4TEX for Windows

Dutch language oriented TEX Users Group NTG

4TEX for Windows

Wietse Dol
Erik Frambach

Dutch language oriented TEX Users Group
Plantage Kerklaan 65/1
1018 CX Amsterdam
The Netherlands
http://www.ntg.nl/
ntg@ntg.nl
http://4tex.ntg.nl/
4tex-support@ntg.nl

ISBN 90–76669–01–5

First printing, June 1999

We are 4TEX.
Resistance is futile.
You will be assimilated.

Those who know ST:TNG and
meet 4TEX will understand.
To all others we say:

May the 4's be with you!

Contents

Preface

The name of the game

Although TEX is about 20 years old and very much alive, many people have never even heard about it. And those who have will often say that their impression is that it is a very difficult system, or that it is a program from the 'stone age'.

But how many programs do you know that after so many years are still so popular? How old do *your* programs become before they die? And did you ever wonder why those programs always die before they get a chance to mature?

TEX is old. Right, and how old are *you*? Do *you* feel you should be replaced by a brand new person who has had little education, experience or field training? Wisdom comes with age. You will have a hard time finding a program of the size of TEX that has *fewer* bugs. In fact, if you can find a bug, you will get a reward from the author!

We are not going to tell you that you should switch to TEX because it is the best there is. That would be a naïve simplification of the situation. Nevertheless, TEX is a program that shines and has been shining in several areas that have been neglected by many competitors. And in other areas TEX simply is no match to its competitors. But of course there are also areas in which TEX is easily defeated by other programs. As usual, you will have to make choices. Sometimes you may need TEX, at other times you may need something else, depending on what you are trying to realize.

With this book and the 4allTEX CDROMs we want to show you how well TEX fits a modern computer environment, and how you can benefit from it.

It is up to you to decide if this way of working with text suits you or whether you would rather stick to some other system. If you read the book and try the software you will at least get an idea of what TEX is about. And you will know how it compares to other systems. It doesn't. But maybe you will agree with us that it is an exciting system.

The history of TEX

In the late 1970's, when Don Knuth, the author of TEX, was working on his book series 'The Art of Computer Programming', he found that none of the typesetting systems available were capable of producing output at the quality level that he needed. He decided to implement his own document formatting language, so he would be able to typeset his own books the way he had envisioned them. This project lead to the first incarnation of a program that he named TEX.

The name TEX is the uppercase version of the Greek root starting with '$\tau\epsilon\chi$' (tau, epsilon, chi). In Greek the word means both *technology* and *art*. The Greek origin also explains its pronounciation. The 'T' and 'E' are as in 'technology', but the 'ch' should sound as in the German name 'Bach', the Scottish word 'loch', the Spanish 'j', the Dutch 'g' or the Russian 'kh', whatever you prefer.

The first incarnation of TEX (now called TEX78) was improved in many ways, which lead to the definitive version of TEX that was released in 1982. Since then only small changes were introduced and several bugs were removed.

The current version number of TEX is 3.14159. Indeed, this number is very close to π. With every new version another decimal of π is added, so we already know that the next version will be 3.141592. Every new version will bring TEX closer to the real value of π — which has an infinite number of decimals. Think of it as reaching for perfection.

TEX as it exists now is 'frozen' by the author. New versions will only contain bug fixes and other minute changes that will not cause any TEX document written since 1982 to fail. TEX comes with its own test files to ensure this. The so-called 'trip test' should produce predefined output. If it doesn't, the system simply cannot be called 'TEX'. This method also ensures that TEX = TEX, regardless of the operating system it runs on.

Though Don Knuth wrote TEX as a document formatting system for his own books, the system proved to be much more universal. Many people all over the world started using the system. Traditionally the academic world uses it a lot, but TEX is certainly not limited to this niche. In fact, any environment in which production of (complex) documents is important, TEX could be a very valuable tool. Some big publishing companies chose TEX as their document production system.

Clearly the system that Don Knuth developed turned out to be more versatile than anyone expected. Millions of users all over the world are using it for their work, hobby or both. TEX user groups are formed, international conferences are held, courses are given, journals are published: a true TEX community has evolved. In short, TEX is alive and kicking!

The history of 4TEX

It was back in February 1991 when Maarten van der Vlerk and Wietse Dol, both Ph.D. students at the Faculty of Economics of the University of Groningen in the Netherlands, became interested in TEX, and devised the first version of what was to become the 4TEX system. It started as modest MS-DOS batch files that implemented the edit-compile-view cycle for LATEX jobs. Very soon more features were added and the simple batch file became a more sophisticated interactive system based on the 4DOS batch file programming language. This system was named the 'Mally & Dolly menu', after the nicknames of the authors.

The system became more and more popular among econometricians who were changing from ChiWriter to TEX, and the system grew as the authors became more familiar with TEX and friends, and users asked for more features.

In the same year Erik Frambach was employed by the department of Econometrics as a scientific programmer. He soon became infected with the TEX virus that was still spreading. Together we revised, improved and extended the 'Mally & Dolly menu', and it was renamed '4TEX'. This name reflects both its roots in the 4DOS batch programming language and the notion of a user-friendly system *for* TEX users.

In 1993 we made the system available on a large set of diskettes, maintained by Phons Bloemen, and distributed by the Dutch language oriented TEX Users Group NTG. This system was named '4allTEX' because it was now available to a much larger group of users, and because it contained not only MS-DOS specific programs, but also large quantities of operating system-independent stuff such as METAFONT sources and PostScript fonts.

Soon it became apparent that such a set of diskettes was very hard to maintain and tedious to install. CDROMS were becoming more and more popular as a medium for software distribution. 4allTEX on CDROM would make installation much easier and there would be plenty of space to add many more goodies.

In May 1994 the first edition of 4allTEX on CDROM was produced. But this was much more than just a reworking of the diskette set. The CDROM was set up as a true 'plug and play' system. In fact, it was the very first TEX CDROM that would install TEX in less than a minute, and run almost entirely from the CDROM. Because of the huge amounts of space on a CDROM (compared to the diskette set) we were able to add many extras (TEX files, programs, documentation, etc.). We even felt confident (or arrogant) enough to call it 'The ultimate TEX resource'.

This CDROM soon became more popular than we could ever have expected. In July the 250 copies we made were sold out. In September 1994 a second edition was released. 1250 copies were made and they were sold in half a year, not only to NTG members but worldwide.

From the experience with this edition we learned a lot. In 1995 we felt that it was time for a complete revision. The third edition was produced. This time not a single, but a *double* CDROM. We simply couldn't fit everything we thought valuable onto one CDROM. This third edition sold over 4000 copies worldwide in less than 2 years.

In 1997 we started working on the fourth edition. It would be similar to the third, but it would now take full advantage of graphic and multitasking abilities of the Microsoft Windows environment. The fourth edition was finished in October 1997. The fame and reputation of the 4allTEX CDROMs made four TEX User Groups (TUG, UKTUG, DANTE and NTG) decide to buy the CDROMs for all their members (about 4800 in total, within the first three months after its release).

Although 4TEX 4 takes advantage of Windows features, it is still not a true Windows environment. Under the hood many programs are still MS-DOS-based. We realized that a true Win32 (Windows 95, 98, NT) implementation was needed. So we started looking for programs to build such an environment based completely on freeware and shareware. Luckily the complete Web2c TEX system (*the* standard on Unix systems) was being ported to Win32 in the same year. It was soon decided that this implementation should be the basis of the Windows environment we envisioned.

We evaluated many, many Windows programs and 'TEX shells' to find that not one of them offers all the flexibility and options we expected from a 'control center' for

using TEX on Windows. So, again we wrote our own system, and named it '4TEX for Windows'.

Acknowledgements

We would like to give credit to all people who helped us in building this system, but there are just too many to list all of them here. Nevertheless, here is a list of people who made very valuable contributions, sorted by the 4th character of their names:

> Gerard van Nes, Sebastian Rahtz, Barbara Beeton, Eric Fookes, Marcin Adamski, Jacek Kmiecik, Pavel Sekanina, Peter Davis, Peter Gordon, Doc Evans, Ian Gibson, Sasha Berdnikov, Richard Silberstein, Kathy Hargreaves, Luzia Dietsche, Maciej Walkowiak, Christina Thiele, Ewa Kmiecik, Thomas Mittring, Bernard Gaulle, Bernd Raichle, Frank Mittelbach, Taco Hoekwater, Siep Kroonenberg, Andreas Eder, Fabrice Popineau, Bjørn Volkerink, Petr Olsak, Igor Podlubny, Petr Sojka, Maarten van der Vlerk, Andrzej Borzyszkowski, Hans Bessem, Hans Hagen, Kees Praagman, Rens Strijbos, Ernst Willand, Kees van der Laan, Staszek Wawrykiewicz, Morten Risager, Martin Bartels, Piet van Oostrum, Alex Simonic.

We want to thank all the people who tested the system, made suggestions for improvements, reported bugs, proofread the manual, gave pointers to other sources, programs, documentation, or who contributed in various other ways.

Wietse Dol
Erik Frambach

How to use this book

We have tried to write this book in such a way that people who never heard about TEX can learn fast and get the software running. At the same time we know (or hope) that people who are familiar with TEX will also read it. Of course we don't want to bother them with superficial introductions. Therefore we will give recommendations for both types of users which parts to read and/or skip. But let us start with an overview of the contents of this book.

Part I: *Getting started with TEX*. This part is a general introduction that explains TEX concepts in a nutshell. It also gives recommendations on when and where (not) to use TEX. Finally an overview is given on different ways that are available to TEX users to get support and advice.

Part II: *Using 4TEX*. This part focusses on how to use the 4TEX program. It start with installation of the software, and it explains all menus, button, and other features that 4TEX supports.

Part III: *The technical ins and outs*. This part starts with a detailed explanation on how 4TEX's features are implemented, which files are involved and how 4TEX expects them to appear. Details on 4TEX configuration are discussed as well. After that an exhaustive explanation of all Web2c programs that make up the entire TEX system is given.

Part IV: *The many roads to TEX*. This part explains the ideas of different macro packages for TEX. An introductory course on three such packages is included, as well as a comparison of these three. We will also briefly discuss future developments.

Appendix A contains a set of tables in which you can quickly look up the meaning of file types that you may run across when using TEX.

Appendix B contains flowcharts that describe the relations among the files and programs in a TEX system.

Appendix C lists all the software that you will find on the CDROM. All this software is either freeware or shareware. Freeware is absolutely free of charge: you got it free of charge and you can use it free of charge. You can also distribute it to anyone else, provided that you distribute the whole package without any changes and you don't charge for it. Shareware is somewhat different. You can use/evaluate shareware only for a limited period. If you want to keep using it after the evaluation period, you are required to pay a (usually small) license fee. Distribution of shareware software is usually permitted and encouraged, as long as you don't charge for it and you distribute the software without any change.

Appendix D lists all electronic documents available on the CDROM. Throughout this book we have marked such references like this: (cdrom).

Appendix E contains a glossary of terms used in this book. It also contains a graph of all programs cooperating in a typical TEX system.

Appendix F is the bibliography. It lists books, articles, online publications, tutorials, etc. that are referenced in this book, or could be interesting to TEX users in general. Items marked (cdrom) are available as electronic documents on the 4allTEX CDROM. See also appendix D.

Appendix G is the index that can be used to find all major references to important items discussed in this book.

Depending on how familiar you are with TEX, you may choose to skip parts that you think you don't need right away.

If you are a completely new user we recommend that you start with part I and then move on to part II. We recommend that you read at least two chapters of part IV, so that you acquire basic knowledge of the TEX language. You can skip part III, at least for the time being.

If you already know TEX from other environments, it may be sufficient to read part II, which explains how 4TEX works. You may want to read one or two chapters from part IV if you feel that you don't know enough about a certain TEX dialect.

System managers and other people who want to get a deeper understanding of how all this software works will want to read part III. Others should rarely need such detailed information on program syntax, configuration files, etc. Appendices A and B can be helpful, too.

Typographic conventions

In this book different typefaces and other typographical features are used to distinguish between file names, commands, parameters, syntax descriptions, etc.

In general, everything that you would type at the command prompt and output that programs write to the terminal is written in `typewriter font`.

File names are written like this: `myfile.tex`. In general file name *extensions* (`.tex` in this case) are important, but in many cases there are defaults, depending on the program that reads a file. The *case* (upper or lower) of names is not significant.

Paths are written like this: `c:\tex\fonts`. Note that in many cases paths should be written using a *forward* slash: `c:/tex/fonts`. 'Long' path names and file names (e.g. `c:\aLongDirectoryName\MyFirstTeXDocument.latex`) are usually allowed but there are some pitfalls that we will warn you about. The *case* (upper or lower) of paths is not significant.

Commands that you can enter at the command prompt are written like this:

```
▷ dosomething -nice to me
```

`dosomething` is the actual *command*, which can be an executable (with extension `.exe`), a standard batch file (with extension `.bat`), or a 4DOS batch file (with extension `.btm`). The differences between standard and 4DOS batch files will be

explained later.
`-nice to me` are *parameters*. The hyphen in the parameter `-nice` indicates that this is actually an *optional* parameter. The *case* (upper or lower) of commands is not significant, but parameters often *are* case-sensitive.

Syntax descriptions of programs run from the command line are written like this:

≡ command [*option*] *variable* fixed ...

Options are specified between square brackets, but you never actually use square brackets when you enter a command.

Syntax descriptions of TEX macros (including LATEX and CONTEXT) are written like this:

`\somemacro(`*parameters*`)[`*options*`]{`*argument*`}{...}`

Let us assume that the macro `\somemacro` accepts `here` as an option, then a valid construction would be `\somemacro(1,2)[here]{in}{out}`. Writing `\somemacro(1,2){in}{out}` would also be valid, but `\somemacro{in}{out}` would be invalid. Because of the nature of the TEX language many other syntactic constructions are possible.

Keystrokes are written like this: A or Esc. Keystroke *combinations* are written like this: Ctrl B. it means that you should keep the Ctrl-key depressed while you press the B-key.

Windows buttons are displayed like this: Help or OK. You can click on them with your mouse or other Windows pointing device. When we write 'click on ...' we mean: press the *left* mouse button when the mouse pointer is positioned over a button. When we write 'right-click on ...' you should press the *right* mouse button.

Environment variables are written like this:
`TEXMFCNF=c:/texfiles/texmf/web2c;!!d:/texmf/web2c`
There are several ways in which environment variables can be set. The simplest way is to use the Windows command `SET` from the command line. Other methods will be described later.

Examples of TEX code are displayed like this:

Hello World!

```
Hello World! \bye
```

The output on the left, the exact typed input on the right.

Hints and warnings are indicated with a special sign like this:

Whenever this symbol occurs you should read that paragraph carefully. It could be very important!

PART I

Getting started with TEX

Chapter 1

A quick introduction

1.1 What TEX is (and what it is not): an overview

Communicating is something we all do, and in various ways. Often we write letters, memos, papers, books and so on. When you use a computer for entering text, you need a program to *typeset* the information you want to express in a way that appeals to the reader. TEX is a system designed for performing this task.

We use the term *system* here on purpose. Many people seem to regard word processing and desktop publishing and typesetting as one and the same thing. We don't. Desktop publishing may be regarded as a special case of word processing, but typesetting is not. In fact, typesetting is just one of the processes that run continuously when you are entering words in a word processor or DTP program.

Most word processors do typesetting on-the-fly. As you type, the characters are positioned on a line and lines are broken when they become too long.

TEX takes a different approach. The process of entering text is completely separated from the typesetting process.

One of the major reasons is to make it easy for you to concentrate on the *content* of your text. In some cases you may need direct visual feedback, but in many cases a WYSIWYG interface ('What You See Is What You Get') is distracting. And, for that matter, the acronym WYSIWYG would often better be replaced with WYSI**A**YG — what you see is *all* you get. What you see on the screen is not exactly what you will get on paper — of course not, you want something much better.[1]

That is why TEX is different. It takes the text file you wrote and typesets it, producing an output file that you can print or view on the screen. Naturally you will have to tell TEX how you want your text typeset. TEX has hundreds of commands that you can

1 For an interesting discussion on the virtues of WYSIWYG systems see C. Taylor's article *What has WYSIWYG done to us?* (cdrom).

use in your text. But don't worry, you don't need to know them all, and in fact you will probably only use a few in your document.

Most of the commands will be placed in another file that defines the *style* in which you want certain documents to be typeset. This allows for even stronger separation of layout definitions and your actual text. You can (and will) even have many different styles that will handle the same input but give completely different output. With TEX it is easy and it feels natural to use a *logical document structure*. Such a logically structured document could look like this:

title: title of the document

author: the one who wrote it

table of contents

list of tables

chapter: preface
  text...

chapter: intro
  text...

chapter: main
  text...

chapter: conclusion

appendix

bibliography

index

You can divide your work in as many smaller parts as you like. Reusing parts in other documents is almost trivial.

TEX files can be written using any editing program, as long as that program can save the file in plain ASCII — which is a standard feature of just about every editor or word processor you can find. Therefore you are not limited to the specific editing program that is built into a word processor but you can choose your own and configure it as you like. If you prefer Notepad, that is fine too. If you prefer a professional programmer's editor such as PFE, MED or WINEDT, go right ahead.

Since the editing phase and the typesetting phase are separated, there is a repeating cycle that you will go through:

- enter text
- typeset this text
- review the results

Figure 1.1 is a graphic representation of this cycle. If you have any experience in com-

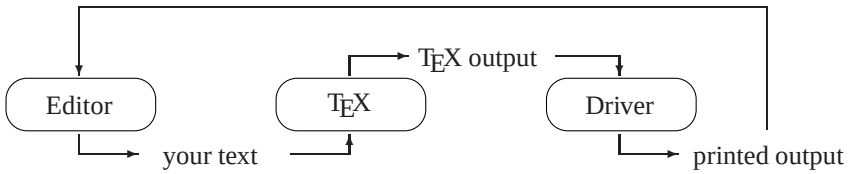


Figure 1.1: A text editor, TEX compiler and printer driver working together

puter programming this cycle will look familiar to you. Indeed you can regard TEX as a 'compiler' that takes a plain ASCII text as input, and generates typeset pages. But anyway, by now you will probably wonder what the point of all this is.

1.2 Why TEX?

In a time where WYSIWYG desktop publishing is so familiar it may seem odd to promote a system such as TEX. But for certain needs you may even feel that TEX is the best or even the only suitable system.

Let us have a look at some of TEX's strong points.

- High quality output: TEX does a great job in following the rules of traditional typesetting. This may not be obvious at first but once you have compared TEX's output and the output of many modern word processors, typesetting the same text, you will begin to see. You may even get hooked.
- Stability: TEX has been around since 1978. Only minor changes and bug fixes were applied over the years. This means that a TEX file written 15 years ago can still be processed by any current TEX system. The output will even be identical. TEX is probably the only program on earth that has its full source code interleaved with documentation published in hard cover. Stability in a different meaning also applies: TEX rarely crashes, and what is more important: you can produce texts of any size (thousands of pages if you like) and TEX will typeset it, even if your computer has very little memory available.
- Programmability: TEX is a macro language that allows you to do incredibly complex things with just a few simple commands. If you want you can redefine just about anything TEX does for your specific application.
- Flexibility: although TEX has been around for so many years there have been very few changes to its core. However, because of the way the core is set up, TEX users all over the world have been able to make TEX do anything they need. To name but a few examples: in TEX you can typeset text in just about any language, including non-Latin languages. Even languages written from right to left (e.g. Arabic) or languages with huge character sets (e.g. Chinese) can be typeset by TEX. You can mix any number of languages within one text. You can typeset music, or chess games, or ...

- Simplicity: TEX documents are written in plain ASCII. Even if you don't have a TEX system around you can still read almost all of it using any editor or word processor near you. In case a TEX file becomes corrupted (e.g. when copied on a bad diskette) you can usually repair it without special tools because it has no proprietary file format. This also explains why it is so easy to exchange TEX files or pieces of TEX code through electronic mail.
- Availability: TEX has been implemented on almost any computer system you can think of. To name but a few: Atari, Apple Macintosh, Unix (all flavors), VMS, CMS, MVS, MS-DOS, OS/2 and MS-Windows[2] (all flavors). Files can be exchanged from any system to any other and will run just fine.
- Low price: TEX is free software. The sources of the program are free too. You don't pay anything for them. You may have to pay for a specific implementation or the media on which a TEX system is shipped to you, as well as any manuals or books that come with it. There are several very good TEX implementations around for free, and there are a few commercial versions that usually have some extra features (e.g. a more or less WYSIWYG interface) and include a professional helpdesk.
- Superb support: since there is no one company exploiting TEX, users and implementors around the world have set up global means for giving support. Usually this support is free, too — that is the spirit of TEX. Support is given mostly through Internet (email, WWW, Usenet and FTP) and sometimes through telephone or fax.
- And last but not least: TEX is fun to work with. It is both challenging and rewarding. It is addictive. You wouldn't be the first TEX user to come to the conclusion that TEX has it all and will serve you a lifetime.

We like to share our enthusiasm with you but we think it is only fair that we also tell you about the disadvantages.

- Learning curve: when you start using TEX you will often have to refer to a manual to look up certain commands. You will also find that TEX often doesn't understand what you are trying to do. In such cases it will confront you with an error message. You will get used to this but the learning curve will be steep.
- Poor error recovery: in case an error message appears it may be very difficult to deduce its true meaning or relevance. Often the messages are obscure, especially to new users.
- Strange language: a macro language such as TEX is probably quite different from any other computer language you may be familiar with. If you intend to write your own macros you need to gain quite some insight before you can write robust and reliable macros with confidence.
- Not WYSIWYG: some people prefer that. There are some commercial WYSIWYG implementations on the market, but so far they never quite feel like ordinary word processors.

2 'MS' stands for Microsoft. In the rest of this book we will simply write 'Windows' when we mean Microsoft Windows.

- Everyone uses *SomethingElse*: TEX is not a mainstream system. If you are writing a book together with colleagues, chances are that they don't have a TEX system on their computer.

Now that you have some idea of what TEX is about, let us see in what areas and for what purposes TEX is suited or not.

1.3 Where and when TEX?

Although TEX can be used for almost any text related task we don't think TEX is always the best choice. In this section we want to list areas in which TEX shines and areas in which it may be wise to choose a different tool. Remember that 'the right tool for the right job' always works out best.


Traditionally TEX has been very popular in scientific environments. In areas where people write about mathematics we dare to state that TEX is still by far the best tool. Writing math in TEX is a pleasure and feels natural: TEX's math syntax is almost like you would dictate it over the phone.


Other kinds of documents require direct control over the exact position of all sorts of elements, such as text blocks, pictures, backgrounds and logos. If you design glossy magazines that depend heavily on a designer's touch on each page, then you will be better off using a dedicated desktop publishing program.


If, however, your documents are by nature (or can be) logically structured then TEX can make your life very easy. A logical document structure will allow you to leave many tedious tasks to TEX. And it will make your text easily reusable and convertible to HTML, so you can publish on the Internet.


Some people produce a thousand pages per year. Others only a few. Some people write many short letters, others one or two books. Remember that learning TEX takes some time. Unlike most modern word processor you can't simply 'start TEX' and then start typing. And if you use TEX only occasionally you will find that you never really get the hang of it. So use it only for big or regular tasks for which the investment will pay off.


In case you need to create documents with lots of cross-references, citations, footnotes, endnotes, marginal notes, bibliographic references, etc., TEX should be your ideal partner. Believe it or not, all such references, including bibliographies, glossaries and indices can be resolved by TEX automatically. You will never use 'hard-coded' numbers again. TEX will number anything you want automatically.


TEX, in its infinite wisdom, has it own ideas about fonts. It doesn't use the fonts that your computer operating system uses. Fonts to be used by TEX need to be installed specifically for TEX. However, you need not worry about this because

De uitbreiding van de wet voorziet in een koppeling van de bestanden van de Informatiebeheergroep (IBG) en de bevolkingsregisters. Deze maatregel is getroffen om de aanspraak van studenten op een zogenoemde uitwonendenbeurs te kunnen beoordelen.

De uitbreiding van de wet voorziet in een koppeling van de bestanden van de Informatiebeheergroep (IBG) en de bevolkingsregisters. Deze maatregel is getroffen om de aanspraak van studenten op een zogenoemde uitwonendenbeurs te kunnen beoordelen.

Figure 1.2: Typesetting a paragraph

The left column is a paragraph as it was typeset in a Dutch newspaper (De Volkskrant, February 7, 1998). The right column is the same paragraph typeset by TEX. Notice how TEX speads spaces more equally over the entire paragraph, avoiding many spurious hyphenations.

almost everything you may ever need comes with any modern TEX distribution and is ready to run.

Over the years TEX has been known for producing exceptionally good looking documents. Not that it is impossible to make ugly documents using TEX — we have seen many examples — but TEX adheres to fine typesetting traditions. For instance, it automatically takes care of ligatures and kerning, and unlike most word processors it formats text not line by line but takes a whole paragraph at a time. This way it is able to find the optimal spread of interword space in combination with a minimum of hyphenations (see figure 1.2 for a demonstration). If you want the best, you want TEX.

As explained before, TEX is 'only' a typesetting engine. If you need integrated drawing and painting programs, spreadsheets and more, TEX will feel rather old-fashioned. Of course you can incorporate pictures, tables and such produced by other programs, but integration this is not.

Nevertheless, TEX is very good at automatically placing so-called 'floating objects' such as tables and figures, so you don't have to worry about these moving targets. Instead you can concentrate on your text.

If you write documents that contain text pieces in different languages TEX is an excellent choice. It can deal with many different languages, taking care of hyphenation rules and specific typographical details such as diacritics.

Using TEX you can create wonderful documents with that may include incredibly complex mathematics. But if you want to publish your work and your publisher doesn't accept TEX text, he will at least have to reformat the whole document using his own typesetting system. There are only very few programs that can import TEX text. To be more exact: all programs can import ASCII but very few understand that, e.g., the ASCII string `\sqrt[3]{\pi}^\alpha` represents '$\sqrt[3]{\pi}^{\alpha}$' when typeset.

Many scientific and technical journals accept TEX text for submitting articles. Because of TEX's generalized and logical markup the risk of conversion errors are reduced dramatically. As a result turn-around time can be much shorter. See G. Valiente Feruglio's article *Do journals honor LATEX submissions?* cdrom for an overview.

We hope this short description has given you an idea of the kind of tool that TEX is.

1.4 What do you need?

To get started with TEX you will need at least the following:

- Basic knowledge of what TEX is. Read this chapter and the next chapter carefully if you don't.
- A computer. If you want to install TEX from the 4allTEX CDROM it should be a computer running Windows 95, 98, NT 4.0, or 2000.
- This book. It contains a description of the system as it runs on Windows (part II) and an introduction to the TEX commands that you need (part IV).
- You may need other books since we can't explain everything in detail in this book. In chapter 5 we will give some recommendations. On the 4allTEX CDROM you can also find many interesting documents. See appendix D for an overview. Documents available on the CDROM are marked cdrom throughout this book.
- You may need additional support. If you bought a commercial TEX implementation a reliable helpdesk should be at your service. If you want support for the free software available from the 4allTEX CDROM we recommend that you join a TEX users group. A detailed description of support facilities is given in chapter 5.

Chapter 2

TEX through the looking glass

Although we didn't actually tell lies in the previous chapter, there is a lot that we didn't tell you. In this chapter we will dig a bit deeper to explain what TEX is about and how it does its job.

2.1 TEX's origins

TEX was developed by Donald E. Knuth,[1] a computer science professor at Stanford University, California. While working on his magnum opus *The Art of Computer Programming* he found that there was no electronic typesetting system available that could produce his work in a way that would satisfy him. So he started writing his own program. The main work on TEX was done between 1978 and 1984. Afterwards only relatively small features were added and some bugs were removed.

It is true that TEX is very old, compared to many popular word processors, but don't let that fact confuse you. TEX is not outdated; rather, it was way ahead of its time when it was conceived. And in several respects it still is. Thanks to its open nature it is able to adapt to developments that no one could have foreseen.

An unusual aspect of TEX is that its sources are free. The sources themselves are also unusual. They were written in a format that combines documentation and pure source code. Two simple programs called 'Tangle' and 'Weave' are used to produce just the source code or just the documentation. Knuth calls this style of writing 'literate programming'. Sources in this format are called 'web' files (cdrom).

1 Pronounced 'kah-NOOTH'.

The source code of TEX is written in a restricted version of Pascal. The reason for this is that Knuth wanted the program to be as portable to other operating systems as possible. At that time Pascal was the programming language best suited for that job. His 'web' sources provide for 'change files' that implement the changes necessary for any specific operating system or compiler. This clever set-up made it possible to implement TEX on almost any computer system you can think of. And that is what actually happened.

Another important feature of TEX is that its output is *device independent*. The output file is called a 'DVI' file. Device independent means that once TEX has typeset your text it can be sent to any output device (printer, screen, ...) and it will always look the same — within the limits of that device. The exact position of all items on the pages is fixed, down to the smallest character. This means that line breaks and page breaks will not move. For almost any output device you can think of (from simple matrix printers, through laser printers up to professional photo typesetting devices) drivers have been developed to print TEX output on them. Knuth didn't want TEX to be restricted to any particular computer environment or printer, and indeed his generic approach to building TEX has worked out and is still working out.

2.2 TEX is many things

So far we have discussed TEX in rather general terms, as if it were one program. Actually, TEX is a whole bunch of programs. And, to add to the confusion, the word 'TEX' can have many different meanings. When we talk about TEX we could mean:

- the typesetting system
- the whole system of programs of which any implementation consists
- the macro language
- the macro language plus the macros that Knuth wrote ('plain TEX', we will discuss it in detail in chapter 16)
- the macro language plus any other 'macro package', such as LATEX (which we will discuss in chapter 17)

In practice all these different meanings are being used and it isn't always clear what particular meaning you should assume. Sometimes even someone who talks about TEX doesn't really know. But then again, in most cases the context is so clear that there is no need for tedious formalizations.

As stated above, a TEX system consists of a whole bunch of programs. In chapter 13 we will introduce you to all the components. For now, it is sufficient to know that there are two major programs:

- the 'compiler': the program that reads your text, formats it and generates a DVI file as output

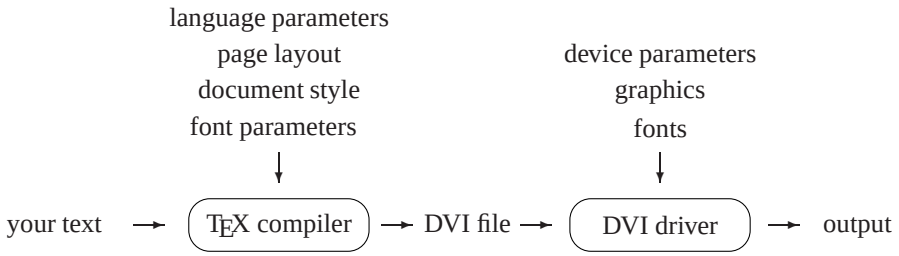


Figure 2.1: The files used by the TEX compiler and DVI

- the DVI driver: the program that reads the DVI file and renders it on an output device such as a computer screen or a printer

Actually, you will be using more programs. An ASCII text editor is essential. Strictly speaking this component is not part of a TEX system although you will be spending most of your time entering and editing your texts, not running the TEX compiler.

To make things even more complicated, there is more than one TEX compiler available, and of course there are more DVI drivers, simply because there are so many different output devices out there. In chapter 13 we will explain the options.

2.3 TEX uses many file types

So far we have only mentioned two types of files that you will use: your own text files and the DVI file that TEX produces from your input. But TEX will actually be using many more files.

In figure 2.1 you can see a few more of the different file types that the TEX compiler and a DVI driver will typically need. Of course you don't have to provide all these files yourself: almost all of them are already installed in any TEX implementation and you will probably never even notice them. We mention them here anyway so you will get some insight into what is going on when TEX is compiling your document. Understanding the process will make it easier for you to diagnose problems when TEX generates warnings or error messages.

The file types shown in figure 2.1 still represent only a fraction of the whole picture. Table A.1 in appendix A should give you an impression of the file types and their meaning that can participate in a complete TEX system. But let us first take a closer look at the file types depicted in figure 2.1.

As we have explained before, 'your text' is a plain ASCII file that you have written. You can name the file anything you like, but it is a good idea to use `.tex` as the file name extension. If you do so you can feed the file to the TEX compiler without specifying its

file name extension, because `.tex` is the default. So far we have assumed that plain ASCII is a simple and universal concept. Unfortunately that is not entirely true. It is true as long as you restrict yourself to so-called '7-bit' input. This includes all non-accented letters, digits, punctuation marks and other characters such as &, $ and the backslash (\). The backslash is usually used as the starting character of TEX commands (in chapter 15 we will explain the ins and outs of the TEX language). In more technical terms, you can use all characters from 32 to 127 from the ASCII chart, nothing else.

But what if the language in which you write your text requires accented letters or other special characters? Basically, there are three ways to solve this problem.

1. TEX supports plain ASCII representations for any character you can think of. For example, if you need accented e's in the French word as `élève` you can write `\'el\‘eve`. So in fact you use TEX commands to enter characters that may or may not be available on your keyboard, yet you are still using plain ASCII. Many, many such commands are available.

2. If you hate to write so many commands, TEX can make your life a little easier by 'intelligently' interpreting your text. Suppose you want to write the Dutch word `geïnd`. There are (at least) four ways to do that, all of them awful: `ge\"{\i}nd` or `ge\"\i{}nd` or `ge\"\i nd` or `ge{\"\i}nd`. It is illegible and (therefore) prone to error. But the good news is that you can tell TEX that whenever you write a double quote followed by an 'i' you want an ï. Then you could write `ge"ind`, which looks a lot better. Similar constructions can be made for other accented letters. Note that using this simplified notation you are still writing in plain ASCII.

3. If you want to enter accented letters as such it means you will be using '8-bit' input because these letters are not part of plain ASCII. There are many definitions of the 8-bit tables for specific languages that you may want to use. They are usually called 'codepages'. If you tell TEX what codepage your text is based on, TEX will know whether a certain character you have written represents a 'ñ' or a 'ø'. See the ISO/IEC report 10646-1 on *Universal Multiple-Octet Coded Character Set* for an exhaustive description of codepages defined in ASCII and Unicode. In section 10.1 we will explain in more detail how different input encodings can be dealt with by TEX.

You can imagine that similar problems occur when using TEX for non-Latin languages. But it can be done. TEX can typeset just as well Cyrillic, Arabic and Chinese or Icelandic and mix them in any combination. In section 10.3 we will explain in more detail TEX's multilingual capabilities and limitations.

In figure 2.1 we have presented 'language parameters' as a separate item because there is more to writing in a specific language than just codepages. Hyphenation is another important aspect. You will have to tell TEX in what language you wrote your text, and TEX will have to know the hyphenation rules for that specific language. Specifying those rules is far from trivial so it is better left to specialists. Fortunately hyphenation rules have been specified for many languages so you probably don't have to worry about that. But there are a few pitfalls. For instance, you can't tell TEX to load a set of hyphen-

ation rules at any time you like. Neither can you tell TEX to load rules for *all* possible languages. That would overwhelm the TEX compiler. In sections 10.3 and 13.3.1 we will explain strategies to make TEX typeset your text just the way it should.

Another complication (or challenge, if you prefer) is the font parameters. The TEX compiler doesn't need to know exactly what shape a certain character has; it only needs to know how much space the character needs, so it can build lines, paragraphs and pages according to your specifications. These font parameters are stored in so-called 'TEX font metric' (TFM for short) files. As a rule, font metrics are freely available for any font you can think of, even though the font itself may be only commercially available. For instance, the font metrics for the font 'Times Roman' that was used to typeset this book can be found on the 4allTEX CDROM. So TEX can compile your text using Lucida Bright font parameters, but you will not be able to *print* or *preview* the typeset text unless you buy the fonts. Experts often use the terms font *metrics* and font *glyphs* to distinguish between character measures and actual shapes. The *shapes* are what you pay for. A special case of shapes are the fonts built into printers. Almost any PostScript printer has the shapes of the font 'Palatino' built in. Therefore you don't need to buy that font if you own such a printer. So you could use TEX to typeset your text using Palatino font metrics and print right away. However, if your printer were a standard laser printer, then the DVI driver would tell you that the font shapes you requested were not available.

