classes (article, report, book, and letter) have a certain American look and feel, which not everyone likes. Moreover, the language-dependent strings, such as “Chapter” and “Table of Contents” (see Table 13.2 on page 305 for a list of commands holding language-dependent strings), come out in English by default.

The babel package, originally developed by Johannes Braams [23] and now maintained and developed further by Javier Bezos [19], provides a set of options that allows the user to choose the language or language(s) in which the document is typeset. It has the following basic characteristics:

- Multiple languages can be used simultaneously.
- The list of hyphenation patterns to be loaded when the LaTeX format is generated can be defined via an external file.
- Translations for the language-dependent strings and commands for facilitating text input are provided for more than 60 languages (a selection of them is listed in Table 13.1 on the facing page).

CHAPTER 14

Index Generation

To find a topic of interest in a large document, book, or reference work, you usually turn to the table of contents or, more often, to the index. Therefore, an index is a very important part of a document, and most users' entry point to a source of information is precisely through a pointer in the index. You should, therefore, plan an index and develop it along with the main text [40]. For reasons of consistency, it is beneficial, with the technique discussed below, to use special commands in the text to always print a given keyword in the same way in the text and the index throughout the whole document.

This chapter first reviews the basic indexing commands provided by standard LaTeX and explains which tools are available to help you build a well-thought-out index. The *LaTeX Manual* does not contain a lot of information about the syntax of the `\index` entries. However, several articles in *TUGboat* deal with the question of generating an index with TeX or LaTeX. The syntax described in Section 14.1 is the one recognized by *MakeIndex* [39, 105] and the other indexing programs we discuss: `upmendex`, `xindy`, and `xindex`.

Section 14.2 describes how the *MakeIndex* processor is used. The interpretation of the input file and the format of the output file are controlled by style parameters. Section 14.2.4 lists these parameters and gives several simple examples to show how changing them influences the typeset result.

Section 14.3 presents `upmendex`, an Unicode-aware extension to *MakeIndex*. It is preferable to use this program whenever you have non-English documents or other special demands, such as the production of technical indexes.

① A raw index (`.idx` file) is generated by running LaTeX.

② The raw index, together with some optional style information (`.ist` file in the case of *MakeIndex* or `upmendex`), is used as input to the index processor, which creates an alphabetized index (`.ind` file) and a transcript (`.ilg` file).

③ The index (`.ind` file) is read by LaTeX to give the final typeset result.

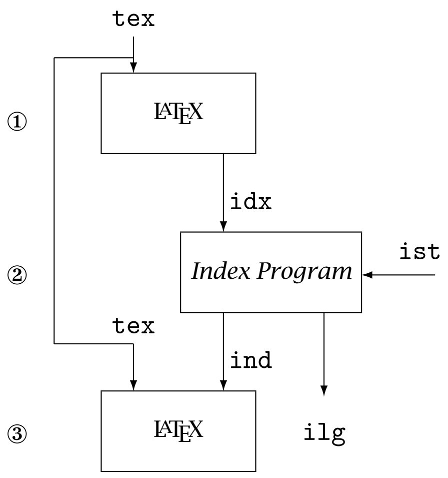


Figure 14.1: The sequential flow of index processing and the various auxiliary files used by LaTeX and an external index processor, e.g., *MakeIndex*

In Section 14.4 we briefly discuss two further alternatives that you may come across. The final section describes several LaTeX packages to enhance the index and to create multiple indexes such as those in the book you are reading.

* * *

The process of generating an index is shown schematically in Figure 14.1. The steps for generating an index with LaTeX and *MakeIndex* are illustrated in this figure. The same flow is used with other index processors; only the configuration of the processors is done differently in some cases, i.e., not with `.ist` style files.

Figure 14.2 on the next page shows, with an example, the various steps involved in transforming an input file into a typeset index. It also shows, in somewhat more detail, which files are involved in the index-generating process. Figure 14.2(a) shows some occurrences of index commands (`\index`) in the document source, with corresponding pages listed on the left. Figure 14.2(b) shows a raw index `.idx` file generated by LaTeX. File extensions may differ when using multiple indexes or glossaries. After running the `.idx` file through the index processor, it becomes an alphabetized index `.ind` file with LaTeX commands specifying a particular output format [Figure 14.2(c)]. The typeset result after formatting with LaTeX is shown in Figure 14.2(d).

LaTeX and *MakeIndex* (or `upmendex` or `xindex`), when employed together, use several markup conventions to help you control the precise format of the output. The `xindy` program has a *MakeIndex* compatibility mode that supports the same format. In Section 14.1, which describes the format of the `\index` command, we always use the default settings.

```
Page vi:   \index{animal}
Page 5:    \index{animal}
Page 6:    \index{animal}
Page 7:    \index{animal}
Page 11:   \index{animalism|see{animal}}
Page 17:   \index{animal@\emph{animal}}
           \index{mammal|textbf}
Page 26:   \index{animal!mammal!cat}
Page 32:   \index{animal!insect}
```

(a) The input file

```
\indexentry{animal}{vi}
\indexentry{animal}{5}
\indexentry{animal}{6}
\indexentry{animal}{7}
\indexentry{animalism|see{animal}}{11}
\indexentry{animal@\emph{animal}}{17}
\indexentry{mammal|textbf}{17}
\indexentry{animal!mammal!cat}{26}
\indexentry{animal!insect}{32}
```

(b) The `.idx` file

```
\begin{theindex}
  \item animal, vi, 5-7
    \subitem insect, 32
    \subitem mammal
      \subsubitem cat, 26
  \item \emph{animal}, 17
  \item animalism, \see{animal}{11}
  \indexspace
  \item mammal, \textbf{17}
\end{theindex}
```

(c) The `.ind` file

animal, vi, 5–7
    insect, 32
    mammal
        cat, 26
animal, 17
animalism, *see* animal

mammal, **17**

(d) The typeset output

Figure 14.2: Stepwise development of index processing

14.1 Syntax of the index entries

This section describes the default syntax used to generate index entries with LaTeX and either *MakeIndex* or one of the other programs described in this chapter. Different levels of complexity are introduced progressively, showing, for each case, the input file and the generated typeset output.

Figures 14.3 and 14.4 on page 353 show the input and generated output of a small LaTeX document, where various simple possibilities of the `\index` command are shown, together with the result of including the showidx package (see Section 14.5.2). To make the index entries consistent in these figures (see Section 14.1.7), the commands `\Com` and `\Prog` were defined and used. The index-generating environment `theindex` has been redefined to get the output on one page (Section 14.5.1 explains how this can be done).

Generating the raw index

After introducing the necessary `\index` commands in the document, we want to generate the index to be included once again in the LaTeX document on a subsequent run. If the main file of a document is `main.tex`, for example, then the following changes should be made to that file:

- Include the makeidx package with a `\usepackage` command.
- Put a `\makeindex` command in the document preamble.

- Put a `\printindex` command where the index is to appear — usually at the end, right before the `\end{document}` command.

You then run LaTeX on the entire document, causing it to generate the file `main.idx`, which we shall call the `.idx` file.

14.1.1 Simple index entries

Each `\index` command causes LaTeX to write an entry in the `.idx` file. The following example shows some simple `\index` commands, together with the index entries that they produce. The page number refers to the page containing the text where the `\index` command appears. As shown in the example below, duplicate commands on the same page (such as `\index{stylistic}` on page 23) produce only one "23" in the index.

```
style, 14                   Page iii:  \index{style}
style , 16                  Page xi:   \index{Stylist}
style, iii, 12              Page 12:   \index{style}
style , 15                             \index{styles}
style file, 34              Page 14:   \index{ style}
styles, 12                  Page 15:   \index{style }
Stylist, xi                 Page 16:   \index{ style }
stylist, 34                 Page 23:   \index{stylistic}
stylistic, 23                          \index{stylistic}
                            Page 34:   \index{style file}
                                       \index{stylist}
```

Spaces can be harmful

Pay particular attention to the way spaces are handled in this example. Spaces inside `\index` commands are written literally to the output `.idx` file and, by default, are treated as ordinary characters by *MakeIndex*, which places them in front of all letters. In the example above, look at the `style` entries on pages 14 and 16. The leading spaces are placed at the beginning of the index and on two different lines because the trailing blank on page 16 lengthens the string by one character. We end up with four different entries for the same term, an effect that was probably not desired.

It is therefore important to eliminate such spurious spaces from the `\index` commands when you use *MakeIndex*. Alternatively, you can specify the `-c` option when running the index processor. This option suppresses the effect of leading and trailing blanks (see Sections 14.2.2 and 14.3.1). We recommend always using this option; it should really have been the default.

Inconsistent spelling

Another frequently encountered error occurs when the same English word is spelled inconsistently with initial lowercase and uppercase letters (as with `Stylist` on page `xi` in the preceding example), leading to two different index entries.

Of course, this behavior is wanted in languages like German, where "Arm" (arm) and "arm" (poor) are really two completely different words.[1] In English, such spurious double entries should normally be eliminated.

[1] As the joke goes: "Besser arm dran als Arm ab" (better poor than without an arm).

CHAPTER 15

Bibliography Generation

While a table of contents (see Section 2.3) and an index (discussed in Chapter 14) make it easier to navigate through a book, the presence of bibliographic references should allow you to verify the sources and to probe further subjects you consider interesting. To make this possible, the references should be precise and lead to the relevant work with a minimum of effort.

There exist many ways to format bibliographies, and different fields of scholarly activities have developed very precise rules in this area. An interesting overview of Anglo-Saxon practices can be found in the chapter on bibliographies in *The Chicago Manual of Style* [40]. Normally, authors must follow the rules laid out by their publisher. Therefore, one of the more important tasks when submitting a book or an article for publication is to generate the bibliographic reference list according to those rules.

Traditional ways of composing such lists by hand, without the systematic help of computers, are plagued with the following problems:

- Citations and references, particularly in a document with contributions from many authors, are hard to make consistent. Difficulties arise, such as variations in the use of full forenames versus abbreviations (with or without periods); italicization or quoting of titles; spelling “ed.”, “Ed.”, or “Editor”; and the various forms of journal volume number.

- A bibliography laid out in one style (e.g., alphabetic by author and year) is extremely hard to convert to another (e.g., numeric citation order) if requested by a publisher.
- It is difficult to maintain one large database of bibliographic references that can be reused in different documents.

In the present chapter we concentrate on the formatting of reference lists and bibliographies, and we discuss possibilities for managing collections of bibliographic references in databases. The chapter is heavily based on the BibTeX program, written by Oren Patashnik, and the biber program written by François Charette and now maintained by Philip Kime, which both integrate well with LaTeX. In Chapter 16 we are then mainly concerned with the citation of sources within the text.

We start by showing how a bibliography can be created manually in LaTeX. Because the standard environment is also used in the output of most BibTeX styles, this prepares the ground for the BibTeX workflow. We then introduce the two programs. This is followed by a detailed introduction to the database format[1] used to specify bibliographical data in a suitable form for processing with BibTeX or biber. We continue with a description of how to produce bibliographic input files for LaTeX from these databases.

Instead of collecting your own bibliographical data, there is also the possibility of drawing information from various online sources that offer such data in BibTeX format. Some of them are introduced in Section 15.5.

Having collected data for BibTeX databases, the next natural step is to look for tools that help in managing such databases. Section 15.6 offers tools of various flavors for this task, ranging from command-line utilities to GUI-based programs for various platforms.

Finally, in Section 15.7 we return to the task of typesetting a bibliography and discuss how different BibTeX and biblatex styles can be used to produce different bibliography layouts from the same input. Because there may not be a suitable style for a particular set of layout requirements available, Section 15.7.2 discusses how to generate customized styles using the custom-bib package without the need for any BibTeX style programming. How to create or adapt styles for biblatex is discussed in Chapter 16 where we review that package in detail.

15.1 The standard LaTeX bibliography environment

The standard LaTeX environment for generating a list of references or a bibliography is called `thebibliography`. It is not defined by LaTeX itself but by the document class, so the layout can vary. In its default implementation in the standard classes, it automatically generates an appropriate heading and implements a vertical list structure in which every publication is represented as a separate item. It also creates

[1]Due to its origin, the format is commonly referred to as the BibTeX database format, even if used with biber. Thus, if you read "BibTeX database format" in the remainder of this chapter, it refers to the format usable by either of the programs.

for every item a label to allow referencing it in the document. The `thebibliography` environment can be created manually, but it is also used in the output of most BibTeX styles. It is not used by the biblatex package, which typesets bibliographies with its own command, `\printbibliography`.

The different traditions regarding how such sources are then referred to in a document and how to produce such citations with LaTeX is the topic of Chapter 16; here citation commands are mentioned only as far as necessary to understand how they affect the bibliography.

```
\begin{thebibliography}{widest-label}
\bibitem[label1]{cite-key1}  bibliographic information
\bibitem[label2]{cite-key2}  bibliographic information
      ...
\end{thebibliography}
```

The *widest-label* argument is used to determine the right amount of indentation for individual items. If the works are numbered sequentially, for example, it should contain the number of items.

Individual publications are introduced with a `\bibitem` command. Its mandatory argument is a unique cross-reference *cite-key* that is used to refer to this publication in the text of the document. For this purpose LaTeX offers the `\cite` command, which takes such a *cite-key* as its argument. Its precise syntax is described in Section 16.2.1 on page 475.

The optional argument defines the textual representation that is used in the citation and as the *label* in the list. If this argument is not specified, the publications are numbered with arabic numerals by default. Within a bibliographic entry the command `\newblock` may be used to separate major blocks of information. Depending on the layout produced by the document class, it may result in a normal space, may result in some extra space, or might start a new line.

The text of the heading is produced — depending on the class — by either the command `\bibname` or `\refname`. A language package, such as babel, localizes such commands and changes to them should be done by using the methods described in Chapter 13.

[1], [GOO97]

References

[1] Goossens, M., S. Rahtz, and F. Mittelbach (1997). *The LaTeX Graphics Companion: Illustrating Documents with TeX and PostScript*. Reading, MA, USA: Addison-Wesley Longman.

[GOO97] Goossens, M., B. User, J. Doe (1997). Ambiguous citations.

15-1-1

```
\cite{LGC97}, \cite{GUD97}
\begin{thebibliography}{2}
\bibitem{LGC97}  Goossens, M., S.~Rahtz,
    and F.~Mittelbach (1997).
  \newblock \emph{The \LaTeX{} Graphics Companion:
  Illustrating Documents with \TeX{} and
  PostScript}. \newblock Reading, MA, USA:
  Ad\-di\-son-Wes\-ley Longman.
\bibitem[GOO97]{GUD97} Goossens, M., B.~User,
  J.~Doe (1997). \newblock Ambiguous citations.
\end{thebibliography}
```

Producing a large bibliography manually in this way is clearly a tedious and difficult task, and the result is normally not reusable, because nearly all journals and publishers have their own house styles with different formatting requirements. For this reason it is generally better to use programs, such as BibTEX or biber, that generate ready-to-use LaTEX input from a database of bibliographical information. This is discussed in the next section.

Order by first citation done manually

Note that the references are typeset and numbered in the order in which they appear in the bibliography. Thus, if you produce the bibliography manually, numbering and sorting the entries into the correct order becomes your task, which is especially difficult if they should be ordered by first reference in the text instead of alphabetically. In contrast, when using BibTEX or biber, either order can be achieved automatically.

Handling the requirement to use a `thebibliography` environment explicitly

Some journals provide templates in which you are required to provide the reference list as explicit `\bibitems` in a `thebibliography` environment. When generating the bibliography with BibTEX, this can be easily achieved by copying the final `.bbl` content into your document before submitting. However, when using biblatex/biber, this is not possible. In this case, the biblatex2bibitem package by Nikolai Avdeev can be of some help. Its usage is simple: load the package and issue `\printbibitembibliography` at the end of the document. This then typesets at this point a source representation of the `thebibliography` environment, which can be copied and pasted from the PDF into the source file intended for the journal. However, please note that when using the engine pdfTEX, you need to ensure that your document uses `\usepackage[T1]{fontenc}` so that you get real accented characters into the output — with the default `OT1` encoding, cut and paste goes horribly wrong otherwise.

Copying from a PDF is not fool-proof even then, and the result should be checked for faulty characters and missing spaces. But it can save you from having to manually retype a possibly lengthy reference list.

15.2 The biber and BibTEX programs

The BibTEX program was designed by Oren Patashnik to provide a flexible solution to the problem of automatically generating bibliography lists conforming to different layout styles. It automatically detects the citation requests in a LaTEX document (by scanning its `.aux` file or files), selects the needed bibliographical information from one or more specified databases, and formats it according to a specified layout style. Its output is a file containing the bibliography listing as LaTEX code that is automatically loaded and used by LaTEX on the following run. Section 15.4 on page 409 discusses the interface between the two programs in some detail.

At the time of this book's writing BibTEX was available as version `0.99d`, but if you look into the second edition of this book (a decade back), you find that it already talks about version `0.99c`. Version `0.99a` probably dates back to 1986. In other words, the program has been kept stable for a very long period of time. As a consequence, the BibTEX database format is very well established in the LaTEX world (and in fact even far beyond), with many people having numerous citation entries collected over the

CHAPTER 16

Managing Citations

16.1 Introduction

Citations are cross-references to bibliographical information outside the current document, such as to publications containing further information on a subject and source information about used quotations. It is certainly not necessary to back everything by a reference, but background information for controversial statements, acknowledgments of other work, and source information for used material should always be given.

The previous chapter showed numerous ways to compile bibliographies and reference lists. They can be prepared manually, if necessary, but usually they are automatically generated from a database containing bibliographic information. This chapter now introduces the many presentation forms of bibliographical sources, and it reviews different traditions regarding how such sources are referred to in a document.

We start the chapter with a short introduction to the major citation schemes in common use. Armed with this knowledge we then plunge into a detailed discussion of how LaTeX supports the different citation schemes. At the time we wrote the first edition of this book, LaTeX basically supported only the “number-only” system. Nearly three decades later, the situation has changed radically. Today, all major citation schemes are well supported by extension packages.

We end this chapter by discussing packages that can deal with multiple bibliographies in one document. This is not difficult if the reference lists are prepared manually or if biblatex together with biber is used, but it poses some challenges if you want to interact with BIBTEX, as well.

16.1.1 Bibliographical reference schemes

There are five common methods of referring to sources: the "number-only", "author-date" (or "author-year"), "author-number", "author-title", and "verbose" systems. The last two of these are often used in books on humanities and jurisdiction; the second appears mainly in science and social science works. The other two are less often used, although the first is quite common within the LATEX world, because it has been actively promoted by Leslie Lamport and originally was the only form of citation supported by LATEX. Outside the LATEX world, a variation of it, called "numeric by first citation", is quite popular as well.

The number-only system

In the number-only system, publications are sequentially numbered in the bibliography. Citations in the text refer to these numbers, which are usually surrounded by brackets or parentheses. Sometimes raised numbers are used instead.

One argument against this system — put forward, for example, in *The Chicago Manual of Style* [40] — is that it raises the costs of publication, because a late addition or deletion of a reference may require renumbering and consequently costly (and error-prone) changes to many pages throughout the manuscript. With automatic cross-referencing facilities as provided by LATEX, this argument no longer holds true. In fact, the number-only system is the default system provided with LATEX.

In a slight variation, known as "alpha" style, citations comprise the author's name and the year of the publication. Thus, the bibliographic label and the citation may look like "[Knu86]". This is still fairly compact compared with an author-date or author-title style and has the advantage that one can often deduce the reference from the context.

Numerical by first citation

A fairly popular form of the number-only system numbers the publications sequentially by their first citation in the text (and presents them in that order in the bibliography). This is fairly easy to provide with either standard LATEX or biblatex. However, in that case it is important to avoid that references in the table of contents mess up the expected order; see Section 16.2.3 for advice on this.

The author-date system

In the author-date system (often referred to as the Harvard system after one of its better known typographical variants), references to sources are also given directly in the text. They show the author's name (or names) and the year of the publication. The full citation is given in a list of references or a bibliography. If the author published more than one work in a given year, that year is suffixed with lowercase letters (e.g., 2001a, 2001b).

There have been many attempts over the years to provide author-date citation support for LATEX. With the natbib package (discussed in Section 16.3.2), the jurabib package (discussed in Section 16.5.1) and the biblatex package (discussed in Section 16.7 and 16.3.3), there are now three very flexible and general solutions available.

It should be possible for a reader to look up a cited reference in the bibliography lists. In all citation schemes that use author or editor names, the bibliography list therefore should normally be sorted alphabetically by these names. This can require special care if some of the names contain accented chars or are written in a non-Latin script; see Section 15.3.3 for some advice. The sorting should follow the rules of the main language of the document. This is easy to achieve with biblatex, because biber contains the necessary Unicode libraries, but with BibTeX it can be necessary to add sort keys to some entries to guide the sorting.

Watch out for ambiguous citations

Care must also be taken to get unique citation "labels", which allow identifying the reference without ambiguity. For example, a work by three or more authors is usually referred to by using the name of the first author followed by *et al.* Especially with the author-date system, this may lead to ambiguous citations if different groups of authors with the same main author published in the same year. This problem can be seen in the following example:

```
\usepackage{chicago} \bibliographystyle{chicago}
Entries with multiple authors can be problematical, e.g., \shortcite{TLC94},
\shortcite{LGC97} and \shortcite{test97} or worse \shortcite{TLC94,LGC97,test97}.

\bibliography{tlc,tlc-ex}
```

Entries with multiple authors can be problematical, e.g., (Goossens et al. 1994), (Goossens et al. 1997) and (Goossens et al. 1997) or worse (Goossens et al. 1994; Goossens et al. 1997; Goossens et al. 1997).

References

Goossens, M., F. Mittelbach, and A. Samarin (1994). *The LaTeX Companion*. Tools and Techniques for Computer Typesetting. Reading, MA, USA: Addison-Wesley Longman.

Goossens, M., S. Rahtz, and F. Mittelbach (1997). *The LaTeX Graphics Companion: Illustrating Documents with TeX and PostScript*. Tools and Techniques for Computer Typesetting. Reading, MA, USA: Addison-Wesley Longman.

Goossens, M., B. User, J. Doe, et al. (1997). Ambiguous citations. Submitted to the IBM J. Res. Dev.

16-1-1

In the above example the bibliography is produced from the sample BibTeX databases `tlc.bib` and `tlc-ex.bib` that we introduced in the previous chapter in Figure 15.1 on pages 382–383 and Figure 15.2 on page 391. Those databases are also used in most examples throughout this chapter. Above we applied the BibTeX style `chicago` and its accompanying support package to it, a style that aims to implement a bibliography and reference layout as suggested by *The Chicago Manual of Style* [40].

One way to resolve such ambiguous citations is to use more or all author names in such a case, although that approach can lead to lengthy citations and is not feasible if the number of identical authors exceeds a certain limit. Another solution is to append a, b, and so on, to the year, even though the citations are actually for different author groups. This strategy is, for example, advocated in [28]. If the bibliography is compiled manually, as outlined in Section 15.1, this result can be easily achieved.

When using BIBTEX, you have to use a BIBTEX style file that recognizes these cases and provides the right data automatically. For example, the style `chicago` cannot be used in this case, but all BIBTEX styles produced with custom-bib (see Section 15.7.2) offer this feature:

Entries with multiple authors might be problematical, e.g., Goossens et al. [1997a] and Goossens et al. [1997b] or even Goossens et al. [1997a,b]. But then they might not.

References

M. Goossens, S. Rahtz, and F. Mittelbach. *The LATEX Graphics Companion: Illustrating Documents with TEX and PostScript*. Tools and Techniques for Computer Typesetting. Addison-Wesley Longman, Reading, MA, USA, 1997a. ISBN 0-201-85469-4.

M. Goossens, B. User, J. Doe, et al. Ambiguous citations. Submitted to the IBM J. Res. Dev., 1997b.

```
\usepackage{natbib}
\bibliographystyle
          {abbrvnat}
Entries with multiple
authors might
be problematical,
e.g., \cite{LGC97} and
\cite{test97} or even
\cite{LGC97,test97}.
But then they might not.

\bibliography{tlc,tlc-ex}
```

16-1-2

When using biblatex with biber, the citations are almost[1] always unambiguous from the start. The package adds, if needed, initials, given names, more authors, or an extra year marker. The behavior can be fine-tuned with the options `uniquelist`, `uniquename`, `maxnames`, `maxcitenames`, and `maxbibnames`. The settings in the next example force short author lists and automatically append letters to the year if needed. Details on the customization possibilities of the biblatex package are covered in Section 16.7 starting on page 541.