So now we have already seen an example of the 'device parameters' in figure 2.1: the DVI driver needs to know what fonts are built in. It also needs to know how to download fonts if necessary. In case font shapes have to be generated from METAFONT sources the DVI driver will tell Metafont which fonts need to be generated, in what resolution, and which set of font generation parameters apply to the given output device. In section 8.3 we will explain how METAFONT uses this information to achieve the best possible results on any output device.

In figure 2.1 we listed 'document style' and 'page layout' as another two items that TEX needs as input when typesetting your document. By 'document style' we mean a general set of parameters and commands that define the layout and function of a document. For instance, a technical manual will require a different style than a book of poems. The 'page layout' is a more specific definition of e.g. the text height, text width and margins. It depends greatly on the TEX dialect that you will be using how exactly these items are implemented. In chapter 15 we will discuss a three such dialects. You will see that document style and page layout can be specified in many ways and that there are lots of TEX parameters and commands that enable you to produce beautiful documents.

Looking more closely at the DVI driver in figure 2.1 you may be surprised to see that we listed the item 'graphics' as input for the DVI driver, not for the TEX compiler. The reason for this is simple. Any graphic not written in the TEX language itself is not included in the DVI file. Instead only a *reference* to that graphic is included. It is the DVI driver's job to insert the graphic at the point indicated in the DVI file. This may seem odd but it allows TEX to deal with any kind of graphic, even those that don't exist yet. TEX only needs to know the file name of the graphic and the amount of space it requires, the rest is left to the DVI driver.

This is how the TEX system deals with new developments (relative to TEX itself, that is) such as different types of graphic formats (e.g. GIF, EPS and PNG), new output devices (e.g. color deskjet printers and Braille printers) and new output formats. Examples of these are HTML (HyperText Markup Language) and PDF (Portable Document Format)).

Another advantage to this approach that we would like to mention is that your document always stays very compact. Graphics are not included in your document source, they are only referenced. Only when you need real output will the graphics be used. This explains why TEX is able to typeset thousands (or millions, we haven't tried) of pages on which again thousands of graphics may appear. It will not slow down and just one or two megabytes of internal memory is still enough. You may object that TEX doesn't solve any problem here; it merely puts the burden on the DVI driver. This is correct. But then again, we think that is exactly what a typesetting system *should* do. It should typeset, no more, no less. The right tool for the right job instead of one bulky tool for any job.

2.4 TEX is many programs

A TEX system consists of a large set of separate programs that cooperate. In appendix B you can find complete flow charts that show the relations between (nearly) all programs and file types in a complete TEX system.

In chapter 13 we will explain the purpose of many of the programs involved. Many of them are invoked automatically, without any user interaction. Only a few of them are very 'visible' to any common user, most notably of course the TEX compiler.

Many of the programs may launch each other on the fly when needed. If, for instance, the TEX compiler does not find the metrics of the font that your text requires, it may pause and call METAFONT to generate it first. A DVI driver may call METAFONT to render a certain font at a certain resolution if it is not available yet.

You may be dazzled by all this complexity, but remember that you are not required to understand all or even part of it. If you simply want to work with TEX you can forget the details and remember only that a description is available in case you become enthusiastic and want to know more. It happens.

2.5 The future: new developments

We have stated that TEX has not changed significantly over a decade and we considered that a good feature.

Over the years TEX has been able to adapt to new requirements quite easily because of its generic character. For instance, support for PostScript fonts or for inclusion of graphics produced by other systems could be implemented without any change to the TEX kernel. TEX was even the first text system to generate HTML files (for publishing on the World Wide Web) automatically. All this was done either by clever TEX

macro programming or some additional programs that cooperate with TEX programs and files. More recently TEX was extended in order to generate PDF without any extra (commercial) tools such as Adobe Acrobat Distiller.

Nevertheless, the TEX community understands quite well that TEX has to develop in order to survive. Today's demands are not the same as ten or twenty years ago.

In section 15.4 we will discuss in more detail the future of macro programming. Here we list a few new developments that go beyond programming macros. These new programs change the TEX kernel in order to add new features.

PDFTEX This is a TEX compiler that is fully compatible with any standard TEX compiler, but has the ability to output PDF instead of DVI. It can still produce standard DVI output.

ε-TEX This TEX compiler is also fully compatible with any standard TEX compiler, but supports an 'extended mode'. In extended mode a number of new features and enhancements are available. ε-TEX's extensions mainly consist of better programming tools and more typographic control over output. Eventually ε-TEX will replace TEX, we think.

Omega This is almost a complete rewrite of the TEX program that supports Unicode (see the glossary, page 495 for a description). Omega has a lot of other features that go beyond Knuth's original ideas. It takes a different approach to characters and their flow, which is necessary for typesetting many non-Latin texts that may not run simply left to right, top to bottom, even in all kinds of combinations. 'It's TEX, Jim, but not as we know it', to paraphrase a character from a science-fiction series.

NTS 'NTS' stands for 'New Typesetting System'. Knuth has stopped developing TEX, deciding that anything that goes beyond TEX should not be called TEX. This should prevent any confusion over what TEX is not. NTS is not an actual system yet, but a collection of ideas and concepts that will be implemented in a successor to TEX. Naturally it will inherit many features that we love about TEX, but it will be much more advanced in many respects. Eventually NTS is expected to replace TEX/ε-TEX.

MLTEX This extended version adds the command `\charsubdef`, which allows for easier usage of 8-bit input and hyphenation. 8-bit input is especially important when writing text in languages that require many accents. Note also that 8-bit input can also be enabled through different mechanisms described in more detail in section 10.1. MLTEX is a minor extension when compared with ε-TEX.

At the moment PDFTEX is a serious alternative to standard TEX which can only output DVI. It has a few minor drawbacks but they will probably never bother you, especially if the ability to produce PDF is very valuable to you. See section 6.3 for more details on this program.

ε-TEX is fully functional, though the ε-TEX team is still working on it. So far we have not seen many applications that require the extended mode supported by ε-TEX, but we expect that they will appear soon. ε-TEX may become the new standard since

it provides full compatibility with 'older' systems. Very recently PDFTEX and ε-TEX were merged into PDF-ε-TEX.

Omega takes a very different direction. In the future we will certainly need support for Unicode. The improved memory management and filtering capabilities that Omega supports are quite impressive. However, at this time there are still very few fonts available for Omega, and there is very little experience with the system. We feel that for the moment it may be too premature to deploy this system in a production environment. That is why it is not supported in the run-time system supplied on the 4allTEX CDROM. PDFTEX and ε-TEX are. However, if you feel confident, you can install it yourself from the 4allTEX CDROMs.

Chapter 3

Installation of 4TEX

In order to integrate TEX into a full-fledged Windows text processing environment you need more than a TEX compiler and all related programs.

First, you will need a Windows editing program. Secondly, to make life easier, you will want a true Windows interface that is much more user friendly than a bare command line. And sooner or later a lot more wishes will come up...

4TEX is an attempt to implement such an environment. Many of the TEX programs and several others can be selected and started from a small program that supports the familiar Windows look and feel.

4TEX can start programs as concurrent processes. This means that you can have several programs running at the same time. For example, while you are compiling a large document you could edit another document, or you could *copy and paste* information from an Internet web page to your bibliographic database while drawing a graph with a plotting program.

Starting a program is as easy as a mouse click. 4TEX tries to shield you from tedious command line switches and weird syntaxes that some programs require by providing defaults or presenting them in a more user friendly format.

Nevertheless, if you don't know what a program is supposed to do, you may have a hard time using it. But don't worry about that now, it will all come to you if you are a little patient. In the following sections we will guide you through the (very simple) installation stage, and we will show you how to run your (possibly very first) TEX job successfully.

After that we will explain how to modify and customize 4TEX to your specific environment and preferences. We will describe what resources 4TEX uses in order to run effectively so you will know where to look if you want to change anything. Naturally we will also give you hints for troubleshooting, just in case something doesn't work as you (or we) thought it would. If you are able to install 4TEX and you want to stick to the defaults we have chosen you could skip those sections entirely.

Starting from section 6 we will take a closer look at 4TEX from a pure user point of view. We will explain in detail how to use 4TEX and all the programs that you can run from within the 4TEX environment. Many examples are included to make you familiar with 4TEX and the programs within.

Of course you don't need to learn all this by heart. There is always help available. And remember that no one ever became an expert on anything overnight. Especially if you are new to TEX we advise you to start with very simple, almost trivial documents. Once you understand the basics you can move on to more advanced features such as including graphics, making indices and setting up bibliographic databases.

We will assume that you are familiar with common features of Windows programs such as 'buttons', 'radio buttons', 'check boxes', 'edit boxes'. If not, you can find a short description in the glossary (page 495). Nevertheless, we will try to avoid technical terms if other, more understandable, terms are available.

3.1 Installing the software

In order to make 4TEX and all associated programs work right away in a Windows environment, some changes in the Windows *registry* have to be made, and a few files will be installed on your hard disk. The installation program will do this for you.

In case you only want to run TEX programs from the command line, that is, *without* a graphic interface then installation is a bit different. In fact, it should be sufficient to add the path `d:\bin\win32` to your `PATH` environment variable (assuming drive `d:` is your CDROM). Further details about how to configure TEX will be discussed in chapter 13. If you choose to take this route we assume that you know enough about TEX and the operating system to get things up and running in a few minutes.

If you prefer the more usual installation programs featured by most Windows programs then read on carefully. We will explain all the ins and outs of your options for installing 4TEX.

On most Windows systems the installation program will start automatically when you insert the 4allTEX CDROM in your computer. If it doesn't, you can launch it manually by double clicking on the `setup.exe` file in the root of the CDROM. The installation program will make some changes to the Windows registry that 4TEX, TEX and/or other associated programs require. It will also allow you to select which parts of the whole system you want to install on your hard disk or network. After that you will be able to start 4TEX. Figure 3.1 show the opening screen of the installation program.

The most crucial decision you have to make is about running the software from the CDROM, or copying it to your hard disk. A mixed situation is also possible, in which case you use, for example, TEX input files can be read from the CDROM, while all executables are run from your hard disk.

The more you install on your hard disk, the better the system will perform (in terms of execution time). But there may be insufficient space on your hard disk to hold everything, so we advise you to consider your options carefully. Besides, performance

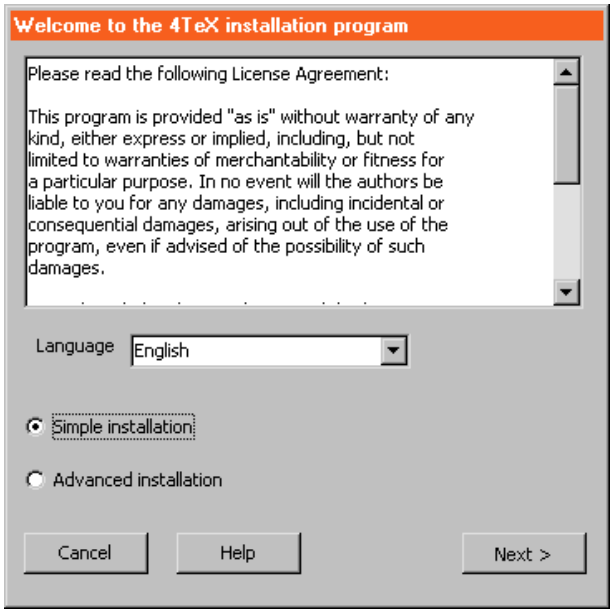


Figure 3.1: The installation program

may be more than adequate when running entirely from CDROM. It may be wise to run 4TEX from CDROM to start with, and decide later if you want to try other options.

Read the installation instructions that the installation program provides, and then choose:

- **Simple installation**
  This option provides two options once you have pressed Next >:

  1. **Run entirely from CDrom**
     in which case only an absolute minimum of files will be installed on your hard disk (less than 50 kilobytes). Installation only takes a few seconds. Choose this option if you just want to experiment with the system, or if your hard disk is already overfull. The whole 4TEX system will be read from the CDROM, so your options for configuring the system are limited. However, this may not be a problem at all.
  2. **Install everything on hard disk**
     This options will install 4TEX in such a way that you will not need the CDROM anymore to run the system.

- **Advanced installation**
  This option provides ways to specify exactly which parts you want to install on your hard disk. Naturally this option requires that you have more than basic knowledge

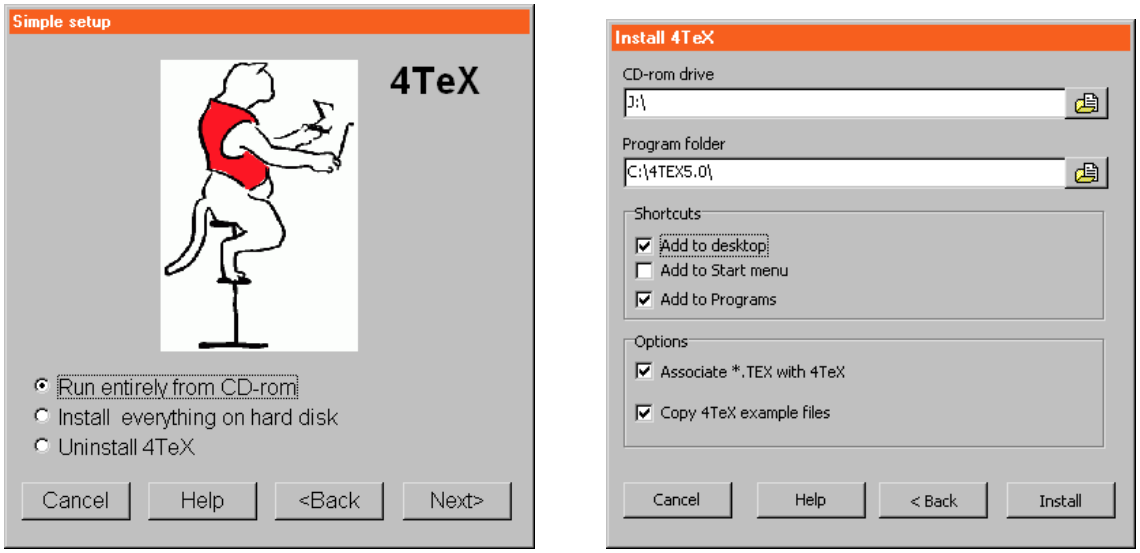


Figure 3.2: Simple installation

of the parts that make up a TEX runtime system. The installation program will provide many 'modules' from which you can select the ones you want. In section 3.1.2 we will explain this procedure in detail.

Note that a complete TEX system consists of thousands of files that will occupy several hundreds of megabytes altogether. In case your hard disk is formatted as a standard 'FAT' partition (not FAT32, not NTFS), you may find that installation on the hard disk requires much more space than you anticipated. A typical TEX system consists of lots of very small files (1 to 4 kilobytes) that may still occupy as much as 16, 32 or 64 kilobytes each because that is the minimum space for any file. FAT is simply not suited for large disks, especially in combination with lots of small files. FAT32 (under Windows 95/98) and NTFS (under Windows NT) are much better choices.

In case you want to install the system to a hard disk that is formatted as a FAT partition, make sure the partition size does not exceed 1 gigabyte. Less than 500 kilobyte is even better.

If you have chosen 'Simple installation', and 'Run entirely from CDROM', you will only need to specify a few more options (see figure 3.2). You can determine the folder in which a few files will be installed; whether you want to add shortcuts to your desktop, start menu and/or programs. If you want you can also associate files with extension `.tex` with 4TEX. If you do you will be able to start 4TEX by clicking on a `.tex` file within the Windows Explorer. To get you started, 4TEX comes with a set of example files. We

advise you to install them if you are new to TEX. Press Install when you have made all choices. When the installation process is finished you can start 4TEX right away.

3.1.1 Online help information

4TEX can display context-sensitive online help information in two ways:

1. As plain HTML files. These will be displayed by your default World Wide Web browser, e.g. Microsoft Internet Explorer or Netscape Navigator.
2. As 'HTML Help' files. These can only be displayed if Microsoft Internet Explorer version 3.02 or higher is installed on your system, preferably version 4 or higher. These files are much easier to navigate than plain HTML files, they offer superiour search facilities, and they are very compact. See also section 6.10 (page 6.9) and figure 6.9.

In order to use HTML Help, your system may need to be updated. Windows 98 and Windows 2000 systems should already be up to date. You can download Microsoft Internet Explorer for free from the World Wide Web (`http://www.microsoft.com`). If your system has Internet Explorer 3.02 or higher already installed, it may be sufficient to run the program hhupd.exe, which you can find on the 4allTEX CDROM. If after running this program online help still doesn't work, then you will have to install Microsoft Internet Explorer version 4 or higher.[1] 4TEX will use HTML Help if it detects that this format is supported by your system.

3.1.2 Advanced installation

The advanced installation will require some more time and thinking, because there are many, many options to choose from.

You should attempt an 'Advanced installation' *only* if you are very familiar with TEX and related programs. If your system doesn't work properly after this, you are on your own.

You will be given the opportunity to select a number of *modules*. These modules make up a specific part of the system. Some modules are *required*, some are *recommended* and some are *optional*. They are listed in three colums (see figure 3.3). Each module may require other modules. If you click on the name of a module it will be displayed in the module information window. You can select or deselect a module by clicking on the checkbox before the module's name.

The installation program will calculate the total size of the selected modules. Use the Recalculate button to get an estimation of the actual disk space required, based on the real size that each file will occupy (see remarks on FAT partitions above). The total

1 Make sure you download and install the Internet Explorer version that matches your Windows system with respect to language.

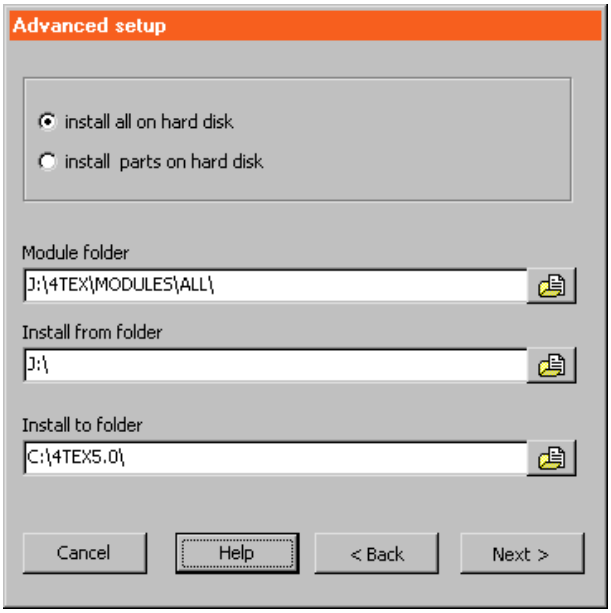


Figure 3.3: Advanced installation

required space should not exceed the available disk space. If it does, you should deselect some modules, or you will not be able to proceed with the installation.

You can run the installation program again later if you want to install other modules or if you want to deinstall modules.

During installation the file `4TEX.INI` will be generated. It will be stored in either the Windows directory (usually `c:\windows` or `c:\winnt`) or in the directory in which `4TEX.EXE` resides. A few more `.ini` files and other configuration files will be installed as well.

In case 4TEX cannot find the file `4TEX.INI` either in the Windows directory or in its own directory, or if the file is invalid, the 4TEX initialization screen will appear on start-up. Unless you know exactly what you are doing, we recommend that you abort 4TEX and rerun the installation program.

In chapter 12 you can find detailed information about customizing 4TEX. All files that are part of 4TEX are explained there.

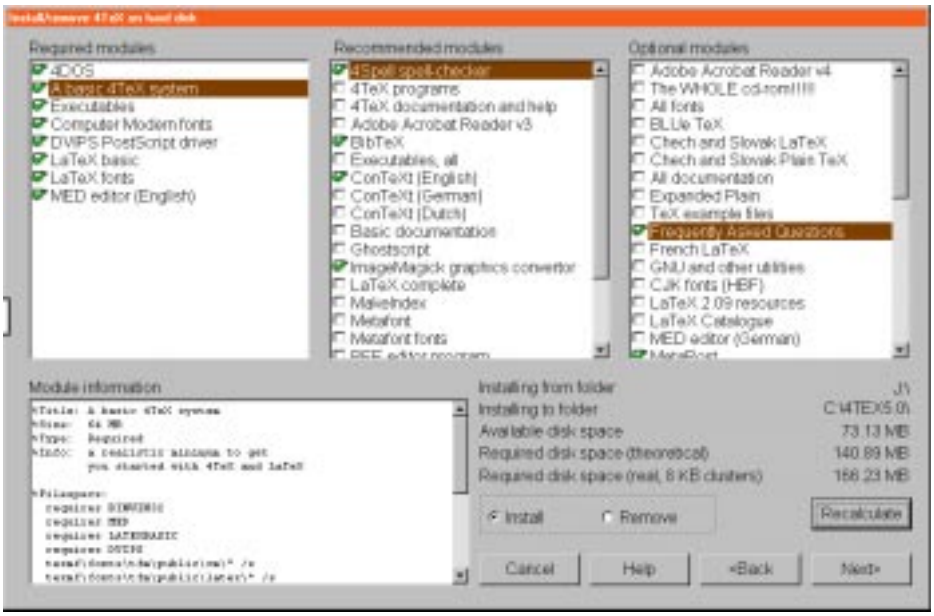


Figure 3.4: Advanced installation options

3.2 Updating the system

The TDS that 4TEX builds on your local hard disk looks like the tree in figure 3.5

This set-up is determined by the file `OWNTREE.LST`. Initially this tree is nearly empty: only one file is stored in `\texmf`: `aliases`; three very important configuration files are stored in `\texmf\web2c`: `mktex.cnf`, `texmf.cnf` and `windvi.cnf`. These are slightly modified versions of the ones you can find on the corresponding TDS on the 4allTEX CDROM. See section 13.8 for a full description of their purpose.

In this local TDS you can store any updates of files because this TDS is searched *before* the CDROM TDS is searched. So, if you have a new version of, say, the LATEX package `graphicx.sty` that is stored on the CDROM in `\texmf\tex\latex\graphics`, you could store the new version in a directory with the same name in your local TDS. If you put it in `\texmf\tex\latex` it will work, too, but things may become difficult to manage.

3.3 Uninstalling

Uninstalling 4TEX is quite straightforward and can easily be done by running the installation program. The program will remove all files in the directory that contains the 4TEX program (provided it is not on the CDROM or on a read-only network drive) and

```
yourdirectory/articles/
             /books/
             /letters/
             /reports/
             /spell/
             /texmf/bibtex/bib/
                          /bst/
                   /dvips/config/
                   /etex/generic/
                        /plain/
                        /latex/
                        /context/
                   /fonts/afm/
                         /pk/
                         /source/
                         /tfm/
                         /type1/
                         /vf/
                   /makeindex/
                   /metafont/
                   /metapost/
                   /tex/context/
                       /generic/hyphens/
                       /latex/4tex/
                       /plain/
                   /web2c/
```

Figure 3.5: TDS tree on your local hard disk

all its subdirectories. If the file `4TEX.INI` exist in the Windows system directory it will be removed as well. File associations, desktop icons and menu entries will also be removed.

Note that uninstalling 4TEX may not remove all files or registry entries that were in generated during the use of 4TEX. Some programs that accompany 4TEX may have written their own configuration files or registry entries that will not be removed by the 4TEX installation program.

3.4 Testing

If you have managed to successfully install 4TEX you are now ready to run the first test. If all is well you will see the 4TEX window much as in figure 3.6.

A few sample files are provided to get you going. The file `latexsample.tex` would be a good starting point. Try to edit, compile and preview it.

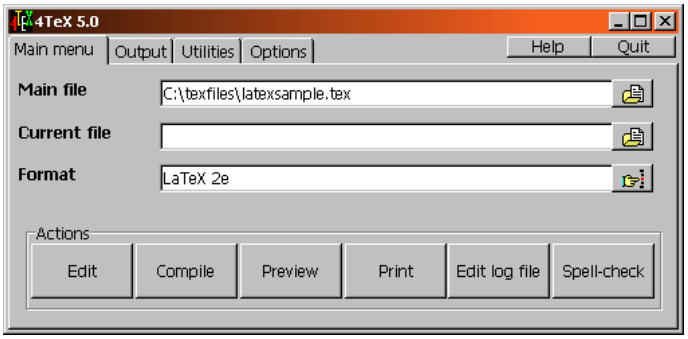


Figure 3.6: The 4TEX main menu

In case you want to start 4TEX from the command line, or if you want to change the short-cut to 4TEX, it is important to understand the syntax of the 4TEX program:

≡ 4tex.exe [*–ini*] [*Main file*] [*Current file*]

Here are some examples of valid ways to start 4TEX:

```
4tex.exe
```

or

```
4tex.exe c:\tex\letters\to_garry.ltx
```

or

```
4tex.exe /private/jobapplication.tex /mylife/curriculum.tex
```

or

```
4tex.exe -ini
```

⚠ Unlike most other Windows programs, 4TEX allows you to use both backslashes and forward slashes in its command line. Actually, 4TEX automatically replaces all forward slashes with backslashes here. See section 12.1 for a description of where and when to use forward slashes and backslashes.

The `-ini` option should only be used by people who understand the system in depth. It can be used to regenerate the files `4TEX.INI` and reset some paramters. However, in general it is recommend to use the installation program to regenerate a lost or damaged `4TEX.INI`.

3.5 Long files names and other pitfalls

Almost all Web2c programs and most other programs assimilated in the 4TEX system are true 32-bit Windows executables. Therefore they can handle *long file names*, in contrast to MS-DOS executables, which expect file names to obey the so-called 8.3 rule. This rule states that a file has a *name* of up to 8 characters, optionally followed by an *extension* of up to three characters. Name and extension are separated by a dot (= a period).

MS-DOS file names are also much more restricted with respect to the characters that can be used within a name. Most notably spaces and multiple dots are not allowed. The (upper or lower) case of characters is not preserved in MS-DOS file names.

In a 32-bit Windows environment these limitations are gone, which is a good thing. Nevertheless we recommend that you don't indulge in exploiting all these features, at least not in a TEX environment.

As a rule, it doesn't matter if you use lowercase or uppercase characters (or any mixture) when specifying *file names* or *folders* anywhere in Windows. In this book we did not attempt to use either lowercase or uppercase file names consistently, simply because it is futile. If you browse the CDROM you will notice that there is no consistency at all in this respect. There are no rules. Note, however, that TEX as a programming language *is* case sensitive. Do not confuse the properties of file names and their content.

TEX programs depend heavily on resources that have specific file name *extensions*. Usually the programs assume these file name extensions (such as `.tex`, `.sty`, `.dvi` and `.tfm`) by concatenating such an extension to a file name that you supply. You can probably imagine that if you enter

```
4tex.exe a sample file
```

the program doesn't understand that `a sample file` is one parameter, not three. Actually, you *can* make it one parameter by putting double quotes around the whole name, but we think it is better to avoid the problem from the start.

Another pitfall is the dot. In Windows this is a character like any other, *not* a separator. So, if you enter

```
DVIPS sample.
```

the program will actually try to open `sample..dvi` — and fail.

Things get even more tricky when writing a TEX document. The TEX command to input another file within the current file is `\input`, and this command takes one parameter. The file name extension `.tex` is default. You will now understand that if you write

```
\input a long file name
```

TEX will try to input `a.tex` (which will probably fail) and `long file name` will be treated as some words you typed.

Note that in TEX documents some characters have a special meaning, so you should avoid them in file names.

To summarize, here are a few rules that we recommend you follow to keep you out of trouble in a TEX environment:

- Do *not* use spaces in file names. You could use hyphen character instead, if you want.
- Do *not* use more than one dot in a file name. And do not *start* a file name with a dot.
- Stick to the characters `a`–`z`, `A`–`Z` and `0`–`9` if you can. At the very least you should avoid: `#$%^&{}[]’‘`

Note that in general upper- and lowercase file names are equivalent in a Windows environment.

Chapter 4

Running TEX

Once you have written your first TEX document, be it in Plain TEX, LATEX, CONTEXT or any other dialect, you will want to run it through the TEX compiler.

In any typical TEX run you will see many messages scrolling over the computer screen. These messages inform you about the progress TEX is making, what files it is reading, what pages it is writing to the DVI file.

These messages may well run much too fast to be readable. And even if you can read them you may need more time to study them in order to really understand what is going on.

Therefore the TEX compiler, METAFONT compiler and METAPOST compiler also write these messages to a *log file* for later reference.[1] This log file actually contains even more information than is displayed on the screen.

When you first browse through a log file you will probably feel uncomfortable about all kinds of mysterious messages, but we hope you will soon find out that you only need to scan for a few important messages, such as warning and error messages.

4.1 Dealing with warnings and errors

Sooner or later (probably very soon) TEX will confront you with warning messages and error messages. It is important that you learn how to interpret these messages and how to deal with them.

All warnings and error messages that are written to the console will also go to a log file. But remember that the log file usually contains more than that. It is an elaborate listing of all events that occurred during the last TEX run. The log file has the same file name as the main TEX file you are compiling, but its extension is `.log`. From

1 Unfortunately not all TEX (related) programs write a log file so it may not always be easy to trace everything.

the main menu you can simply click on Edit log file. Right-clicking this button will show the last (often most important) lines of the log file.

4.1.1 Warnings

Warning messages are relatively harmless. They may indicate, e.g., that TEX found that it had to stretch spaces inside a paragraph a little more than usual, so the output may look less attractive. Or maybe it couldn't stretch spaces enough to prevent a word from sticking out into the right margin. In such a cases TEX will notify you but it will continue so you can ignore the message for the time being. Maybe you will even not notice the slight imperfection that TEX reported. Here are two examples of such a messages:

```
Overfull \hbox (1.26472pt too wide) in paragraph at lines 16--25
[]\OT1/hlh/m/n/10 It was back in Febru-ary 1991 when Maarten
 []

Underfull \hbox (badness 2035) in paragraph at lines 289--293
\Times stu-den-ten op een zo-ge-noemde
 []
```

The first warning tells you that, given the current constraints on paragraph formatting, this line could only be typeset overflowing into the right margin by about one third millimeter. So what should you do now? First of all: watch carefully how it *looks* in the output. Then you could decide to disregard this 'problem' (for now). Or perhaps you can loosen the paragraph constraints: see e.g. section 16.14 for hints. Or maybe you can only solve the problem by rewriting or rearranging that paragraph.

In the second warning TEX tells you that the spaces in that line had to be stretched more than usual, possibly leading to ugly output. The same rules apply here. Note that the hyphens in these messages indicate what possible hyphenations TEX considered.

More important are warning messages about unresolved references. TEX dialects such as LATEX and CONTEXT keep track of references by writing them to an auxiliary file. During the next TEX run this auxiliary file will be read, so only then all references, whether they are forward or backward, will be resolved. So if you introduce a label and a reference in your text it is very probable that during the first run TEX will issue a warning message. When running TEX a second time that warning message should not reappear. If it does, you may have made a typing error that you should correct.

Warning messages about unresolved references to bibliographic items are similar. In section 13.7.1 methods to use BIBTEX references are discussed in detail.

4.1.2 Error messages

Errors cause the compiler to stop and wait for you to decide how to proceed. There are 5 major classes of errors:

1. TEX needs a file that it can't find. Typically the error message looks like this:

   ```
   ! I can't find file 'myfile'.
   l.1 \input myfile
   Please type another input file name:
   ```

   This error may be caused by a misspelled file name. The file may also actually exist but in a directory that TEX doesn't search. Regenerating the index file(s) 'ls-R' (see section 13.8) may also help.

2. Your document contains an unknown TEX macro or control sequence. Typically the error message looks like this:

   ```
   ! Undefined control sequence.
   l.1 \nopindent
   ?
   ```

   You will have to correct the text and rerun this job.

3. There is a mismatch of *begingroup*s and *endgroup*s. In case there is one curly brace too much the error message will look like this:

   ```
   ! Too many }'s.
   l.2 Try this.}
   ?
   ```

   In case there is one (or more) curly brace too much this may be completely harmless. In general it will cause no error message but at the end of the compilation TEX will write something like:

   ```
   (\end occurred inside a group at level 1)
   ```

4. A control sequence that is only allowed in math environment was found in normal text. The error message will look like this:

   ```
   ! Missing $ inserted.
   <inserted text>
                   $
   <to be read again>
                      \alpha
   l.2 \alpha
   ?
   ```

TEX proposes to insert $ so a math environment is started, which may or may not work, but anyway you will have to correct your document.

5. You requested a font that TEX can't find. The error message will look like this:

```
! Font \myfont=tims not loadable: Metric (TFM) file not found.
<to be read again>
                   \myfont
l.3 \myfont
?
```

   You may have made a typing error, the index file(s) ls-R may need updating, or the font is simply not available on your system.

By default TEX runs in so-called `\errorstopmode` which means that it will stop at any error and wait for user input.

When an error message appears and you are prompted to input something after the question mark, you can type one of the following characters and then press Enter:

h The compiler will give you more, hopefully helpful, information on the error, if any is available. After that you can select how to proceed again.

Enter Ignore *this* error and continue.

s Ignore this error, continue and scroll any further errors messages. This is equivalent to `\scrollmode`.

r Ignore this error, continue and run the rest of this job without stopping. This is equivalent `\nonstopmode`.

i Insert a line here. Note that this line is *not* inserted in your text, it is only a line that should put the compiler back on track for this run. You should correct your input file later, so it is a good idea to remember the line number that TEX reported.

e End this run and start the editor. If correctly configured, the editor will load the file in which the error occurred, and it will show the line that the compiler stumbled on.

q Ignore this error and continue in quiet mode. Note that this is not the same as **r** or **s**. In scrolling mode all errors are ignored; in quiet mode a missing file is considered a fatal error on which the compiler will abort. This is equivalent to `\batchmode`.

x Stop this run, 'exit'.

If your text requested (directly or indirectly) an input file that TEX couldn't find it will pause and ask you to supply another file name. It will tell you what file it was trying to find, so maybe you can determine that it was just a typing error. In that case you can simply type the correct name and proceed. Beware that if you supply a file name with a full path, you should use forward slashes, not backslashes.

If you think TEX doesn't really need that file you may enter `null.tex` and proceed. This is actually an existing empty file, a dummy.

If you think this is a serious error that needs to be corrected first you can abort the compilation after pressing Ctrl Z. A more graceful way to end the compilation is to enter `x`. In that case the file `x.tex` is read. Hopefully there is no such file in your own directory of TEX files, so a file from the standard distribution is used. It contains a few TEX commands to end the job.

4.2 Finding errors

Now it is time to point out a few pitfalls and tricks that may be helpful to know. After all, TEX error messages *will* appear, even if you are an experienced TEX programmer. This is not something to panic about, as long as you know how to deal with them.

In some cases you may want to abort a TEX compilation, e.g., if you realize that you have made an error that makes this (time consuming) compilation useless. Pressing Ctrl C will interrupt the process. TEX will ask you how to proceed and probably you will enter `x` as explained in the previous section. Ctrl C can also be used if you think that TEX has entered an endless loop. This is unlikely to happen but TEX is a complete programming language so it is possible.

Even if TEX did not stop at an error and ask you how to proceed, it may have encountered a serious problem that it couldn't recover from. Typically such problems are caused by incorrect settings of memory parameters. We advise you to always check the last page of the log file.