Entries with multiple authors might be problematical, e.g., Goossens et al. 1997b and Goossens et al. 1997a or even Goossens et al. 1997b; Goossens et al. 1997a. But then they might not.

```
\usepackage[style=authoryear,
  uniquelist=false,maxnames=1]{biblatex}
\addbibresource{tlc.bib}
\addbibresource{tlc-ex.bib}
Entries with multiple authors might be
problematical, e.g., \cite{LGC97} and
\cite{test97} or even \cite{LGC97,test97}.
But then they might not.
```

16-1-3

With just `\usepackage[style=authoryear]{biblatex}` it would have produced the far lengthier result instead:

Entries with multiple authors might be problematical, e.g., Goossens, Rahtz, and Mittelbach 1997 and Goossens, User, Doe, et al. 1997 or even Goossens, Rahtz, and Mittelbach 1997; Goossens, User, Doe, et al. 1997. But then they might not.

16-1-4

[1]It can fail if two references have identical authors *and* title and an author-title style is used.

CHAPTER 17

LaTeX Package Documentation Tools

In this chapter we describe the doc system, a method to document LaTeX macros and environments. A large proportion of the LaTeX code available is documented using its conventions and support tools. The underlying principle is that LaTeX code and comments are mixed in the same file and that the documentation or the stripped package file or files are obtained from the latter in a standard way. In this chapter we explain the structure that these files should have and show how, together with the program DOCSTRIP, you can build self-installing procedures for distributing your LaTeX packages and generating the associated documentation. The chapter also helps you understand the code written by others, install it with ease, and produce the documentation for it (not necessarily in that order).

The third section then introduces the l3build program, which offers a flexible development environment. It supports a package developer in all important steps: code and documentation development, testing, and all aspects of the release management including upload to CTAN. It is the workflow environment used by the LaTeX Project Team for the code that they manage.

We end the chapter with a section on version control systems and explain how to extract and use their information with LaTeX. Applying version control methods is definitely useful for any larger documentation project, but in fact is advisable for any document that goes through a number of “revisions”.

17.1 doc — Documenting LATEX and other code

The idea of integrated documentation was first employed by Donald Knuth when he developed the TEX program using the WEB system, which combines Pascal-like meta source code and documentation. Thanks to his approach, it was particularly easy to port TEX and its companion programs to practically any computer platform in the world.

Subsequently, authors of LATEX packages started to realize the importance of documenting their LATEX code. Many now distribute their LATEX macros using the framework defined with the doc package (by Frank Mittelbach) and its associated DOCSTRIP utility (originally by Frank Mittelbach with later contributions by Johannes Braams, Denys Duchier, Marcin Woliński, and Mark Wooding).

The doc package was written roughly 30 years ago (version 1 appeared in 1988, predating LATEX 2ε). Since then it has been extensively used to document the LATEX kernel and most of the packages that are available on today's LATEX distributions. The core code of version 2 has existed since 1998, and that version was distributed with LATEX until 2022. At the TUG conference in Rio 2018 the author sketched out plans for a version 3 to improve it in several respects, but given it is so widely used, any change would need to be very light-weight, basically adding hyperlink support and adding a way to provide additional doc elements (not just macros and environments), with full compatibility to support all the existing uses.[1]

Both systems together allow LATEX code and documentation to be held in the same TEX source file. The obvious advantage is that a sequence of complex TEX instructions becomes easier to understand with the help of comments inside the file. In addition, updates are more straightforward because only a single source file needs to be changed.

The doc package provides a set of commands and establishes some conventions that allow specially prepared source files to contain both code and its documentation intermixed with each other. This is discussed in Sections 17.1.1 to 17.1.3.

To produce the documentation you need a driver (file) that loads the doc package and then interprets the source file. In its simplest form the driver for the documentation is an external file. However, the driver is more commonly made part of the source file so that all you have to do to produce the documentation is to run the source file through LATEX. The possibilities are discussed in detail in Section 17.1.4.

To produce a ready-to-run version of your code you need to first process the source package with DOCSTRIP (see Section 17.2). This step is usually implicitly done by providing an `.ins` file that is run through LATEX. The most important commands and concepts are discussed in the next sections. Tables 17.1 to 17.5 on pages 595–597 provide an overview of all doc user commands. Further details on any of them can be found in the documented source `doc.dtx` of the doc package, which can also serve as a prime (though somewhat aged) example of the doc system.

[1]If I restarted from scratch, I would do a lot of things differently now, and in fact several other people have tried to come up with better solutions. However, as the saying goes, a bad standard is better than none, so doc has prevailed, and changing it now in incompatible ways would be a nonstarter. Some of the ideas for the extensions have been borrowed from Didier Verna's DoX package, even though I did not use the document-level interfaces.

17.1.1 General conventions for the source file

A LaTeX file to be used with the **doc** system consists of *documentation parts* intermixed with *code parts*. Every line of a documentation part starts with a percent sign (`%`) in the first column. It can contain arbitrary TeX or LaTeX commands, but the `%` character cannot be used as a comment character. User comments are created by using the `^^A` character or, if you prefer, the `^^X` character instead. Longer text blocks can be turned into comments by surrounding them with `%\iffalse` ... `%\fi`. All other parts of the file are called code parts. They contain the code described in the documentation parts.

Depending on how the code parts are structured, it is possible to use such a file directly with LaTeX, although these days this is seldom done. Instead, DOCSTRIP is typically used to produce the production files. If the former approach is taken, LaTeX bypasses the documentation parts at high speed and pastes the macro definitions together, even if they are split into several code parts.

On the other hand, if you want to produce the documentation of the macros, then the code parts should be typeset verbatim. This is achieved by surrounding these parts by the `macrocode` environment.

```
%␣␣␣␣\begin{macrocode}
    ⟨one or more lines of code⟩
%␣␣␣␣\end{macrocode}
```

It is mandatory that you put *exactly* four spaces between the `%` character and `\end{macrocode}` and no space between `\end` and its argument. The reason is that when LaTeX is processing the `macrocode` environment, it is actually looking for that particular string and not for the command `\end` with the argument `macrocode`.

Inside a code part all TeX commands are allowed. Even the percent sign can be used to suppress unwanted spaces at the ends of lines.

If you prefer, you can use the `macrocode*` instead of the `macrocode` environment. It produces the same results except that spaces are displayed as ␣ characters when the documentation is printed.

17.1.2 Describing new macros and environments

Most packages contain commands and environments to be employed by users in their documents. To provide a short manual describing their features, a number of constructs are offered by the **doc** package.

```
\DescribeMacro[keys]{\cmd_1,...}   \Describe⟨element⟩[keys]{entry_1,...}
```

The `\DescribeMacro` command takes one argument containing a comma-separated list of command names, which are shown in the margin, and for which special (usage) index entries are generated, for example,

```
% \DescribeMacro{\DocInput,\IndexInput}
% Finally the \meta{input commands} part ...
```

In the optional argument you can specify either `noindex` or `noprint` to suppress indexing or printing for that particular instance. Using both would be possible too, but pointless because then the command would do nothing.

A similar command, `\DescribeEnv`, can be used to indicate that at this point one or more LaTeX environments are being explained, and if you declare additional doc elements, e.g., ⟨*element*⟩, then `\Describe`⟨*element*⟩ becomes available too.

`\begin{macro}[`*keys*`]{\`cmd_1`,...,\`cmd_n`}`
`\begin{environment}[`*keys*`]{`env_1`,...,`env_n`}`

To describe the definition of one or more commands, you use the `macro` environment. It takes one mandatory argument: a comma-separated list of the new commands. This argument is also used to print the names in the margin and to produce appropriate index entries. The index entries for usage (produced by `\Describe...`) and for definition are different, which allows for easy reference. Here is an example taken from the sources of the doc package itself:

```
% \begin{macro}{\check@angle}
%    Before looking ahead for the |<| the |%| is gobbled by the
%    argument here.
%    \begin{macrocode}
\def\check@angle#1{\futurelet\next\ch@angle}
%    \end{macrocode}
% \end{macro}
```

The optional *keys* argument lets you suppress indexing or printing of the command names in the margin, through specifying either `noindex` or `noprint` for that particular instance. If any such setting is made on the environment level, it overwrites whatever default was given when the doc element was defined or when the package was loaded.

By default the commands listed in the mandatory argument of the `macro` environment are not allowed to be declared with `\outer` (which is a plain TeX concept not officially supported by LaTeX). However, should you really want to document such a command, you can do so by specifying the key `outer` in the optional *keys* argument.

Another environment, with the unimaginative name `environment`, documents the code of environments. It works like the `macro` environment but expects the name or names of environments as its argument.

These two (`macro` and `environment`) are the doc elements supported out of the box by the package. How to define further elements is covered in Section 17.1.6 on page 592.

`\MakeShortVerb{\`*c*`}`   `\MakeShortVerb*{\`*c*`}`   `\DeleteShortVerb{\`*c*`}`

When you have to quote a lot of material verbatim, such as command names, it is awkward to always have to type `\verb+...+`. Therefore, the doc package provides an abbreviation mechanism that allows you to pick a character *c*, which you plan to use

APPENDIX A

LaTeX Overview for Preamble, Package, and Class Writers

This appendix gives an overview of the basic programming concepts underlying the LaTeX formatter. We explain how to define new commands and environments, including those with optional arguments and other specialized input syntax. We discuss how LaTeX handles counters and their representation; we also introduce horizontal and vertical space parameters and explain how they are handled.

The third section reviews the important subject of LaTeX boxes and their use. A good understanding of this topic is very important to fully appreciate and exploit the information presented in this book.

In the fourth section we then take a deeper look at LaTeX's new hook management that was introduced in 2020. The fifth section is devoted to two package files, calc and ifthen, that make calculations and building control structures with LaTeX easier. They have been used in many examples of LaTeX code throughout this book.

Finally, we describe in detail the LaTeX interfaces that allow you to define your own packages and class files.

A.1 Linking markup and formatting

This section reviews the syntax for defining commands and environments with LaTeX. It is important that you exclusively use the LaTeX constructs described below, rather than the lower-level TeX commands. Then, not only are you able to take advantage of LaTeX's consistency checking, but your commands are also portable, (probably) without modification, to future versions of LaTeX.

LaTeX offers three major ways to define commands and environments. Simple commands can be defined with `\newcommand` and `\newenvironment` that have been available for several decades. These are discussed first.

More recently the `\NewDocumentCommand` and `\NewDocumentEnvironment` declarations were added to LaTeX. They allow you to easily define commands and environments with a richer argument structure, not only involving a single optional argument, but also those with multiple optional arguments, special argument delimiters, and much more. This is discussed in Section A.1.4 on page 632.

Finally, there are low-level methods of TeX, through `\def`, `\let`, and similar commands, and, of course, all the declarations from the L3 programming layer (formerly known as expl3), e.g., `\cs_new:Npn` and many more. None of that is covered in this book because these constructs should be used only for low-level programming, and their discussion would easily fill a book by itself.

A.1.1 Command and environment names

Commands

In current LaTeX, whether used with pdfTeX or with one of the Unicode engines, it is possible to enter accented characters and other non-ASCII symbols directly into the source, so it would seem reasonable to expect that such characters could also be used in command and environment names (e.g., `\größer`). However, this is not the case — LaTeX multicharacter command names should be built from basic ASCII letters (i.e., `a`...`z` and `A`...`Z`).[1] This means that `\vspace*` is actually not a command by itself; rather, it is the command `\vspace` followed by the modifier `*`. Technically, you could write `\vspace␣*` (as the space is ignored) or even put the `*` on the next line of your document.[2]

Environments

On the other hand, names of environments are different. In this case the `*` is part of the name, and spaces preceding it are not ignored. Thus, when writing something like `\begin{figure␣*}`, the space would become part of the name, and this is therefore not recognized as the start of a `figure*` environment. This is due to implementation details and seems to indicate that with environment names some additional ASCII characters work. For example:

```
\newenvironment{foo.bar:baz␣with␣space}{}{}
```

However, this is not true in general because, depending on additional packages being

[1] Strictly speaking this is not true, as TeX can be configured to support other configurations. There are, however, valid reasons why this is not being done for standard LaTeX. Some of these reasons are discussed in Section 9.9 describing LaTeX's encoding model.

[2] It is bad style to use this in your documents, but there is unfortunately no way to prevent it.

loaded, such environment names may no longer be recognized or may produce strange errors. Thus, it is best not to explore that implementation (mis)feature and instead to rely on officially supported names — those containing only lowercase and uppercase letters and the star character.

Citation and label keys

Strictly speaking, `\cite` and `\label` keys have (or had) the same kind of restriction. Nevertheless, it has become common practice to use keys containing colons (e.g., `sec:cmds`) so that most packages provided extra support to allow for at least the colon character in such keys. Around 2020, when LaTeX moved to UTF-8 as its default input, that restriction was lifted as part of the necessary rewrite so that now most characters outside the ASCII range can be safely used in label keys. However, characters used in LaTeX's syntax (e.g., `$`, `_`, or `#`) can never be used in names, whether they are keys, counters, environments, or multicharacter command names.

Control symbols

With single-character command names, the situation is different again: any (single) character can be used. Symbols that are not "letters" cannot participate in multiletter command names, e.g., `\foo$bar` would be interpreted as the command `\foo` followed by the start of a math formula (signaled by `$`) followed by the (math) characters `b`, `a`, and `r`. Any following text is then also typeset in math mode. However, such symbols can appear directly after the backslash and form a so-called control symbol; e.g., `\$` is perfectly valid.

In contrast to multiletter commands, such control symbols do not ignore spaces after them, because that is what is normally desired in situations like `30\%` or `cut \& paste`. Note, however, that something like `\S` would not be a control symbol but a multiletter command, consisting of a name with only a single letter and thus spaces are ignored after it, as shown in the following example:

$ 100 is different from $100. You have to write $100 to avoid the space! But §1.2 and §1.2 give the same spacing.

A-1-1

```
\$ 100 is different from \textdollar 100.
You have to write \$100 to avoid the space!
But \S 1.2 and \S1.2 give the same spacing.
```

Categories of LaTeX commands

LaTeX commands (i.e., those constructs starting with a backslash) are classified into three basic categories: document-level commands, package and class writer commands, and internal "kernel" commands.

Document-level commands

- Document-level commands, such as `\section`, `\emph`, and `\sum`, usually have (reasonably) short names, by convention all in lowercase.

Class and package writer commands

- Class and package writer commands, by convention, have longer mixed-case names, such as `\InputIfFileExists` and `\RequirePackage`. Some of them can be usefully applied in the document source, but many stop working after `\begin{document}` has been processed, producing an error message if you try.

Internal LaTeX commands

- Traditionally most of the internal commands used in the LaTeX implementation, such as `\@tempcnta`, `\@ifnextchar`, and `\z@`, contain `@` in their name. This effectively prevents these names from being used in documents for user-defined commands. However, it also means that they cannot appear in a document, even in the preamble, without taking special precautions.

Modern internal commands use the L3 programming layer where commands use "`_`" and "`:`" as part of their names, which makes them also (deliberately) unusable in documents.

Careful with internal commands!

As a few of the examples in this book demonstrate, it is sometimes necessary to have such bits of "internal code" in the preamble. The commands `\makeatletter` and `\makeatother` make this easy to do; the difficult bit is to remember to add them — failure to do so can result in some strange errors. For an example of their use, see page →I 210. Note that package and class files should never contain these commands: `\makeatletter` is not needed because this is always set up when reading such files, and the use of `\makeatother` would prematurely stop this behavior, causing all kinds of havoc. For modern L3 programming layer code (`expl3`), there are `\ExplSyntaxOn` and `\ExplSyntaxOff` that serve the same purpose.

The distinction between internal and public commands

Unfortunately, for historical reasons the distinction between these categories is often blurred. For example, `\hbox` is an internal command that should preferably be used only in the LaTeX kernel, because it does not deal properly with color changes in its argument, whereas `\m@ne` is the constant `-1` and would have been better named `\MinusOne` back then.

Nevertheless, this rule of thumb is still useful: if a command has `@` in its name, then it is not part of the supported LaTeX language — and its behavior may change in future releases! Any such command should be used with great care. On the other hand, mixed-case commands or those described in the *LaTeX Manual* [106] are guaranteed to be supported in future releases of LaTeX.

In the L3 programming layer there is much more standardization. Command names follow a predictable pattern, and what is supported and usable elsewhere is well defined: any name starting with `\__` is private and should not be used outside its module; all others are officially part of the programming layer. Similarly, private (local) variables start with `\l__`, whereas public ones have only one underscore, etc.

A.1.2 Defining simple commands

It is often advantageous to define new commands (e.g., for representing repetitive input strings or recurring combinations of commands). Here we describe how do this for simple cases, where the new commands take at most one optional argument. This is the method introduced with LaTeX 2_ε in 1994. How to define commands with more complex argument structures is shown in Section A.1.4.

Complexity comes with a price tag in terms of processing speed: so for "simple" tasks we recommend simple tools, i.e., the declarations discussed here. However, already when defining commands with one optional argument, you should consider using the declarations discussed later.

`\newcommand{`*cmd*`}[`*narg*`]{`*command definition*`}`

A new command is defined using the `\newcommand` declaration. The number of arguments is in the range $0 \leq narg \leq 9$. If your new command has no arguments,

APPENDIX B

Tracing and Resolving Problems

In an ideal world all documents you produced would compile without problems and give high-quality output as intended. If you are that lucky, there will be no need for you to consult this appendix, ever. However, if you run into a problem of some kind, the material in this appendix should help you to resolve your problem easily.

We start with an alphabetical list of all error messages, those after which LATEX stops and asks for advice. “All” in this context means all LATEX kernel errors (their text starts with `LaTeX Error:` or `LaTeX` ⟨*module*⟩ `Error`), practically all TEX errors (i.e., those directly produced by the underlying engine), and errors from the packages amsmath, babel, calc, color, fontenc, graphics, graphicx, and inputenc. Errors reported by other packages — those that identify themselves as

```
! Package ⟨package⟩ Error: ⟨error text⟩
```

where ⟨*package*⟩ is not one of the above — are not included. For such errors you should refer to the package description elsewhere in the book or consult the original package documentation.

But even if there are no real errors that stop the processing, warning and information messages might be shown on the terminal or in the transcript file. They are treated in Section B.3, where you will find all LATEX core messages and all relevant TEX messages that may need your attention, together with an explanation of their possible causes and suggestions on how to deal with them.

The final section deals with tools for tracing problems in case the error or warning information itself is not sufficient or does not exist. We will explore ways to display command definitions and register values and then take a look at diagnosing and solving page-breaking problems. This is followed by suggestions for identifying and solving paragraph-breaking problems and includes a brief look at two LuaTEX-only packages for dealing with hyphenation issues. We finish with a description of the trace package, which helps in thoroughly tracing command execution, if your own definitions or those of others produce unexpected results.

Some of the material in this appendix can be considered "low-level" TEX, something that, to the author's knowledge, has never been described in any other LATEX book. It is, however, often important information. Directing the reader to books like *The TEXbook* does not really help, because most of the advice given in books about plain TEX is not applicable to LATEX or produces subtle errors when used. We therefore try to be as self-contained as possible by offering all relevant information about the underlying TEX engine as far as it makes sense within the LATEX context.

B.1 Error messages

When LATEX stops to display an error message, it also shows a line number indicating how far it got in the document source. However, because of memory considerations in the design of TEX itself, it does not directly show to which file this source line number belongs. For simple documents this is not a problem, but if your document is split over many files, you may have to carefully look at the terminal output or the transcript file to identify the file LATEX is currently working on when the error occurs.[1]

Finding the source line of an error

Whenever LATEX starts reading a file, it displays a "(" character that is immediately followed by the file name. Once LATEX has finished reading the file, it displays the matching ")" character. In addition, whenever it starts preparing to output a page, it displays a "[" character followed by the current page number. Thus, if you see something like

```
(./trial.tex [1] (./ch-1.tex [2] [3] (./table-1.tex [4] [5]) [6]
! Undefined control sequence.
<argument> A \textss
                    {Test}
l.235 \section{A \textss{Test}}
                               \label{sec:test}
?
```

you can deduce that the error happened inside an argument of some command (`<argument>`) and was detected when LATEX gathered material for page 7. It got as far as reading most of line 235 in the file `ch-1.tex`. In this example the error is readily

[1] A useful little helper for this is the structuredlog package. Once loaded, it uses the file hooks to identify the start and end of file-reading actions in the `.log` with lines such as "`== (LEVEL 2 START) table-1.tex`" and corresponding `STOP` lines. Thus, to find out where some error has occured you only have to search backwards for the string "`(LEVEL`" and if that is a `START` line you are done; otherwise, you have to search further for the `START` line with the next lower level number.

visible in the source line: `\textsf` was misspelled as `\textss` inside the argument to the `\section` command. In some cases, however, the relationship between error and source line is blurred or even nonexistent.

For example, if you define `\renewcommand\thepart{\Alp{part}}`, then the typo appears only when you use the `\part` command that executes your definition. In that case you get

```
! Undefined control sequence.
\thepart ->\Alp
                {part}
l.167 \part{Test}
```

In this particular case the actual error is not on line 167 and most likely not even in the current file — the `\part` command merely happens to call the faulty definition of `\thepart`.

Sometimes an error is detected by LaTeX while it is preparing a new page. Because this is an asynchronous operation, the source line listed in the error message is of no value whatsoever. So if you do not understand how the error should be related to the source line, you may well be right — there is, indeed, no relationship. Here is an example:

```
! Undefined control sequence.
\thepage ->\romen
                  {page}
l.33 T
      his is a sample text to fill the page.
```

One way to obtain additional information about an error (or information about how LaTeX intends to deal with it) is to press the key `h` in response to the `?` that follows the error message. If used with a TeX error such as the one above, we get

```
? h
The control sequence at the end of the top line
of your error message was never \def'ed. If you have
misspelled it (e.g., `\hobx'), type `I' and the correct
spelling (e.g., `I\hbox'). Otherwise just continue,
and I'll forget about whatever was undefined.
```

You probably already see the problem with advice coming directly from the TeX engine: you may have to translate it, because it often talks about commands that are not necessarily adequate for LaTeX documents (e.g., for `\def` you should read `\newcommand` or `\NewDocumentCommand`). With real LaTeX errors this is not the case, though here you sometimes get advice that is also not really helpful:

```
You're in trouble here.  Try typing  <return>  to proceed.
If that doesn't work, type  X <return>  to quit.
```

Well, thank you very much; we already knew that! Typing `h` ⟨*return*⟩ is, however, worth a try, because there are many messages with more detailed advice.

Displaying the stack of partially expanded macros

Another way to get additional information about an encountered error is to set the counter `errorcontextlines` to a large positive value. In that case LaTeX lists the stack of the current macro executions:

```
 1  ! Undefined control sequence.
 2  \thepage ->\romen
 3                    {page}
 4  \@oddfoot ->\reset@font \hfil \thepage
 5                                         \hfil
 6  \@outputpage ...lor \hb@xt@ \textwidth {\@thefoot
 7                                                    }\color@endbox }}\globa...
 8
 9  \@opcol ...lumn \@outputdblcol \else \@outputpage
10                                                    \fi \global \@mparbotto...
11  <output> ...specialoutput \else \@makecol \@opcol
12                                                    \@startcolumn \@whilesw...
13  <to be read again>
14                     T
15  l.33 T
16        his is a sample text to fill the page.
```

You read this bottom up: LaTeX has seen the `T` (lines 15 and 16) but wants to read it again later (`<to be read again>`, lines 13 and 14) because it switched to the output routine (`<output>`). There it got as far as executing the command `\@opcol` (lines 11 and 12), which in turn got as far as calling `\@outputpage` (lines 9 and 10), which was executing `\@thefoot` (lines 6 and 7). Line 4 is a bit curious because it refers to `\@oddfoot` rather than `\@thefoot` as one would expect (`\@thefoot` expands to `\@oddfoot`, so it is immediately fully expanded and not put onto the stack of partially expanded macros). Inside `\@oddfoot` we got as far as calling `\thepage`, which in turn expanded to `\romen` (lines 2 and 3), which is finally flagged as an undefined command (line 1).

Fortunately, in most cases it is sufficient only to display the error message and the source line. This is why LaTeX's default value for `errorcontextlines` is `-1`, which means not showing any intermediate context.

Error during expansion

Errors can occur when LaTeX is doing expansions rather than typesetting. This is a problematical time, because in that mode TeX is not able to do any assignments, which is a prerequisite to display LaTeX error or warning messages with useful help texts. It is also difficult to stop the expansion in time when an error condition is identified without stopping it when there is no error condition. Nevertheless, with some gymnastics it is possible to produce reasonable error messages (as long as you ignore the help text that you get by typing `h` in response to the error). The trick used by LaTeX is to produce a low-level TeX error, but to do this in a way that it nearly looks like a normal LaTeX error.

```
! Use of \??? doesn't match its definition.
<argument> \???
      ! LaTeX cmd Error: Required argument missing for \foo.
l. ...  }
```

APPENDIX C
Going beyond

While we certainly hope that your questions have been answered in this book, we know that this cannot be the case for all questions. This appendix tries to help in that unfortunate case.

The first section deals with the fundamental problem that our book (despite being rather large) is not meant for beginners. To make good use of it, you already need some basic knowledge of LaTeX. If this is missing, then there is a great online course to teach you the foundations.