The tail of the log file should look similar to this:

```
Here is how much of TeX's memory you used:
 196 strings out of 10991
 1929 string characters out of 198076
 44691 words of memory out of 263001
 3128 multiletter control sequences out of 10000+0
 3640 words of font info for 14 fonts, out of 200000 for 1000
 14 hyphenation exceptions out of 1000
 23i,4n,18p,128b,160s stack positions out of 300i,100n,500p,30000b,4000s

Output written on test.dvi (1 page, 1044 bytes).
```

If it looks like:

```
! TeX capacity exceeded, sorry [input stack size=300].
\xx #1->\xx {#1
                }\begingroup
l.7 \end
         {document}
If you really absolutely need more capacity,
you can ask a wizard to enlarge me.
```

```
Here is how much of TeX's memory you used:
 194 strings out of 10982
 1891 string characters out of 198671
 43959 words of memory out of 263001
 3135 multiletter control sequences out of 10000+0
 3640 words of font info for 14 fonts, out of 200000 for 1000
 14 hyphenation exceptions out of 1000
 300i,0n,299p,111b,40s stack positions out of 300i,100n,500p,30000b,4000s
No pages of output.
```

you have to read carefully to understand what went wrong and take appropriate actions. In the example TEX reported that its 'input stack' was too small to complete the task. So you could increase the value of `stack_size` in you configuration file `TEXMF.CNF`. But before you try that you should check thoroughly if you didn't make a mistake yourself. TEX's error message may well be a symptom of an ill-behaved macro. In that case increasing the `stack_size` to whatever value will not solve the real problem.

Sometimes TEX files you get from others (who may be running TEX under a different operating system) contain lines that are many thousands of characters long. While porting the file to your system the line terminations were probably lost. By default there is an input buffer of 10,000 characters. This value is determined by the variable `buf_size` in the configuration file `TEXMF.CNF` (see section 13.8.1). However, you should *not* change that value; instead, you should edit the input file that contains the long lines and reformat them into lines of 'normal' length, say, 70 characters.

If you mistakenly forget to write a closing statement in your text (such as `\end{document}` in LATEX or `\bye` in plain TEX), TEX will process your text and then show a 'prompt': `*`. Now TEX expects you to type something: `\end` would be a good idea. Of course, you should correct your text to avoid this strange behavior.

A more cryptic error message is this one:

```
! Runaway argument?
```

It means that you invoked a macro that takes a parameter, but the parameter you supplied is more than one paragraph. This particular macro considers that an error. LATEX's `\centerline` command is an example. Beware that this error message can also occur if you simply forget to write the closing brace.

Once you are more familiar with TEX and TEX macro writing you may want to play with TEX's error tracing commands. They allow you to trace a number things that go on while TEX is processing your text. When an error occurs TEX al-

ways shows a few lines to indicate in what context the error occurred. By setting `\errorcontextlines` to a higher values like this:

```
\errorcontextline=15
```

you will get more contextual information.

A number of switches are available to get extensive *tracing* information. You can switch on a tracing switch like this:

```
\tracingmacros=1
```

Most tracing switches can be set to either 0, 1 or 2. It is beyond the scope of this chapter to explain these switches in detail so we will only list them here. Consult D. Knuth's *The TEXbook* for details on these switches:

`\tracingmacros`, `\tracingcommands`, `\tracinglostchars`, `\tracingparagraphs`, `\tracingpages`, `\tracingrestores`, `\tracingstats`, `\tracingonline`, `\tracingall`.

In section 16.15.5 some of these switches are explained briefly.

Another useful switch for debugging is `\pausing`. If the value of this counter is bigger than 0, the TEX compiler will give you a chance to edit each line of input as it is read from the file. You can also use this switch to make TEX run more slowly because you have to press Enter after each line.

A common source of errors is a mismatch in braces. If you write `\begin[itemize]` instead of `\begin{itemize}` TEX will complain about the `}` that doesn't match any `{`. Or worse, if it *does* match an opening brace that you wrote some lines above you are bound to get some very confusing error messages. A simple trick to find brace mismatches can be done from inside your editor (we assume here that you use MED). If you put the cursor on one of the following characters: `{}[]()<>` and then press Ctrl M the editor will try to find a match. If it doesn't or if the editor points to an unexpected match, that should give you a clue in tracing the problem. A neat trick is to insert an opening brace at the top of the document and then search for a match. Likewise you can insert a closing brace at the end and search for a match.

There are a few tools at your service that can help you determine the source of an error. The program 'LaCheck' can analyse the structure of document and check if you made errors in starting and ending LATEX environments and TEX groups. Note that LaCheck is not restricted to LATEX. It can be useful to check any TEX file (see section 13.7.19 for more details on LaCheck).

With LaCheck program you may be able to trace the origin of this warning that may appear at the end of compilation:

```
(\end occurred inside a group at level 1)
```

The message means that you 'opened' a group somewhere, perhaps using {, but no matching } appeared. The opening may have occurred literally anywhere, so it can be hard to trace. The error could be due to an unclosed font switching command, for instance. Fortunately this error is usually harmless.

The program TEXchk has similar features. You could also use the option 'View TEX project' from the Utilities menu and see if it shows any unexpected structures (see also section 8.9). The spell-checker 4Spell understands TEX syntax upto a certain level. It uses different colors to display commands, parameters, comments, equations and more (see also section 6.9). The editor programs MED and WINEDT also support syntax highlighting for TEX texts (see sections 6.2.2 and 6.2.3).

Remember that the line that the compiler stumbles on is not always the *source* of the problem.

A simple example should prove this statement. Consider the following example:

```
1  And so we conclude that $\alpha x^2 + \beta y^2 = \Omega.
2  This should now be obvious.
3  If $\alpha$ is greater than ...
```

Now the TEX compiler will complain about \alpha on line 3. But there is nothing wrong with that line. The *real* problem is that the $ on line 3 doesn't *start* a math environment, it actually *ends* the one started on line 1. The problem is solved by adding a $ after the \Omega on line 1.

And here is another tricky error. Consider the following lines:

```
1  They pay 200 an hour\dots
2  \vskip 3mm
3  plus expenses!
```

Did you expect the following message?

```
! Missing number, treated as zero.
<to be read again>
                   e
l.3 plus e
          xpenses
```

If not, then check the syntax of the \vskip command (page 284) and you will understand that TEX is still evaluating \vskip's parameters. The word 'plus' is a legal pa-

rameter that should be followed by a dimension. So, what should you do? Put a `\relax` command right after `\vskip 3mm` (see page 280).

Often problems occur when TEX expects a new paragraph (implicitly by an empty line or explicitly by a `\par` command). Or TEX finds a new paragraph at a point where this is not allowed. So be very careful where you put empty lines.

Chapter 5

Support for TEX users

Sooner or later you will find that you are trying to do something that just won't work, and you can't find out why.

Whatever you do, don't despair. There are several ways of getting support even though (or because?) there is no single person, institute or company that you can complain to. In fact, you will find that there are thousands of people all over the world who are quite willing to help you for free. Just like someone else helped them when they had problems they couldn't solve. And maybe one day you in turn will be in a position to help...

In this chapter we will describe various ways of getting support. Except for printed books, the support channels mentioned here are either for free or at a non-profit basis. If you prefer to engage a commercial company you also have a several options. However, these are not listed here. Most of them are present on the World Wide Web, and `http://www.tug.org/consultants.html` is a good starting point to find consultants and commercial TEX vendors.

5.1 Support media

There are various ways of getting support. At least one of them you are already using: reading books. In the next section we will give pointers to other books and articles that you may find useful if you want to learn more about TEX and friends.

The rest of this chapter is about support via the Internet. We assume that you are familiar with the major Internet applications e-mail (electronic mail), World Wide Web (usually called WWW, or simply 'the Web'), FTP (file transfer protocol) and Usenet (also known as 'News'). If you are not, then we suggest you read a book about it. The book shops are full of them. Or ask a friend to explain. Soon Internet skills will be considered as common as being able to make a telephone call.

5.2 Books and courses

On the 4allTEX CDROM you will find lots of additional information such as FAQ's (Frequently Asked Questions), courses on TEX/LATEX in several languages and documentation on TEX related software.

There are several ways to become proficient in TEX or LATEX. Some of the user groups (see next section) offer courses in TEX or LATEX at levels ranging from absolute beginners to advanced macro writers, or METAFONT users, or METAPOST users.

Many books have been written on the subject, and there are some introductory texts available. M. Doob's *A Gentle Introduction to TEX, a Manual for Self-study* is a fine article to start with (cdrom). It is even available in several languages. See appendix D for a complete list of available courses and introductions to TEX and friends on the CDROM.

If you plan to use LATEX you may want to read *The Not So Short Introduction to LATEX 2ε* by T. Oetiker, H. Partl, I. Hyna and E. Schlegl (cdrom). That introduction is based on J. Knappen, H. Partl, I. Hynax and E. Schlegl's *LATEX 2ε Kurzbeschreibung* in German. Translations into Finnish, Spanish, Russian, French, Hungarian and Mongolian are also available (cdrom).

But after you have mastered the essentials of LATEX you will soon feel the need for more documentation. This is provided by the L. Lamport's *LATEX, A Document Preparation System*. The most complete book on LATEX is *The LATEX Companion* by M. Goossens, F. Mittelbach and A. Samarin. We will mention three other recommendable books on LATEX: H. Kopka and P. Daly: *A Guide to LATEX*, H. Kopka's *LATEX: Erweiterungsmöglichkeiten* in German, and R. Seroul and S. Levy's *A Beginner's Book of TEX*. CONTEXT users should read H. Hagen's *ConTEXt manual* (cdrom).

Advanced TEX users will often write their own macros and need much more insight. Their reference manual is *The TEXbook* by D. Knuth, the author of TEX. This is the most comprehensive work on TEX, but some consider it not very easy to read and understand for beginners. Another commendable book on TEX is V. Eijkhout's *TEX by Topic*, which is not for the novice, but for users with basic understanding of TEX, who want to explore the full potential of TEX.

METAFONT users should get a copy of D. Knuth's *The* METAFONT*book*. METAPOST users should read J. Hobby's *The MetaPost System* and his *A User's Manual for MetaPost* (cdrom), the first in English as well as in Polish.

The bibliography near the end of this book provides lots of references to books about TEX, METAFONT, METAPOST and many other TEX related subjects.

5.3 TEX user groups around the world

All over the world TEX users have formed networks of *user groups* on an informal basis. These user groups consist of enthusiastic TEX users who share their problems and solutions with anyone who wants to join the TEX community. They usually communicate through e-mail and often produce printed journals with contributions from members.

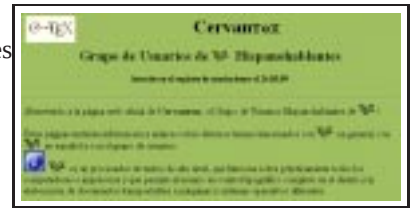
The journals and discussions on e-mail are essential for users who want to be informed about the latest developments. E-mail is important for getting information on, e.g. what is the newest version of program *x*, where do I find a macro for problem *y*, how do I install program *z*.

Below we have listed all TEX user groups currently known to us.

AsTEX (French-speaking)

Michel Lavaud, President
Association pour la diffusion de logiciels scientifiques lies à TEX
Association AsTEX
BP 6532
45066 Orleans cedex 2
France
tel: 33 2 38 64 09 94
e-mail: `astex-admin@univ-orleans.fr`
discussion list: `astex@univ-orleans.fr`
WWW page: `http://www.univ-orleans.fr/EXT/ASTEX/astex/doc/en/web/html/astex000.htm`

CervanTEX (Spanish-speaking)



Grupo de Usuarios de TEX Hispanohablantes
Mailing list: `spanish-tex@eunet.es`
(send subscription requests to this list).
José Ra Portillo Fernández
Departamento de Matemática Aplicada I
Escuela Técnica Superior de Arquitecturs
Avenida de la Reina Mercedes, 2
E-41012 Sevilla
Spain
email: `josera@gordo.us.es`
WWW page:
`http://gordo.us.es/Actividades/CervanTeX/CervanTeX.html`

CSTUG (Czech and Slovak Republics)

Petr Sojka, President
Československé sdružení uživatelu TEXu
CsTUG, c/o FI MU
Botanická 68a
CZ-602 00 Brno
Czech Republic
fax: +420-5-755896
e-mail: `cstug@cstug.cz`
WWW page: `http://www.cstug.cz/`
journal: *Zpravodaj CSTUG*

CyrTUG (Russia)

Eugenii V. Pankratiev, President
Associaciia Pol'zovateleĭ Kirillicheskogo TEX'a
Mir Publishers
2, Pervyĭ Rizhskiĭ Pereulok
Moscow 129820
Russia
tel: +7 95 286 0622, 286-1777
fax: +7 95 288 9522
e-mail: cyrtug@cemi.rssi.ru
WWW page: http://www.cemi.rssi.ru/cyrtug/

Dante e.V. (German-speaking)

Thomas Koch, President
Deutschsprachige Anwendervereinigung TEX e.V.
Postfach 101840
D-69008 Heidelberg
Germany
tel: +49 6221 29766
fax: +49 6221 167906
e-mail: dante@dante.de
WWW page: http://www.dante.de/
journal: *Die TEXnische Komödie*

DK-TUG (Danish)

Thomas Widman, President
DK-TUG
Department of Mathematical Sciences
University of Århus
Ny Munkegade Building 530
DK-8000 Århus
Denmark
email: dk-tug-bestyrelse@sunsite.auc.dk
WWW page: http://sunsite.auc.dk/dk-tug/

Estonian User Group

Enn Saar, Tartu
Astrophysical Observatory, Toravere
EE 2444 Estonia
e-mail: saar@aai.ee

Greek TEX Friends Group (Greek speaking)
Apostolos Syropoulos, President
366, 28th October Str.
GR-671 00 Xanthi
Greece
tel: +30 541 28704
e-mail: apostolo@platon.ee.duth.gr
WWW page: http://obelix.ee.duth.gr/eft/

Grupo de Utilizadores de TEX (Portuguese)
No formal user group yet.
For information, contact
Pedro Quaresma de Almeida
Email: pedro@mat.uc.pt
WWW page:
http://www.mat.uc.pt/~pedro/ntcientificos/TeXportugues.html

GUST (Poland)
Tomasz Plata Przechlewski, President
Polska Grupa Użytkowników Systemu TEX
Instytut Matematyki Uniwersytetu
Gdańskiego
ul. Wita Stwosza 57
80 - 952 Gdańsk
Poland
e-mail: ekotp@univ.gda.pl
WWW page: http://www.gust.org.pl/
journal: *GUST Bulletin*

GUTenberg (French-speaking)
Michel Goossens, President
Groupe francophone des Utilisateurs
de TEX
Association GUTenberg
BP 10
F-93220 Gagny principal
France
tel: +33 1 44 32 37 96
fax: +33 1 44 32 20 80
e-mail: gut@irisa.fr
WWW page: http://www.ens.fr/gut/
journal: *Les Cahiers GUTenberg*

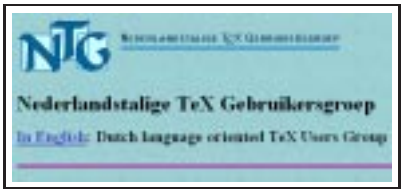
HunTEX (Hungarian TEX Users Group)
Gyöngyi Bujdosó
HunTEX
4010 Debrecen, Pf. 12
Hungary
email: huntex@math.klte.hu
WWW page: http://neumann.math.klte.hu/~huntex/

ITALIC (Irish)
No formal user group yet.
Public mailing list: ITALIC-L@irlearn.ucd.ie
(send subscription requests to listserv@irlearn.ucd.ie).
Peter Flynn
Computer Centre
University College Cork
Ireland
e-mail: pflynn@www.ucc.ie

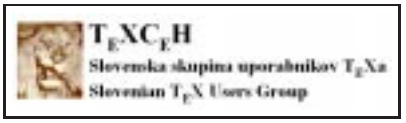
Lietovos TEX'o Vartotojø Grupë (Lithuanian TEX Users Group)
Vytas Statulevičius, Chair
Akademijos 4
LT-2600 Vilnius
Lithuania
tel: +370 2 359 609
fax: +370 2 359 804
e-mail: vytass@ktl.mii.lt
WWW page: http://www.vtex.lt/tex/

Nordic TEX Users Group (Scandinavian countries)
Dag Langmyhr, Chair
Nordic TEX Users Group
Department of Informatics
PO Box 1080 Blindern
University of Oslo
N-0316 Oslo
Norway
tel: +47 22 85 24 50
fax: +47 22 85 24 01
e-mail: dag@ifi.uio.no
WWW page: http://www.ifi.uio.no/~dag/ntug/

NTG (Dutch-speaking)
Erik Frambach, Chair
Nederlandstalige TEX Gebruikersgroep
Postbus 394
NL-1740 AJ Schagen
The Netherlands
e-mail: ntg@ntg.nl
WWW page: http://www.ntg.nl/
journal: *MAPS*



TEXCeH (Slovenian TEX User Group)
Vladimir Batagelj
Jadranska 19
SI-61111 Ljubljana
Slovenia
e-mail: Tex.Ceh@fmf.uni-lj.si
WWW page: http://vlado.mat.uni-lj.si/texceh/texceh.htm



Tirant lo TEX (Catalan TEX Users Group)
Gabriel Valiente Feruglio
Technical University of Catalonia
Department of Software
Mòdul C6, Campus Nord
Jordi Girona Salgado, 1-3
E-08034 Barcelona, Catalonia
Spain
mailing list: catala-tex@aligna.cesca.es (send subscription requests to listserv@cesca.es)
e-mail: valiente@lsi.upc.es
WWW page: http://www-lsi.upc.es/~valiente/tug-catalan.html

TUG (International user group)
Mimi Jett, President
TEX Users Group
1466 NW Front Avenue
Suite 3141
Portland, OR 97209
USA
tel: +1 503 223 9994
fax: +1 503 223 3960
e-mail: office@tug.org
WWW page: http://www.tug.org/
journal: *TUGboat*

TUGIndia (Indian)
K.S.S. Nambooripad, Chairman
Indian TEX Users Group
Kripa, TC 24/548, Sastha Gardens
Thycaud, Trivandrum 695014
India
tel: +91 471 324341
fax: +91 471 333186
email: tugindia@mailexcite.com

TUG-Philippines (Philippines TEX Users Group)
Felix P. Muga II, President
Mathematics Department
Ateneo de Manila University
Loyola Heights
Quezon City
Philippines
tel: (63-2) 426 6001 local 2515
fax: (63-2) 426 6008
email: fpmuga@admu.edu.ph, fpmuga@philonline.com

UK TUG (United Kingdom)
Philip Taylor, Chairman
UK TEX Users' Group
For information:
Peter Abbott
1 Eymore Close
Selly Oak
Birmingham B29 4LB
England
e-mail: uktug-enquiries@tex.ac.uk
WWW page: http://uk.tug.org/
journal: *Baskerville*

5.4 Mailing lists

Many of the user groups listed above communicate through e-mail, by means of *mailing lists*. A mailing list is an electronic mail system that anyone can subscribe to. All mail that is posted to the list is automatically distributed to all subscribers. These lists are often used to ask for help and to discuss problems, new developments, etc.

Table 5.1 lists some active mailing lists. The common language used in discussions on these mailing lists is English, unless stated otherwise.

Table 5.1: Mailing lists

List name	Listserver address
tex-nl	listserv@nic.surfnet.nl (Dutch)
tex-euro	listserv@urz.uni-heidelberg.de (European)
tex-d-l	listserv@vm.gmd.de (German)
4tex	4tex@nic.surfnet.nl (on 4TEX)
tex-k	tex-k-request@tug.org (on Web2c)
fptex	fptex-request@tug.org
gut	listserv@ens.fr (French)
ellhnika	listserv@urz.uni-heidelberg.de (Greek)
gust-l	listserv@vm.cc.uni-torun.pl (Polish)
spanish-tex	listserv@eunet.es (Spanish)
italic-l	listserv@irlearn.ucd.ie (Irish)
yunus	listserv@bilkent.edu.tr (Turkish)
ling-tex	ling-tex-request@ifi.uio.no (on TEX and linguistics)
typo-l	listserv@listserv.hea.ie (on typography)
latex-l	listserv@urz.uni-heidelberg.de (on LATEX3)
tetex	majordomo@informatik.uni-hannover.de (on teTEX)
metafont	listserv@ens.fr (on METAFONT)
ntg-context	majordomo@ntg.nl (on CONTEXT)
ntg-ppchtex	majordomo@ntg.nl (on the chemistry package)
ntg-tools	majordomo@ntg.nl (on tools around TEX)
ntg-toekomst	majordomo@ntg.nl (on TEX's future)
pdftex	majordomo@tug.org (on PDFTEX)

You can subscribe to a mailing list by sending a message to the list server, e.g. `listserv@nic.surfnet.nl`. The body of the message should contain just one line:

```
subscribe tex-nl Foo Bar
```

where 'Foo Bar' is your real name. Note that you should *not* send your request for subscription to, e.g. `tex-nl@nic.surfnet.nl`, because this is the mailing list itself. In that case your request would be distributed to all subscribers, which will not be appreciated.

In case the list server's email address is 'majordomo' you should *not* supply your first and last name. Just '`subscribe somelist`' will work.

Once your request has been accepted you will receive an introductory greeting mail, explaining how to use the list.

5.5 'Usenet' or 'News'

Another forum for discussions, exchanging opinions and helping each other is 'News' or 'Usenet'.

The following newsgroups are dedicated to TEX and friends:

`comp.text.tex` Discussions about TEX, LATEX, TEX implementations and macros

`de.comp.tex` Similar discussions, in German

`fr.comp.text.tex` In French

5.6 4TEX Support

If you have trouble installing 4TEX or need more information you can send e-mail to

```
4TeX-support@ntg.nl
```

However, don't expect an answer within the hour. We will try to help you as soon and as best as we can, but 4TEX is an 'after-hours' project.

4TEX users can join the 4TEX mailing list. On this list, users can pose/answer questions regarding 4TEX. New or desired developments and features are also announced and discussed on this list.

Subscribing to this list is very easy. Send the following message to `listserv@nic.surfnet.nl`:

```
subscribe 4tex Foo Bar
```

where 'Foo Bar' is your real name. At the moment, about 280 people from 36 countries have joined the list.

Updates and bug fixes will be available through 'anonymous FTP' from the server `ftp://4tex.ntg.nl/`.

5.7 File servers

To aid the archiving and retrieval of TEX-related files, a TUG working group developed the Comprehensive TEX Archive Network (CTAN). Each CTAN site has identical material, and maintains authoritative versions of its material. These collections are extensive; in particular, many things described in this book is archived at the CTAN, even if not explicitly stated. The participating hosts in the Comprehensive TEX Archive Network are:

`ftp.dante.de` (Germany)
– anonymous FTP, directory `/tex-archive (/pub/tex /pub/archive)`

– e-mail via ftpmail@dante.de
– World Wide Web access on http://www.dante.de/
– Administrator: ftpmaint@dante.de

ftp.tex.ac.uk (England)
– anonymous FTP, directory /tex-archive (/pub/tex /pub/archive)
– NFS mountable from nfs.tex.ac.uk:/public/ctan/tex-archive
– World Wide Web access on http://www.tex.ac.uk/tex-archive
– Administrator: ctan-uk@tex.ac.uk

ctan.tug.org (Massachusetts, USA)
– anonymous FTP, directory /tex-archive (/pub/archive)
– World Wide Web access on http://ctan.tug.org/ctan
– Administrator: ftpmaint@tug.org

In order to reduce network load, it is recommended that you use the CTAN host which is located in the closest network proximity to your site.

A current list of CTAN hosts and mirrors can be found by executing the command

▷ finger ctan@ctan.tug.org

The information is stored in the file CTAN.sites in the primary archive directory at the participating hosts. Known mirrors of the CTAN reside on:

FTP host address (country)	Directory
ctan.unsw.edu.au (NSW, Australia)	/tex-archive
mirror.aarnet.edu.au (QLD, Australia)	/pub/tex-archive
ftp.univie.ac.at (Austria)	/packages/tex
gd.tuwien.ac.at (Austria)	/publishing/tex/CTAN
ftp.belnet.be (Belgium)	/packages/TeX
ctan.math.mun.ca (Newfoundland, Canada)	/tex-archive
scratchy.emate.ucr.ac.cr (Costa Rica)	/pub/ctan
ftp.cstug.cz (The Czech Republic)	/pub/tex/CTAN
ftp.net.uni-c.dk (Denmark)	/mirror/ftp.tex.ac.uk/tex-archive
sunsite.auc.dk (Denmark)	/pub/tex/ctan
ftp.gwdg.de (Germany)	/pub/dante
ftp.informatik.uni-hamburg.de (Germany)	/tex-archive
ftp.mpi-sb.mpg.de (Germany)	/pub/tex/mirror/ftp.dante.de
ftp.tu-chemnitz.de (Germany)	/pub/tex
ftp.uni-augsburg.de (Germany)	/pub/tex/ctan
ftp.uni-bielefeld.de (Germany)	/pub/tex
ftp.uni-stuttgart.de (Germany)	/tex-archive(/pub/tex)
ftp.ut.ee (Estonia)	/tex-archive
ftp.funet.fi (Finland)	/pub/TeX/CTAN

... continued on next page

`ftp.jussieu.fr` (France)	`/pub4/TeX/CTAN`
`ftp.loria.fr` (France)	`/pub/ctan`
`ftp.oleane.net` (France)	`/pub/mirrors/CTAN/`
`ftp.uvsq.fr` (France)	`/pub/TeX/CTAN`
`ftp.ntua.gr` (Greece)	`/mirror/ctan`
`ftp.comp.hkbu.edu.hk` (Hong Kong)	`/pub/TeX/CTAN`
`ftp.sztaki.hu` (Hungary)	`/pub/tex`
`ftp.heanet.ie` (Ireland)	`/pub/ctan/tex`
`cis.uniRoma2.it` (Italy)	`/TeX`
`ftp.unina.it` (Italy)	`/pub/TeX`
`ftp.lab.kdd.co.jp` (Japan)	`/CTAN`
`ftp.meisei-u.ac.jp` (Japan)	`/pub/CTAN`
`ftp.riken.go.jp` (Japan)	`/pub/tex-archive`
`ftp.u-aizu.ac.jp` (Japan)	`/pub/tex/CTAN`
`sunsite.sut.ac.jp` (Japan)	`/pub/archives/ctan`
`ftp.kreonet.re.kr` (Korea)	`/pub/CTAN`
`ftp.ntg.nl` (The Netherlands)	`/pub/tex-archive`
`sunsite.icm.edu.pl` (Poland)	`/pub/CTAN`
`ftp.radio-msu.net` (Russia *only*)	`/tex-archive`
`tex.ihep.su` (Russia)	`/pub/TeX/CTAN`
`ftpserver.nus.sg` (Singapore)	`/pub/zi/TeX`
`ftp.rediris.es` (Spain)	`/mirror/tex-archive`
`ftp.nada.kth.se` (Sweden)	`/pub/tex/ctan-mirror`
`sunsite.cnlab-switch.ch` (Switzerland)	`/mirror/tex`
`ftp.ccu.edu.tw` (Taiwan)	`/pub/tex`
`dongpo.math.ncu.edu.tw` (Taiwan)	`/tex-archive`
`sunsite.bilkent.edu.tr` (Turkey)	`/pub/tex/ctan`
`sunsite.doc.ic.ac.uk` (UK)	`/packages/tex/uk-tex`
`ftp.cdrom.com` (West coast, USA)	`/pub/tex/ctan`
`ftp.cise.ufl.edu` (Florida, USA)	`/tex-archive`
`ftp.duke.edu` (North Carolina, USA)	`/tex-archive`
`ftp.rge.com` (New York, USA)	`/pub/tex`
`joshua.smcvt.edu` (Vermont, USA)	`/pub/tex`
`sunsite.unc.edu` (North Carolina, USA)	`/pub/packages/TeX`
`wuarchive.wustl.edu` (Missouri, USA)	`/packages/TeX`

To find software at a CTAN site, use anonymous FTP to the host, and then enter the command

▷ `quote site index` *`search-term`*

where *search-term* is the thing you are trying to find. If you want to find, e.g., a file called `book.cls` you would enter:

```
▷  quote site index book.cls
```

and the CTAN server would reply:

```
200-index book.cls
200-NOTE. This index shows at most 20 lines. for a full list of files,
200-retrieve /tex-archive/FILES.byname
200-1997/02/13 | 9956 | macros/latex/contrib/other/misc/abstbook.cls
200-1997/03/26 | 44789 | macros/latex/required/amslatex/inputs/amsbook.cls
200-1999/03/23 | 22715 | macros/latex/unpacked/book.cls
200-1998/07/20 | 1014 | nonfree/language/arabtex/texinput/arabbook.cls
200 (end of 'index book.cls')
```

As you can see now, you should navigate to the directory `macros/latex/unpacked` to retrieve the file `book.cls`.

Note that not all CTAN mirrors provide a 'quote site' service. If not, you could transfer the file `FILES.byname` to your own system. This is a plain ASCII file that contains an index of everything available on CTAN.

If you don't have a (fast) Internet connection, you could use CDROMs instead. The complete CTAN content is published regularly on CDROM by the German TEX user group DANTE.

PART II

Using 4TEX

Chapter 6

The main menu

Figure 6.1 shows the main menu. This is the screen that opens up when you start 4TEX. It contains the most basic functions that we think you need all the time.

6.1 Choosing a Main file and a Current file

The 'Main file' is the document that you want TEX to process. In case you are just starting with TEX, or if you are just writing a letter or another small document, you will only use a 'Main file'.

However, if you are working on a larger or more complex document, you may want to split the work into several files. In that case the 'Main file' could be a tiny file that only contains the set-up of the document and some TEX commands to include other TEX files. In a LATEX document (see chapter 17) your 'Main file' could use `\include`

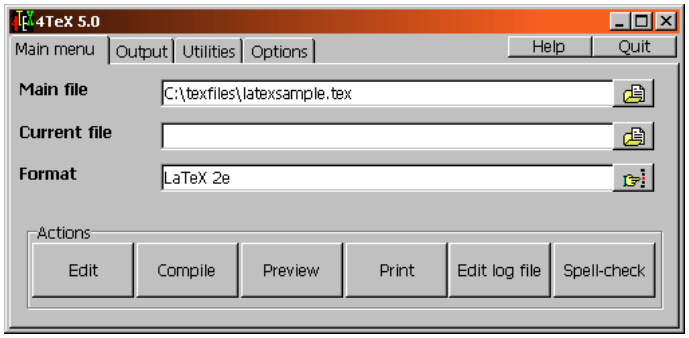


Figure 6.1: The 4TEX main menu

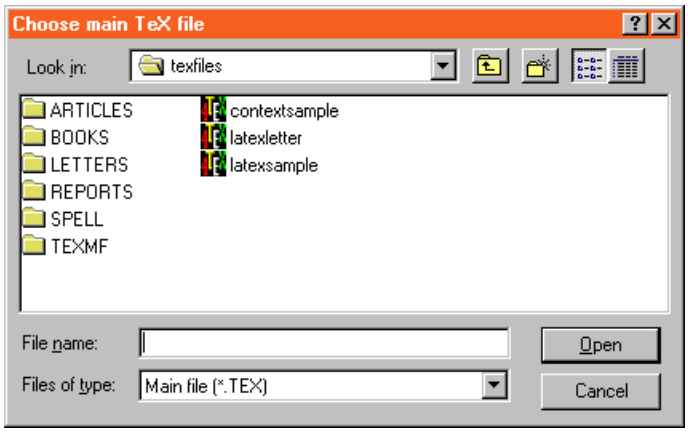


Figure 6.2: Choosing a Main file or Current file

commands to insert chapters of a book, for instance, that you put in separate files. In a Plain TEX (see chapter 16) or CONTEXT document (see chapter 18) you could use `\input` commands to accomplish the same. The 'Current file' could be such a file that is included in the main TEX file.

4TEX makes *no* assumptions whatsoever on the contents of the 'Main file' or 'Current file'. It does *not* generate any include statement for you. 4TEX regards the 'Main file' and the 'Current file' as two separate files. It is not aware of their relation, if any.

The 'Current file' typically contains part of the text that is to be typeset by TEX, without the outer framework which is defined in the 'Main file'. The 'Current file' could also contain nothing but macro definitions and no actual text at all. If you are working on a large document it makes sense to divide the complete text into logical parts like chapters or sections that you put in separate files that you will 'include' in the 'Main file'. If the document at hand is only a small one, perhaps a letter, it makes little sense to use an 'Current file' because in that case it is easier to have the whole document in one file.

When clicking on the 'Main file' field or on the icon to the right, an 'Open file' dialog box will appear from which you can select a file. See figure 6.2 for an example. You can also navigate to the directory you want and type in a new file name to create a new file. The dialog box also allows you to create new directories and delete or rename files.

Note that if you select a 'Main file' for which a parameter file with the extension `.4PAR` is available, this parameter file will be used to restore the settings associated with the 'Main file' (see section 12.9 on `.4PAR` files).

By *right*-clicking on the 'Main file' field or on the icon to the right, a list of the 10 most recently used files will be presented to choose from. This makes it easy to retrieve previous work. Selecting a 'Current file' works similarly.

6.2 Editing your documents

By clicking on Edit the editor will be started. Both the 'Main file' and the 'Current file' (if any) will be loaded. Depending on the configuration of 4TEX, the program LATEX Mac may be started simultaneously. This program provides menus with (LA)TEX symbols, commands, etc. Simply by clicking on a button the required TEX code will be pasted into you document at the current cursor position. See section 8.13 for details on LATEX Mac.

There is no need to quit the editor or LATEX Mac every time you want to compile a document or use any other 4TEX function. You can leave these windows open because you will probably use the editor quite frequently in the Edit–Compile–Preview cycle.

If you use PFE, MED or WINEDT as your editor, the file you are currently editing will be automatically saved before starting the compiler or any other function within 4TEX.

You can use any editing program ('editor' for short) you like to enter your text, as long as the file that you save, and which TEX will read, is in ASCII format. On the CDROM you will find a few editing programs that we think are good choices. They all have somewhat different features, so it is matter of taste and money which one you prefer.

If you are used to 'Notepad', a small editor that comes with Windows, and you want to stick with it that will be just fine, but remember that Notepad can't handle a file bigger than 64 kilobytes. If you try, Windows will ask if you would like to use its more powerful word processor 'WordPad' instead. You could do that, but you should be careful to save the file in 'Text Document' format, not 'Microsoft Word Document' or 'Rich Text Document' format.

We do strongly recommend *against* `edit.com`. Frankly, we don't understand why this embarrassingly poor editing program is still part of Windows. It doesn't even understand long file names.

On the 4allTEX CDROM you can find a few other *freeware* and *shareware* editors that you may want to try.

TSE A 32-bits shareware programmer's editor that runs in 'console mode', written by SemWare Comp.. It is very fast, loaded with of useful features and it supports a powerful macro language.

NoteTab Pro A trial version, written by E. Fookes, combined with a large set of convenient features for TEX users written by I. Podlubny is certainly worth trying.

NoteTab Light A free, but less sophisticated version of NoteTab Pro.

Ultra-Edit A powerful shareware editor, written by I. Mead.

You can also specify your own editor. In section 12.3 on `4TEX.INI` we will explain how to do that.

Any serious editor program can handle files of arbitrary size. Only the operating system sets limits to the maximum size, and the method that a program uses to allocate memory. On Windows this is typically up to twice the amount of fysical memory in your computer, which is probably more than you will ever need. Nevertheless, we recommend that you keep files small. Small meaning no more 100 or 200 kilobytes.[1] It makes your files easier to manage and it can keep errors more local (e.g., a global replace action that can't easily be undone will be less destructive). To TEX it is irrelevant if your text consists of just one large file or a hundred small files.

It is up to you to select the editor program that works best for you. You can select PFE, MED or WINEDT as your editor from the 'Options' menu (see section 9). Below we will describe the major features of the three editor programs we just mentioned.

6.2.1 PFE

A. Phillips's 'Programmer's File Editor' (PFE for short) is a powerful editor that will serve you very well when writing TEX documents. Important features of PFE are:

- No charge: in spite of its professional features this editing program doesn't cost you anything.
- Edit files of any size: only Windows memory sets a limit to the size of files you can edit.
- Multiple windows: you can have several files open at the same time. You can arrange the windows in several ways.
- It is highly configurable: you can set colors, fonts, toolbar, key mappings, menu items and a lot more.
- Macros: you can automate functions in macros.
- Dynamic Data Exchange (DDE) is supported: through DDE 4TEX is able to save the file you are editing before it is starts compiling it. It is so easy to forget this that we think it is better to automate it. See section 12.4 for details on this item.
- It can give context sensitive help information: by right-clicking on a word in your text PFE can search user-installed help files, e.g. the LATEX 2_ε help file.
- Remember position: when PFE reloads a file that you have recently edited using PFE it will automatically put the cursor at the position where you last put it.