For questions related to specific packages discussed in the book, it can be helpful to read the original documentation provided with the package. Appendix C.2 suggests ways to find that documentation on your system.

Appendix C.3 then shares how to find online information to research questions that you may have, and how to interact with people in the TeX community when you experience problems.

This is followed in C.4 by explaining how best to obtain LaTeX packages that are missing on your installation. It also gives a brief overview of TeX distributions and LaTeX services in the cloud.

The final section talks about the community itself and how you can join it to make a difference and keep the ship afloat.

C.1 Learn LaTeX — A LaTeX online course for beginners

Worry-free experimentation in a browser

If you are new to LaTeX and picked up this book hoping that it also covers the LaTeX basics (which it understandably does not, if you look at the current page number) or if you need to refresh your memory on the basics, we recommend that you take a look at the Learn LaTeX website at `https://learnlatex.org`, which offers a great online course, teaching you the fundamentals and allowing you to experiment with LaTeX online — even without installing the system locally.

Available in several languages

The Learn LaTeX project was born out of the desire to provide any easy-to-access approach for LaTeX beginners. The site has been designed and implemented by members of the LaTeX Project Team, primarily by David Carlisle and Joseph Wright, as well as a larger group of volunteers who helped shape the content and did a marvelous job of translating it to several languages so that the full course is now available in Catalan, English, Español, French, German, Marathi, Portuguese, and Vietnamese — and, who knows — soon perhaps additional languages.

18 lessons to get you going

The curriculum consists of eighteen easy-to-consume lessons with interactive examples and exercises that can be processed and modified (!) in the browser, covering the basics that you need to get started with LaTeX and that underpin all the information provided in the current book. The central idea of the site is to get people started with one of the most common LaTeX tasks: write some academic document such as an article or report. The topics start with simple LaTeX structure, cover things such as including images or building tables, and move on to longer documents and bibliographies. Typesetting mathematics is clearly a core LaTeX strength, but here the course remains deliberately lightweight and touches on only a few concepts; it is enough to get you going but certainly not enough to write a complex math paper — the thoughts being that details like those covered in Chapter 11 are better mastered in a second step.

… more on each topic

Each lesson comes with an extra “More on this topic” page that can be bypassed when working through the course for the first time but that offers additional details if you are interested in a particular topic. For example, the lesson on tables focuses on the array package as the fundamental extension to LaTeX’s basic `tabular` environment and explains the main concepts such as the different kinds of column specifiers, how to add rules, and how to merge cells. The “More on this topic” page then goes into styling columns with the help of specifiers, manipulating space between columns, or producing customized rules and much more.

Being closely linked with the LaTeX Project Team that maintains and develops LaTeX for you, the site is expected to stay up-to-date (in contrast to many other sites sprinkled across the Internet that have been set up with good intentions, but which then sadly declined in quality over the years). By offering a couple of ways to process the examples online (or off-line because the examples are plain text that is easily copied) and by presenting the course in several languages, it provides a low-barrier introduction to learning LaTeX. As it stands, it is a great resource for starting with LaTeX, and even if you personally have no need for it at your level of experience, it may be good to take a look and see how it functions. There may be a need to convince a colleague or friend to take the first steps with LaTeX, and this site may just convince them that LaTeX is not too hard to learn and worth the trouble — and for the rest there are books like the one you are currently holding in your hands.

Recommend this to new users

C.2 Finding information available on your computer

When you want to use a LaTeX package, it would be nice if you could study the documentation without having to remember where the relevant files are located on your TeX system. Two important command-line tools exist to help you in your search: kpsewhich and texdoc. The first is most important if you want to study code, while the second helps you to easily find available documentation in your distribution.

C.2.1 kpsewhich — Find files the way TeX does

When TeX loads files (at least those loaded with `\input` or similar), it uses a special search library named kpathsea by Karl Berry that determines where to look and which file to load in case there is more than one with the same name.

The file name loaded is written into the `.log` file so you can find out what was loaded there, but this is not necessarily the most convenient way given that the transcript file contains a lot of other information. It also would not help if you ask yourself the question "which `array.sty` would LaTeX load if I use `\usepackage{array}`?" unless you first make a test file, run it through LaTeX, and then look at the transcript.

To help with this, the library is also available as the standalone command-line tool kpsewhich. In most cases it is enough to pass it the name of the file you are interested in, e.g., `kpsewhich array.sty`, and it responds with the full path for the file:

```
/usr/local/texlive/2022/texmf-dist/tex/latex/tools/array.sty
```

You can load it into your editor if you want to look at the code. You can also call kpsewhich with the option `-all`, in which case it returns

```
/usr/local/texlive/2022/texmf-dist/tex/latex/tools/array.sty
/usr/local/texlive/2022/texmf-dist/tex/latex-dev/tools/array.sty
```

i.e., all files with that name that it knows about.

You may wonder when (if ever) the other files are used by LaTeX. The answer is it depends on the program name used for running LaTeX. For example, if you use any of the development formats, e.g., pdflatex-dev, the array package from the `latex-dev` directory is used, because the search library has different search rules based on the program that is initiating the search. To mimic that with kpsewhich, pass it the option `-progname=`⟨*prog*⟩, and you get the result that would be returned to ⟨*prog*⟩ for the search:

```
kpsewhich -progname=pdflatex-dev array.sty
```

would return

```
/usr/local/texlive/2022/texmf-dist/tex/latex-dev/tools/array.sty
```

i.e., the package version for the next LaTeX release.

If you want to find out where the LaTeX format file resides that is used by LaTeX, it is not enough to just specify the name of the format file. You also have to tell `kpsewhich` which engine is loading it, using the option `-engine=`⟨*engine*⟩. Possible engine names are `pdftex`, `xetex`, or `luahbtex`. Together with the `-all` option, this is a great help if you receive a "Mismatched LaTeX support files" error; see page 730.

The program offers several other options; a list can be obtained with `-help`, but most of them are needed only for developers who set up a TeX distribution. The one that may be of interest occasionally is `-debug=32`. This produces detailed tracing output about how `kpsewhich` does its search and where it looks, which can be helpful if LaTeX seems to be unable to find a file that you know is on your system. The `-debug` option can be given other numeric values to show other aspects of its behavior. For details on that and other features of TeX's search library, take a look at the documentation with `texdoc kpathsea`.

C.2.2 texdoc — A command-line interface to local TeX information

The `texdoc` program has a long history: it started out as a Unix shell script developed in the early nineties by Thomas Esser. The first Lua-based version was then written by Frank Küster with contributions by Reinhard Kotucha. It offered some operating system independence, because Lua, through LuaTeX, is available with any TeX (decent) distribution without the need to install separate programs.

Today's greatly enhanced version is developed and maintained by Manuel Pégourié-Gonnard and ASAKURA Takuto and the TeX Live team.

Initially you had to know the exact name of the documentation except for the file extension, so it was quite similar to `kpsewhich` in that regard. The current implementation, however, has a much more general search mechanism that accounts for the fact that many developers use schemes such as *package*`-doc.pdf`, *package*`-manual.pdf`, etc., for their documentation. It also has an alias mechanism through which individual documentation files can be associated with some keyword(s) people are likely to search for. For example, the file `interface3.pdf` (the interface documentation for the L3 programming layer) is one of the results if you search for `expl3`. This is managed through a configuration file that is part of the TeX distribution and regularly updated. You can also have your own configuration file as explained in the program documentation.

Opening the best result for viewing

Most of the time all you have to do is to pass `texdoc` the name of a package as an argument, e.g., `texdoc fewerfloatpages`, and it replies by opening the (what it thinks) best result in a suitable viewer for you; in this case it would open the file `fewerfloatpages-doc.pdf`.

Getting a list of "good" results to choose from

If this turns out not to be the documentation you have been looking for, your next possibility is to use it with the option `-l`. If you do that, it replies with a list of "useful" documentation files based on your input, and you can then choose one of them to be opened. For example, `texdoc -l array` would return five results: the official package documentation, the (possibly updated) version in the LaTeX development release, as well as a French translation of the package documentation.

Getting a list of "all" results

If that is still not what you want, you can use `-s` instead. In that case `texdoc` returns all files it has found, even those that it considered to be a bad match. For

Bibliography

[1] Adobe Systems Incorporated. Adobe Type 1 Font Format. Addison-Wesley, Reading, MA, USA, 1990. ISBN 0-201-57044-0.

The "black" book contains the specifications for Adobe's Type 1 font format and describes how to create a Type 1 font program. It explains the specifics of the Type 1 syntax (a subset of PostScript), including information on the structure of font programs, ways to specify computer outlines, and the contents of the various font dictionaries. It also covers encryption, subroutines, and hints. `https://adobe-type-tools.github.io/font-tech-notes/pdfs/T1_SPEC.pdf`

[2] American Mathematical Society. User's Guide to AMSFonts Version 2.2d. Providence, Rhode Island, 2002.

This document describes AMSFonts, the American Mathematical Society's collection of fonts of symbols and several alphabets. `https://www.ctan.org/pkg/amsfonts`

[3] ———. Using the amsthm Package (Version 2.20.3). Providence, Rhode Island, 2017.

The amsthm package provides an enhanced version of LaTeX's `\newtheorem` command for defining theorem-like environments, recognizing `\theoremstyle` specifications and providing a `proof` environment. `https://www.ams.org/arc/tex/amscls/amsthdoc.pdf`

[4] ———. AMS Author Handbook (website). Providence, Rhode Island, 2021.

Entry point to documentation and support packages for preparing articles and books for publication with the American Mathematical Society.

`https://www.ams.org/arc/handbook/index.html`

[5] American Mathematical Society and LaTeX Project. User's Guide for the amsmath Package (Version 2.1), 2020.

The amsmath package, developed by the American Mathematical Society, provides many additional features for mathematical typesetting. It is now maintained by the LaTeX Project team.

Locally available via: `texdoc amsmath`

[6] American Psychological Association. Publication Manual of the American Psychological Association. APA, 7th edition, 2020.

The official style guide of the American Psychological Association (APA) for the preparation of manuscripts for publication, as well as for writing student papers, dissertations, and theses.

`https://apastyle.apa.org/`

[7] Anonymous. "'thanks' note (footnote) placed below right column even though there is enough space on the left". *StackExchange*, 2012.
An example of a question on a strange footnote placement by LaTeX. The answers explains some of the reasons why this can happen. https://tex.stackexchange.com/questions/43294

[8] Apple Inc. "TrueType Reference Manual", 2019.
Apple's reference manual for the TrueType font format.
https://developer.apple.com/fonts/TrueType-Reference-Manual

[9] Donald Arseneau. url.sty—Version 3.4, 2016.
Documentation provided by Robin Fairbairns (1947–2022). Locally available via: texdoc url

[10] Michael Barr. A new diagram package, 2016.
A rewrite of Michael Barr's original diagram package to act as a front end to Rose's xypic; see [55, Chapter 7]. It offers a general arrow-drawing function; various common diagram shapes, such as squares, triangles, cubes, and 3×3 diagrams; small 2-arrows that can be placed anywhere in a diagram; and access to all of xypic's features. Locally available via: texdoc diagxy

[11] Claudio Beccari and Apostolos Syropoulos. "New Greek fonts and the greek option of the babel package". *TUGboat*, 19(4):419–425, 1998.
Describes a new complete set of Greek fonts and their use in connection with babel's greek extension. https://tug.org/TUGboat/tb19-4/tb61becc.pdf

[12] Nelson Beebe. "Bibliography prettyprinting and syntax checking". *TUGboat*, 14(4):395–419, 1993.
This article describes three software tools for BibTeX support: a prettyprinter, syntax checker, and lexical analyzer for BibTeX files; collectively called bibclean.
https://tug.org/TUGboat/tb14-4/tb41beebe.pdf

[13] Barbara Beeton. "Mathematical symbols and Cyrillic fonts ready for distribution". *TUGboat*, 6(2):59–63, 1985.
The announcement of the first general release by the American Mathematical Society of the Euler series fonts. https://tug.org/TUGboat/tb06-2/tb11beet.pdf

[14] Alexander Berdnikov, Olga Lapko, Mikhail Kolodin, Andrew Janishevsky, and Alexei Burykin. "Cyrillic encodings for LaTeX 2_ε multi-language documents". *TUGboat*, 19(4):403–416, 1998.
A description of four encodings designed to support Cyrillic writing systems for the multi-language mode of LaTeX 2_ε. The "raw" X2 encoding is a Cyrillic glyph container that allows one to insert into LaTeX 2_ε documents text fragments written in any of the languages using a modern Cyrillic writing scheme. The T2A, T2B, and T2C encodings are genuine LaTeX 2_ε encodings that may be used in a multi-language setting together with other language encodings.
https://tug.org/TUGboat/tb19-4/tb61berd.pdf