1 You will be surprised how much you can write in so little space: the TEX sources of this whole book easily fit on a 1.4 MB diskette. All those pictures occupy much more space, but you can work on both independently.

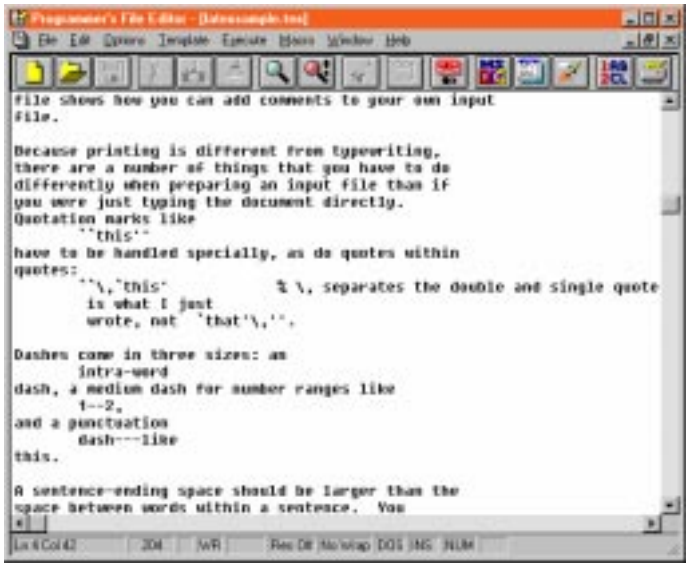


Figure 6.3: The PFE editor

- Jump to a line: when invoking PFE you can specify what line in the file it should jump to. This is very convenient in case TEX detects an error in your text: PFE will take you directly to the line that TEX stumbled on.
- Multiple 'undo': in case you regret a series of modifications you made during the current editing session you can roll them back.
- Unix files: on Unix line endings are differently marked than in MS-DOS or Windows. Unix typically uses only a 'Carriage Return' character while MS-DOS uses a *Carriage Return* character plus a *Line Feed* character. Macintosh files are also different: they typically use only a *Line Feed* character to mark a line ending. PFE can easily handle all three types.

Unfortunately PFE also lacks a few important features:

- There is no 'AutoSave' function. Many text processors save your file every now and then (e.g. every 10 minutes) to avoid loss in case of system failure. With PFE you will have to save your file regularly yourself. However, if you use 4TEX you can enable a special 'AutoSave' function through DDE. See section 12.4 for details on this subject.
- Syntax highlighting is not supported. Many modern programmer's editors understand a little about the syntax of the programming language in which you are writing your text, and they color elements accordingly. This can be very helpful in providing a good overview and avoiding mistakes.

- You can't select *columns* from your text to be deleted or moved elsewhere, you can only select lines (rows). Only a few very good editing programs actually support this function, but it is very convenient.
- There is no sorting function. Sometimes you may wish to sort lines or columns alphabetically. You will have to use another program to do that.
- You can edit multiple files but you can't make changes over multiple files.
- The 'Find' function doesn't support 'regular expressions'. Regular expressions allow you to use 'wildcards' and other special characters in strings to be searched for. For instance, using the expression `a.used\.$` you could find any instance of the word 'amused.' or 'abused.' that appears at the end of a line.

PFE comes with extensive online help information that we think is well worth reading. Below we will list only the most essential keys that you may want to remember when editing a file with PFE:

[←], [→], [↑], [↓], [Page Up], [Page Down], [Home], [End] Move through the text.

[Ctrl] [←], [Ctrl] [→] Move to previous/next word.

[Ctrl] [Home], [Ctrl] [End] Move to top/end of file.

[Ctrl] [Page Up], [Ctrl] [Page Down] Move to top/bottom of screen.

[Ctrl] [S] Save the file you are currently editing. There is no shortcut to 'Save all files': you will have to use PFE's File menu to do that.

[Ctrl] [G] Go to line...

[Ctrl] [Shift] [K] Delete current line.

[F2] Find text...

[F3] Replace text...

[Ctrl] [Z] Undo typing sequence.

[Ctrl] [B] Find matching character. This works with `{}[]()<>`.

If you don't like the current key mappings you can change them to anything else. You could make PFE emulate keystrokes that you are familiar with from other editors. Even an EMACS-like style using [Esc] as a prefix key is possible.

6.2.2 MED

This is a shareware editor written by M. Pfersdorff that is a lot more powerful than PFE. Here are a few of the features that make this a truely professional programmer's editor:

- Column manipulation is supported.

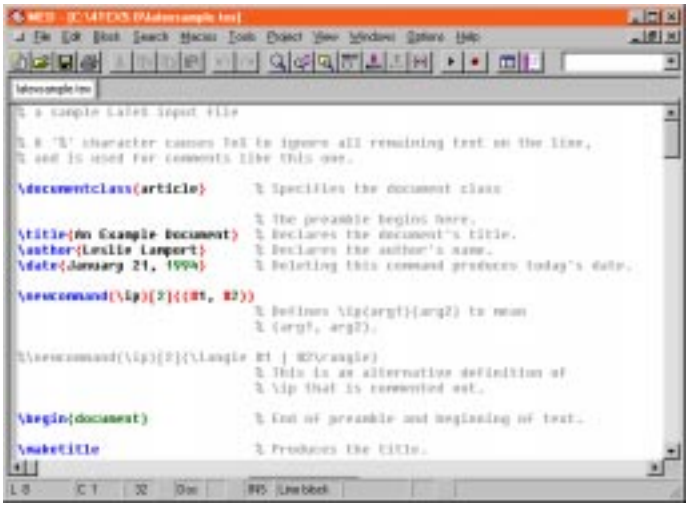


Figure 6.4: The MED editor

- Syntax highlighting rules can be specified for any language.
- There is a (global) 'Autosave' feature.
- DDE is supported for nearly every command (of course 4TEX takes advantage of this feature).
- A 'section indicator and browser' can help you keep track of what section of your document you are currently editing. You can instantly jump to any other section.
- Files are displayed as 'tabbed windows', not overlapping windows. This provides a much better overview while editing multiple files.
- The 'Find/Replace' function can run on multiple files. It supports 'regular expressions'.
- A 'Project manager' provides for easy access to all files related to a project.

MED comes with online help information that we advise you to read. Below we will list only the most essential keys that you may want to remember when editing a file with MED:

←, →, ↑, ↓, Page Up, Page Down, Home, End Move through the text.

Ctrl ←, Ctrl → Move to previous/next word.

Ctrl Home, Ctrl End Move to top/end of file.

Ctrl Page Up, Ctrl Page Down Store current cursor position, move to stored cursor position.

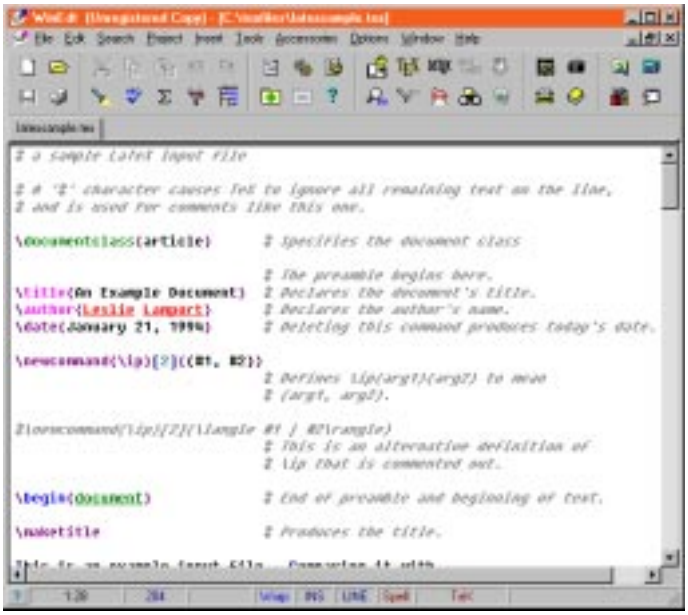


Figure 6.5: The WINEDT editor

Ctrl S, Shift Ctrl S Save the file you are currently editing, save all files.

Ctrl G Go to line...

Ctrl BackSpace Delete current line.

Ctrl F Find text...

Ctrl H Replace text...

F3 Find next.

Ctrl Z, Ctrl Y Undo last action, redo last action.

Ctrl M Find matching character. This works with {} [] () <>.

On the CDROM you will find a *licensed* version of MED. This license allows you to use *this* version of MED for as long as you like. However, you are not entitled to updates or support. If you require any of those, you will have to buy a full license from the author.

6.2.3 WINEDT

This is a very powerful shareware editor written by A. Simonic. Here are a few of the features that make this a very popular editor program:

- Easily insert TEX commands from a menu.
- Syntax highlighting.
- Transparent conversion of TEX equivalents to 8-bit international characters (such as *umlaute* and accented characters) during reading and saving.
- A powerful macro language.
- A built-in spell-checker.
- Completely customizable Menu Bar and Tool bar.
- User-defined pop-up menus for context sensitive response to the right mouse button and/or (double-stroke) shortcut keys.
- A definition of double-stroke (EMACS-like) shortcuts through a flexible pop-up menu mechanism.
- 'Active strings' and command completion functionality.

We advise you to read the online help information that comes with WINEDT. Below we will list only the most essential keys that you may want to remember when editing files with WINEDT:

[←], [→], [↑], [↓], [Page Up], [Page Down], [Home], [End] Move through the text.

[Ctrl][←], [Ctrl][→] Move to previous/next word.

[Ctrl][Home], [Ctrl][End] Move to top/end of file.

[Ctrl][F] Find text...

[Ctrl][R] Replace text...

[F3] Find next.

[Ctrl][G] Go to line...

[Ctrl][S] Save the file you are currently editing.

[Ctrl][F5] Spell-check the current document.

[F12] Find matching character. This works for `{}[]()$` and even the strings `begin` and `end`. It is configurable.

[Ctrl][Z], [Shift][Ctrl][Z] Undo, Redo.

WINEDT will show several icons and options for processing TEX files. However, these are not configured to run the same way as in 4TEX. You will have to configure them yourself.

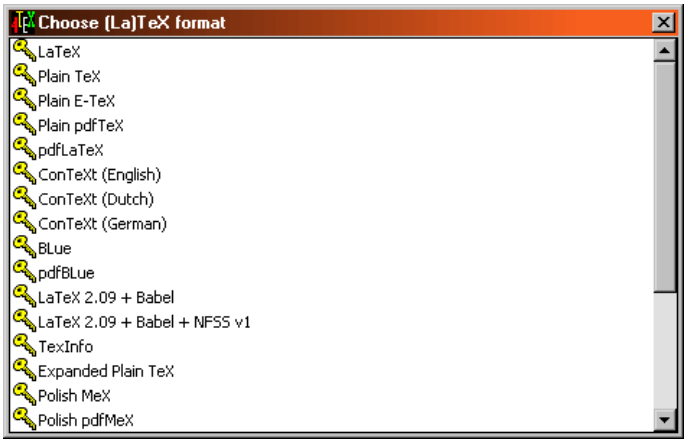


Figure 6.6: Choosing a TEX format

6.3 Choosing a TEX format

The 'Format' field shows the currently selected (LA)TEX format that will be used for compiling the 'Main file'.

After clicking on the 'Format' field or on the icon to the right, a menu of available formats will be presented (see figure 6.6).

The formats all have their own specific features. Currently 4TEX supports the following TEX formats:

Plain TEX D. Knuth's standard macro package.

ε-TEX An extended version of TEX, with all functionality of Plain TEX and more. Do not confuse this format with 'eplain', an extended version of the Plain TEX format. 'ε-TEX' is an extension of the TEX *program* whereas 'eplain' is an extension of the Plain TEX *macro package*. The ε-TEX manual (cdrom) by the NTS-team and P. Breitenlohner provides detailed background information.

LATEX L. Lamport's macro package.

PDFTEX The same as Plain TEX, but this format (or rather, compiler) can produce PDF output rather than DVI output. See Hàn Thế Tahn's *PDFTEX manual* (cdrom) for details.

Expanded Plain TEX A format that extends the Plain TEX format with various commands that are very useful for routine production of articles, books, etc.

PDFLATEX The same as 'LATEX', but now with the ability to produce PDF output instead of DVI output.

CONTEXT H. Hagen's macro package. It comes in three versions: one with an English interface, one with a German interface and one with a Dutch interface. A Czech interface is currently under construction. Note that CONTEXT runs PDFTEX to produce either DVI or PDF.

LATEX 2.09 An older version of LATEX. Officially it is no longer supported but some older documents may not run well on the new LATEX 2_ε. We recommend that you use LATEX 2.09 only in such cases.

BLUe A macro package based on Plain TEX, written by K. van der Laan.

MeX A Polish version of Plain TEX.

PLATEX A version of LATEX that only includes hyphenation patterns for English and Polish, and which does not use Babel.

CSPLAIN A little extension of Plain TEX that uses CS fonts instead of CM fonts. Czech and Slovak hyphenation patterns are included. ISO-8859-2 input encoding is assumed.

CSLATEX An adaptation of standard LATEX for Czech and Slovak. Note that it uses a way of including hyphenation patterns and fonts that is not compatible with 'normal' LATEX.

TEXinfo A format used for generating online help info.

FRLATEX A LATEX format geared for French-speaking TEX users.

Most of these formats are available in one or two variants. A variant may, e.g., be based on PDFTEXor on ε-TEX, or it may use a different input encoding scheme. See sections 13.3.2 and 17.16.2 for details on input encoding.

You can produce your own TEX formats by selecting 'Generate (LA)TEX format' from the 'Utilities menu'. See section 8.5 for a detailed explanation.

6.4 Compiling a document

When you click on Compile, the TEX compiler will be started. The compiler will load the selected format and start processing the 'Main file'. The compiler will write messages about its progress to a separate window, shown in figure 6.7. Messages are written to the log file simultaneously, so you can always review them. The messages window typically closes when the compiler is done. If you want to see the last messages, you can right-click on Edit log file and browse through the messages.

See section 4.1 for tips on how to deal with errors reported by the compiler.

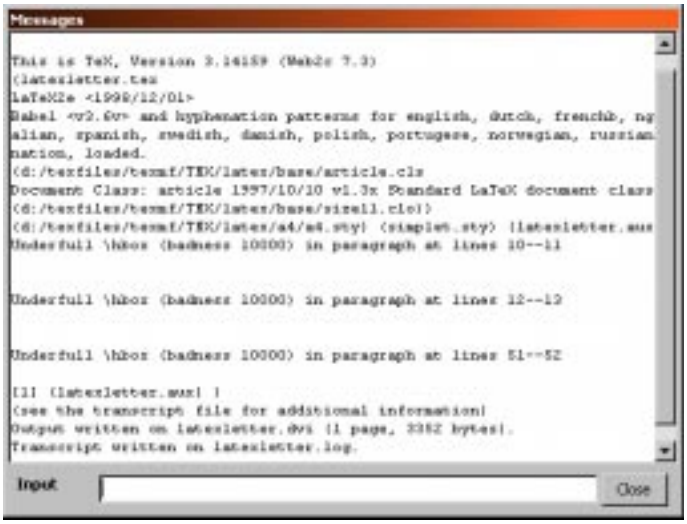


Figure 6.7: Messages from the compiler

6.5 Compiling a selected block

A special feature of 4TEX is its ability to compile an arbitrary small part of a document. The method is quite simple:

1. Copy the lines you want to compile to the Windows clipboard.
2. Right-click on Compile.

In any standard Windows editor you can select a block: click at the start of the block you want to copy. Keep the mouse button depressed and move to the end of the block. Then press Ctrl C or Ctrl Ins (or use the editor's 'Edit' menu) to copy the selected block to the Windows clipboard.

What actually happens when 'block compilation' is started is that 4TEX writes a temporary file by copying all lines from the 'Main file' up until it finds the line specified in the 'Advanced options' (see section 8.11) as `BlockMarker`. If you use LATEX this should be `\begin{document}` so the complete LATEX preamble (which should contain all macro definitions, `\usepackage` commands and other global settings) will be compiled. After that the text from the Windows clipboard is copied. Finally, a line as specified as `EndBlockMarker` is added. This temporary file is then compiled and, if all went well, the results are displayed on screen.

If you use PFE, MED or WINEDT, you only need to *mark* the block. 4TEX will take care of copying the block to the clipboard when you right-click on Compile.

6.6 Previewing the results

Clicking on Preview will display the output on screen, using the selected previewer. You can select a previewer from the Output menu (see chapter 7) or by right-clicking on Preview.

Note that if the compilation of the 'Main file' was not successful nothing may be displayed.

6.7 Printing a document

By clicking on Print, the selected printer program or batch file will be executed in order to print the output.

Right-clicking on Print will pop up the 'Printer options' menu (see chapter 7).

6.8 Viewing the log file

All the messages that the TEX compiler displayed on screen are also written to a log file for later reference. In fact, the log file contains even more information.

By clicking on Edit log file the log file of the last compilation of the 'Main file' will be shown. Right-clicking on Edit log file has a special function: the last lines of the screen output will be shown, typically the final messages of the last TEX compilation.

6.9 Spell-checking

If you click on Spell-check the spell-checker 4Spell, written by W. Dol and E. Frambach (see figure 6.8) will be started. It will spell-check the 'Current file', or the 'Main file' if no 'Current file' is specified.

The language used for spell-checking can be selected by right-clicking on Spell-check or from the Options menu (see chapter 9).

6.10 Online help

In general there are always two ways to invoke the online help system in any 4TEX menu.

- If you press F1 you will get information on the current menu.
- If you click on Help you can start the online help 'from the top'.

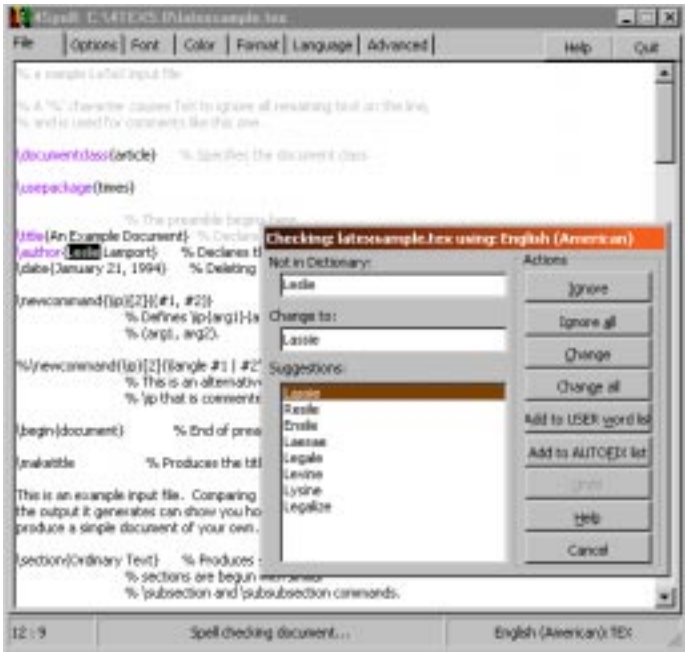


Figure 6.8: Spell-checking with 4Spell

In case your system supports HTML Help, a help window much like figure 6.9 will be displayed. If not, your default World Wide Web browser (e.g. Microsoft Internet Explorer or Netscape Navigator) will be launched. In that case the browser will load and display an HTML file. An active Internet connection is not required.
If you click on Help you can select either 'Help', 'About' or 'Debug'. The 'About' function will be explained in section 6.11.

Another way of getting help is by clicking on the info button that is available in some menus. Clicking this button will start a viewer that will show the 'manual page' of a specific program. The manual page is the documentation that comes with a program; it is not part of the 4TEX online help system.

6.11 About 4TEX

If you click on Help and select 'About' the window shown in figure 6.10 will be presented. This window shows some basic information about 4TEX. By clicking on OK the window will be closed. If you click on Debug the debug window will appear. See section 6.12 for details on this subject.

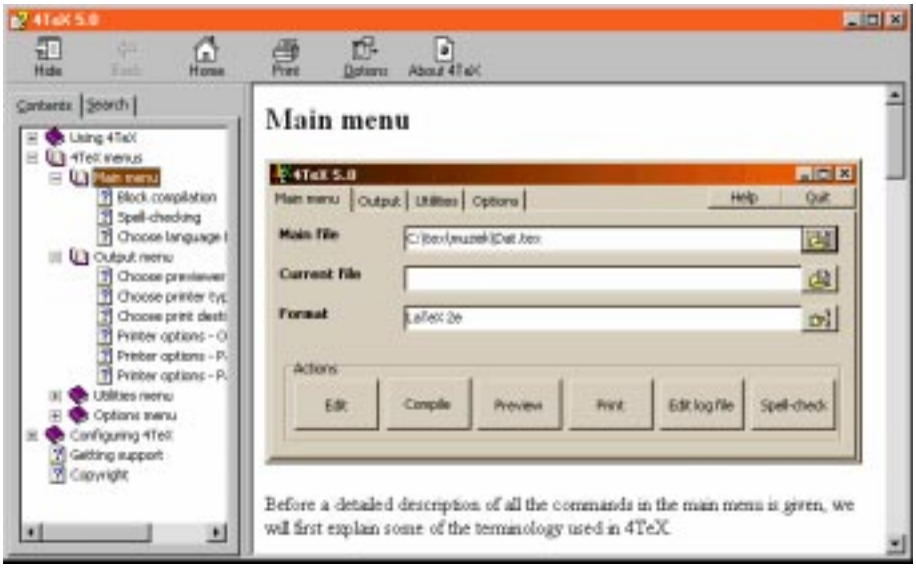


Figure 6.9: Online help

6.12 Debug

4TeX can generate an analysis report of its own current status, which can be helpful in diagnosing problems if 4TeX is not behaving as expected. This report can be accessed by clicking on the 'Help' button which is available in all menus, and subsequently selecting 'Debug'. It will pop up the window shown in figure 6.11.

You can use this information to determine the problem yourself, or you can send it (preferably by email) to the 4TeX support team. See also chapter 5 on getting support.

Whenever you submit questions or bug reports to the 4TeX support team, please include the analysis report generated by the debug feature. It provides essential system information that will greatly improve the changes of locating and solving any problem.

Clicking on Clipboard will copy the text to the Windows clipboard from which you can paste it (by pressing Ctrl V or Ctrl Ins) into any other Windows program, possibly your email program. By clicking on Save you can save the information to a file.

If you (or someone who is assisting you) needs even more information about your system, you could click on SysInfo. The 'System Information' window shows details about all vital elements of your system. In figure 6.12 you can see an example.

You can click on X in the window caption to quit the System Information window. Click on OK to quit the 'Debug' window.

Figure 6.10: The About window

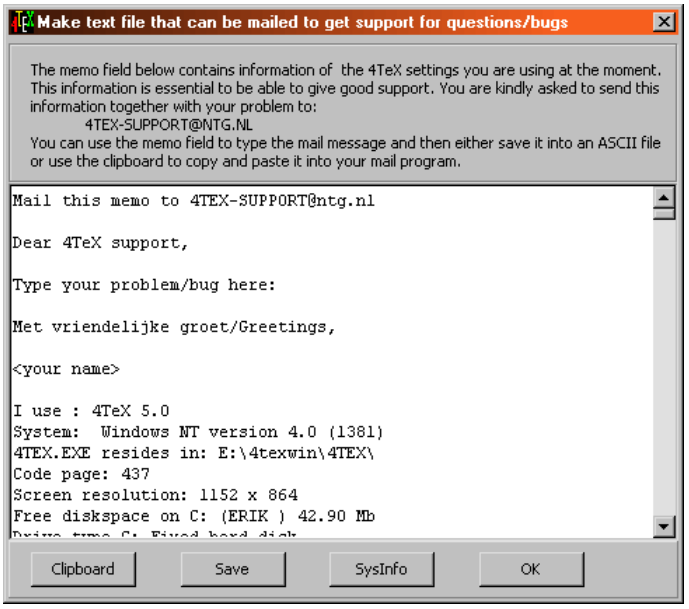


Figure 6.11: The Debug window

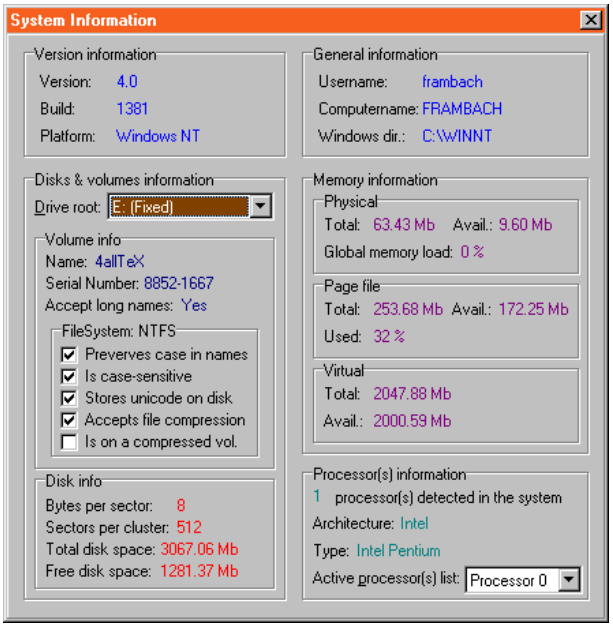


Figure 6.12: The System Information window

6.13 Quitting 4TEX

By clicking on Quit you will quit 4TEX. In the process of quitting 4TEX can do some cleaning up (such as deleting temporary files) and it can 'kill' some other programs if you wish. In section 8.11 we will explain in detail how to specify which files you want 4TEX to delete and which programs to kill.

When quitting 4TEX the file `4TEX.INI` will be rewritten and the current state (screen position, window size, name of 'Main file', language, etc.) will be saved. The next time you start 4TEX it will restore the last saved state.

Chapter 7

The output menu

From the output menu (figure 7.1) you can select your preferred screen previewer and printer program. Many options are available such as selecting paper size, selecting pages to be printed and output resolution.

When clicking on the ‘Printer type’ field or the icon to the right of it a printer type menu will be shown. Figure 7.2 shows an example.

If you click on the ‘Print to’ field or the icon to the right of it, a print destination menu will be shown. Figure 7.3 shows an example. If you select ‘Print to File’ as destination, a dialog window will pop up from which you can choose the file name (and folder) to which the output will be written.

When clicking on the ‘Previewer’ field or the icon to the right of it, a previewer menu will be shown. Figure 7.4 shows an example. See section 13.6.11 for a detailed discussion and comparison of previewers and print programs.

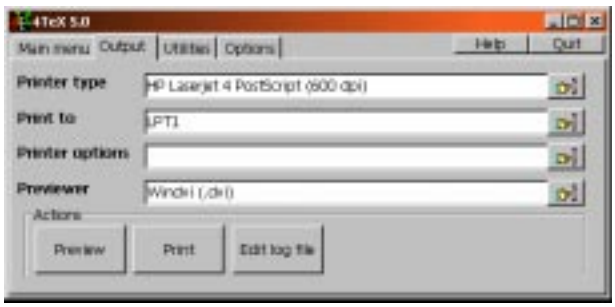


Figure 7.1: The 4TEX output menu

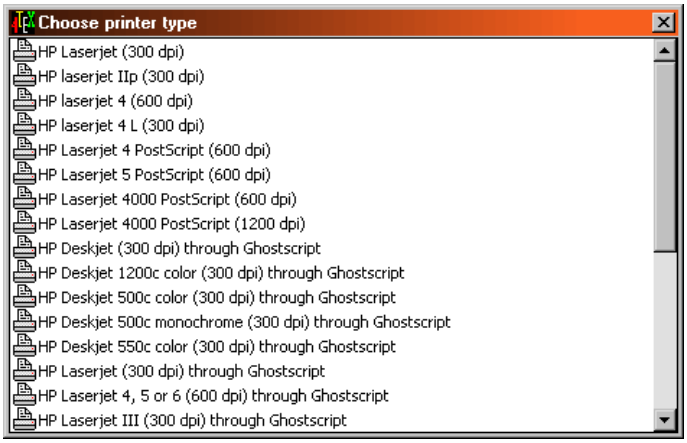


Figure 7.2: Selecting a printer type

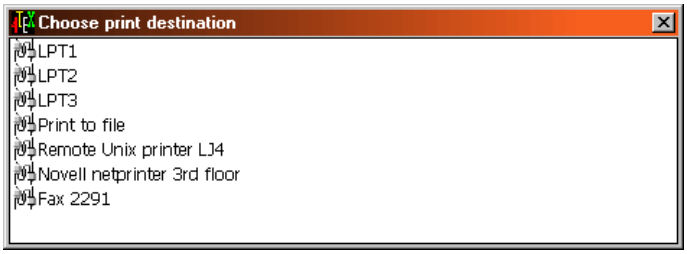


Figure 7.3: Selecting a print destination

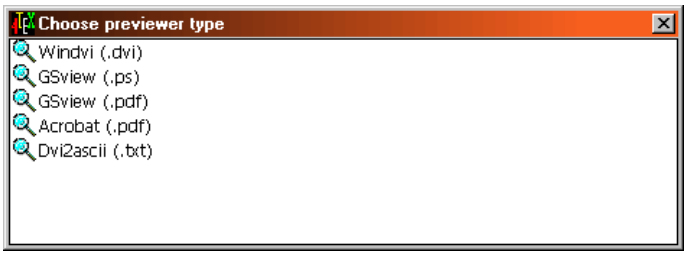


Figure 7.4: Selecting a previewer

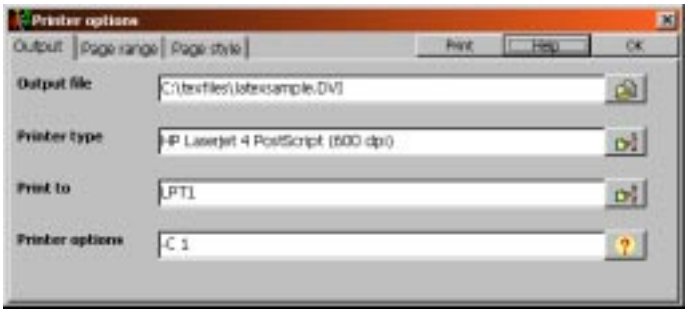


Figure 7.5: Setting printer options

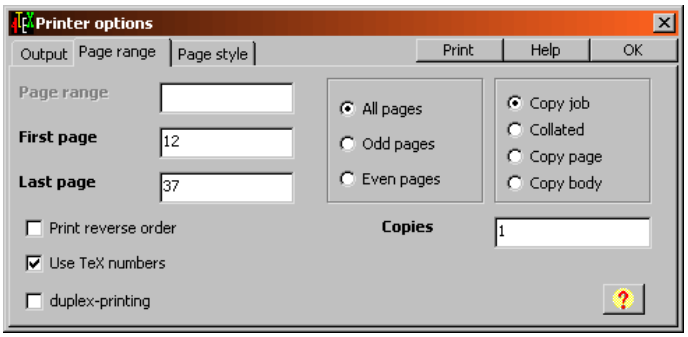


Figure 7.6: The Page range menu

By clicking on Preview you launch the selected previewer which will display the output file corresponding to the 'Main file'.

By clicking on Print you launch the selected printer program which will print the DVI file corresponding to the 'Main file'.

If you click on the 'Printer options' field or the icon on the right of it, another menu will pop up in which you can set printer options. Figure 7.5 shows this menu.

Printer options can be rather complex but we hope this menu will make things a bit easier.

Most of the buttons in the output menu you have already seen. New in this submenu is the option to choose a DVI file. This option can be used to print a DVI file even if the TEX source of it is not available. More important are the 'Page range' and 'Page style' menus.

7.1 The Page range menu

The 'Page range' menu (figure 7.6) allows you to specify the pages you want to print.

If an option is not supported by a specific printer program that option will be shown in gray, and you will not be able to enter a value.

In the ‘Page range’ field you can specify a page range. For the printer program for PostScript devices (DVIPS, see section 13.6.6) the range can be specified e.g. like this:

```
3:10,21,73:92
```

This means: print pages 3 to 10, page 21, and pages 73 to 92.

As an alternative, in the ‘First page’ field you can specify the first page that should be printed. In the ‘Last page’ field you specify the last page to be printed. Since filling out the first page and/or a last page is the equivalent to filling out the ‘Page range’ field, 4TEX will not allow you to fill out both the ‘Page range’ field *and* the ‘First page’ or ‘Last page’ field.

By checking the ‘Print reverse order’ checkbox the last page will be printed first and the first page is printed last. This can be handy if your printer puts printed pages on top of each other face up.

Checking the ‘Use TEX numbers’ checkbox indicates that all the numbers specified in the page range fields represent page numbers that TEX wrote in the DVI file, not physical page numbers. The first page in a DVI file always has physical page number one, but in your TEX document it may represent page 112 or whatever.

If the selected printer type supports ‘duplexing’ (printing on both sides of a sheet), you can switch this feature on or of using the ‘Duplex printing’ checkbox.

The ‘All pages’, ‘Odd pages’ and ‘Even pages’ group allows you to select only even or only odd (physical) pages to be printed. This options can be handy if you want to print a document double sided, but your printer can’t do that in one run: first print all odd pages, then print all even pages on the back of the prints you just made.

The ‘Copies’ field allows you to print multiple copies of the same DVI file. Depending on the printer type you selected you may have up to four methods for printing multiple copies. If you selected a PostScript printers you can choose from all four methods:

Copy job Copies are generard simply by calling the printer driver multiple times. This is equivalent to pressing [Print] multiple times.

Collated Copies are generated by replicating the data in the PostScript output. This method makes the output file grow and it is relatively slow. Each page is printed a number of times, then the next page, and so on. Therefore this option should not be used on printers that print double sided.

Copy body Copies are generated of each page by duplicating the page body rather than regenerating the full page. This option can be useful in conjunction with a PostScript header file setting `\bop-hook` to do color separations or other neat tricks. But this is really for advanced PostScript technicians...

Copy page Copies are generated by means of the PostScript internal copy feature. Obviously this option generated the smallest output file.

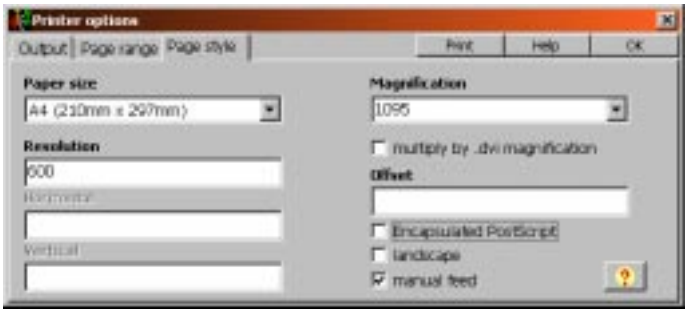


Figure 7.7: The Page style menu

7.2 The Page style menu

The ‘Page style’ menu can be used to specify some details related to the pages you will be printing (figure 7.7).

In the ‘Paper size’ field you can specify the paper type you will be printing on.

In the ‘Resolution’ field you can specify the printer resolution. Alternatively you can specify horizontal and vertical resolution separately in case they are different using the ‘Horizontal’ and ‘Vertical’ field.

The ‘Magnification’ field will set the magnification ratio. If you specify x the magnification will be set to $x/1000$. Note that it will override any magnification specification within the DVI file, unless you check the ‘Multiply DVI magnification’ checkbox. The magnification must be between 10 and 100,000. It is recommended that you use one of standard magnification values (1095, 1200, 1440, 1728, 2074, 2488, 2986, and so on) from the menu, but you *can* enter any number you wish.

In the ‘Offset’ field you can specify how much the page margin/offset should be moved from its origin. The offset is a comma-separated pair of dimensions, such as `.1in,-.3cm`. The origin of the page is shifted from the default position (which is one inch down, one inch to the right from the upper left corner of the paper) by this amount.

In case you selected a PostScript printer program a checkbox will be available to specify that you want to generate ‘Encapsulated PostScript’ instead of ‘standard’ PostScript. Note that the printer program may ignore this option if your DVI file is not suited for generating EPS. The most important restriction for generating EPS is that the DVI file should contain no more than one page.

Checking the ‘Landscape’ checkbox will make the printer generate output in in landscape mode, as opposed to portrait mode which is default.

Checking the ‘manual feed’ checkbox will make the printer wait at the start of each page for a manual feed.

7.3 Types of output devices

When it comes to generating output on paper, TEX is a bit different from most other Windows programs. In general, if you use TEX you don't rely on any Windows printer driver, but you generate printable output by TEX's own printer drivers. The reason for this is that TEX was designed to deliver state-of-the-art output on any computer platform. This output in turn can be fed to a printer, either directly (e.g. to `lpt1`) or through another program that can communicate with Windows (e.g. Ghostscript, WINDVI or PrintFile).

For previewing the results there a specialized previewer programs that are also capable of printing, but this is not always a good idea. We will explain all your options below.

4TEX supports several types of output devices. Output can be displayed on the computer screen, or it can be printed. But unfortunately things are a little more complex than that:

- Output on screen can be produced by WINDVI or GSview.
- Output on an inkjet printer can be produced by DVIHP or Ghostscript.
- Output on a matrix printer can be produced by Ghostscript.
- Output on a laser printer can be produced by DVILJ*xx* or Ghostscript.
- Output on a PostScript printer can be produced by DVIPS.
- WINDVI and GSview are also capable of printing to a Windows printer.

As you can see, in several cases you have more than one option, so you have to choose. You may or may not have noticed that in all cases it is a safe bet to adopt PostScript as your output format, even if you don't have a PostScript printer. TEX and PostScript go together extremely well. T. Rokicki's DVIPS is probably the most powerful DVI driver around, and Ghostscript (or GSview as its front-end) can print PostScript output on almost any output device.

7.4 The WINDVI previewer

You can use WINDVI to preview a DVI file. Most features of this program can be activated by clicking on buttons located at the top of the windows. These buttons allow you to:

- Select a file.
- Print the current file.
- Zoom in.
- Zoom out.
- Move backward 10 pages.

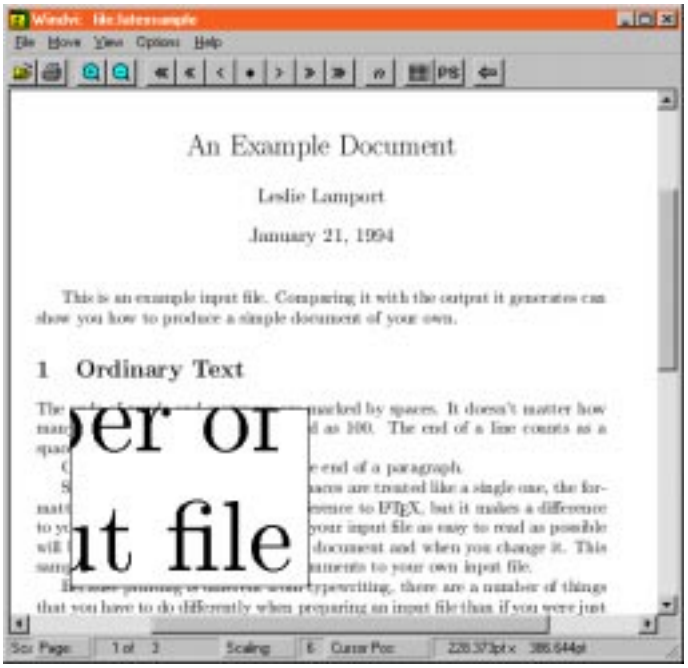


Figure 7.8: WINDVI previewer

- Move backward 5 pages.
- Move backward 1 page.
- Redraw current page.
- Move forward 1 page.
- Move forward 5 pages.
- Move forward 10 pages.
- Move to page. . . .
- Draw a grid on the page (toggle).
- Use Ghostscript to render EPS graphics (toggle).

When previewing you can use the following shortcut from the keyboard:

[←], [→], [↑], [↓], [L], [R], [U], [D], [Page Up], [Page Down] Move within the current page.

[Home] Go to the upper left corner of the page. If margins are active, use them.

[Backspace] **or** [B] Move to previous page.

[Enter] **or** [N] Move to next page.

Ctrl Home Go to first page.

Ctrl End Go to last page.

K Normally, when WINDVI switches pages it moves to the home position as well. 'k' toggles toggles a 'keep-position' flag which, when set, will keep the same position when moving to another page.

– Zoom out.

+ Zoom in.

T **(in lower case!)** Change units in which the cursor position is displayed. You can cycle through pt, sp, bp, cc, cm, dd, in, mm and pc.

Clicking a mouse button will pop up a small 'magnifying glass'. As soon as you release the mouse button the magnifying glass will disappear.

left mouse button Pop up a small magnifying glass, as long as the button is down. See the rectangle in figure 7.8.

middle mouse button Pop up a medium magnifying glass, as long as the button is down.

right mouse button Pop up a big magnifying glass, as long as the button is down.

Shift **+ left mouse button** Change the arrow cursor for a crossbar cursor and enter 'setting home position' mode. Home position is set when the button is released.

WINDVI is capable of printing to any Windows printer. However, you will have to set up a few parameters if you want this to work properly. Choose the menu item 'Options'. Then select the correct 'MF Mode' and enter the corresponding 'Pixels per inch'. Select the correct 'Paper type' and make sure that 'Make pk' is checked. The other options are not important for printing.

Once you have entered these parameters correctly, you should be able to print, using the WINDVI's printer icon, or the 'Print' item from the 'File' menu. Note that 'Print Setup…' and 'Page Setup…' do not work in the current version of WINDVI.

Using WINDVI for printing is not recommended. It can be very slow, and pictures in EPS or PCX format may not print. Color printing may also work unsatisfactory. In general, specialized printer programs, such as DVIPS, work more reliably and a lot faster. GSview is also a good alternative, though it works a bit slower.

You can quit WINDVI by pressing Alt F4.

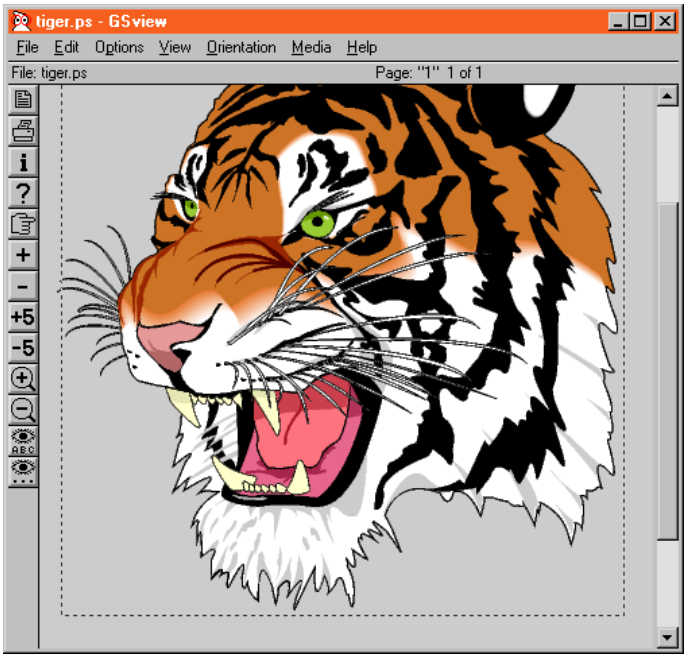


Figure 7.9: GSview previewer

7.5 The GSview previewer

GSview is a program written by R. Lang that can display PostScript or PDF output by using Ghostscript as its rendering engine.

The most important functions can be activated by clicking on an icon on the toolbar at the left side of screen or via the menu at the top. Their functions are (top down):

- Open a file...
- Print...
- Information on the currently displayed file.
- Online help.
- Move to page...
- Move to next page.
- Move to previous page.
- Move forward 5 pages.
- Move backward 5 pages.

- Zoom in.
- Zoom out.
- Find text…
- Find next.

Note that if the file that GSview is currently displaying is changed (e.g., regenerated by a new TEX run), GSview will automatically reload the (new) file when you click anywhere on the displayed image.

Although all basic functions can be operated by clicking on buttons, it may be worth while to remember a few of the keyboard shortcuts listed below:

O Open and display a file.

C Close file.

N **or** + Go to next page.

Space Go to next page and home.

V **or** – Go to previous page.

BackSpace Go to previous page and home.

G Go to page…

I File information.

R Redisplay page.

S Select file: open but don't display.

A Save file as…

P Print all or some pages to a printer.

F Print all or some pages to a file.

E Extract some pages to another file.

M Show Ghostscript messages.

< **or** , Decrease resolution by 1/1.2 (zoom out).

> **or** . Increase resolution by 1.2 (zoom in).

F1 Show online help.

Ctrl C **or** Ctrl Insert Copy displayed bitmap to clipboard.

Ctrl F Find text…

F3 Find next…

↑ Scroll up 16 pixels.

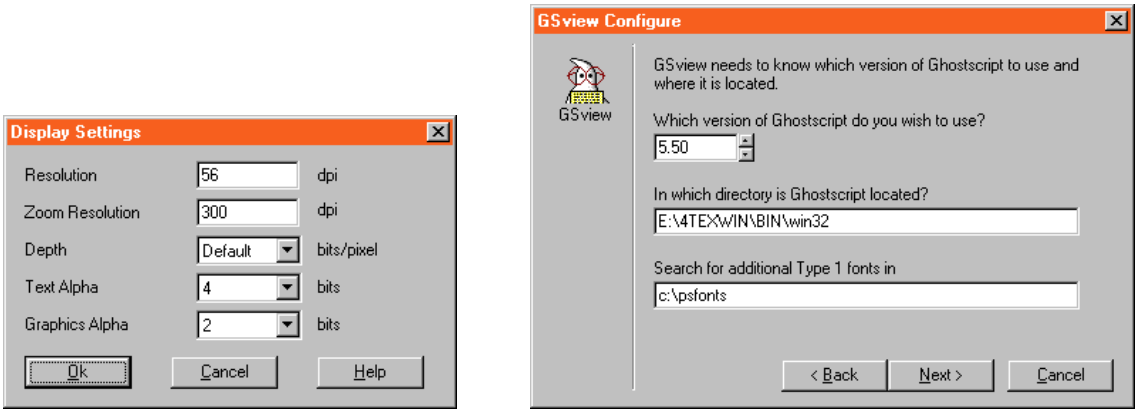


Figure 7.10: GSview previewer: media parameters (left), configuration (right)

↓ Scroll down 16 pixels.

← Scroll left one screen (window width).

→ Scroll right one screen.

Page Up Scroll up one screen (window height).

Page Down Scroll down one screen.

Home Scroll to top of page.

End Scroll to bottom of page.

Using GSview you can print PostScript files to any printing device that Windows supports, which naturally includes PostScript printers. PDF files, however, files can only be printed to non-PostScript devices. For printing PDF files you will have to use Adobe Acrobat viewer (see section 7.6).

When you select the print feature from GSview, the window displayed in figure 7.11 will pop up. Though it may sound illogical, the first thing you must decide is whether you will be printing to a PostScript device or not. If so, you will notice that several options will be grayed out once you check the 'PostScript Printer' option. The grayed out options are not relevant in that case. Now you only need to specify which pages you want to print and to which 'Queue'.

In case you are *not* printing to a PostScript printer, you will have to specify a few more parameters. First you should select a printer 'Device' such as `ljet3` (HP Laserjet III and compatibles), `bjc800` (Canon Bubblejet Color 800 and compatibles), `cdjcolor` (HP Deskjet Color and compatibles). If you are not sure that your printer will work with any of the listed devices, you should select the device `mswinpr2`, which will allow you to print to any printer that you have installed on your Windows system. Probably this is a good choice in any case. If you select a more specific printer device, you will also

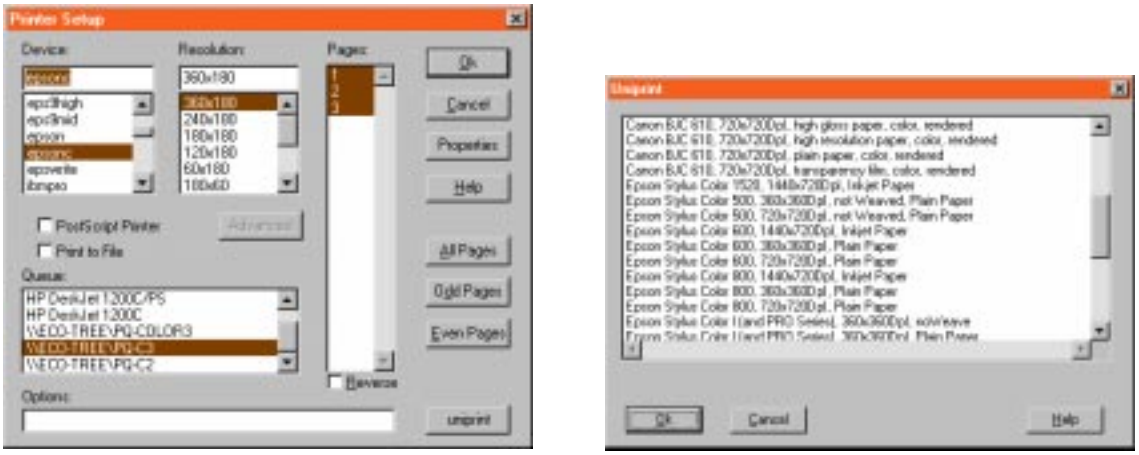


Figure 7.11: Printing with GSview: selecting a print device (left), selecting a 'uniprint' configuration (right)

have to select the required 'Resolution' in dots per inch. Then you can specify which pages you want to print and to which 'Queue'.

GSview comes with a set of predefined parameter files for specific printers. Such a parameter file can be selected if you specify `uniprint` as 'Device'. In that case you can click on uniprint and select a set of parameters. Using this method your prints may look better, especially if they are in color.

Note that the predefined parameter files have the file name extension `.upp`. They can be found in the `\bin\win32` directory. They are ASCII files that you can edit, if you know enough about Ghostscript's parameter syntax.

7.6 The Adobe Acrobat PDF viewer

In case you are producing PDF output, you may want to use Adobe's Acrobat PDF viewer to display the results. From this viewer you can also easily print to any Windows printer you have installed on your system.

All features can be activated by clicking on icons or selecting options from menus, but you can also use key strokes. The most important keys are listed below:

←, →, ↑, ↓ Move to previous page, move to next page.

Page Up, Page Down Move to top or bottom of current page, previous of next page.

Home, End Move to first page, move to last page.

Ctrl – Move to previous position.

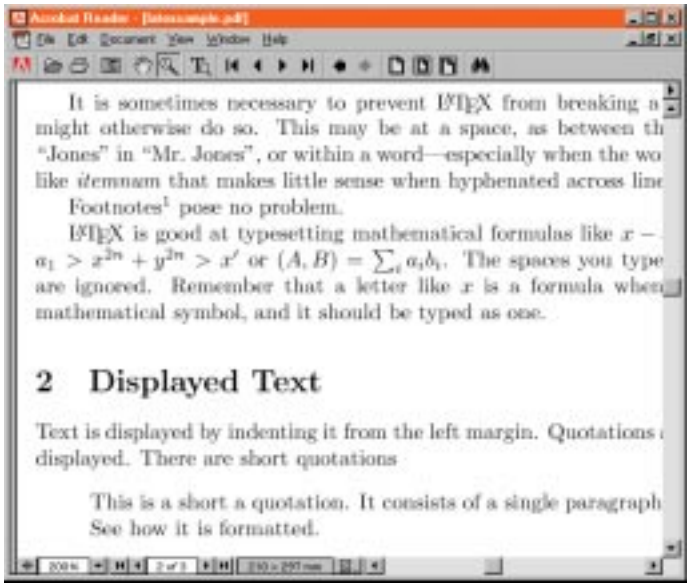


Figure 7.12: Adobe Acrobat PDF viewer

Ctrl = Move to next position.

Ctrl 5 Go to page...

Ctrl F Find text...

Ctrl P Print...

Ctrl Shift L Full screen mode. Use Esc to return to windowed mode.

Note also that you can use the hand shaped cursor to pull the page up or down.

7.7 Printer drivers

TEX comes with its own printer drivers that have no relation at all with the Windows printer drivers. These printer drivers are programs just like any other. They produce output that printers will understand. They can send their output directly to a printer (e.g., to `lpt1`) or to a file.

The Web2c TEX implementation comes with drivers for printers that support the so-called PCL printer language, and for printers that support the PostScript printer language.

Almost all inkjet and laser printers support PCL; some of them support both PCL and PostScript; some only support PostScript or something completely different. Check

the user manual of your printer for PCL and/or PostScript support, so you will know what printer type to select from 4TEX.

If your printer is not on the list that 4TEX presents, or if you don't know what printer language(s) your printer supports, try 'Laserjet IIp' first. If you you think your printer may support PostScript, try 'Laserjet 4 PostScript'.

A very general way of printing from 4TEX that should work on any Windows printer you have installed will be used if you select 'Windows printer through Ghostscript'. In this case the following procedure will be executed:

1. 4TEX will generate PostScript output from the DVI file that TEX produced.
2. 4TEX will send the PostScript output to Ghostscript, a PostScript interpreter.
3. Ghostscript will render the output on a Windows printer that you select.

This procedure will take a little more time than sending PCL or PostScript output directly to a printer. The advantage is that you can use all the advanced features of PostScript and still print on a very cheap printer. Think of it as upgrading your printer to a full-fledged (color) PostScript printer...

7.8 Color support

Using color in TEX is not very difficult, but you must remember that TEX is not a desktop publishing program, so you should not expect to be able to design the most exotic colored pages in a few minutes.

In case you use Plain TEX you should definitely read section 16.12 on color support using `colordvi.tex`. This method is easy and straightforward. LATEX users should read section 17.19, which describes the `color` package. This package is a bit more powerful than `colordvi` and has a more user-friendly interface. CONTEXT users should read section 18.15. CONTEXT is a lot more sophisticated than LATEX with regard to color management. You can quite easily define colors and backgrounds for pieces of text, or for whole pages.

Not all previewer programs and printer drivers support colors. The best results can be obtained by using PostScript or PDF as your output format. From PostScript or PDF you can print to any (color) printer you have installed on your system. We recommend that you use GSview as your previewer. From this program you can also easily print.

Section 7.5 describes the GSview program in detail. See section 13.6.11 for an overview of features of previewers and printer drivers.

7.9 Printing ‘binary’ files

When choosing a print destination in the Output menu, you can select ‘Output to file’. The output file that you generate is usually interpreted by your printer. Often it is completely unreadable to humans. We refer to these files as ‘binary files’ because they cannot (reasonably) be edited by humans.

Now suppose that you generated such a file, or you got one from someone else, from the World Wide Web, through e-mail, or whatever, and you want to send it to a printer.

You would think that using Windows Explorer, you could simply use ‘Send to’ (right-clicking on a file) for that. But Windows doesn’t provide for such a basic function. You can only *associate* files with *programs*, nothing else. Fortunately there are other options.

- You could open the ‘Settings’, ‘Printers’ from the Windows Start menu. Then drag and drop a file on a printer icon. Using the ‘Copy’ option from Windows Explorer and ‘Paste’ from the printer icon or printers window will not work.
- You could use the ‘Run’ command from the Windows Start menu to type in the command

  ▷ `copy /b myfile lpt1`

  Note that the `/b` parameter is important.[1] The Copy command will send the file to whatever `lpt1` means on your system. It could be network printer or a fax. Do *not* try the Xcopy command instead: it will fail hopelessly. Of course you can also issue the Copy command from a Dos-box, and you can use `prn` instead of `lpt1`.
- Using the program PrintFile you get a familiar Windows interface from which you can select the printer that the file will be sent to. You even have several other interesting options. In section 8.17 we will explain them in more detail. On your Windows desktop you could make a shortcut to the PrintFile program, and then drop ‘binary’ files on its icon. If you know Windows well enough, you will be able to add PrintFile to your ‘Send to’ menu, which is even more convenient.
- In case the ‘binary’ file is a PostScript file, you can use GSview (see section 7.5) to preview it and send it (or parts of it) to any Windows printer.

1 If you forget to specify `/b`, there is chance that the Copy command will stop long before the whole file is copied. This is because the Copy program may interpret any ASCII-26 character (Ctrl-Z) in the file as ‘end of file’. `/b` stands for ‘binary’: copy without any interpretation.

Chapter 8

The utilities menu

From this menu you can start a utility program. There are two types of utilities:

1. Utilities and windows defined by 4TEX.
2. Other tools specified as external programs.

Utilities and windows defined by 4TEX are:

- BIBTEX
- MakeIndex
- METAFONT
- METAPOST
- Generate (LA)TEX format
- Clean up files
- Conversion tools
- Graphic conversions
- Select and show manual page
- Select and show Windows help file
- View TEX project
- Run program
- Advanced 4TEX options

Other tools specified as external programs are:

- Regenerate the ls-R file(s)
- Gnuplot (plotting program)

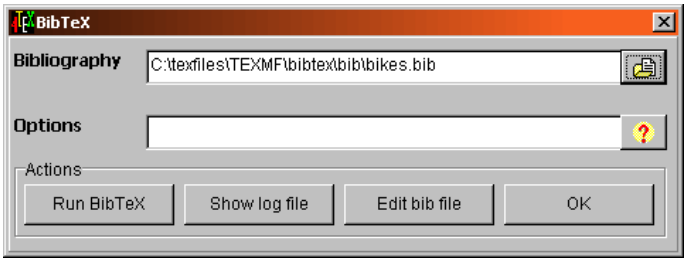


Figure 8.1: The BIBTEX menu

- LATEXcad (drawing program)
- LATEX Mac (generates LATEX code)
- IrfanView (graphics viewer)
- Paint Shop Pro (graphics editor)
- Mayura Draw (drawing program)
- LATEX Wizard (generates LATEX framework)
- Windows Explorer

In the next sections we will explain the utilities in detail.

8.1 BIBTEX menu

This menu presents the options that have to do with maintaining the bibliographic databases in BIBTEX format and running BIBTEX.

When writing an article, book or report you often refer to other literature. At the end of the document you usually include a bibliography. With LATEX and the program BIBTEX this becomes easy.

Using one or more bibliographic databases that contain references to all books, articles, etc., you never have to create a bibliography yourself. In a LATEX document you can simply include a command such as

```
\bibliography{articles,books,reports}
```

This statement tells LATEX to use the databases `articles.bib`, `books.bib`, and `reports.bib` to select the documents you referred to in your document and put them in a bibliography. Note that the databases must have the extension `.bib`.

Referring to other documents is done by one of the following commands: `\cite{keyname}` or `\nocite{keyname}`, where `KEYNAME` is the key or identification name you attached to the reference in the databases. The difference between `\cite` and `\nocite` is that `\cite` produces some output on the spot where the command is issued

(e.g. the number of the document in bibliography). The command \nocite produces no output. It only includes an entry in the bibliography.

```
Knuth~(1986)\nocite{KNUTH}
```

will produce nothing more than Knuth (1986) and the book will appear as, e.g. the 10th entry in the bibliography. If you had used the '\cite' command instead, e.g.:

```
Knuth~\cite{KNUTH}
```

the result would have been Knuth [10] and again an entry in the bibliography.

The layout of the bibliography and the effect of a \cite command is defined by the \bibliographystyle{BIBSTYLE} command where BIBSTYLE is the bibliography style file with the extension .bst. The standard bibliography style files are plain.bst, alpha.bst, abbrev.bst, and unsrt.bst, but many publishers have defined their own bibliography layout and their own .bst'. On the CDROM you will find many .bst files. Most probably one of those will satisfy your requirements.

To generate a bibliography you have to compile your document using the LATEX format, which produces one or more auxiliary files (files with the extension .aux). The auxiliary files contain (amongst other information) everything that BIBTEX needs.

After having compiled your document successfully, you can run BIBTEX. BIBTEX will read the auxiliary files created by LATEX and query the databases for the keynames of the \cite and \nocite commands. The output is written to a file with the extension .bbl. It contains the (LATEX) commands to produce a list of references according to the specified style file.

The second time you run your text through LATEX the \bibliography command reads the file .bbl file and typesets a bibliography. The bibliography is created on the location where the '\bibliography' command is issued. This means that this command should be at the end of the document, although the \bibliography command can be issued anywhere after the \begin{document} command.[1]

A detailed discussion how to use BIBTEX can be found in L. Lamport (1994). We also refer to O. Patashnik's articles (cdrom) for a discussion of how to make your own bibliography style. The following example illustrates how Knuth's TEXbook is entered in the BIBTEX database:

```
@BOOK{KNUTH,
  author = {Knuth, D.E.},
  title = {The {\TeX}book},
  year = 1984,
  publisher = {Addison-Wesley},
```

1 You have to run LATEX at least twice to get a bibliography typeset. This is time consuming but the only way to see which references should be included in the document. Once the .bbl file exists you can edit this file for corrections or add/remove some references. This saves you from editing the databases and running LATEX and BIBTEX again but you should remember that changes in the .bbl file do not change the citations (e.g. the numbering) and will be defeated as soon as you run BIBTEX again.

```
  address = {Reading, Massachusetts}
}
```

There are many options and details to be aware of when building a bibliographic database. Appendix B of L. Lamport's reference manual explains how to do this, but there are easier solutions: BIBDB, the interactive BIBTEX bibliography database manager (see section 8.1.1), or BibEdit (see section 8.1.2).

We will end this section with an example of a document that is set up to produce a bibliography.

```
\documentclass{article}
\begin{document}
As explained by Knuth~(1986)\nocite{KNUTH} in his well-known book

\bibliographystyle{plain}
\bibliography{articles,books,reports}

\end{document}
```

Note that Plain TEX and CONTEXT do not support BIBTEX natively. You will have to use extra macros or modules.

8.1.1 BIBDB

BIBDB is a program written by E. Doron for managing BIBTEX databases.

BIBTEX is a good way of keeping a reference library, but managing it is a pain. Finding entries using criteria such as year or keywords, and so forth, is cumbersome, and entering data is fraught with errors.

With BIBDB you can browse through the database, extract selected entries, add and edit entries in a convenient manner. The user-friendly interface shields you from many technical details that would otherwise often cause errors.

For a detailed discussion BIBDB works we refer to E. Doron (1997) (cdrom). Although BibDB is very user-friendly, you need to learn how BIBTEX works before you can work with BIBDB properly.

8.1.2 BibEdit

J. Björnerstedt's BibEdit is, as its name suggests, a program specifically designed for editing bibliography files. It may be less sophisticated than BIBDB, but its user interface is so easy that it is hard to get lost.

By clicking on the 'Bibliography' field or on the icon to the right of it you can select a BIBTEX bibliography file.

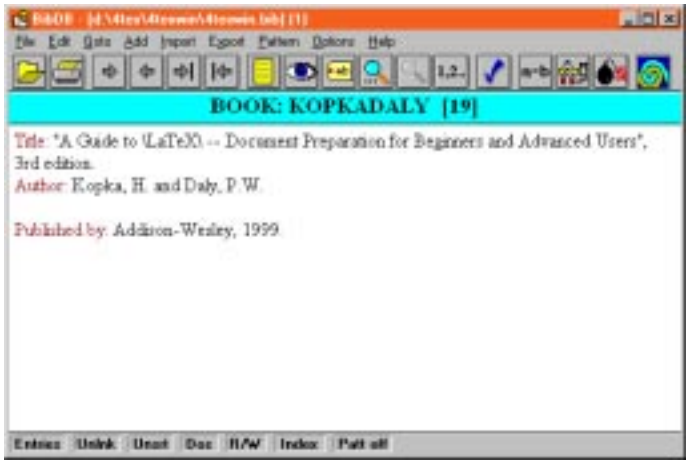


Figure 8.2: BIBDB

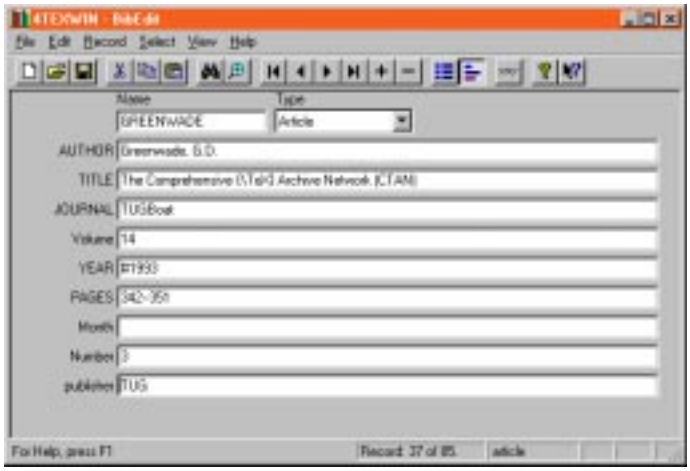


Figure 8.3: BibEdit

8.1.3 How to use the BIBTEX menu

In the ‘Options field’ you can enter options for the BIBTEX program. For an overview of available options click on : the manual page of BIBTEX.EXE will be displayed.

By clicking on Run BIBTEX you can start the BIBTEX program which will generate a list of references based on citations in the ‘Main file’ (and any files that the main file included).

Clicking on Edit log file will display the log file of the last BibTeX run. This file contains all remarks, warnings and error messages produced during the last BibTeX run.

After clicking on Edit bib file you can start editing the selected bibliography. You can select a BibTeX editing program from the options menu (see chapter 9).

The OK button can be used to quit the BibTeX menu.

8.2 The MakeIndex menu

The finishing touch of your book or report may be an index. Creating a detailed index is very time consuming for the author but essential for the reader. With MakeIndex creating an index in LaTeX documents is relatively easy, technically speaking.

Creating an index with LaTeX is much like creating a bibliography. First you add the package `makeidx`, like this:

```
\documentclass{book}
\usepackage{makeidx}
```

Then you put a `\makeindex` command in the preamble (between the `\documentclass` and the `\begin{document}` command). At the location where you want your index to appear you give the command `\printindex` (usually right before the `\end{document}` command). When this is done you have to specify the entries you want in the index and MakeIndex will find the correct page numbers and produces an index table. If you want, say, the word '4TeX' to be included in the index you put the command `\index{4\TeX}` right after the word `4\TeX` in your document, like this: `4\TeX\index{4\TeX}`. There are several other options to include words in an index. For a detailed discussion about all possibilities we refer to L. Lamport's *MakeIndex: An Index Processor for LaTeX* (cdrom) and P. Chen and M. Harrison's. *Index Preparation and Processing* (cdrom).

Suppose that the document you are writing is called `sample.tex`. When you run LaTeX for the first time, the style option `makeidx` together with the command `\makeindex` will produce a file called `sample.idx`. In this file you can find all the index entries. Then you run MakeIndex. The result is a file called `sample.ind` that contains the index entries with the corresponding page numbers. The second time you run LaTeX the `\printindex` command will include the file `sample.ind` so your index gets typeset.

The 'Index file' field shows the name of the index file that will be generated when running MakeIndex on the current 'Main file'. By default the name will be file name of the main file with the file extension `.idx`.

In the 'Options' field you can enter options for the MakeIndex program. For an overview of available options click on  and the manual page of `MAKEINDEX.EXE` will be displayed.

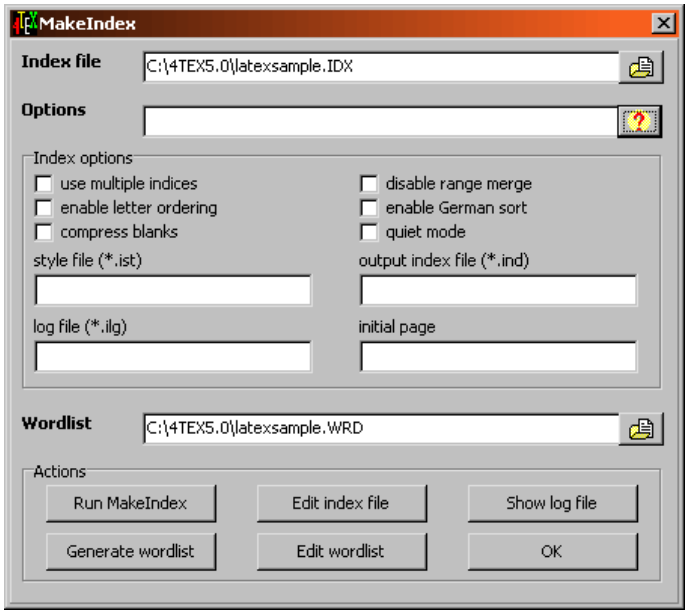


Figure 8.4: The MakeIndex menu

In the ‘Index options’ panel you can choose some frequently used MakeIndex options. By checking an option or typing options the ‘Options’ field will automatically be updated.

The ‘Wordlist’ field will show the name of the file that will contain an alphabetically ordered list of all words that are used in the ‘Main file’ and any files included by the main file. By default the name will be file name of the main file with the file extension `.wrd`. This file is useful to decide which words should be included in the index of your document. This could be the first step in creating a useful index.

By clicking on Run MakeIndex the MakeIndex program will read the indexing commands from the `.ind` file and generate output in the file that is specified in the ‘Index file’ field.

By clicking on Generate wordlist a list of all words in the ‘Main file’ will be generated. The file name of the wordlist is specified in the ‘Wordlist’ field.

By clicking on Edit index file you can start editing the index file created by MakeIndex. This file contains all code needed for (LA)TEX to make up the index.

By clicking on Edit wordlist you can edit the wordlist file.

Clicking on Edit log file will display the log file of the last MakeIndex run. This file contains all remarks, warnings and error messages produced during compilation.

Click on OK to quit the MakeIndex menu.

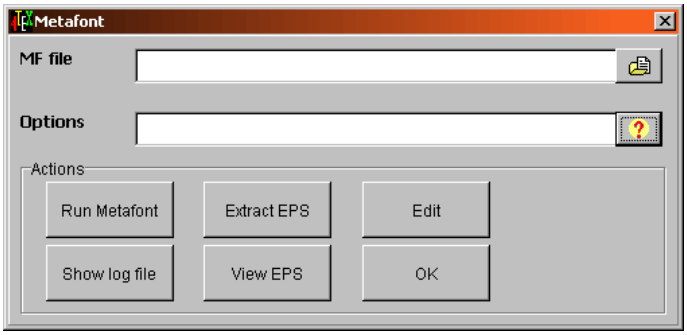


Figure 8.5: The METAFONT menu

8.3 The METAFONT menu

By clicking on the 'MF file' field or the icon to the right of it you can choose a METAFONT file that you want to edit, change, run through METAFONT, etc.

The 'Options' field can be used to specify options for the METAFONT program. For an overview of available options click on : the manual page of `MF.EXE` will be displayed.

Click on Run Metafont to start METAFONT with the options specified and the font specified by the 'MF file' field.

By clicking on Edit log file the log file created by METAFONT will be displayed. This file contains all remarks, warnings and error messages produced during compilation.

In case you use the MFtoEPS package written by B. Jackowski, P. Pianowski and M. Rycko, you may want to extract the Encapsulated PostScript code that was written in the log file. Clicking on Extract EPS button will extract all EPS files from the log file.

After you have extracted the EPS files you may want to view them on screen to check the results. This is done by clicking on View EPS.

Clicking on Edit will start the editor with the METAFONT file specified in the 'MF file' field.

Click on OK to quit the METAFONT menu.