[15] Jens Berger. "The jurabib Package", 2017.
Manual for the jurabib package, translated to English by Maarten Wisse.
Locally available via: texdoc jurabib

[16] Karl Berry. "Filenames for fonts". *TUGboat*, 11(4):517–520, 1990.
This article describes the consistent, rational scheme for font file names that was used for at least the next 15 years. Each name consists of up to eight characters (specifying the foundry, typeface name, weight, variant, expansion characteristics, and design size) that identify each font file in a unique way. https://tug.org/TUGboat/tb11-4/tb30berry.pdf
The latest version of this scheme is available at: https://tug.org/fontname/html/

[17] Javier Bezos. The accents Package, 2019.
Miscellaneous tools for mathematical accents: to create faked accents from non-accent symbols, to group accents, and to place accents below glyphs. Locally available via: texdoc accents

[18] ———. Customizing lists with the enumitem package, 2019.
Extended and customizable list environments. Locally available via: `texdoc enumitem`

[19] Javier Bezos and Johannes Braams. Babel — Localization and internationalization — TeX, pdfTeX, XeTeX, LuaTeX, 2023.
The official user manual for Babel, describing how to use it with different flavors of TeX. A second part covers the complete source code documentation. Locally available via: `texdoc babel`

[20] Charles Bigelow. “Oh, oh, zero!” *TUGboat*, 34(2):168–181, 2013.
A survey of attempts to make “O”, “0”, and similar look-alike characters distinguishable in historical and modern typography.
https://tug.org/TUGboat/tb34-2/tb107bigelow-zero.pdf

[21] ———. “About the DK versions of Lucida”. *TUGboat*, 36(3):191–199, 2015.
A description of the design of the Lucida DK fonts and their history, developed as a response to [98]. https://tug.org/TUGboat/tb36-3/tb114bigelow.pdf

[22] The *Bluebook*: A Uniform System of Citation. The Harvard Law Review Association, Cambridge, MA, 21st edition, 2020. ISBN 978-0-578-66615-0.
The *Bluebook* contains three major parts: part 1 details general standards of citation and style to be used in legal writing; part 2 presents specific rules of citation for cases, statutes, books, periodicals, foreign materials, and international materials; and part 3 consists of a series of tables showing, among other things, which authority to cite and how to abbreviate properly.
Can be ordered at: https://www.legalbluebook.com

[23] Johannes Braams. “Babel, a multilingual style-option system for use with LaTeX’s standard document styles”. *TUGboat*, 12(2):291–301, 1991.
The babel package was originally a collection of document-style options to support different languages. An update was published in *TUGboat*, 14(1):60–62, April 1993.
https://tug.org/TUGboat/tb12-2/tb32braa.pdf
https://tug.org/TUGboat/tb14-1/tb38braa.pdf

[24] Peter Breitenlohner et al. “The eTeX manual (version 2)”, 1998.
The current manual for the eTeX system, which extends the capabilities of TeX while retaining compatibility. While the eTeX engine is considered obsolete these days, its extensions have been implemented in all major modern systems and LaTeX expects them to be available. Thus, the documentation remains relevant. Locally available via: `texdoc etex`

[25] Robert Bringhurst. The elements of typographic style (20th Anniversary edition). Hartley & Marks Publishers, Point Roberts, WA, USA, and Vancouver, BC, Canada, 4th edition, 2013. ISBN 0-88179-212-8 (hardcover).
A very well-written book on typography with a focus on the proper use of typefaces.

[26] ———. Palatino — The Natural History of a Typeface. David R. Godine, Publisher, Boston, 2016. ISBN 978-1-56792-572-2.
Palatino is one of the most widely used typefaces today. Designed by Hermann Zapf (1918–2015), it has been used as a model for many other typefaces. This book discusses its evolution and history and is a fascinating read if you are interested in type design.

[27] Bureau International des Poids et Mesures. SI Brochure: The International System of Units (SI), 9th edition, 2019.
The 9th edition of the brochure explaining and promoting the International System of Units (SI) — the preferred system of units for science, technology, industry and trade. Available in English and French. https://www.bipm.org/en/publications/si-brochure/

[28] Judith Butcher, Caroline Drake, and Maureen Leach. Butcher's Copy-editing: The Cambridge Handbook for Editors, Copy-editors and Proofreaders. Cambridge University Press, New York, forth edition, 2006. ISBN 0-521-40074-0.
A reference guide for all those involved in the process of preparing typescripts and illustrations for printing and publication. The book covers all aspects of the editorial process, from the basics of how to mark a typescript for the designer and the typesetter, through the ground rules of house style and consistency, to how to read and correct proofs.

[29] Florian Cajori. A History of Mathematical Notations. Dover Publications, New York, 1993. ISBN 978-0-486-67766-8.
This classic book on mathematical notations, originally published in two volumes in 1928–1929, devotes a full chapter to several numeral systems from antiquity, including Greek.

[30] David Carlisle. "A LaTeX tour, Part 1: The basic distribution". *TUGboat*, 17(1):67–73, 1996.
A "guided tour" around the files in the basic LaTeX distribution. File names and paths relate to the file hierarchy of the CTAN archives. https://tug.org/TUGboat/tb17-1/tb50carl.pdf

[31] ———. "A LaTeX tour, Part 2: The tools and graphics distributions". *TUGboat*, 17(3):321–326, 1996.
A "guided tour" around the "tools" and "graphics" packages. Note that *The Manual* [106] assumes that at least the graphics distribution is available with standard LaTeX.
https://tug.org/TUGboat/tb17-3/tb52carl.pdf

[32] ———. "A LaTeX tour, Part 3: mfnfss, psnfss and babel". *TUGboat*, 18(1):48–55, 1997.
A "guided tour" through three more distributions that are part of the standard LaTeX system. The mfnfss distribution provides LaTeX support for some popular METAFONT-produced fonts that do not otherwise have any LaTeX interface. The psnfss distribution consists of LaTeX packages giving access to PostScript fonts. The babel distribution provides LaTeX with multilingual capabilities.
https://tug.org/TUGboat/tb18-1/tb54carl.pdf

[33] ———. "XMLTEX: A non validating (and not 100% conforming) namespace aware XML parser implemented in TeX". *TUGboat*, 21(3):193–199, 2000.
XMLTEX is a an XML parser and typesetter implemented in TeX, which by default uses the LaTeX kernel to provide typesetting functionality. https://tug.org/TUGboat/tb21-3/tb68carl.pdf

[34] David Carlisle, editor. Mathematical Markup Language (MathML) Version 4.0. W3C, 1st edition, 2023.
This is the draft specification for a new version of the Mathematical Markup Language; the current version is 3.0 [35]. MathML4 extensions primarily relate to improving accessibility, with new attributes for improving audio rendering. https://www.w3.org/TR/mathml4/

[35] David Carlisle, Patrick Ion, and Robert Miner, editors. Mathematical Markup Language (MathML) Version 3.0. W3C, 2nd edition, 2014.
This is the current specification defining the Mathematical Markup Language; the upcoming version will be [34]. MathML is an XML vocabulary for mathematics, designed for use in browsers and as a communication language between computer algebra systems. The goal of MathML is to enable mathematics to be served, received, and processed on the World Wide Web, just as HTML has enabled this functionality for text. https://www.w3.org/TR/MathML3/

[36] David Carlisle, Patrick Ion, Robert Miner, and Nico Poppelier, editors. Mathematical Markup Language (MathML) Version 2.0. W3C, 2nd edition, 2003.
This is the previous version of the MathML standard [35]. https://www.w3.org/TR/MathML2/

Index of Commands and Concepts

This title somewhat hides the fact that everything (for both volumes) except for names of people is in this one long comprehensive index. To make it easier to use, the entries are distinguished by their "type", and this is often indicated by one of the following "type words" at the beginning of the main entry or a subentry:

> attribute, BIBTEX/biber command, BIBTEX entry type, BIBTEX field, BIBTEX style, boolean, counter, document class, env., file, file extension, folio style, font encoding, hook, key/option, keyword, key, language, length, library, math accent, math symbol, option, package, page style, program, rigid length, syntax, text accent, text symbol, text/math symbol, TEX counter, or value.

type words used in the index

Most "type words" should be fairly self-explanatory, but a few need some explanation to help you find the entries you are looking for.

'option' and 'keyoption' Both type words indicate that the keyword can be used in the optional argument of `\usepackage`. If the option accepts a value, it is marked as a 'keyoption' and otherwise as a classic 'option'. Most packages with 'keyoptions' also offer configuration commands that can be used in the preamble or in the document body — see also next types.

'key' and 'value' Many modern LATEX packages implement a key/value syntax (i.e., ⟨*key₁*⟩`=`⟨*value₁*⟩`,`⟨*key₂*⟩`=`⟨*value*⟩`,...`) as part of the optional argument to `\usepackage`, in arguments of commands or environments, or both. Keywords that can appear to the left of the equal sign are indicated by the type word 'key' (or as type 'keyoption' if allowed in `\usepackage`), those that can appear to the right as 'value'. Sometimes keywords can appear on either side, depending on context, in which case they are indexed according to their use on the particular page.[1]

'syntax' Keywords and strings marked with 'syntax' can appear in arguments of commands but are not part of a key/value pair.

'counter' and 'TEX counter' Names marked as 'counter' are LATEX counters and are altered with `\setcounter`, etc., while 'TEX counters' start with a backslash and use a low-level method for modification.

'length' and 'rigid length' A 'length' register can take values with a `plus` and `minus` component, i.e., can stretch or shrink. In contrast, a 'rigid length' stores only a fixed value. Most lengths in LATEX are flexible, i.e., not rigid.

'text symbol', 'math symbol', and 'text/math symbol' A command is classified as a 'text symbol' if it typesets a glyph for use in text, whereas a 'math symbol' can be used only in math mode and produces an error elsewhere. A few symbols are allowed everywhere and are therefore of type 'text/math symbol'.

In most cases, the actual symbol is also shown in the index entry to help you find the symbol you are looking for more easily (the command names are not always obvious). Note, however, that the glyph shown is only an approximation of reality — in your document it may come out differently depending on the fonts you use.

no type word indication The absence of an explicit "type word" means that the "type" is either a core LATEX command or simply a concept.

Relation to packages or programs

If a particular index entry is defined or used in a special way by a package, then this is indicated by adding the package name (in parentheses) to an entry or subentry. If package aaa builds upon or extends package bbb, we indicate this with (aaa/bbb). There is one "virtual" package name, tlc, which indicates commands introduced only for illustrative purposes in this book. Again, you may see (tlc/bbb), if appropriate.

Interpreting page references

The index contains all entries for both volumes. To which pages the references point is indicated by →I (for part one) and →II (for part two). To save space, these indicators are given only once in each entry. An *italic* page number indicates that the command or concept is demonstrated in an example on that page. When there are several page numbers listed, **blue boldface** indicates a page containing important information about an entry, such as a definition or basic usage. However, **bold** is (normally) not used for concept entries, when all entries are of equal importance, or when there is *only one* page reference.

Sorting within the index

When looking for the position of an entry in the index, you need to realize that both of the characters `\` and `.` are ignored when they come at the start of a command or file extension. Other syntax entries starting with a period are listed in the Symbols section. Otherwise, the index entries are sorted in ASCII order, and the running header gives you an indication where you are in the index. For the rather lengthy Symbols section at the beginning of the index, this is unfortunately of little help because only a few people would know the symbol order in the ASCII encoding. We therefore show the order of symbols in the margin on those pages.

[1]This explains why you might find the same keyword under both type words, but given that one is usually interested only in one type of usage, this distinction is made. This distinction also exhibits the fact that different packages use the same keywords in different ways — an unfortunate side effect of the long history of LATEX package development by independent developers.

Symbols

People

Biographies

Frank Mittelbach

Frank's interest in the automated formatting of complex documents in general, and in LaTeX in particular, goes back to his university days and has always been a major interest, perhaps a vocation, and throughout his IT career certainly a "second job". He studied mathematics and computer science at the Johannes Gutenberg University, Mainz, and afterwards joined EDS, Electronic Data Systems, initially working in a newly formed group for document processing using TeX and other tools and later on in the system management division.

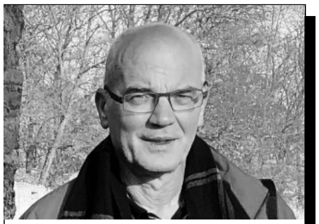
Stanford, 1989 ...