8.4 The METAPOST menu

METAPOST is a program written by J. Hobby for writing graphics in an algorithmic manner. The language is based on METAFONT, but the output of METAPOST is Encap-

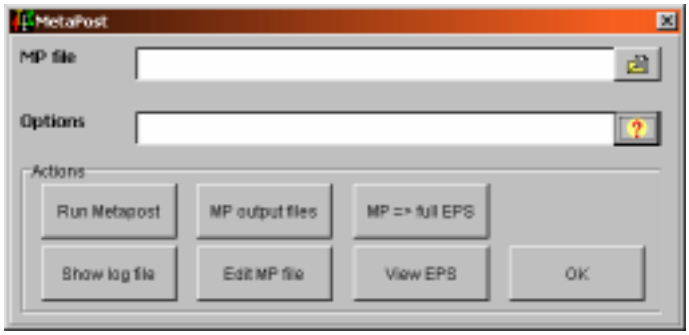
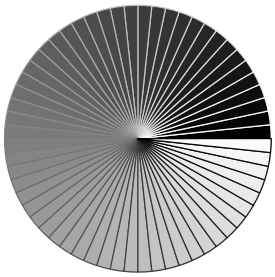


Figure 8.6: The METAPOST menu

sulated PostScript. An introduction to this program and its language is available (cdrom). Here is an example of a METAPOST picture (courtesy H. Hagen).



```
beginfig(1)
path p;
p := (0,0)--subpath (0,8/60)
  of fullcircle scaled 100--cycle;
for i=0 upto 59:
  fill p rotated (i*6)
    withcolor (i/60)*white;
  draw p rotated (i*6)
    withcolor (1-i/60)*white;
endfor;
currentpicture :=
  currentpicture shifted (100,100);
endfig;
```

By clicking on the ‘MP file’ field or the icon to the right of it you can choose a METAPOST file that you want to edit, change, run through METAPOST, etc.

The ‘Options’ field can be used to specify options for the METAPOST program. For an overview of available options click on the help icon: the manual page of `MPOST.EXE` will be displayed.

Click on Run MetaPost to start METAPOST with the options specified and the font specified by the ‘MP file’ field.

By clicking on Edit log file the log file created by METAPOST will be displayed. This file contains all remarks, warnings and error messages produced during compilation.

By clicking on MP output files you can run a TEX job (actually, the TEX file `mproof.tex` is compiled) that includes all METAPOST output files, convert the resulting DVI file to PostScript, and display it on screen.

After clicking on Edit MP file you can edit the MP file specified in the ‘MP file’ field.

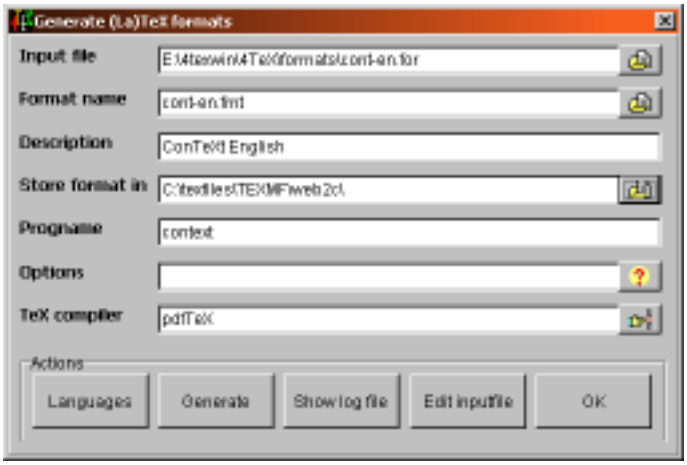


Figure 8.7: The Generate (LA)TEX formats menu

Click on MP => full EPS to convert the METAPOST output to ready-made EPS files. This extra step is necessary if the METAPOST input file called for TEX to typeset pieces of text.

METAPOST produces PostScript output files. You can view these on screen by clicking on View EPS. You can select the output file from a menu.

Choosing a non-PostScript file may generate strange errors. Any PostScript file usually starts with the string `%!PS`.

Use OK to quit the METAPOST menu.

8.5 The Generate (LA)TEX formats menu

To generate TEX formats 4TEX uses files with the extension `.FOR`. These files can be found in the `FORMATS` directory, which is a subdirectory of the directory in which all 4TEX programs reside.

In the ‘Input file’ file you should specify such a file. You can click on that field or on the icon to the right of it to choose a file.

The ‘Format name’ field specifies the name of the newly generated format

The ‘Description’ field describes the format. After successfully generating a format, this line will be appended to the list of available formats, so you can easily select the new TEX format from the main menu.

The ‘Store format in’ field specifies the directory where the newly generated format will be stored.

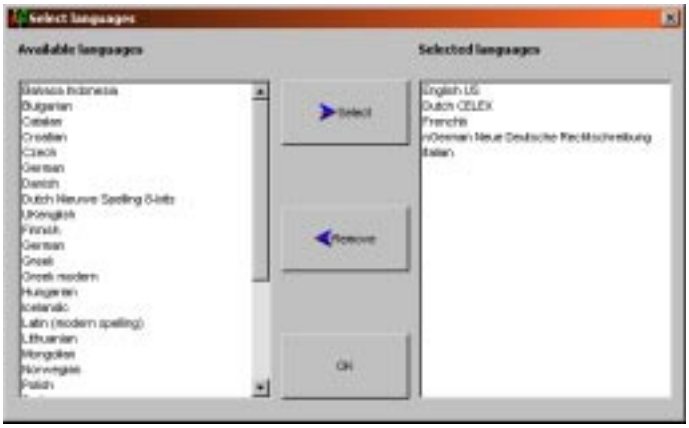


Figure 8.8: Selecting languages for format generation

The ‘Options’ field specifies any options that will be used to generate the format. Click on ? to see the manual page of the TEX compiler.

By clicking on Languages a language selection menu (figure 8.8) will be presented.

You can select one or more languages from this list using Select. If you have selected a language that you didn’t want you can remove it from the ‘Selected languages’ box by using Remove. You can also use the standard Windows ‘drag and drop’ method to move a language from one list to the other.

The hyphenation patterns you select will be included in the format in the order that you specified. If you use LATEX with the Babel package, or if you use CONTEXT you don’t need to worry about that (but do read the documentation on language support!). In Plain TEX it is important to know that `\language=0` selects the first set of hyphenation patterns, `\language=1` selects the second, etc. See section 12.12 if you want to know what happens behind the scenes when generating format files.

By clicking on OK you confirm your selection and you will be returned to the ‘Generate (LA)TEX format’ menu. But first 4TEX will ask you if the index file(s) `ls-R` should be regenerated. If you have generated a new format you should answer ‘Yes’. Otherwise the TEX compiler may not be able to find the new format. If you didn’t generate a new format, or if you only replaced an existing format, there is no need to regenerate the `ls-R` index(es), so you can answer ‘No’.

You can’t include *all* languages in one format. Due to the internal workings of TEX there are limitations. We recommend that you only include those that you really need. Up to, say, 6 languages there should be no problems.

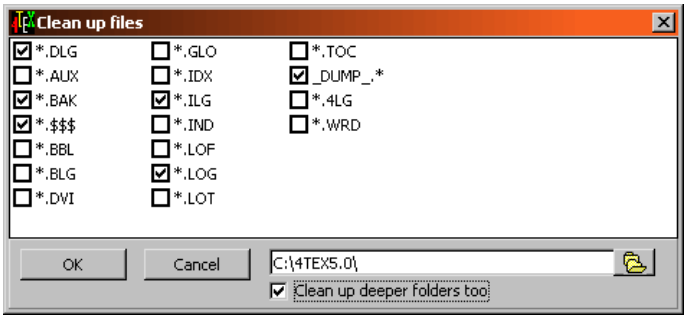


Figure 8.9: The Clean up files menu

By clicking on Edit log file the log file of the TEX format generation (messages, warnings, errors, etc.) will be displayed. By clicking on Edit input file you can start editing the 'Input file'. Click on Quit to quit the format generation menu.

8.6 The Clean up files menu

This menu allows you to select which files you want to delete. Running TEX and friends will generate a lot of 'temporary' files that may be deleted. You can easily change this list of files through the Advanced 4TEX options menu (see section 8.11).

Simply 'check' the files you want to delete. You can use the folder field to select a folder from which the selected files will be deleted. To change the folder click on this field or on the icon to the right of it.

You can use the 'Clean up subfolders' check box to indicate that you want the selected files to be deleted not only from the selected folders but also in all its subfolders.

By clicking on OK all selected files files will be deleted and the 'Clean up files' menu will be closed.

Click on Cancel if you do not want delete any file.

8.7 The conversion tools menu

Sometimes you may want to convert files produced by other word processors to TEX. Or you may want to use the extended ASCII set for accented letters instead of the less readable equivalent TEX commands. This may also be necessary when you want to use the text in other word processors. A more drastic approach is to 'deTEX': strip all TEX commands from a file.

The conversion tools menu (figure 8.10) makes it easy to do many types of conversions:

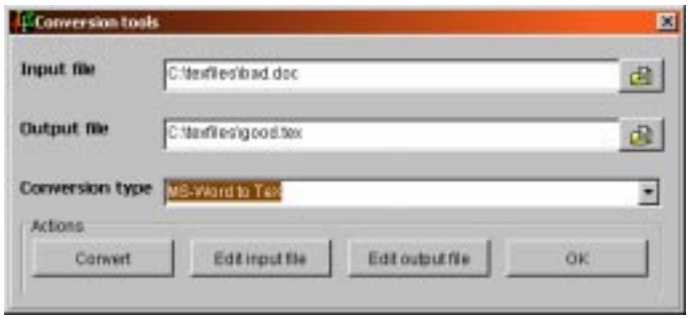


Figure 8.10: The Conversion tools menu

Text or word processor files to (LA)TEX:

- MS-Word to TEX
- WordPerfect 5.1 to LATEX
- TROFF to LATEX
- TROFF to TEX
- ChiWriter 4.x to LATEX
- ChiWriter to TEX
- DisplayWrite to ASCII
- PC-Write to LATEX
- HTML to LATEX
- Program listing to TEX
- RTF (Rich Text Format) to LATEX
- SJIS text to preprocessed (LA)TEX
- BIG-5 text to preprocessed (LA)TEX
- BIG-5 text with CEF macros to preprocessed (LA)TEX
- SJIS text with CEF macros to preprocessed (LA)TEX
- Text with CEF macros to preprocessed (LA)TEX
- DOS text (codepage 437) to TEX (7-bits)

ASCII related conversions:

- MS-Word to ASCII
- TEX 8-bits (ANSI) to TEX ASCII (7-bits)
- TEX ASCII (7-bits) to TEX ANSI (8-bits)
- (LA)TEX to plain ASCII using DeTEX

- (LA)TEX to plain ASCII using unTEX
- DVI file to ASCII
- DisplayWrite to ASCII
- HTML to human readable ANSI text

Files from different operating systems:

- Hard returns (CR) to soft returns (WP input)
- Unix text file (LF) to DOS/Windows text file (CR-LF)
- DOS text (codepage 437) to Windows text (ANSI)
- DOS/Windows text file (CR-LF) to MacIntosh text file (CR)
- DOS/Windows text file (CR-LF) to Unix text file (LF)
- DOS text (codepage 437) to Windows text (ANSI)
- MacIntosh text file (CR) to MS-DOS text file (CR-LF)

(Encapsulated) PostScript tools:

- Plain EPS to EPS with EPSI preview
- Plain EPS to EPS with TIFF4 preview
- EPS with preview to plain EPS (recalculate BB)
- PostScript to canonical EPS (to curves by ps_conv)

Others:

- ASCII screen dump to LATEX picture
- MIDI to MusicTEX
- (LA)TEX to HTML (by TEX4ht)
- (LA)TEXto HTML (by TTH)
- LATEX to RTF
- DVI to devirtualized DVI (by DVIcopy)

The ‘Input file’ field specifies the file that you want to convert. The ‘Output file’ field specifies the file name of the output of the conversion.

You can click on Edit input file to edit the selected input file before conversion if you wish. Likewise you can click on Edit output file to edit the generated output file. By clicking on ▽ at the right of the ‘Conversion type’ field you can select one of the above conversions. Click on Convert to start the conversion. Use OK to quit the ‘Conversion tools’ menu.

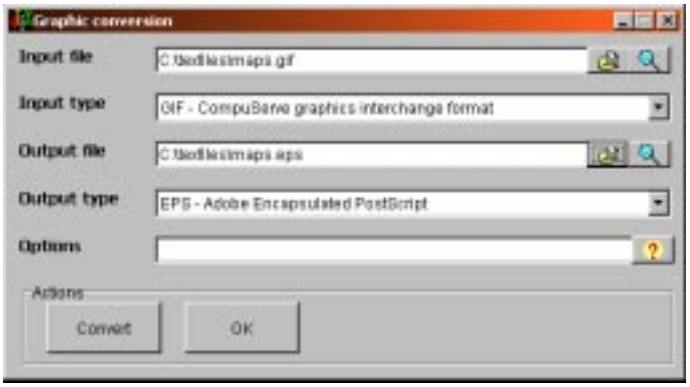


Figure 8.11: The Graphics conversion menu

8.8 The graphics conversion menu

The graphics conversion menu (figure 8.11) makes it easy to convert almost any kind a graphic file to a format that can be used in TEX or any other program you require. The program CONVERT, written by E. du Pont de Nemours, does the actual conversion, in some cases in combination with Ghostscript (see section 13.6.9).

The 'Input file' field specifies the graphic file that you want to convert. The 'Output file' field specifies to the file name of the output of the conversion.

The 'looking glass' button can be used to get a preview on screen of the selected graphics file. The program 'IrfanView', written by I. Skiljan, is used to display bitmap graphics; GSview, written by R. Lang, is used to display PostScript graphics.

Conversion of a large number of graphic formats is supported. By default both input and output formats are determined automatically by the file name *extension* of the files. Table A.4 in appendix A lists exhaustively all supported file name extensions and their meaning. Below is a short list of the most important ones for TEX users:

BMP Windows bitmap image

BMP24 Windows bitmap (24-bit color depth)

EPS Encapsulated PostScript

EPS2 Encapsulated PostScript Level II

GIF87 Graphics interchange format (version 87a)

JPEG/JPG Joint Photographic Experts Group

PCX ZSoft IBM PC Paintbrush

PDF Portable Document Format

PNG Portable Network Graphics

TIFF/TIF Tagged Image File Format

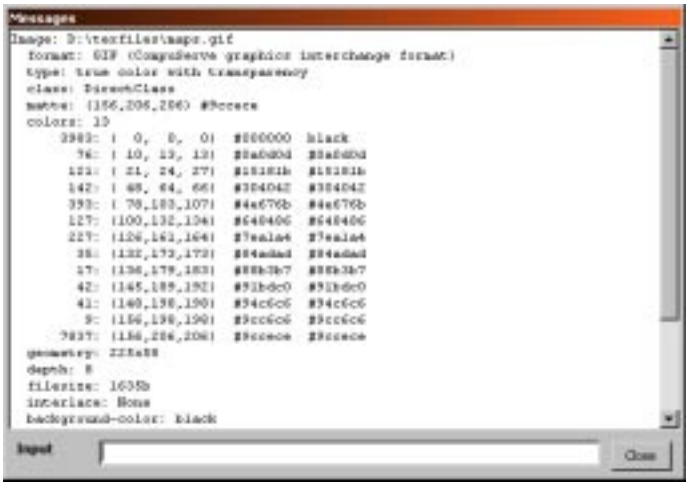


Figure 8.12: Identifying a graphic file

TIFF24 Tagged Image File Format (24-bit color depth)

Instead of selecting just one input file, you can select an arbitrary number of graphics files from one directory. In that case no output file needs to be specified: output files will be written to the same directory as the input files, and their file names will be the same, too, except for their extension.

In case the file you chose is incorrectly identified or named you can manually select the correct type. In case you are uncertain you can right-click on the selected file. The program IDENTIFY (also written by E. du Pont de Nemours) will be started. It will attempt to identify the graphic file for you and show you its findings. See figure 8.12 for an example.

Click on Convert to start the conversion. Click on OK to quit the 'Conversion tools' menu.

8.9 View TEX project

You can get a graphic overview of the current 'TEX project' by selecting the option 'View TEX project' from the utilities menu. This option starts 4Project, a program written by W. Dol and E. Frambach.

The 'Main TEX file' is assumed to be the top level of the project, which includes other files and graphics. As a demonstration an overview of the project of the book you are reading is given in figure 8.13.

The first tab field shows an overview of all files that involved in the current project: nested references to other files are traced, as well as graphics that are included. Files

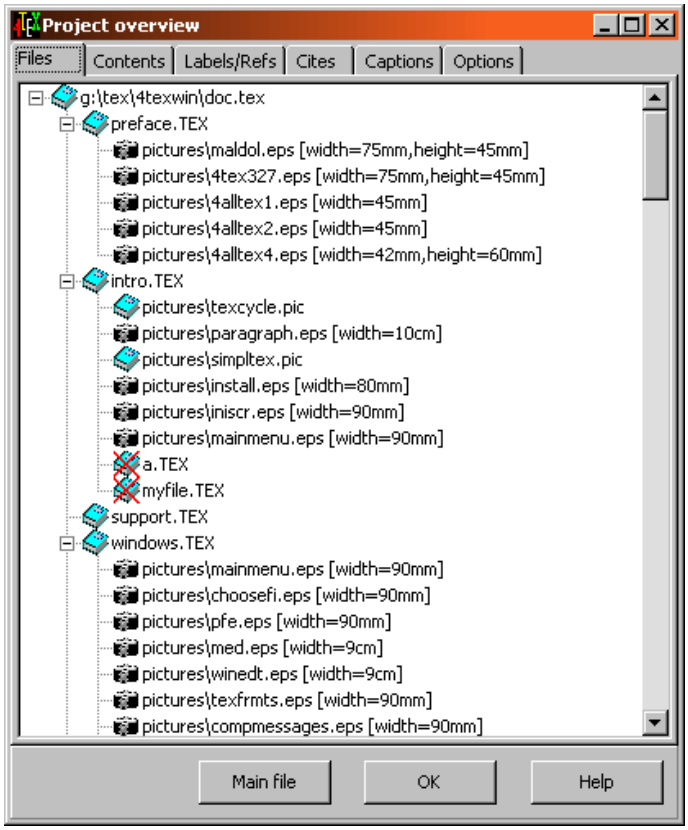


Figure 8.13: View TEX project

that were not found are indicated by a red cross. By clicking on any file it will be loaded in the editor (if it is a TEX file), or displayed by an appropriate viewer (if it is a graphics file). Right-clicking on a TEX file will invoke the spell-checker.

The next tab field ‘Context’ show the structure of the project: chapters, sections, subsections, etc. are presented in a tree format. The names of these headers are given, along with the file name and position therein. By clicking on a header the file will be loaded in the editor and the header will be on the current line.

The ‘Labels/Refs’ tab field shows all labels and references. In case there are references to non-existing labels these will be listed at the top. Again the file an the position therein of any label or reference is given. By clicking on a label or reference the file will be loaded in the editor and the label or reference will be on the current line. Right-clicking on a label or reference will copy the reference to the Windows clipboard, so you can simply paste it into your document.

The ‘Cites’ tab field will list all bibliographic cites. By clicking on a cite the file will be loaded in the editor and the cite will be on the current line. Right-clicking on a

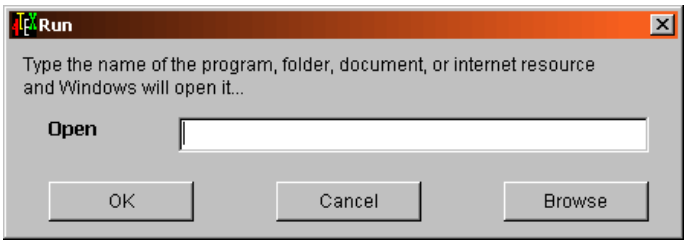


Figure 8.14: The Run menu

cite or reference will copy the reference to the Windows clipboard, so you can simply paste it into your document.

Captions of tables and figures are listed in the 'Captions' tab field. Again the file and the position therein of any caption is given. By clicking on a caption the file will be loaded in the editor and the caption will be on the current line.

How 4Project detects chapters, labels, cites, etc. can be configured in its 'Options' tab field. We advise you to read 4Project's own online help for details on its configuration.

8.10 The Run menu

This menu is similar to the Windows Run option. You can use it to start another program from within 4TEX. It is very similar to the 'Run' option in the standard 'Start' menu of a Windows system.

Use the 'Open' field to specify the program that you want to run, or click on Browse to select the program you want to run. Use OK to start the selected program and quit the 'Run' menu. Use Cancel to quit the 'Run' menu without running a program.

8.11 Advanced 4TEX options

In chapter 9 we will discuss the more basic options of 4TEX that any user can play with. Some less used or tricky options of 4TEX can be configured by clicking on 'Advanced 4TEX options' in the utilities menu. You could also edit `4TEX.INI` by hand, but this interface is easier and it will supply online help on all settings.

These options are called *advanced* options for a reason. If you are not careful or if you don't really understand the meaning of any option, you could seriously mess up the 4TEX system. It is advisable to make a backup of `4TEX.INI` (you can find it in the Windows directory or in the directory where `4TEX.EXE` resides) before you change anything.

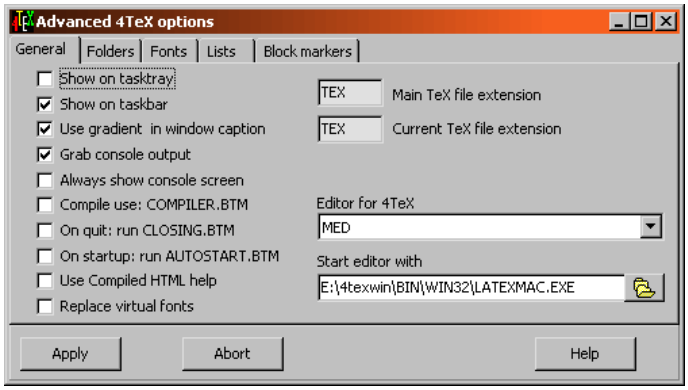


Figure 8.15: Advanced 4TEX options: general

Click on Apply when you are done changing the option, or click on Abort if you want to cancel any changes you just made. Click on Help to get information on what the options mean.

The Advanced 4TEX options menu is divided into 5 'tab sheets': General, Folders, Fonts, Lists, and Block markers.

In the General tabsheet you can select the following options:

Show on tasktray Check this option if you want a small 4TEX icon to appear on the Windows tasktray when it is running. Selecting this option and deselecting the next option at the same time will save some space on your taskbar.

Show on taskbar Check this option if you want 4TEX to appear on the Windows taskbar when it is running. This is the normal behavior of any Windows program. You can deselect it to save space on a crowded taskbar.

Use gradient in window caption To make 4TEX's appearance a bit more attractive you can select gradient captions. In that case captions will appear as a bar that becomes lighter in color from the left to the right. In Windows 98 gradient captions are a standard feature.

Grab console output If you want screen output from console applications, such as the TEX compiler, to be displayed in a Window controlled by 4TEX, you should check this option. Otherwise, screen output will go to a standard Dos-box.

Always show console screen If this option is checked, the console output window will always be visible. If not, it will only be visible when a console application, e.g. the TEX compiler, is running.

Compile use: COMPILER.BTM If you prefer to handle TEX compilation completely from a batch file, you can check this option. In that case 4TEX will not

execute a compile command directly, but execute the batch file `COMPILER.BTM` to which it will pass a number parameters.

On quit: run CLOSING.BTM You can check this option if you want more control over what will happen when you quit 4TEX. Instead of quitting 'normally', 4TEX can execute the batch file `CLOSING.BTM` in which you can execute whatever programs you want.

On startup: run AUTOSTART.BTM Just like 'On quit', you can make 4TEX execute a batch file on start-up. From `AUTOSTART.BTM` you can execute whatever programs you want.

Use Compiled HTML help If Microsoft Internet Explorer 3.02 or higher is installed on your system, and your system is capable of displaying 'Compiled HTML Help' files (with extension `.chm`), you can use this kind of online help. It is the successor of the old-fashioned 'Winhelp'. If your system is not able to display such files, 4TEX can display plain HTML files instead, by launching your World Wide Web browser.

Replace virtual fonts

This is an option that should be used only by those users who understand what virtual fonts are and how DVI drivers deal with them. If you feel uncertain about this, we advise you to leave this option *unchecked*. Chances are that you will never need it.

If you check the 'Replace virtual fonts' option 4TEX will run the program DVIcopy after each compilation in order to replace all *virtual* fonts in the DVI file with *real* fonts. This options is useful when using DVIWIN (a non-supported previewer) as a previewer because it doesn't support virtual fonts. It can also be useful if you want to use the DVI file with other programs that do not support virtual fonts. When distributing the DVI file to others, it may also be useful to devirtualize it. See section 13.7.16 for details on the DVIcopy program.

From the 'General' tab sheet you can also set the extension of the Main file and the Current file. By default they are both `TEX`. Note that you should *not* supply a dot in front of the extension.

You can also select an editor program from the 'General' tab sheet. The editors MED, PFE and WINEDT are pre-installed. Along with the editor program, 4TEX can launch another program. This can be any handy utility such as LATEX Mac (see section 8.13).

In the next tab sheet, 'Folders', you can specify a few specific folders that 4TEX needs (see figure 8.16). The following items can be specified:

Manual pages The folder where 4TEX can find manual pages of programs used by TEX and friends. 4TEX will find `.ps` or `.pdf` files in the folder `manpages` below the folder you specify here.

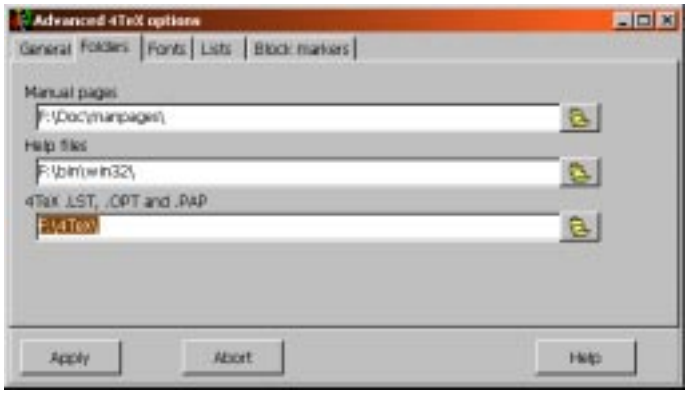


Figure 8.16: Advanced 4TEX options: folders

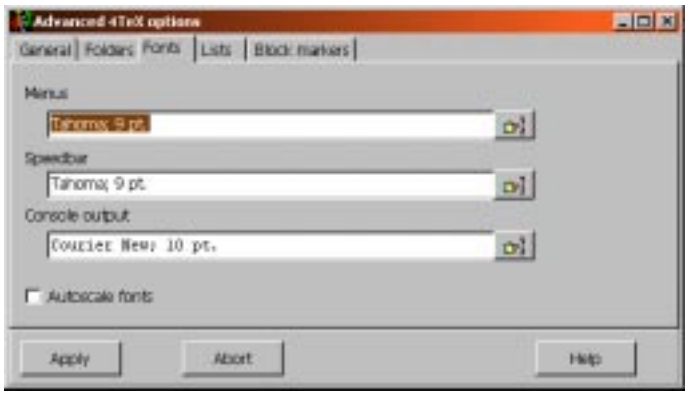


Figure 8.17: Advanced 4TEX options: fonts

Help files The folder where 4TEX can find Windows help files of programs used by TEX and related programs. 4TEX will find `.hlp` or `.chm` files in the folder you specify here.

4TeX .LST, .OPT and .PAP The folder where 4TEX's `.lst`, `.opt` and `.pap` files are located. In a multi-user environment you could have each user make his/her own set, dedicated to specific tasks, without having to install everything locally.

In the 'Fonts' tab sheet you can specify the fonts that 4TEX will use:

Menus The font uses in all the 4TEX screens.

Speedbar The font used for the speedbar displaying buttons with text on them.

Console output The font used for the console output window.

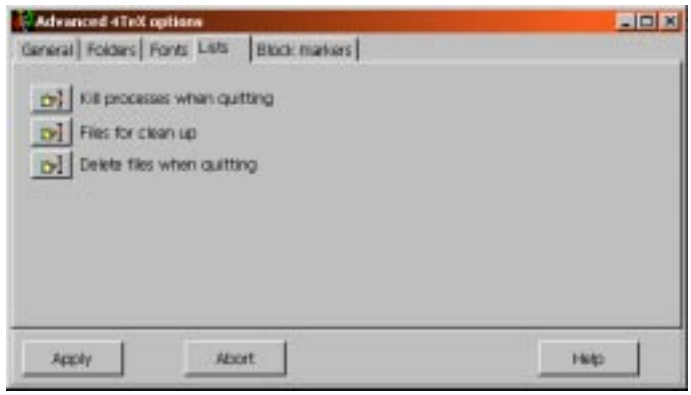


Figure 8.18: Advanced 4TEX options: lists

In the 'Lists' tab sheet you can specify lists of files and programs for specific tasks:

Kill processes when quitting Programs that 4TEX should 'kill' when you exit 4TEX. You have to specify the names exactly as they appear in the task list produced by programs such as `PS.EXE` (in folder `\bin\win32`).

Files for clean up Files that will be displayed as candidates for deletion from the Clean up menu. You can use 'wild cards' (`*` and `?`) in file specifications.

Delete files when quitting Files to be deleted automatically every time you quit 4TEX. You can use 'wild cards' (`*` and `?`) in file specifications, but do be careful.

In the 'Block' markers tab sheet you can set the 'Begin' and 'End' marker that will be used in block compilation (see section 6.5). 4TEX scans the Main file for the 'Begin' marker and uses all text up to this marker as the preamble of a block compilation. Then the text block from the Windows clipboard is appended. Finally the 'End' marker text is added. After this the temporary file should be ready for compiling.

8.12 LATEXhelp

When writing a document in LATEX you sometimes or perhaps often need to look up a control sequence, symbol or other information.

Unfortunately L. Lamport's *LATEX, a Document Preparation System* (and many other LATEX related books) are often too elaborate, or you simply can't find what you are looking for because it is not in the index. Finding documentation can also be hard if the index has 5 or more entries to the subject, as is often the case in D. Knuth's *The TEXbook*.

A more convenient way to get help on LATEX related items is a hypertext help system. The file `latexhlp.hlp` is a standard Windows help file with all the hypertext

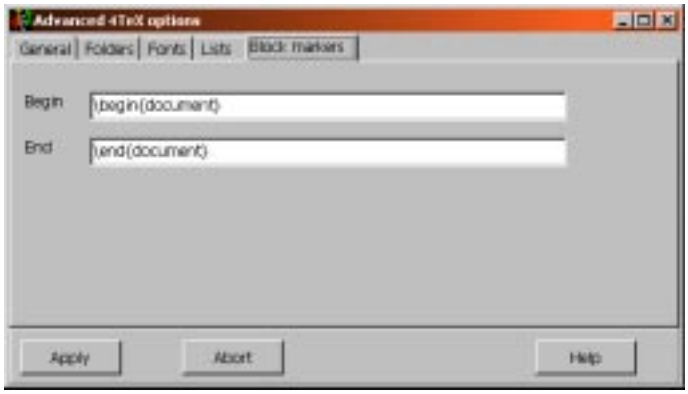


Figure 8.19: Advanced 4TEX options: block markers

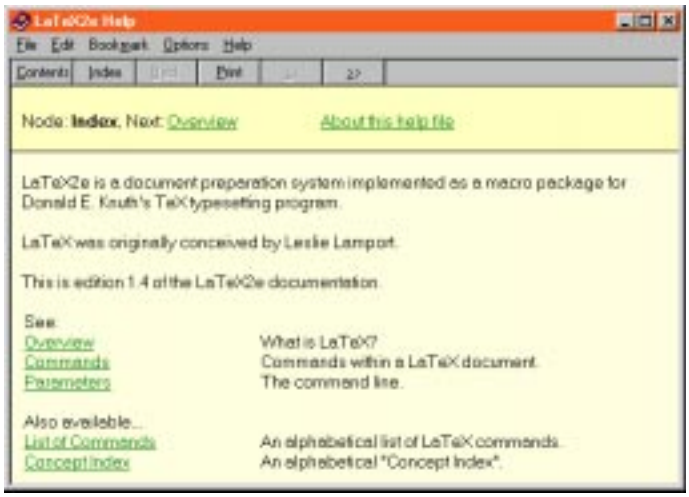


Figure 8.20: LaTeXhelp

and search facilities that the Windows help system supports. This help file describes LaTeX 2_ε in detail.

8.13 LaTeX Mac

LaTeX Mac is a program that can make writing a TeX file more WYSIWYG (What You See Is What You Get). This program, written by J. Aguirregabiria, offers several 'tool-

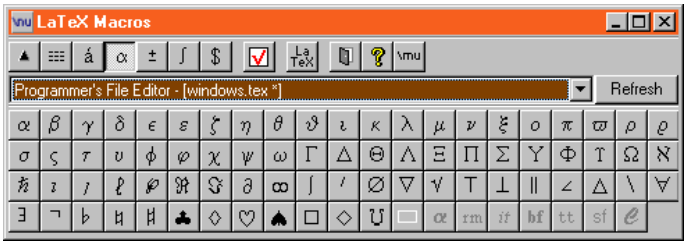


Figure 8.21: LaTeX Mac: inserting Greek letters

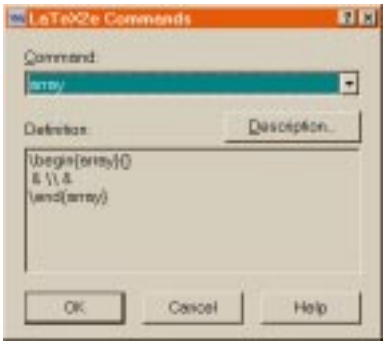


Figure 8.22: LaTeX Mac: inserting LaTeX environments

bars' with all mathematical operators, all Greek symbols, etc. By clicking on, e.g., a Greek lambda the program will generate the TeX code `\lambda`.

Likewise, it can generate complete LaTeX environments, and more. Text is inserted at the current cursor position, so beware that the cursor in your editor is at the right position when you choose an item from LaTeX Mac's toolbar.

LaTeX Mac can only communicate with your editor after you have told LaTeX Mac what your editor is. You can pick an application that LaTeX Mac will communicate with from the *output destination* selector. See figure 8.23. You can make LaTeX Mac automatically try to connect to your editor program: click on ☑, go to the Output window, check one or both the options, and specify the *Class name* as follows:

PFE: `PFE32_Frame`
MED: `MRED_MAINWIN_CLASS`
WINEDT: `TApplication`

Alternatively you could specify the 'Caption' as follows:

PFE: `Programmer's File Editor`
MED: `MED`
WINEDT: `WinEdt`

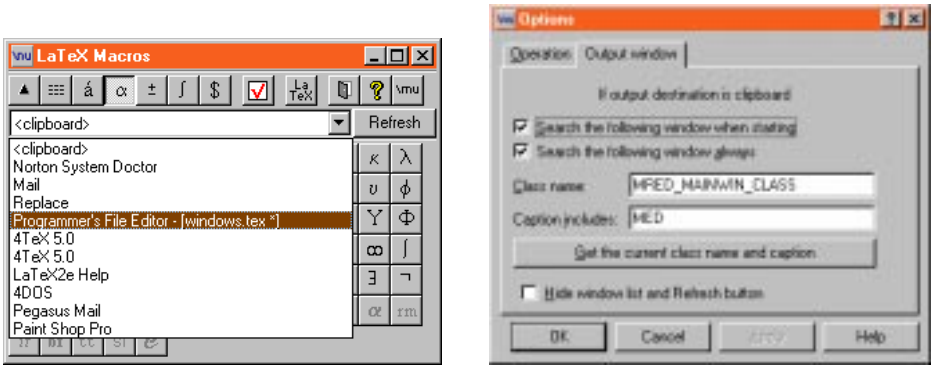


Figure 8.23: LaTeX Mac: selecting an application

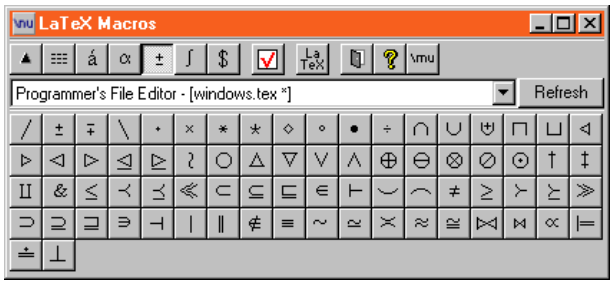


Figure 8.24: LaTeX Mac: math and relational operators

Specifying either *Class name* or *Caption* should be sufficient.

8.14 LaCheck

This program can do a syntax check on LaTeX files. When you select 'LaCheck' from the utility menu you will be prompted to specify the file that you want to check.