At the TeX Users Group (TUG) conference at Stanford University in 1989, he gave a talk about the (many) problems with LaTeX 2.09, which led to him and a small team (known as the LaTeX Project Team) to take on the responsibility for the further development and maintenance of LaTeX from its original author Leslie Lamport.[1] In the capacity of technical lead of this project, he has overseen the original major release of LaTeX 2_ε in 1994 and the 36 (as of this writing) subsequent releases of this software. Right now he and the team are working on the important multi-year project to make LaTeX automatically produce tagged PDF and conform to accessibility standards.

When EDS was bought by HP he joined the global Enterprise Service Management division and eventually worked there as the European lead architect. In 2015 Frank ended his IT-industry career to fully concentrate on research in automated typesetting and Open Source software development, in particular on the future development of the LaTeX typesetting system.

[1] The history is briefly discussed in Section 1.1 of this edition; for a listing of all LaTeX Project team members through the years see `https://latex-project.org/about/team/`.

Frank is author or coauthor of many and varied LaTeX extension packages, such as AMS-LaTeX, doc, multicol, and NFSS: the New Font Selection Scheme. His publication of many technical papers on LaTeX and on general research results in automated formatting brought him in contact with Peter Gordon from Addison-Wesley. Peter and Frank inaugurated the book series *Tools and Techniques for Computer Typesetting* (TTCT), with Frank as series editor. *The LaTeX Companion* (1994) was the first book of this series, which now includes titles that cover LaTeX in all its facets.


... and three decades later

He is on the board of the International Gutenberg Society, an international association for the study of the history and development of printing technology and font-oriented media, where he focuses on the more recent period. He is also involved in the print production of the society and in general enjoys designing and typesetting high-quality documents using computers but also using traditional letterpress. One of his plans after finishing TLC3 is to typeset a book on Japanese authored by Peter Gordon. When not involved with typographical questions or programming Frank enjoys reading (though the books better be of high quality in content *and* form), listening to or playing music (he is an amateur bass player), and winning or losing board games, these days often collaborative ones.

He and his wife Christel live in the Gutenberg city of Mainz; their three grown-up sons have left by now for new pastures.

Ulrike Fischer

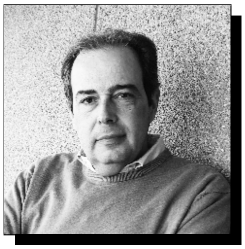
Ulrike Fischer studied mathematics at the University of Bonn. She wrote her thesis in a rather arcane branch called model theory with an Atari text processor called Signum but did not like to have to place lots of sub and superscripts with a mouse. So when seeing an example LaTeX document, she directly ordered the floppy discs and never looked back.

Quite early she got interested in the internal handling of fonts and in chess typesetting as she wanted to print the bulletins of chess competitions she organized in her club. With the help of the first edition of the *LaTeX Companion*, she wrote the chessfss package, which allows users to choose between various chess fonts, and other chess-related packages.

After university, she moved with her husband to Siegburg and later to Mönchengladbach and worked as an assistant for a deputy of the state parliament of North Rhine-Westphalia. She found her way into the LaTeX groups on the Internet and enjoyed answering questions and debugging errors. She also helped her father who after his retirement as a professor for computer science wrote with LaTeX a number of books about Salvador Dalí's illustrations of the *Divine Comedy*. So when her job ended after a lost election, she decided to try something new and to work as a LaTeX consultant.

Her interest in fonts continued over the years and led her to accept the job of maintaining luaotfload. Talks about accessibility and tagged PDF at Dante and TUG meetings induced her to work on the tagpdf package and to present it to LaTeX team members at a workshop at the TUG meeting in Rio de Janeiro. Both together got her an invite to join the team in 2018. She had already had many online contacts with the team members in Usenet groups and chats as well as in person — she met Frank the first time at a Dante meeting in Heidelberg, where she talked about how to misuse biblatex as an address database, and he showed her how to eat some Japanese dish — and so accepted without hesitation. She now lives with her husband Gert in Bonn.

Javier Bezos



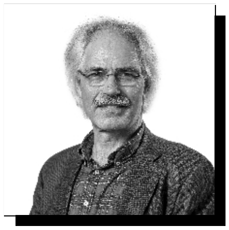
Javier Bezos is passionate about typography, science, language, and writing systems, as well as about computer programming. A curious person and self-taught by nature, his interests are broad, and he has worked in fields like book and journal publishing, design, radio, and classical music. He is an honorary member of the Unión de Correctores, the Spanish professional association of copyeditors.

As a computer programmer, he has developed systems for both LaTeX and InDesign, including tools for technical documentation, automatic typesetting and copyediting, XML, and localization. It is this last facet that he develops most actively as a member of the LaTeX Project Team.

Javier has participated as an author on several style manuals and published the book *Tipografía y notaciones científicas*. He is currently working at FundéuRAE, a foundation linked to the Royal Spanish Academy.

Hiking, reading old science books, and watching classic movies are some of his hobbies.

Johannes Braams



Johannes Braams studied electronic engineering at the Technical University in Enschede, the Netherlands. His master's thesis was on video coding, based on a model of the human visual system. During his research at the *dr. Neher Laboratories* in 1984, he came into contact with LaTeX for the first time. He was a founding board member of the Dutch-speaking TeX User Group (NTG) in 1988 and participated in developing support for typesetting Dutch documents. He started work on developing the babel system for multilingual typesetting following the Karlsruhe EuroTeX conference in 1989.

Since its inception at the EuroTeX conference in Cork in 1990, he has been a member of the LaTeX Project Team. Johannes still maintains a couple of LaTeX extension packages such as the ntgclass family of document classes and the supertabular and changebar packages.

After his studies, Johannes continued to work for the Dutch PTT, which later came to be known as KPN until 2014, mostly as an IT project manager. In 2015 he started a new career as a consultant on cybersecurity for industrial automation and control systems, first at the Software Improvement Group in Amsterdam; then in 2017 he joined Royal HaskoningDHV, a Dutch engineering firm that operates globally. He works on security by design in industrial installations. Johannes is now a board member of the Dutch chapter of (ISC)^2. He lives with his wife in Den Ham; their two sons both graduated from the University Twente in Enschede.

Joseph Wright

Joseph Wright is a synthetic chemist who studied for his degree at the University of Leicester before moving to the University of Cambridge for his PhD. He first heard of LaTeX in about 2001 but did not start using it until writing a final report on his first research assistant position at the University of Southampton in 2004.

Joseph transitioned from being a user to a developer most obviously when he answered a "simple" question on the `comp.text.tex` newsgroup asking for a fix to the SIunits package in 2007. It soon turned out that this was the tip of a much larger task: writing a truly comprehensive units package: siunitx.

Joseph became involved with the LaTeX Project Team's work after Will Robertson asked him what he thought of the experimental ideas in expl3. Joseph liked them, so he used expl3 to develop the second version of siunitx and raised *lots* of questions about how things worked. His questions and suggestions led to an invite to join the team.

As well as writing (La)TeX code, Joseph is one of the moderators on the TeX/LaTeX StackExchange site. He's one of two moderators who've been in place from the very start of the site, helping to ensure there's always been a welcoming approach with the aim of helping end users solve their (La)TeX problems.

Production Notes

This book was typeset using the LaTeX document processing system (which it describes) together with substantial help from some of the extension packages covered herein, and considerable extra ad hoc LaTeX programming effort. We used the base LaTeX release dated 2022-11-01, pdfTeX as the main TeX engine, and LuaTeX for examples that involved OpenType fonts. All packages have been taken from the 2022 TeX Live distribution to ensure that the examples show what you will get when you run them yourself.[1]

Body fonts

The text body fonts used are Lucida Bright and Lucida Sans (Bigelow/Holmes) at 8.47pt/11.72pt. These font families can be used at rather small sizes, which can be seen by the fact that the monospaced font we matched them with is Latin Modern Typewriter (TeX Gyre foundry) at 10.36pt. This particular combination was chosen to get a reasonable amount of material on each page and to optically balance the appearance of the `typewriter font` so that it is distinguishable but without too big a contrast.

Example fonts

The text in the examples mostly uses TeX Gyre's Times Roman (`tgtermes`) with its Helvetica (`tgheros`) for sans serif. For the mathematical material in the examples we stayed with the now classic Computer Modern math fonts, so the symbols will appear familiar to the majority of mathematics users. Of course, examples intended to demonstrate the use of other fonts are exceptions.

Package usage in the book

The LaTeX packages used to produce the book . . .

If you count the examples as part of the book, then every package described in this edition has also been used in it. This is a bit of a red herring, though, because the examples are typeset as separate documents and the results are then included in the book as PDF (Portable Document Format) images. However, many of the packages

[1] Well, at least initially — sometimes, though seldom, packages change in incompatible ways.

have been deployed for real to produce this edition, and for the interested reader we now list the most important ones and why they have been used.

… for the main text …

To help in producing the explanatory text we used acro for acronyms, resetting their use count on a per-chapter basis. Thus, for most acronyms you see the first use in a chapter being displayed in its full form, e.g., as "Comprehensive TeX Archive Network (CTAN)". For quotation marks, commands from the csquotes package have been used throughout.

… for generating references …

Heavy use was made of `\vref` and friends from the varioref package. We often refer to examples or sections "on the facing page", "previous", "next" or "below", etc., and in all such cases these are generated texts and not hardwired in the source. Otherwise, the layout process would have been a nightmare, requiring extensive amounts of textual adjustments whenever material moved from one page to the next and vice versa. Of course, using varioref in a book of this size is bound to produce an *impossible document* as explained on page →I 85. And indeed, it happened a handful of times that the generated text got split across two pages, rendering the wrong result in all cases. Initially, we changed the errors to warnings and ignored them, and then resolved the issues during the final pagination by adjusting the surrounding text (either rewriting parts or adding some strategic line or page breaks).

… applying optical alignment and font expansion …

Since the second edition of *The LaTeX Companion* we used "hanging punctuation" to typeset the paragraphs. In the last edition, this typographical icing was basically produced manually, but this time we have been fortunate to simply[1] apply the microtype package. This package was also used to add some light expansion/suppression using the *hz*-algorithm, which, in my opinion noticeably improved the overall look and feel of the paragraphs. For comparison, look at this paragraph and compare it with the repeat below (where both features are turned off).

… or not … (a repeat) [2]

Since the second edition of *The LaTeX Companion* we used "hanging punctuation" to typeset the paragraphs. In the last edition, this typographical icing was basically produced manually, but this time we have been fortunate to simply[1] apply the microtype package. This package was also used to add some light expansion/suppression using the *hz*-algorithm, which, in my opinion noticeably improved the overall look and feel of the paragraphs. Compare this to the previous paragraph (which has both features turned on) — *which do you prefer?*

… chapter headings, lists, and running headers …

The core design for the book dates back to the first edition and therefore chapter headings, list environments, etc., have been defined using the basic mechanisms of LaTeX rather than using support packages because those were not available back then. For the running headers and the page numbers the fancyhdr package was used.

… making tables …

To produce the various tables in the book, array and tabularx (for the extended syntax) were put to good use and on nearly every table we applied commands from the booktabs package for the formal rules. The few multipage tables have been generated with longtable and in some places we used multicol to produce a table-like appearance.

[1] We had to define our own configuration file, though, because the microtype package has no presets for the commercial Lucida Bright fonts and the default settings, while more or less adequate, were not totally satisfactory. Thus, we applied the advice given in Section 3.1.3.

[2] The paragraph now breaks differently and the margins are no longer optically aligned.

Graphics have been included with the help of graphicx and in a few places we used overpic to add some arrows or other data to the image. The captions have been styled with the caption package; other float-related packages were not necessary as the book requirements have been fairly light. Producing the examples, showing input and output side by side, was more elaborate and involved the use of the fancyvrb package and a fair amount of programming. The general approach we used is explained in Section 4.2.4 on page →I 315.

… showing graphics and examples …

The boxes showing information that is only relevant for Unicode engines or that detail differences between the BIBTEX and biblatex approach have been designed using tcolorbox, which in turn uses tikz.

… displaying informational boxes …

For the bibliography we used BIBTEX with a number-only style that was designed with the help of custom-bib, and natbib was used for the citations. Because this edition was split in two parts, each part got its own bibliography; this was achieved with the bibunits package.

… producing the bibliography …

The book uses several separate indexes (even though only two are shown in the final product). To generate the raw data for them we made use of the index package and the sorting was done with *MakeIndex*. However, due to the complexity of the index (the colored page numbers, etc.) it was necessary to use pre- and post-processing by scripts to produce the final form of the index files. These were then typeset using an enhanced version of the multicol package to add the continuation lines — something that perhaps one day can be turned into a proper package. The “Commands and Concepts” index is a mixture of commands, environments, and other LATEX elements, but also includes concept entries. The latter were produced by typesetting a version of the book with line numbers and giving that to the indexer, who produced “conceptual index entries” that were then added to the source files for the book so that pagination changes would not invalidate the index entries. This was a major testament to the quality of the lineno package, as it worked “straight out of the box”.

… the different indexes …

In anticipation of a PDF and ebook version of the third edition, we already used both the bookmark and the hyperref package. Even though this does not show in the printed form of the book it helped us tremendously during its preparation, because all internal references have been active links, allowing us to quickly jump from one place to another in the draft PDF document. Furthermore, all internal notes we made were enhanced to generate bookmarks so that the bookmark pane of the PDF reader gave us a quick overview of all the tasks still to do and text still to write.

… and interactive PDF features

Packages that should have been used for the book …

You might be surprised that some heavily recommended packages, such as enumitem or geometry, are missing. The reason is simply that the fundamental design decisions, e.g., the page layout or the design of the headings, lists, etc., have been retained across all editions and when the first edition was typeset in 1994 those packages that can now make your life easier were not yet available, and thus the necessary code had to be handcrafted. Altering that setup was only done when improvements or changes were necessary; e.g., the titletoc package was added when we introduced the partial TOCs below the chapter headings. But in a new project such packages would have been used from the start.

… but have not

Packages supporting the production

Running out of files to write to

This book loads more than 140 `.sty` files (not counting those loaded in the examples), which is way beyond what is needed in typical documents. Several of these packages open their own write streams, and this is a rather limited resource in TeX (only 16 can be open at the same time). As a result the book died when processing the list of figures because that tried to open one stream too many. Fortunately, there is a remedy with Bruno Le Floch's morewrites package. It comes with a cost, though, because it needs additional processing time, but the important point is that it enables you to process a document of such complexity!

Color woes

The printing process for the book required two colors: black and some kind of blue as a spot color. For this, we used the packages xcolor and colorspace, but it is not a simple task to ensure that the final document does not contain color instructions in CMYK (or RGB) because some packages include hardwired color specifications using such models. As a result we had to do some patching or avoid using color in a few examples in order to not upset the final production process. This is clearly an area where LaTeX could be improved to make life easier.

Task management

To keep track of all the open tasks that needed resolving, we used the todonotes package and defined a few commands similar to those shown in Example A-1-17 on page 645, except that we also added a bookmark for each task using the bookmark package. At one point this showed several hundred entries, but now as I write this text it is thankfully down to a handful of remaining tasks.

Where are my errors?

Loading the structuredlog package enabled us to quickly identify in which source file an error occurred — not that we ever had any.[1] The footnote in Appendix B.1 on page 712 explains how to make use of this package.

Controlling the source

Using a source control approach is beneficial for nearly every project — for a complex one like this book, it is essential. It enables you and your co-authors to work without stepping on each other's toes; if something breaks, you have a history that helps you identify and resolve problems; and, last but not least, it simplifies project data synchronization across different devices and locations. For the second edition we used SVN; for this edition it was Git, and the package gitinfo2 was deployed to write source branch and date information at the bottom of every page (outside the final printing area).

Tracking file usage

Another tool we used regularly was mkjobtexmf. With its help we collected all files used during compilation in a separate tree, which was also placed under source control. This way, any updates to standard packages, font files, etc., could be easily tracked, which again helped a lot if something went awry.

The production cycle

The production of this book required a custom class and, as outlined above, many additional package files. It also needed a complex "make" process using a collection of "shell scripts" controlled by a "Makefile". One of the major tasks these accomplished

[1] Joke — of course, we did. All kinds of errors occurred, from simple typos to subtle package incompatibilities that resulted in talking to the developers after identifying the root cause of the problem.

was to ensure that the typeset output of each and every example really is produced by the accompanying example input. This “make” process worked as follows:

Generating examples

- When first processing a chapter, LaTeX generated a source document file for each example. These are the one-and-a-half-thousand “example files” you will find on CTAN at `https://ctan.org/pkg/tlc3-examples`.
- The make process then ran each of these “example files” through LaTeX (also calling BibTeX, biber, or whatever else was needed) as often as was necessary to produce the final form of the typeset output.

  The resulting PDF file was post-processed with pdfcrop to drop all white space on the outside (making the PDF pages only as big as the visible output) and, if necessary, to split it into separate pages using pdfseparate.
- The next time LaTeX was run on that chapter, each of these PDF output files was automatically placed in its position in the book, next to (or near) the example input. The process was not complete even then because the horizontal positioning of some elements, in particular the examples, depends on whether they are on a verso or recto (the technique from Example A-5-7 on page 692 was used in this case). Thus, at least one or two additional runs were needed before all the cross-references were correctly resolved and LaTeX finally found the right way to place the examples correctly into the margins.

Due to the example-generating complexity — together with the use of varioref altering the generated text based on distance to the referenced object — the book required not the usual two or three but in fact six complete runs before LaTeX stopped asking for “Rerun to get cross-references right”. Even on a fast iMac from 2021 with an M1 chip, that meant a half-hour wait — and this is not counting the generation of the 1540 examples. Those required another two-and-a-half hours[1] so that it took a total of three hours of computer-processing time to produce the typeset book from scratch.

The layout effort

Manual labor from page 1 to 2000

That was about as far as automation of the process could take us. Because of the many large examples that could neither be broken nor treated as floating material, getting good page breaks turned out to be a major challenge. For this and other reasons, getting to the final layout of the book was fairly labor intensive and even required minor rewriting (on maybe 10% of the pages) in order to avoid bad line breaks or page breaks (e.g., paragraphs ending with a single-word line or a distracting hyphenation at a page break). Spreads were allowed to run one line long or one line short if necessary, and in several cases the layout and contents of the examples were manually adjusted to allow decent page breaks.

Giving widows and orphans a new home

A great help in this final layout process was the package widows-and-orphans, which told us the pages that contained widows, orphans, or hyphenations across page boundaries. We then applied one of the suggestions from Section 5.6.6 on page →I 422, often adding an explicit page break in some strategic place or sometimes rewriting a

[1]The main factor here is the file I/O cost. Each example has to be run several times, and each time, TeX has to start and load all packages and fonts, over and over again.

paragraph to make it longer or shorter (after all, we are the authors and can change the text for aesthetic reasons). In a few places, no fix seemed reasonable, in which case we accepted an orphan or widow and marked the place with `\WaOignorenext`. In total, 1.5% of the pages were treated like this.

Some pagination statistics

Here are a few approximate statistics from this page layout process: 24 long spreads, 41 short spreads, 51 compressed pages (using `\enlargethispage*`) combined with forced page breaks. This is noticeably less than in the previous edition and one of the reasons is that this time we made ample use of TEX's `\looseness` functionality: we ran 125 paragraphs one line shorter than normal and 17 paragraphs one line longer than is optimal in TEX's eyes.

We added 501 forced page breaks in total, which means 25% of all pages have a forced premature ending — usually to avoid half-empty pages later due to a larger example showing up in the wrong place. While I think the results are satisfactory, getting this right was rather time consuming and difficult — here it really hurts that TEX has a global paragraph optimizer but generates its pagination with a simple greedy algorithm. While this gives reasonable results with most documents it totally breaks down in those that contain larger unbreakable parts such as this book.[1]

Searching across the final sources, there are about 1200 adjustments to the vertical spacing[2] and 100 other manual adjustments (other than rewriting or altering the examples). For the latter I do not have any reliable data, but from my recollection I would guess that about a quarter (i.e., 400) of the examples have been adjusted to better blend into the page flow.

A final word of advice

In several places in the book we have given the advice to leave any layout work until the very last minute, and on the whole we followed our advice to the letter with this book. For example, all pagination work was delayed until the copy editor had worked through all chapters and marked up all the mistakes we made (well, hopefully all) — it took me a while to convince her not to mark up ugly-looking page breaks, because they were deliberate. Given that changing a single letter in a paragraph may result in total reflow (and possibly a different number of lines) this saved me a lot of unnecessary work.

However, in one case I diverged from that principle. When I started to write this section, I realized that the chosen typewriter font seemed to be a tiny bit too large compared to the Lucida body fonts. I therefore decided to scale it down by one percent. Given that this is only used for individual words, I expected a few changes but not much. To my dismay I ended up with more than fifty pages where the pagination was totally broken, e.g., paragraphs getting shorter or longer, lines overflowing to the next page, etc. It took me a full day to repair the damage. So again, take this advice seriously and only work on pagination adjustments when you are 100% done with the text and all your other layout choices have been made — easier said than done, I know!

[1] I had secretly hoped that the research results outlined in [134, 135, 138] could already be applied to the production of this book, which would have saved me perhaps a month worth of effort. Unfortunately, turning the prototype into a fully working production system proved to be more complicated than I had bargained for. Thus, in the end I buried the idea and produced the layout the old-fashioned way.

[2] These adjustments are partially due to deficiencies in parameter setup for the custom class. However, by the time I realized this, several chapters had already been paginated and changing the class setup would have invalidated those. Thus, it seemed simpler to live with the imperfection.

PDF and ePub production

The PDF for printing the edition

As you know, we had to split the third edition — due to its size — into two physical books. However, both parts were produced with a single LaTeX source to make it easier for LaTeX (and us) to automatically resolve cross-references across the parts and to generate a consolidated index with entries for the whole edition. This index was included at the end of each physical part to make it easier for our readers to work with the two books.

The tables of contents lists were tailored for each part: they show the sections of the current part in detail, but only give a chapter-level overview of the other part. Similarly, the bibliographies at the end of each part only show entries that have a citation in that part. The few books and articles that are referenced in both parts therefore appear twice in the complete print edition. This way the resulting PDF only needed to be split in the middle and sent to the printer.

However, in that form the PDF is unsuitable as a digital product or as a basis for producing an ePub of the complete edition, because it contains a duplicated index, two separate bibliographies, and no consolidated front matter, e.g., no complete table of contents or list of figures.

The PDF for digital consumption

Thus, for the digital version we augmented the LaTeX sources so that by flipping a switch a slightly different PDF is produced. The differences between the print version and the digital version are as follows:

- In the digital version the front matter of Part I contains a consolidated table of contents for both parts and the partial table of contents in Part II was dropped. The same applies to the list of figures and the list of tables.
- The bibliographies from both parts were combined and placed at the end of Part II. Of course, this alters the citation labels within the edition, and that may result in a slightly different paragraph formatting in one or the other place (due to differently sized labels), but great care was taken to not affect the overall pagination. Thus, if you own both the print and the digital edition they show the same material on pages with the same page number.
- The bibliography change also required me to regenerate two examples: 3-4-17 on page →I 189 and 4-1-44 on page →I 288, because they both refer to entries in the bibliography. These examples therefore differ slightly between the print and the digital versions.
- The preface of Part II was shortened because it repeated the “How to work with the book” section, which is not needed twice in the digital version.
- For the same reason the index for the complete edition at the end of Part I was dropped and only the complete index in Part II was kept. This index of the digital version should be more or less identical to the one in the print version, e.g., you can use the digital index and then look up the page in the printed book. However, given that we dropped one section from the preface that contained index entries, the index in the printed book has a few more page references.
- This section about PDF and ebook production only appears in the digital version.

In all other respects the print and the digital versions have been kept identical, e.g., the page numbers are restarted in Part II, so that most page numbers appear twice. This way they are at most three digits wide and it remains possible to correctly cite pages from the edition, regardless of somebody owning the digital or the print version (with the execption of pages from the front or the back matter).

Navigating the PDF

Most PDF viewers allow you to select pages by entering, for example, `II-39`. Thus, if somebody tells you to have a look at the nice Accanthis font on page 39 in Part II and you want to look this up in the PDF version, this is a simple way to reach that page. Of course, in the ePub version page numbers are of no real help, except that if used in the text they contain a link that you can click on to jump to the correct place in the edition. In the ePub you have to navigate using the book marks if you want to get to a specific section — a possibility that, of course, exists in the PDF as well.

Producing the ePub version

This altered PDF is what you get as one of the digital products, and it is also the one that was used as a basis for producing the ePub version by a company specializing in such work. There are in fact tools in the LaTeX world, such as `tex4ht` by Michal Hoftich or lwarp by Brian Dunn that assist you in a direct conversion of LaTeX sources to HTML or ePub, but the *LaTeX Companion* is so complicated that we did not attempted to use the direct route. Maybe that is going to change in a future edition, when LaTeX is fully capable of automatically producing a tagged PDF.