When LaCheck has finished you will be asked if you want to see the results of LaCheck's analysis.

8.15 ChkTeX

This program, written by J. Berger, can do a syntax check on TeX files. It is not specifically geared for LaTeX like LaCheck. When you select ChkTeX from the utility menu, you will be prompted to specify the file that you want to check.

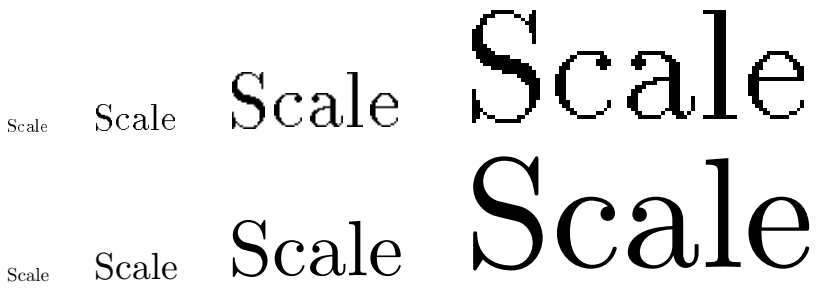


Figure 8.25: Scaling a bitmapped picture (top) and a vector picture (bottom)

When ChkTEX has finished you will be asked if you want to see the results of ChkTEX's analysis.

8.16 Graphics

Of course you can include graphics in a TEX document, and it isn't very difficult to get good results. However, for the best result it is necessary to have a good understanding of what a graphic is. Graphics come in many types, and depending on what you are trying to achieve you may need to choose or convert a graphic file to get optimal output.

8.16.1 Types of graphics

There are two major kinds of graphics: *bitmapped* graphics and *vector* graphics. Bitmapped graphics contain a matrix of pixels. Each pixel's color is defined in the matrix. The size of the matrix determines the printed size or the resolution. E.g., if a bitmapped graphic file contains a matrix of 200 × 100 pixels, it will print as a picture of 1/3 inch × 1/6 inch on a 600 dpi printer. That is, if you print the picture at its natural size. In case you decided to print the picture at 10 inch × 5 inch, it will most probably look bad because the picture had to be magnified far beyond its original resolution. Printing the same picture at 1/10 inch × 1/20 inch is likely to give bad results as well. In general, scaling of bitmapped pictures doesn't work very well.

Vector pictures are quite different. Such pictures are defined as pen strokes, circles, squares, lines, etc. that can be scaled up or down to any size: they are resolution independent. In figure 8.25 we will demonstrate the effect of scaling a bitmapped picture and a vector picture.

So why don't we use vector pictures all the time? Because in some cases bitmaps are simply much easier to use. Below we will list some considerations that may be helpful in deciding which is more appropriate in any specific case.

- Bitmapped graphics are easy to obtain (a screen dump, Internet, scanners, etc.).

- Bitmaps are easy to edit or preview with many common (often freeware) tools. For vector graphics there are only a few good freeware tools available. Importing vector graphics is supported by a only few freeware tools, and (to some extend) by professional commercial tools.
- Bitmapped graphics can all be easily converted from one format to another (e.g. with ImageMagick's 'convert' program). Converting from one vector format to another is usually very hard if not impossible. Converting from vector graphics to bitmap graphics is usually quite simple.
- Vector graphics can be scaled to any size without any decrease in quality. With bitmapped graphics there is always a decrease, which can be unacceptable.
- In vector graphics you can change the attributes (color, thickness, font, etc.) of any element individually. In bitmap graphics you can only change pixels, there is no underlying structure.
- Bitmapped graphics *can* sometimes be converted to true vector pictures using a technique called 'tracing'. This technique often requires some fine tuning, so it can take lots of time. Specialized graphics software is required.
- High resolution bitmap graphics files can be huge (up to tens of megabytes). Choosing the right format and compression can make a big difference. Vector graphics are typically relatively small; their size does not depend on resolution or color depth.
- Vector formats are the best choice for pictures that contain constructed graphic elements such as lines, squares, circles, text, etc. They are not suited for photographs and other pictures that contains extremely fine details that can not be described as graphics elements.

The most popular formats for bitmapped pictures are:

TIF (Tagged Image Format File) A format that supports several subformats that allow for different compression techniques.

BMP (Bitmap) A format understood by all Windows graphics programs. Files of this type tend to be very big.

GIF (CompuServe Graphics Interchange Format) A widely used format, e.g. on World Wide Web pages. There are different subformats, including an 'animated' variant. The color depth is limited to 256 colors (8 bit).

JPG (Joint Photographic Experts Group) A format that supports many different compression techniques, including 'lossy' techniques. The advantage of lossy compression is an extremely high compression rate (up 500 times smaller than an equivalent BMP file), resulting in a very small file size. Of course you loose quality but in some cases the small file size may be more important.

PNG (Portable Network Graphics) A fairly new general purpose format.

PCX (ZSoft IBM PC Paintbrush) A very old and widely supported format. Note that the color depth is limited to 256 colors (8 bit). Also note that DVIPS supports PCX pictures but only if they are black-and-white (1 bit color depth).

There are surprisingly few popular formats for vector pictures. The reason is probably that they are more difficult to interpret. Nevertheless, all professional graphics programs are capable of exporting vector pictures, usually in Encapsulated PostScript format.

WMF (Windows Meta File) A format used by some Windows programs. Unfortunately WMF is a confusing format because there are two versions of it, which are hard to distinguish. The newer version is sometimes called EMF (Extended Meta File), but very often you will find that WMF pictures are in fact EMF pictures. Older Windows programs may not be able to handle WMF pictures of the newer version.

EPS (Encapsulated PostScript) This format is the most common and probably most powerful vector format available. Unfortunately its powerful language (PostScript) also makes it difficult to interpret. Very few graphics programs fully support EPS, some support a subset of the language.

CGM (Computer Graphics Metafile) An older format that is used by some graphics programs.

HPGL (Hewlett-Packard Graphic Language) An old format that is still used by many plotting devices. This plotting language is based on using physical pens to draw directly on paper. Some laser printers can emulate this language.

WPG (WordPerfect Graphic) The graphic program WordPerfect Presentations (nowadays Corel Presentations) supports this format. WPG pictures are typically difficult to convert reliably to any other format.

Note that WMF, EPS and CGM pictures can in fact contain bitmaps, so the file extension `.wmf`, `.eps` or `.cgm` is no guarantee that you are dealing with a vector picture.

In the following sections we will describe programs that specialize in editing or displaying graphic pictures.

8.16.2 Paint Shop Pro

Paint Shop Pro, written by Jasc Software, Inc. is a shareware program that can be used to edit or convert bitmapped graphics.

The program supports a great number of bitmapped formats but it can not import or export vector pictures. Many filters for special effects are available, as well as options to tweak colors, retouch parts, or resize pictures. There is also an option to make pictures of (parts of) your screen and save them as bitmapped graphics (this is how all screen pictures in this manual were made).

Figure 8.26: Paint Shop Pro

Figure 8.27: IrfanView

8.16.3 IrfanView

IrfanView, written by I. Skiljan is a program for viewing bitmapped graphics. It is a freeware program that can display pictures of numerous bitmapped formats. The program is also capable of converting from one bitmapped format to another.

8.16.4 GSview

GSview is a graphic interface around Ghostscript. It can display PostScript files (`.ps` and `.eps`) and `.pdf` files. The program also features a user-friendly interface for printing a PostScript file, or converting it to any of a huge set of bitmapped output formats. See section 7.5 for details.

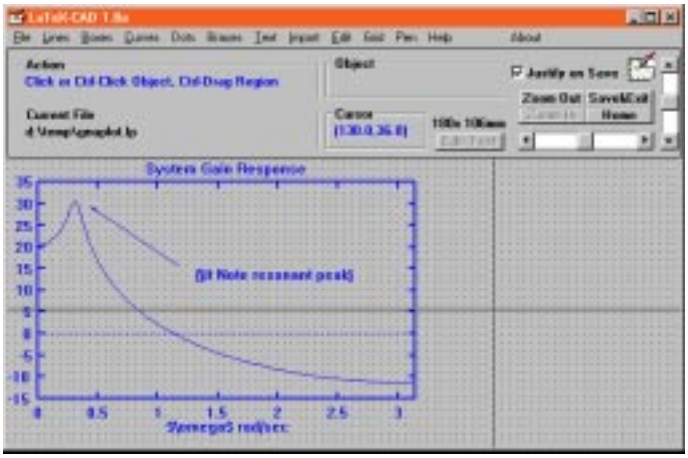


Figure 8.28: LaTeXcad

8.16.5 LaTeXcad

For LaTeX users the program LaTeXcad by J. Leis might be interesting. It is a drawing program that can output a (fully scalable) LaTeX picture environment, that can be included in any LaTeX document with little effort. Note that the TeX input file `latexcad.sty` is required to compile such a picture.

8.16.6 GNUplot

GNUplot is a general plotting program written by C. Kelley and T. Williams. It is capable of plotting curves from mathematical expressions, or from data files. It can export pictures in a variety of ways. Most important for TeX users are EPS (Encapsulated PostScript) and LaTeX picture environment. See figure 8.29 for a screenshot. D. Kotz, T. Williams and C. Kelley and A. Woo wrote extensive documentation (cdrom).

8.16.7 Mayura Draw

This is another fine drawing program, written by Mayura Software. It can output ('export') pictures in EPS, TIFF, Adobe Acrobat PDF, Adobe Illustrator, BMP bitmap and Windows Metafile WMF. Mayura Draw is a vector oriented drawing program, unlike e.g. Paint Shop Pro. See figure 8.30 for a screenshot.

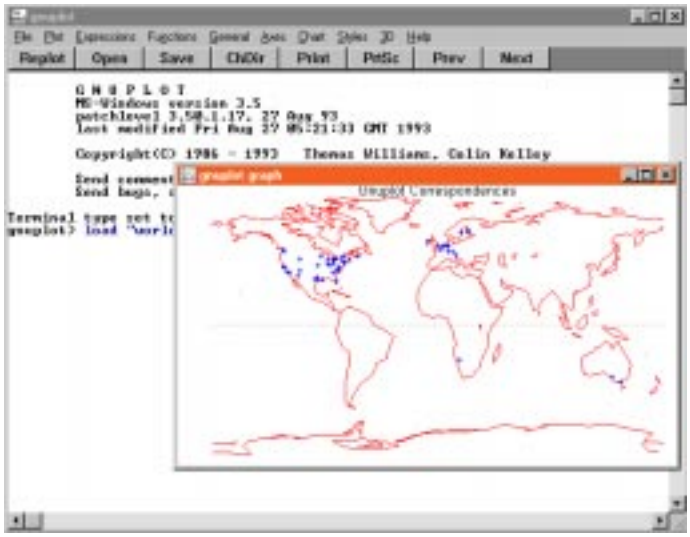


Figure 8.29: GNUplot

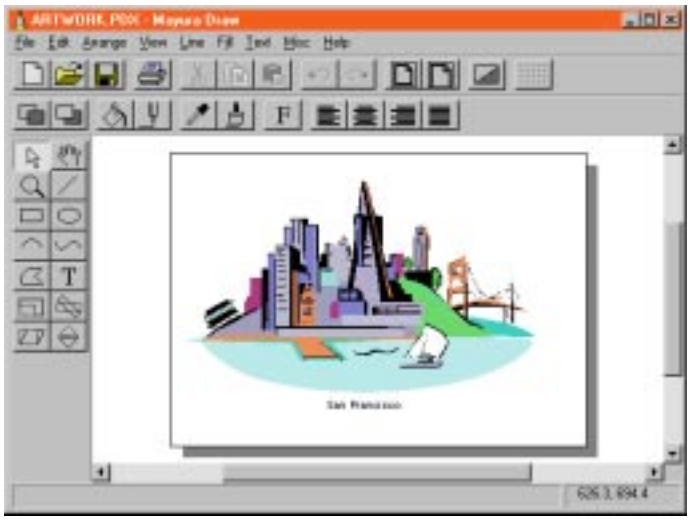


Figure 8.30: Mayura Draw

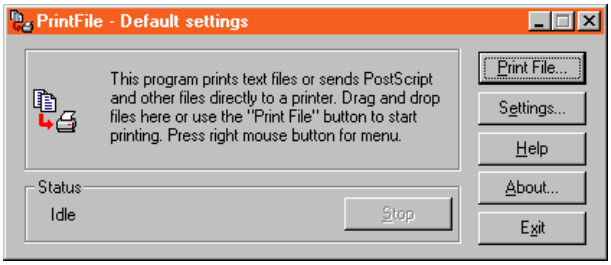


Figure 8.31: PrintFile

8.17 PrintFile

The program PrintFile written by P. Lerup can be used to print an arbitrary PostScript file. Note that this is only useful if you have a PostScript printer. The program has many options, e.g. for scaling and print N-up (multiple pages on one side of a sheet). It can print to any Windows printer.

PrintFile is also capable of printing ASCII text files, such as the sources of TEX documents, C programs or email messages. In that case it offers options for 'pretty printing' (e.g. underlining reserved words in C or Pascal), line wrapping, and header lines.

8.18 LATEX Wizard

The LATEX Wizard, written by P. Wiladt, is a tool that guides you through the initial steps of setting up a framework for a LATEX document. It will ask you what *class* of document it should be, what paper format, what kind of headings, in which language you are going to write, what typeface, and which LATEX *packages* you need, etc.

The LATEX Wizard will generate the complete framework for you. Here is an example of the output:

```
%% done with a little help from the LaTeX Wizard
\documentclass[a4paper,11pt,twocolumn,titlepage]{article}
\pagestyle{headings}
\usepackage[cp1252]{inputenc}
\usepackage[dutch]{babel}
\usepackage{times}
\usepackage[OT1]{fontenc}
\usepackage{rotate}
\usepackage{graphpap}
\usepackage{afterpage}
\usepackage{makeidx}
\usepackage{showidx}
```

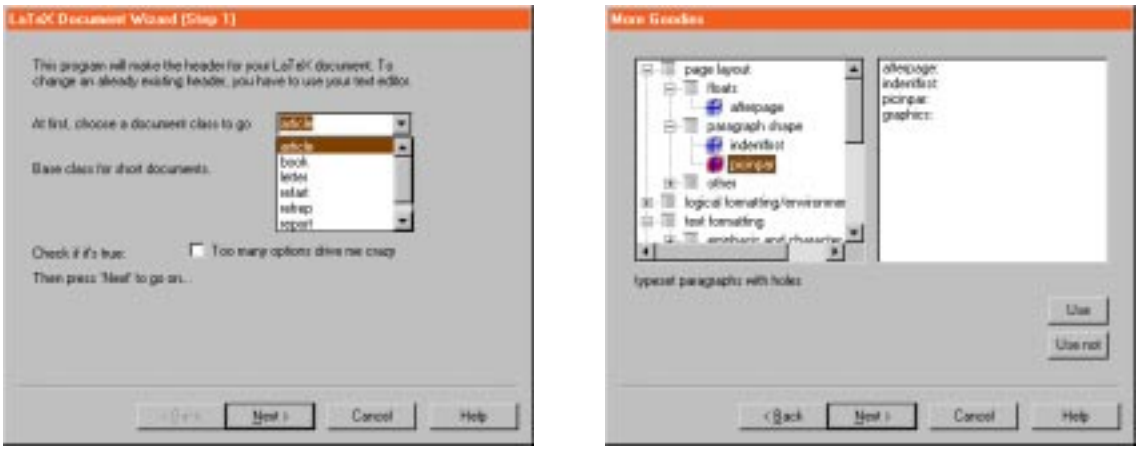


Figure 8.32: LaTeX Wizard

```
\usepackage{verbatim}
% this comes from the Font Matter section
% it is commented out to not bother you
%\usepackage{mymacros}
%\makeatletter
%\@addtoreset{chapter}{footnote}
%\makeatother
% end front matter
\begin{document}
\title{Een leuk artikel}
\author{Jan Doedel}
\date{1998-12-23}
\maketitle
\begin{abstract}
 %% your abstract should be written here
 %%
\end{abstract}
\tableofcontents
\listoffigures
\listoftables
%% here is the place to write your document's text
 %%
\end{document}
```

8.19 Other tools

Select and show manual page This tool will open a file dialog that allows you to specify which PDF document you want to have displayed on screen.

Select and show Windows help file This tool will open a file dialog and you can specify which Windows help file (`.hlp` or `.chm`) you want to have displayed on screen.

Regenerate the ls-R file(s) The TEX system uses index files (called `ls-R`) to quickly locate files. In case you add or delete files from the TEX tree, you need to regenerate the index file.

Windows Explorer Start the Windows Explorer.

Manage Spell-checker dictionaries If you want to change anything in the dictionaries supplied with 4TEX, you will need this program. You could, e.g., merge your own list of words into the standard dictionary, or make a new dictionary based on a word list that you composed.

Chapter 9

The options menu

This menu allows you to set several global options of 4TEX. The menu is divided into four panels (see figure 9.1):

- Editor for TEX files
- Editor for BIBTEX
- Language for spell-checking
- Screen options

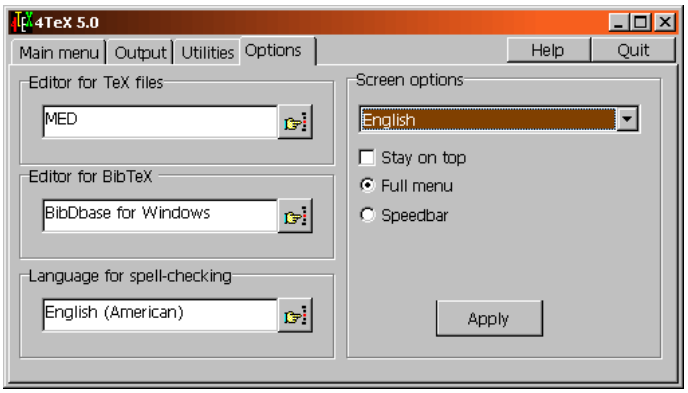


Figure 9.1: The Options menu

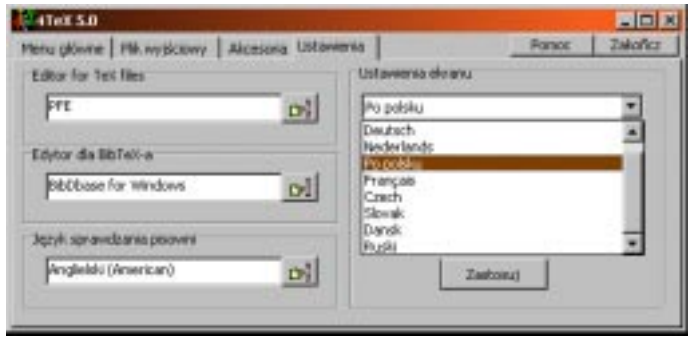


Figure 9.2: 4TEX ‘speaking’ Polish

9.1 Editor for TEX files

4TEX comes with three editor programs pre-installed: MED (see section 6.2.2), PFE (see section 6.2.1) and WINEDT (see section refh:winedt). You can select any of these form this menu.

In the Advanced options menu (see section 8.11) you can select a program to be launched concurrently with the editor program. Programs such as LATEX Mac (see section 8.13 can be handy tools in addition to the editor program.

9.2 Editor for BIBTEX

You can select the editor to be used for editing BIBTEX files. In section 8.1 a few possible BIBTEX editors are discussed.

9.3 Language for spell-checking

You can select the language to be used for spell-checking TEX files. Right-clicking on Spell-check in the Main menu is equivalent. Spell-checking with 4Spell is explained in section 6.9.

9.4 Screen options

The first field in this panel can be used to select the language that is used in all the 4TEX screens. The language can be changed on the fly, so the 4TEX window will change instantly. Figure 9.4 shows 4TEX ‘speaking’ Polish. Currently an English, German, Dutch, Polish, Danish, Russian, French, Czech and Slovak interface are available.

If you check the ‘Stay on top’ option the 4TEX main window will always be displayed on top of all other windows, so it will always be completely visible. The ‘speedbar only’ option will invoke the speedbar at the position you have selected. The ‘top + speedbar’ option will invoke the speedbar in ‘always on top’ mode.

All of 4TEX’s windows can be scaled individually by ‘pulling’ its borders. If you want the fonts that 4TEX uses to scale accordingly, you should check the option ‘Scale fonts of 4TeX screens’. Note that in the Advanced options menu (see section 8.11) you can select the fonts you want 4TEX to use in its windows.

The next option you can choose is whether use 4TEX in ‘Full menu’ mode or in ‘Speedbar’ mode. ‘Full menu’ is the mode that you’re using when e.g. setting these option. In ‘Speedbar’ mode 4TEX will be a lot smaller. It will only display the major functions as small buttons. This mode can be very convenient if your screen is not very big or already crowded with other windows, or if you only need a few 4TEX functions. In ‘text’ mode that looks like this:

In case you selected the more compact ‘icon’ mode of the speedbar it will look like this:

A vertical version of the speedbar (both in ‘text mode’ and in ‘icon mode’) is also available.

This speedbar is as small as possible but nevertheless it has the same buttons as in the main menu ‘Full menu’ mode. It has two additional buttons: Menu and Utilities. The first can be used to switch from ‘Speedbar’ to ‘Full menu’. The meaning of the icons may not be obvious to you at first, but 4TEX will always display a ‘hint’ text whenever you move the mouse pointer over a button. The ‘hint’ will give you information on which file was selected as ‘Main TEX file’, which TEX format was selected, which previewer, which printer and which language for spell-checking.

If you click on the 4TEX icon of the speedbar a tiny menu will pop up that allows you to select ‘Help’ or ‘About’. The first fires up the Windows help system displaying help on 4TEX; the latter will display 4TEX’s ‘About’ window (see section 6.11).

Right-clicking on the 4TEX icon of the speedbar can be used to move the speedbar to a different position. Once you right-click on it the speedbar will follow the mouse as you move it. Right-click again and the 4TEX speedbar will stay at its current position. Note that moving the speedbar is only possible if you selected ‘user position’. In case you selected a fixed position you will not be able to move the speedbar.

Chapter 10

Things to be aware of

10.1 Text encoding

As long as you are using only the standard characters a–z and the standard TEX commands to put accents on characters you can be sure that the output produced by TEX will be as expected.

However, if you want to type your text using a more WYSIWYG manner, i.e., without these silly accent commands but as 'real' accented characters, you will have to tell TEX what *text encoding* you are using.

This is necessary because these characters are not part of the standard ASCII character set, and there are a number of standards that define which input character should represent which output character. For instance, if you type in your editor the keys Alt 1 3 7, what do you get?[1] On some systems you will get 'ë' but it depends entirely on the *code page* that your system is using, so you will have to tell TEX what character 137 actually represents.

In LATEX the text encoding (code page) can easily be selected (see section 17.16.2 for details). In Plain TEX there is no direct support for different encodings, so you will have to define the meaning of, say, character 137 yourself. In section 16.15.4 we will show how you could do that. In CONTEXT the text encoding can be specified in a similar way as in LATEX. See section 18.12.2 for details.

The ISO/IEC report 10646-1 on *Universal Multiple-Octet Coded Character Set* contains an exhaustive description of codepages defined in ASCII and Unicode.

1 Make sure that NumLock is on.

10.2 Importing and exporting documents

Unfortunately TEX is not a main stream document format, so you should not expect to find many import or export facilities in other word processors or other programs. A few exceptions exist, though, and there are several very good convertors available.

You may want to read W. Hennings' overviews on converters from PC text processors to LATEX and vice versa (cdrom). They are available on the World Wide Web at the following addresses: `http://www.kfa-juelich.de/isr/1/texconv/pctotex.html` and `http://www.kfa-juelich.de/isr/1/texconv/textopc.html`.

Importing

First we will discuss *importing* TEX documents into other programs. In general, our experience is that the best strategy is to simply insert the plain TEX code and then replace anything that is not suitable. That may seem disappointing but most other strategies simply fail when the going gets tough. E.g., importing mathematics or pictures nearly always fails, or is far from optimal.

Nevertheless, you may want to try some of the convertors that 4TEX supports (see section 8.7). In case you want to convert TEX to Microsoft Word you could try the following strategy: first translate your TEX document to RTF (Rich Text Format) and then import the RTF version. Alternatively, you could convert the TEX document to HTML (Hypertext Markup Language) and then import the HTML version. The results may not be perfect but still good enough for less demanding purposes.

Importing text from other programs into a TEX document can be hard, too. Again 4TEX supports a few convertors (see section 8.7) that you may want to try. If they don't work (well), you could try to convert to RTF or HTML first and convert from that format to TEX. In any case you should be able to use *cut and paste* to copy text from any Windows program via the clipboard into your TEX document.

Exporting

Only very few programs can export text or pictures in native TEX format. The programs *Mathematica* and *Maple* can export mathematical equations in TEX format. The graphics program GNUplot can export graphics in TEX, LATEX picture or (E)EPIC picture format. But of course pictures in other formats can also be used by TEX, and text can be copied using *cut and paste*. Naturally you will need to edit this imported text because no formatting information is transferred.

4TEX supports a few convertors that allow you to convert documents in, e.g., WordPerfect, ChiWriter, HTML or RTF format directly to TEX. However, don't expect miracles from these programs. Manual intervention is usually still needed. See section 8.7 for details, and the manual of any convertor you want to use.

10.3 Multi-lingual documents

TEX allows you to write documents in any language(s) you like. But before you try, make sure you understand the preparations you should make.

First of all, the way TEX deals with languages depends on the TEX dialect that you use. Plain TEX, LATEX and CONTEXT all take slightly different approaches. In part IV of this book, in which we describe these dialect, their features for dealing with to multi-lingual documents will be discussed.

In case (part of) your document is written in a language that requires accents, you may want to have a closer look at section 16.9, section 17.3.8 or section 18.12.2 for details on how Plain TEX, LATEX or CONTEXT deals with accents and these characters. In case you use LATEX section 17.16.2 is well worth reading.

Depending on the language you are using, you may need specific fonts. 4TEX comes with many different fonts pre-installed (e.g., Cyrillic and Chinese). Most probably you will need to load some extra TEX commands to instruct TEX to use these fonts. Since there are hundreds of languages that you may want to use, and we are not experts on foreign languages, we can't advise you on what fonts or macro packages you should use. We suggest that you contact a Local TEX Users Group or pose your questions at one of the mailing lists on TEX related subjects. In chapter 5 you can read all about it.

Spell-checking multi-lingual documents can be done with the program 4Spell. This spell-checker currently supports English (American and British), Dutch, German, Italian, Spanish, Swedish, South-African, Danish, Czech, Slovak, Polish and Russian. In a multi-lingual document you can switch to another language at any point, using a comment that 4Spell understands. In a LATEX document you could write e.g.:

```
\selectlanguage{english}  %4language=US
Now we are writing in English.
\selectlanguage{dutch}    %4language=NL
Nu schrijven we in het Nederlands.
\selectlanguage{germanb}  %4language=DE
Jetzt schreiben wir auf Deutsch.
```

Note that the switching command can be configured. In the example we choose a TEX comment that 4Spell will scan for, but the TEX compiler will not typeset it. See section 6.9 for details on spell-checking with 4Spell.

10.4 Including graphics

If you are writing LATEX documents, we advise you to use the `graphicx` package. The command `\includegraphics` has so many options that it is unlikely that you can't manage to include your graphics.

If you use Plain TEX, we advise you to use the `epsf` macros to include graphics. You will have to convert any graphics file to EPS first, though, but that is very easy using the 'Graphic conversion' menu (see section 8.8).

In ConTeXt there is a command called `\externalpicture` that can handle PNG, PNG, TIF and EPS graphics. ConTeXt will automatically choose the right type of graphics (if available) depending on the output that will be generated.

If you use PDFTeX (or PDFLaTeX or ConTeXt) to generate PDF output, you should be aware that only PNG, TIF and PDF graphics are supported. Convert any graphics to one of these formats if necessary. 'Normal' TeX, producing DVI output has more options, but it depends on the DVI driver which graphics formats are supported. In general, we recommend that you use DVIPS to generate PostScript output. This is by far the most versatile DVI driver, and the only one that supports EPS.

10.5 Using color

LaTeX, Plain TeX and ConTeXt all have their own methods for specifying colors. You can check sections 17.19, 16.12 and 18.15 respectively.

Much of the ability to use color depends on the DVI driver that you use (if you generate PDF instead, check section 10.4.). In general, we recommend that you use DVIPS to generate PostScript output. PostScript (or PDF) is the most reliable output format when it comes to color. See also section 13.6.11 for a comparison of DVI drivers.

10.6 Running ε-TeX

ε-TeX is a TeX program that can run either in *compatibility mode* or in *extended mode*. We will assume that if you use ε-TeX it is because of its extended mode, so we will focus on that.

All of ε-TeX's extensions and enhancements available in extended mode are activated either by executing some new primitive command or by assigning a nonzero value to some new integer parameter or state variable. Since all these new variables are initially zero, ε-TeX behaves as TeX as long as none of ε-TeX's new control sequences are used, with the following exceptions which should, however, have no effect on the typesetting of error-free TeX documents (produced with error-free formats):

- When `\tracingcommands` has a value of 3 or more, or
  when `\tracinglostchars` has a value of 2 or more, ε-TeX will display additional information not available in TeX.
- Additional tracing commands supported by ε-TeX are: `\tracingassigns`, `\tracinggroups`, `\tracingifs` and `\tracingscantokens`.
- When using a count, dimen, skip, muskip, box, or token register number in the range 256–32767, ε-TeX will access one of its additional registers whereas TeX would produce an error and use register number zero.

ε-TEX supports lots of extensions and enhancements to 'normal' TEX that are especially interesting to TEX macro writers. All of ε-TEX's features are explained in full detail in 'The ε-TEX manual' (cdrom).

10.7 Running PDFTEX

Running PDFTEX is very similar to running 'normal' TEX. In fact, PDFTEX will produce identical output in DVI format unless you instruct the program to generate PDF instead.

In your Plain TEX document you can write the following lines to select PDF output:

```
\pdfoutput=1
\pdfcompresslevel=9
```

Setting `\pdfoutput` to a positive number selects PDF output; setting it to 0 selects DVI output. Note that this setting must be done *before* the first page is output. The second command specifies the compression level to be used for including any graphics. It should be a number from 0 to 9. 0 means no compression (which is faster), 1 means fast compression, 9 means best compression (which is slower), 2–8 are inbetween.

Two other extra commands supported by PDFTEX running in PDF mode are:

`\pdfpagewidth` = *dimen*
`\pdfpageheight` = *dimen*

If not specified then they are calculated as `\hsize` or `\vsize` + 2 × (1 inch + `\hoffset` or `\voffset`), when PDFTEX writes the first page.

PDFTEX is different from 'normal' TEX/DVIPS with respect to graphics inclusions. PDFTEX does not support Encapsulated PostScript, but PNG, JPEG, TIFF and PDF are supported, so you may have to convert your graphics first.

The most low-level command to include graphics in PDFTEX is:

`\pdfimage` `height` *dimen* `width` *dimen* {*filename*}

Dimensions not specified will be treated as zero. If both height and width are zero then the box takes the natural size of the image. If one of them (width or height) is zero and the second is not, then the first one (the zero one) will be set to a value proportional to the second one so to make the box have the same proportion of width and height as the image natural size. If both width and height are positive then the image will be scaled to fit these dimensions.

If you use PDFLATEX to generate PDF output you can still use the *graphics* or *graphicx* package to include graphics, but you should load the package like this:

```
\usepackage[pdftex]{graphicx}
```

CONTEXT users generating PDF output have less things to worry about. CONTEXT takes care of most settings, but conversion of graphics to PNG, JPG or PDF format is still required.

Another useful PDF-specific command supported by PDFTEX is

`\pdfinfo{`*info keys*`}`

This command can be used to identify the PDF file, for instance like this:

```
\pdfinfo{/Title        (The pdfTeX users manual)
         /Author       (Han The Thanh, Sebastian Rahtz, Hans Hagen)
         /Creator      (ConTeXt / Pragma ADE / Hasselt NL)
         /Producer     (pdfTeX)
         /CreationDate (D:19990128141000)
         /ModDate      (D:19990128141434)
         /Subject      (a manual for pdfTeX users)
         /Keywords     (pdfTeX, TeX, PDF)}
```

Note that keywords such as `/Title` can not be chosen freely. The ones listed in the example are recognized by Adobe's Acrobat Reader and will be displayed when choosing 'File', 'Document Info', 'General' from its menu bar.

Chapter 11

Summary

In this chapter we will summarize all menus that are available in 4TEX. For each menu we will list briefly its purpose, how you can get there, for which kind of users it is designed, and where you can find a detailed description.

Main menu

Purpose:	edit, compile, preview, print, edit log file, spell-check
Part of:	top level
Users:	all
More info:	chapter 6 on page 57

About

Purpose:	display info on 4TEX version, system resources, debugging info
Part of:	all menus
Users:	all
More info:	section 6.11 on page 70

Debug

Purpose:	gather info on 4TEX installation for debugging purposes
Part of:	all menus
Users:	users who want to submit a bug report
More info:	section 6.12 on page 71

Debug – SysInfo

Purpose:	display info on vital system parameters
Part of:	Debug
Users:	advanced users who want to check technical details or write a bug report
More info:	section 6.12 on page 71

Help

Purpose:	display online help on 4TEX menus
Part of:	all menus
Users:	all
More info:	section 6.10 on page 69

Output

Purpose:	select printer and previewer, print, preview
Part of:	top level
Users:	all
More info:	chapter 7 on page 75

Output – Page range

Purpose:	select pages to be printed, number of copies, duplex printing
Part of:	Output
Users:	all
More info:	section 7.1 on page 77

Output – Page style

Purpose:	select page size, output resolution, magnification, landscape, manual feed
Part of:	Output
Users:	advanced users
More info:	section 7.2 on page 79

BIBTEX

Purpose:	edit bibliographic database, generate bibliographies
Part of:	Utilities
Users:	mostly LATEX users
More info:	section 8.1 on page 92

MakeIndex

Purpose:	generate index(es), generate word list
Part of:	Utilities
Users:	mostly LaTeX users
More info:	section 8.2 on page 96

METAFONT

Purpose:	generate bitmapped fonts or EPS graphics
Part of:	Utilities
Users:	METAFONT programmers
More info:	section 8.3 on page 98

METAPOST

Purpose:	generates EPS graphics
Part of:	Utilities
Users:	METAPOST programmers
More info:	section 8.4 on page 98

Generate (LA)TeX format

Purpose:	generate format files
Part of:	Utilities
Users:	advanced users
More info:	section 8.5 on page 100

Conversion tools

Purpose:	convert files from one format to another
Part of:	Utilities
Users:	all
More info:	section 8.7 on page 102

Graphics conversion

Purpose:	convert graphics files of one format to another
Part of:	Utilities
Users:	all
More info:	section 8.8 on page 105

Advanced 4TEX options – General

Purpose:	change setting that affect 4TEX's behavior
Part of:	Utilities
Users:	advanced users
More info:	section 8.11 on page 108

Advanced 4TEX options – Folders

Purpose:	set location of manual pages, help files and 4TEX files
Part of:	Advanced 4TEX options
Users:	advanced users
More info:	section 8.11 on page 108

Advanced 4TEX options – Fonts

Purpose:	select fonts used for 4TEX menus, speedbar and console output
Part of:	Advanced 4TEX options
Users:	advanced users
More info:	section 8.11 on page 108

Advanced 4TEX options – Lists

Purpose:	select processes to be 'killed' when quitting, files for clean-up, files to be deleted when quitting
Part of:	Advanced 4TEX options
Users:	advanced users
More info:	section 8.11 on page 108

Advanced 4TEX options – Block markers

Purpose:	set strings that mark begin and end of block for block compilation
Part of:	Advanced 4TEX options
Users:	advanced users
More info:	section 8.11 on page 108

Clean up files

Purpose:	delete files that are not needed any more
Part of:	Utilities
Users:	all
More info:	section 8.6 on page 102

Run program

Purpose:	run an external program
Part of:	Utilities
Users:	all
More info:	section 8.10 on page 108

Options

Purpose:	select editor programs, screen language, LaTeX view (menu or speedbar)
Part of:	top level
Users:	all
More info:	chapter 9 on page 125

PART III

The technical ins and outs

Chapter 12

The 4TEX system

Although strictly speaking 4TEX is 'only' a shell in which many other programs have been assimilated, we think it is useful to explain here in detail how 4TEX manages to do that, what files it uses and how you can configure it.

Understanding the 'wiring' of 4TEX will help you locate spots where you may want to change 4TEX behavior or options. You will even be able to assimilate other programs into 4TEX.

12.1 Forward slashes and backslashes

In the next pages you will notice that we sometimes use forward slashes (/) and sometimes backslashes (\). So much for consistency. And it gets worse. Sometimes we put a forward slash or backslash at the end of a path, and sometimes we don't.

We will gladly admit it is rather confusing. The reason for this inconsistency is that MS-DOS and Windows traditionally use backslashes in paths whereas the Web2c TEX implementation reflects its Unix origin in using forward slashes. Furthermore, TEX uses backslashes in its commands. In some cases this interferes, too.

The backslash at the end of a path is the standard way that Windows returns path names to programs. It is also convenient for batch files that can state, e.g.:

```
set MAIN=%MAINPATH%%MAINFILE%%MAINEXT
```

It may seem irrelevant whether you specify a backslash in an environment variable or in the program code of a batch file, but there is a pitfall here. MS-DOS and Windows don't mind if you supply two (or more) backslashes in a path like this:

```
dir c:\windows\\\command
```

but if you enter

```
dir c:\\windows
```

it will *not* work! So you have to be careful with adding backslashes.

Unfortunately the crucial Windows environment variable `windir` (in lower case! why?) does *not* get a backslash at the end.

The syntax of the Web2c configuration files (see section 13.8) can be another source of confusion because forward slashes and backslashes have once more different meanings...

You can include other files within a TEX document. Note that if you need to specify a path in a TEX command such as `\input` you should *always use forward slashes*. A backslash would be interpreted as the start of a TEX command, which is not what you want.

The Web2c programs support the 'UNC' (Universal Naming Convention). Using this convention you can logically refer to files on any file server connected to your computer. Here are two examples:

```
TEX //texserver/users/dol/tex/test.tex
DVIPS -o//texserver/users/dosl/output.ps output.dvi
```

You can also use UNC names within files that Web2c programs read, e.g. in a TEX document:

```
\input //texserver/tex/generic/plain/story.tex
```

Note that again you have to use *forward* slashes. Also note that except for the Web2c programs, very few (if any) other programs used within 4TEX support the UNC.

All in all, the \ and / situation is a mess, and unfortunately we are not in a position to change that. We advise you to watch carefully how the original 4TEX ini file and batch files work and experiment if necessary. And don't say we didn't warn you...

12.2 Principles

4TEX is composed of a central executable (`4TEX.EXE`), a number of configuration files, batch files and more. We will describe all the parts here in detail. The syntax of the 4TEX program is:

≡ 4tex.exe [option] [mainfile [currentfile]]

Here is an explanation of the parameters you can supply:

–ini Start the program in initialization mode (all other parameters will be ignored). If the configuration file `4TEX.INI` is not found, 4TEX will automatically start in 'initialization' mode. If you think there is something wrong with the configuration you can force 'ini' mode with this option.

–inifile *inifile* Start the program using the *inifile* instead of the usual `4TEX.INI` file. This alternative initialization file should be specified with its complete path.

mainfile You can specify a file that 4TEX will treat as the 'Main file'.

currentfile If a main file is specified, you can additionally specify a 'Current file'.

Note that 4TEX will always run as 'one instance'. It means that if 4TEX is already running, you cannot start the program once more and have two copies running simultaneously. The reason for this is that communication with other programs (such as the editor or previewer) would be out of control.

You can use the common Windows 'drag and drop' method to start 4TEX. If you 'drop' a file on the 4TEX icon, the program will start, using the dropped file as 'Main file'. In case you *associated* `.tex` files with 4TEX (see section 3.1) you can simply click on a `.tex` file in Windows Explorer to start 4TEX.

The 4TEX environment consists of the following file types:

4TEX.EXE The main program.

4TEX.INI The file from which 4TEX reads many initial values. We will describe the variables in `4TEX.INI` in detail in section 12.3.

***.4mod** Modules that describe parts of the TEX system that can be installed separately. They are only used during installation of 4TEX. See section 12.5 for details.

***.btm and *.bat** Batch files that 4TEX calls to do more complex tasks. The shareware program 4DOS is used to run these tasks. Note that the 4allTEX CDROM comes with a pre-paid license for 4DOS. See appendix C for more details on software used within 4TEX.

***.lst** Files that describe selections that can be made within 4TEX. For example, what kind of printers you have, what programs can be called, and how they should be called. Note most of these `.lst` files are language dependent. All files starting with `US_` are used if you have selected 'English' as the interface language that 4TEX uses. Likewise the German versions start with `DE_` and the Dutch versions with `NL_`. The `.lst` files that don't start with a country (or rather language) code are used regardless of the chosen language. A detailed description of all `.lst` files is given in section 12.7.

***.scr** Files that 4TEX, 4Spell and 4Project use to switch the language of their user interface. In section 12.8 these files will be discussed.

***.4par** Files that 4TEX generates for each 'Main file' you use. Settings of language, format, print options, etc. are stored in these files. These files are not involved in installation or configuration of 4TEX. In section 12.9 we will describe them in more detail.

***.opt** Files that describe the options of printer drivers. We will describe them in section 12.10.

***.pap** Files that define paper sizes for printing. In section 12.11 we will describe them.

***.for** Files that contain information and TEX input for generating format files. See section 12.12 for details.

***.4spell** These files contain user defined settings for the program 4Spell, the spell-checker that comes with 4TEX. See section 12.13 for detail.

***.chm** These are the help files in HTML HELP format. This new format is the successor of the standard Windows help file format. In section 12.14 we will describe these files.

Note that, except for the `.chm` files, these are all ASCII text files that can be edited. The syntax of (most of) these files is quite trivial: one line is the text that 4TEX will display in a menu that the user can choose from; the next line is technical information that 4TEX needs. Leading spaces on the second line are ignored by 4TEX, so you can make the files easier to read for humans by indenting the second line, as illustrated below:

```
LaTeX 2e
   tex.exe &latex
plain TeX
   tex.exe
e-TeX
   etex.exe
e-LaTeX
   etex.exe &latex
pdfLaTeX
   pdftex.exe &pdflatex
```

If the second line of an item represents a program that will be started, you can specify either an executable (extension: `.exe` or `.com`), a standard batch file (extension: `.bat`) or a 4DOS enhanced batch file (extension: `.btm`). Both types of batch files will be handled by 4DOS. Documentation on the extended features of 4DOS batch files are available on (cdrom).

Some `.lst` files are even simpler than the example above, e.g. `us_prd.lst`:

```
Parallel port LPT1
  lpt1
Serial port COM1
  com1
Print to file
```

```
  file
Network printer X
  netprintX.bat
Network printer Y
  //server/printer
```

Note that the line following `Print to file` should contain the string `file` or be completely empty. If you choose to print to file, 4TEX will ask you to specify a file name.

If you want to send output to a network printer, you have two options. You can either specify the network printer as a UNC or you can specify a batch file that will execute the necessary redirection commands. Such a batch could look like this, if you are using a Novell 4.11 network:

```
:LPT2
@echo off
capture l=2 endcap
capture l=2 q=pq-c2 nff nt noti
```

Note that the first line *must* contain the actual printer port to which output can be sent. 4TEX executes this batch file immediately after you have selected the network printer from its menu. 4TEX remembers settings for each main file you use (in a `.4PAR` file), so if you reselect such a file, the batch file may be executed.

A `.pap` file describes the parameters needed to specify the paper size for specific DVI driver. Unfortunately different DVI drivers have different parameters to do this. Below we show `DVIPS.PAP` as an example:

```
A4 (210mm x 297mm)
  -t A4
A3 (297mm x 420mm)
  -t A3
legal (8.5in x 14in)
  -t legal
letter (85in x 14in)
  -t letter
ledger (17in x 11in)
  -t ledger
tabloid (11in x 17in)
  -t tabloid
```

In section 12.11 we will describe them in more detail.

12.3 4TEX.INI

Windows 95, 98 and NT, the Win32 platform for short, uses a *system registry* to store program settings. This has certain advantages and disadvantages over older methods

such as 'ini' files. We will not discuss these methods in detail here, but simply state that we feel that for 4TEX the 'ini' file method is much more suited than the registry method.

The ini file `4TEX.INI` is automatically created by the 4TEX installation program `setup.exe` (see section 3.1). 4TEX will search for `4TEX.INI` in the Windows directory, or, if not found, in the same directory as where the 4TEX program resides. Every time you start 4TEX the file `4TEX.INI` file is read and all the settings in this file are used. Every time you quit 4TEX, the file will be updated to reflect the settings used at that moment.

`4TEX.INI` contains the following sections:

[System] This part contains all the *system* settings. A user normally doesn't need to update these program settings.

[UserChoice] This part contains all the *User Choices*. These settings are used to restore all user settings (TEX file names, printer choice, language, etc.) of the last 4TEX session.

[Options] Some extra options.

[Screens] In this section 4TEX stores scaling factors of all screens. By default all variables in this section are set to 1.

[Editor:xxxxxxx_yy] For each editing program that 4TEX support `4TEX.INI` will contain a section. The string 'xxxxxxx' represents the name of the editor, e.g., 'Med', 'WinEdt' or 'PFE'. The string 'yy' represents the language of the editor. In case there are language specific versions of the editor available, they will have their own section. The editor MED, e.g., is available in an English and a German version. In that case there should be a section `[Editor:MED_US]` and `[Editor:MED_DE]`. If there is only one version, you should always specify `_US`.

[4Spell] This section contains all settings that the spell-checher 4Spell needs.

[4Project:xxxxx] In these sections the program 4Project stores its settings for dealing with 'xxxxx' documents, where 'xxxxx' can be e.g., 'LaTeX', or 'ConTeXt'.

[Environment] This section contains specifications for environment variables that will be set when 4TEX starts. These settings are local to the running 4TEX process and they are passed on to any program launched from 4TEX.

Below is a description of all the variables in `4TEX.INI`:

[System]

Web2cDir Specifies the root directory of the Web2c TEX implementation This could be the root directory of the 4TEX CDROM (e.g. `d:\`) or wherever you installed everything.

Web2cBINDir Specifies the directory in which all Web2c program (e.g. `tex.exe`) are stored. On the CDROM this would be `d:\bin\win32\`

MainExtention Specifies the default file extension of a 'Main file'. Do *not* specify a dot in the extension. `TEX` is the default.

CurrentExtention Specifies the default file extension of the 'Current file'.

4DosProgram The 4DOS batch file interpreter that can be used instead of `COMMAND.COM` (on Windows 95/98) or `CMD.EXE` (on Windows NT). This could be, e.g. `d:\4tex\4dos\4dosrt.com` or `d:\4tex\4dos\4ntrt.exe` on the 4TEX CDROM. Please read carefully the notes on 4DOS in section 12.6.

4DosOptions 4DOS settings used when executing 4DOS batch files.

4TeXBTMDir The directory where 4TEX BTM files are stored. On the CDROM this would be `d:\4tex\`

4TeXSelectionsOptionsDir The directory where `.lst` files can be found. On the CDROM it would be `d:\4tex\`

SpellChecker The spell-checker program (including parameters) that 4TEX will use. By default, 4TEX will use 4Spell, that will be executed like this:
`&4TeXBTMDir4Spell.exe &spellfile -L=&lang -SL=&4TeXLanguage -INIFILE=&inifile -R`
4TEX will replace all strings starting with '&' with their real value. `&spellfile` will be the 'Main file' or 'Current file', if any. `&lang` will be the currently selected spelling language (e.g, US, DE or PL). `&4TeXLanguage` will be the currently selected 4TEX menus language (e.g., SK, RU or NL). `&inifile` will be the ini file that 4TEX is currently using. This will usually be `4TEX.INI`.

LaTeXMac Path and file name of LATEXmac program (see section 8.13). On the CDROM this would be `d:\bin\win32\latexmac.exe`.

GSView Path and file name of GSview program (see section 7.5). On the CDROM this would be `d:\bin\win32\gsview32.exe`.

ShowINIErrors Whenever 4TEX is unable to write new settings to `4TEX.INI` it can issue an error message. If this variable is set to 1 (the default) it will do so. If set to 0, it will not.

[UserChoice]

4TeXLanguage Specifies the language of the 4TEX menus. Currently US (English), DE (German), NL (Dutch), PL (Polish), FR (French), RU (Russian), IT (Italian), ES (Spanish), SE (Swedish), DK (Danish), CZ (Czech) and SK (Slovak) are supported.

OwnTeXTreeDir Path of the user TDS tree (where the user stores his/her TEX files, macros, style files, fonts, etc.).

MainFileName The 'Main file' (path + file name).

CurrentFileName The 'Current file'.

Format Description of the currently used TEX format.

FormatName The TEX compiler program.

Viewer The DVI previewer (program).

ViewerTXT Description of the DVI previewer.

Printer The printer program.

PrinterOptions Printer options.

PrinterPort Print destination.

PrinterTXT Description of the printer.

PrinterPortTXT Description of the print destination.

BitmapViewer The program to be used for displayed bitmap graphics.

GraphicsConverter The program to be used to convert graphics into other formats.

BibTeXFile The BIBTEX database file.

BibTeXEditor The BIBTEX editor program.

BibTeXEditorTXT Description of the BIBTEX editor.

SpellLanguage Language used for spell-checking.

SpellLanguageText Description of the language used for spell-checking.

KillProcesses A semicolon separated list of programs (or rather, the strings that identify them in the Windows task list) that you want 4TEX to 'kill' when it exits.

BlockMarker Used to indicate the end of the TEX document preamble (used for block compilation, see section 6.5).

EndBlockMarker Used to indicate the end of the TEX document (used for block compilation, see section 6.5).

ManualPageDir Directory where manual pages of programs can be found.

HelpFilesDir Directory where to find Windows help files of programs.

CleanupFiles A semicolon separated list of file specifications that you want 4TEX to display when you select 'Clean up files' from the Utilities menu.

EditorSection A pointer to another section in `4TEX.INI` that will specify all editor program related variables.

DelFiles A semicolon separated list of files that will be deleted when you quit 4TEX.

ShowOnTaskTray 0 = do not use Tasktray; 1 = use Tasktray.

ShowOnTaskBar 0 = do not use Taskbar; 1 = use Taskbar.

RunDosWindow 0 = run real DOS-box (default for Windows NT); 1 = Redirect the DOS-box output to a Window screen.

CompileUseBTM 0 = use 4TEX internal compiler call; 1 = run the batch file `COMPILER.BTM`.

QuitUseBTM 0 = normal quitting procedure; 1 = run the batch file `CLOSING.BTM`.

ScreenFont Specifies the Windows font to be used in 4TEX menus. Recommended choices are 'Tahoma' or 'Arial'.

RedirectFont Specifies the font to be used in the window that displays compiler messages, BIBTEX messages, etc. 'Courier New' is a good choice.

ScreenFontSize Specifies the size of the font used in 4TEX menus. 8 is a good choice.

SpeedbarFontSize Specifies the size of the font used in the text version of the 4TEX speedbar. 8 is a good choice.

RedirectFontSize Specifies the size of the font used in the window that displays compiler messages, BIBTEX messages, etc. 8 is a good choice.

SpeedbarImages Specifies whether you want a speedbar with images (1) or with text buttons (0).

UserSpeedbarPosition Specifies whether the speedbar should be positioned at used specified coordinates (1) or at a fixed position (0).

AlwaysShowRedirectScreen 0 = only display the console window when it is being used; 1 = always show display it.

[Options]

XScreenPos The X-position of the upper left corner of the 4TEX menu (in pixels).

YScreenPos The Y-position of the upper left corner of the 4TEX menu (in pixels).

ScreenWidth The width of the 4TEX menu (in pixels).

ScreenHeight The height of the 4TEX menu (in pixels).

XScreenPosSpeedbar The X-position if the speedbar.

YScreenPosSpeedbar The Y-position if the speedbar.

RedirectXScreenPos The X-position if the console window.

RedirectYScreenPos The Y-position if the console window.

RedirectScreenWidth The width of the console window.

RedirectScreenHeight The height of the console window.

StayOnTop 0 = menu doesn't stay on top; 1 = stay on top.

Menubar 0 = full-size 4TEX menu; 1 = use speedbar.

Verticalbar 0 = do not use vertical speedbar; 1 = use vertical speedbar

SpeedbarLeft 0 = put speedbar to the right of the screen; 1 = put speedbar to the left of the screen.

SpeedbarTop 0 = put speedbar on the bottom of the screen; 1 = put speedbar on the top of the screen.

AlwaysDevirtualize 0 = do not automatically replace virtual fonts in DVI files by real fonts ('devirtualize'); 1 = automatically devirtualize DVI file after each compilation.

RunAutostartBatchfile 0 = do not run `AUTOSTART.BTM` when 4TEX starts; 1 = run `AUTOSTART.BTM`.

MainHistory0 History list of the last 10 'Main' files. Likewise there are: **MainHistory1**, **MainHistory2**, **MainHistory3**, **MainHistory4**, **MainHistory5**, **MainHistory6**, **MainHistory7**, **MainHistory8**, **MainHistory9**.

CurrentHistory0 History list of the last 10 'Current' files. Likewise there are: **CurrentHistory1**, **CurrentHistory2**, **CurrentHistory3**, **CurrentHistory4**, **CurrentHistory5**, **CurrentHistory6**, **CurrentHistory7**, **CurrentHistory8**, **CurrentHistory9**.

Any `[Editor:xxx]` section should contain the following variables:

Editor The editor program. E.g., `d:\bin\win32\pfe\pfe32.exe`

EditorClassName The class name that identifies the editor program for DDE purposes. E.g., `PFE32_Frame`

TeXEdit The editor program that should be called when you enter 'e' at the TEX prompt when an error has occurred. Note that you should specify the file name and a parameter to jump to the erroneous line. The string `%s` represents the file name, and `%g` represents the line number. For PFE the command should be:
`D:\BIN\WIN32\PFE\PFE32.EXE /g%d "%s"`

KillEditorProcesses The string that identifies the editor program in the Windows task list, so 4TEX can attempt to kill it when you quit 4TEX. For PFE this would be `PFE32_Frame`

LogFileViewer The program to be used to display log files. the same as specified in **[Editor]**. Usually this is the same as **Editor**.

AutoSave 4TEX can implement an 'autosave' function for editors that don't support such a feature (e.g., PFE). You can set a timer (in minutes) that will trigger 4TEX to send a DDE command to save the current file.

DDE... Several DDE commands are used for communications between 4TEX and the editor. See section 12.4 for details.

StartEditorWithProgram The program (if any) that is started every time the editor is called (default: LaTeX Mac, see section 8.13).

[Environment]

This section of `4TEX.INI` can be used to specify environment variables that 4TEX or any of the programs (executable or batch files) it launches may need.

These settings overrule any existing global setting, but only for the current 4TEX process. No other processes are affected.

Note that by specifying a variable with an empty value, you effectively delete the variable from the current environment. This method can be useful to resolve conflicting variable settings. Below we have listed an example of the [Environment] section:

```
[Environment]
CLS=
DELFILES=*.bak /s
COMSPEC=D:\4TEX\4TEX\4DOS\4DOSRT.COM
4DEBUG=
WEBROOTDIR=D:\4TEX\
4TEXPATH=D:\4TEX\4TEX\
TEXMFCNF=C:/4TEX5.0/TEXMF/WEB2C;C:/4TEX/TEXMF/WEB2C
DELEGATE_PATH=D:\4TEX\BIN\WIN32\
UTILSDIR=D:\4TEX\BIN\WIN32\
KILL=Juan.PFE32.dviwin.gsview
PERLLIB=D:\4TEX\BIN\WIN32\PERL\LIB
GS_PATH=D:\4TEX\BIN\WIN32\GSWIN32C.EXE
GS_LIB=D:\4TEX\BIN\WIN32\
```

Note that there is no `PATH` variable specified here. 4TEX will construct this variable automatically by concatenating the `Web2cBINDir` value, the `4TeXBTMDir` value (from the [System] section in `4TEX.INI`), and the global `PATH` value. However, if you do specify the `PATH` variable in the [Environment] section, your setting will overrule 4TEX's own setting.

4TEX will dynamically set a few other environment variables, most notably `TEXEDIT`, `MPEDIT` and `MFEDIT`. The values of these variables depend on the editor program you have chosen. If you select another editor program from within 4TEX the value of these variables will be changed automatically. Therefore it is not a good idea to specify them in the [Environment] section.

12.4 Dynamic Data Exchange

Windows provides a standardized interface for inter-program communication, called Dynamic Data Exchange or DDE. If your editor programs supports DDE, (as PFE, MED

and WINEDT do), 4TEX can communicate with it to automate a few functions. In the file 4TEX.INI all DDE variables are listed to implement these functions. The list below describes all commands that 4TEX uses to communicate through DDE:

1. Test whether the current file you are editing has been changed since the last 'Save file' operation. The variable is called: DDEEditChanged
2. If the file has changed, you want to save the current file: DDEEditSaveCommand
3. Bring the editor and file into focus (come to the foreground): DDEEditForeground
4. If performing a block compilation, 4TEX will:
   (a) Remember the file name of the selected text.
      i. This is done by the DDEBlockFileName command if 4TEX is able to ask for the file name. In other words, if DDEBlockFileNameMode is available. E.g., for PFEthe line number is given by the command DDEBlockFileName=FileName
      ii. If the file name can not be obtained (DDEBlockFileNameMode=Execute) then the DDEBlockFileName command will invoke the editor to send a DDE command to 4TEX in which the file name is stored. E.g., for WINEDT the command will be:
      DDEBlockFileName=DDEOpen("4TeX 5.0","","forall");
      DDEExe("%%FILE:%f");DDEClose
      The result of this macro will be that, for instance, the string %%FILE:c:\texfiles\latexsample.tex is being sent to 4TEX. It indicates that the file name is c:\texfiles\latexsample.tex.
   (b) Remove the block from your file: DDEBlockCutCommand=
   (c) Remember the line number at the position of the cursor. This will always be the first line of the block.
      i. This is done by the DDEBlockLineNumber command if 4TEX can obtain the current line number. In other words, if DDEBlockLineNumberMode=Request. E.g., for PFE the line number is given by the DDE command DDEBlockLineNumber=LineNumber
      ii. If 4TEX can not obtain the line number (DDEBlockLineNumberMode=Execute) then the command DDEBlockLineNumber will instruct the editor to send a DDE command to 4TEX in which the line number is stored. E.g., for WINEDT the command will be:
      DDEBlockFileName=DDEOpen("4TeX 5.0","","forall");
      DDEExe("%%LINE:%l");DDEClose
      The result of this macro will be that, e.g., the string %%LINE:122 is being sent to 4TEX. It indicates that the line number is 122.

(d) Copy the block from the clipboard back to the file, restoring the old situation: `DDEBlockPasteCommand`

5. When quitting 4TEX several programs started by 4TEX may be terminated (see section 8.11) You could add the editor to that list, but if the editor supports a DDE command to terminate then this command should be used. The `DDEEditQuit=` variable specifies the command to quit the editor.

DDE commands are always sent after the connection between 4TEX and the editor is made. This connection is established by sending a `DDE Service` and a `DDE Topic` message to the editor. For block compilation these are:
`DDEBlockService=PFE32`
`DDEBlockTopic=Editor`
For other editing functions these are:
`DDEEditService=PFE32`
`DDEEditTopic=Editor`

Currently 4TEX is known to communicate well through DDE with the editors PFE, MED and WINEDT. However, given the description above, it should be possible to make 4TEX communicate with other editors as well. Below we will list the settings needed for the three editor we just mentioned. Note that 4TEX will automatically generate these settings in `4TEX.INI`. From the 'Advanced options' menu (see section 8.11) you can simply select one of these editors.

12.4.1 PFE

All DDE commands needed for the editor PFE:

```
DDEEditService=PFE32
DDEEditTopic=Editor
DDEEditCommand=[FileSave()]
DDEEditChanged=FileChanged
DDEEditQuit=
DDEEditSaveCommand=[FileSave()]
DDEEditForeground=[ComeToForeground()]
DDEBlockService=PFE32
DDEBlockTopic=Editor
DDEBlockFileName=FileName
DDEBlockFilenameMode=Request
DDEBlockLineNumber=LineNumber
DDEBlockLineNumberMode=Request
DDEBlockCutCommand=[EditCut()]
DDEBlockPasteCommand=[EditPaste()]
```

12.4.2 MED

All DDE commands needed for the editor MED:

```
DDEEditService=med
DDEEditTopic=system
DDEEditQuit=FileSaveAndExit()
DDEEditSaveCommand=FileSaveAll()
DDEEditChanged=
DDEEditForeground=AppWinActivate()
DDEBlockService=med
DDEBlockTopic=system
DDEBlockFilenameMode=Request
DDEBlockFileName=ReqFileName
DDEBlockLineNumberMode=Request
DDEBlockLineNumber=ReqLineNumber
DDEBlockCutCommand=EditCutToClipbrd()
DDEBlockPasteCommand=EditPasteClipbrdLine()
```

12.4.3 WINEDT

All DDE commands needed for the editor WINEDT:

```
DDEEditService=WinEdt
DDEEditTopic=DDEServer
DDEEditQuit=[CMD("Exit")]
DDEEditSaveCommand=[CMD("Save All")]
DDEEditChanged=
DDEEditForeground=[SetFocus("")]
DDEBlockService=WinEdt
DDEBlockTopic=DDEServer
DDEBlockFilenameMode=Execute
DDEBlockFileName=DDEOpen("4TeX 5.0","","forall");
   DDEExe("%%FILE:%f");DDEClose
DDEBlocklinenumberMode=Execute
DDEBlockLineNumber=DDEOpen("4TeX 5.0","","forall");
   DDEExe("%%LINE:%l");DDEClose
DDEBlockCutCommand=[CMD("Erase Clipboard");CMD("Cut")]
DDEBlockPasteCommand=[CMD("Paste")]
```

Note that the lines 10 and 11 are should be typed in as *one* line, without any spaces inbetween. The same goes for lines 13 and 14.

12.4.4 4Spell

4TEX can also communicate with the spell-checker 4Spell. If you start 4Spell, it will run as a process concurrent to 4TEX. Therefore, you could decide to compile a TEX file in

4TEX while you are spell-checking it at the same time. Though we think this is not a very clever idea, we can't (or rather, won't) forbid that. However, in such a case 4TEX will use DDE to instruct 4Spell to save the file to disk before 4TEX will start to compile it.

12.5 The .4mod files

During installation you have the option to install parts (or all) of the 4allTEX CDROM on your hard disk. These parts are defined as 'modules' that have the file name extension `.4mod`. Below is a an example that displays all features supported in `.4mod` files:

```
%Title: Install the whole CD-rom
%Size:  500 MB
%Type:  Optional
%Info:  Install **everything**
        This includes not olny the TeX executables
        and resources, but also extra utilities
        and electronic documentation

%Filespecs:
        tools\* /s
        requires 4TEX
        requires EXAMPLES
        requires BINARIES
        requires TEXMFALL
        require  PSTOOLS
        require  TOOLS
        require  ELECTRODOCS
```

Comment lines always start with %. They can be as many comment lines as you like, but note that some comment lines starting with the following keywords have a special meaning:

%Title: After this keyword a short description on one line is expected. This line will be displayed by the installation program.

%Size: After this keyword a rough estimation of the total size of all files involved in this module should appear. The words 'MB' and 'KB' can be used to denote 'megabytes' and 'kilobytes', respectively. Otherwise, 'bytes' is assumed.

%Type: Type should be either `Required`, `Recommended` or `Optional`. The installation program will display them in separate columns (see figure 3.3).

%Info: A more detailed description of the module. To make displaying by the installation program easier it is recommended to keep the line length below 40 characters.

%Filespecs: Here the actual files that need to be copied are specified. Individual files can be specified and wild cards can be used. The optional parameter `/s` after any file specification means that it also applies to all subdirectories. A module can also '`require`' other modules recursively.

12.6 Batch files

From the 4TEX menu *batch files* may be started to accomplish certain tasks, such as printing a DVI file or converting a file from one type into another. These batch files set up the necessary environment for the program(s) that will run subsequently.

In 4TEX we prefer to use 4DOS batch files. 4DOS is a command line interpreter, just like `COMMAND.COM` (DOS/Windows 95/98) or `CMD.EXE` (Windows NT), but it is much more powerful and flexible. The 4allTEX CDROM comes with a pre-paid license for running 4DOS batch files from within 4TEX.

4DOS is a shareware program written by R. Conn and JP Software Inc. that is available as a MS-DOS/Windows 95/98 version (`4DOSRT.COM`) and as a Windows NT version (`4NTRT.EXE`). The versions on the CDROM are *run-time* systems: they can only be used to execute batch files, not as command line interpreters.

4DOS batch files can have the file extension `.btm` or `.bat`. BTM stands for *batch to memory*, which means that such a batch file is read completely into memory before execution, unlike 'normal' batch files that are read and executed line by line. The advantage is better performance. Furthermore, 4DOS offers a lot of commands and functions to perform more complex tasks. 4DOS makes it easy to do fairly sophisticated programming for a very low price. If you care to dig a little deeper in 4DOS you will find that it can handle tasks that would otherwise require (expensive) programming tools. Programming languages such as Perl, REXX or TCL/Tk could be good (or better) alternatives but 4DOS has the advantage of simplicity and seamless integration with 'standard' batch files.

Just by reading the 4DOS help file and studying the examples you should soon be able to change/add BTM files to suit your own needs.

If you leave the value of the variable `4DosProgram` in `4TEX.INI` empty, 4TEX will use the program specified by the `COMSPEC` environment variable. On Windows 95/98 that would usually be `COMMAND.COM`; on Windows NT it would be `CMD.EXE`. If you don't use 4DOS then 4TEX will not start the BTM files as described below (since these may not work), but 4TEX will start a batch file with the same name but with the extension `.bat`. So, if you prefer not to use 4DOS you can write your own `.bat` files. Note that all `.btm` files that come with 4TEX contain code that will *only* work with 4DOS. All `.bat` files are standard batch files that do not require 4DOS to be processed.

Below you will find a list and short description of BTM files that 4TEX uses:

4NEWS.BTM A batch file that starts your World Wide Web browser to display the 'news' page from the 4TEX WWW site.

AUTOSTART.BTM This batch file is called every time you start 4TEX and `4TEX.INI` states `RunAutoStartBatchFile=1`.

BIBTEX.BTM This batch file is used when you click on Run BibTeX in the BibTEX menu (see section 8.1).

BLCKCOMP.BTM Used for 'block compilation'. If you select a block in the editor and then right-click Compile this batch file is run.

BLKERROR.BTM This batch file is run in case an error occurred during block compilation. It is used to jump to the line in the file that TEX stumbled on.

CLOSING.BTM This batch file is run when exiting 4TEX if you specified `QuitUseBTM=1` in `4TEX.INI`.

COMPILER.BTM The batch file that is run when you click on Compile if you specified `CompileUseBTM=1` in `4TEX.INI`.

CONTEXT.BTM A batch file for compiling a CONTEXT file and running TEXUTIL in one go.

DEBUG.BTM This batch file contains a few commands that are executed to obtain information on the current status of the TEX system. This batch file is called by the 'Debug' feature described in section 6.12.

DEVIRT.BTM If the 'Replace virtual fonts' option is selected (see section 9). this batch file will perform that task.

LATEXWIZARD.BTM Batch file that will start the LATEX Wizard program (see section 8.18).

MAKEINDX.BTM Batch file to run the MakeIndex program.

MAKELS-R.BTM This batch file will (re)generate the `ls-R` index file(s) that the Web2c program need to find all TEX related resources.

METAFONT.BTM Batch file that is run when you click on Run Metafont in the Metafont menu.

METAPOST.BTM Batch file that is run when you click on Run MetaPost in the MetaPost menu.

MFORMAT.BTM Batch file that is run when you click on Generate in the 'Generate (LA)TEX format' menu.

WORDLIST.BTM Batch file that is run when you click on Generate word list in the MakeIndex menu.

The following batch files are used for generating output on the screen or on a printer:

BJ10E.BTM Canon Bubblejet 10e (360 dpi) through Ghostscript.

CDJ550.BTM HP Deskjet 550c color (300 dpi) through Ghostscript.

CDJCOLOR.BTM HP Deskjet 500c color (300 dpi) through Ghostscript.

CDJMONO.BTM HP Deskjet 500c monochrome (300 dpi) through Ghostscript.

DESKJET.BTM HP Deskjet (300 dpi) through Ghostscript.

DVI2TTYP.BAT Output to printer in ASCII format.

DVI2TTYV.BAT Output to viewer (= editor program) in ASCII format.

EPSON.BTM Epson FX matrix printer (240x216 dpi) through Ghostscript.

GSVIEW.BTM Batch file to view a PostScript or PDF file.

GSPRINT.BTM Batch file to convert a DVI file to PostScript or PDF file.

LJET2P.BTM HP Laserjet IIp (300 dpi) through Ghostscript.

LJET3.BTM HP Laserjet III (300 dpi) through Ghostscript.

LJET4.BTM HP Laserjet 4, 5 or 6 (600 dpi) through Ghostscript.

LJETPLUS.BTM HP Laserjet Plus (300 dpi) through Ghostscript.

MSWINPR2.BTM Windows printer through Ghostscript. Choose this option if your printer is not specified in the list.

PDFVGA.BTM Adobe PDF through Ghostscript (110 dpi/VGA screen).

PDFWLJ4.BTM Adobe PDF through Ghostscript (600 dpi).

PDFWRITE.BTM Adobe PDF through Ghostscript (300 dpi).

PJXL300.BTM HP Paintjet XL color (300 dpi) through Ghostscript.

PS2PDF.BTM PostScript to PDF conversion.

In the subdirectory `\4TEX\CONVERT` you will several more batch files. Each of them takes care of a specific conversion. See section 8.7 for details on file type conversions.

The special batch file `4TEXSET.BAT` is generated by 4TEX on the fly. This batch file is generated in the current directory and it contains commands to set up environment variables for other batch files that may need these. Any batch file can simple “call” this batch file. Below is an example of this batch file.

```
set TEMP=D:\TEMP
set TMP=D:\TEMP
set CLS=
set DELFILES=*.bak /s
set COMSPEC=E:\4TEX\4DOS\4NTRT.EXE
set 4DEBUG=
set MODE=ljfour
set WEBROOTDIR=E:\
set 4TEXPATH=E:\4TEX\
set TEXMFCNF=C:/4TEX5.0/TEXMF/WEB2C;E:/TEXMF/WEB2C
```

```
set DELEGATE_PATH=E:\BIN\WIN32\
set UTILSDIR=E:\BIN\WIN32\
set KILL=Juan.PFE32.dviwin.gsview
set PERLLIB=E:\BIN\WIN32\PERL\LIB
set GS_PATH=E:\BIN\WIN32\GSWIN32C.EXE
set GS_LIB=E:\BIN\WIN32\
set PATH=E:\BIN\WIN32;C:\WINNT\system32;C:\WINNT;d:\utils;
set TEXEDIT=E:\bin\win32\MED\med.exe %s %d
```

In 4TEX several conversion tools use this batch file. 4TEX removes this batch file each time you select a main TEX file in another directory, and when you quit 4TEX.

12.7 The .lst files

Part of 4TEX's functionality is defined in user customizable files with file extension `.lst`. These `.lst` files can be found in the same directory as the 4TEX program. Most of these files are language specific. The first two characters of the file name indicate the language in which they are written. For instance, the file that contains the list of all TEX format files for the US English language is called `US_FRM.LST`. The Dutch version is called `NL_FRM.LST`, the German version is called `DE_FRM.LST`, etc. There are also a few language-independent .LST files: `HYPHEN.LST`, `MAGSTEPS.LST`, `OWNTREE.LST`, `COMPILER.LST` and `GRAPHTYP.LST`.

All `.lst` files are plain ASCII files and most of them have the same structure. The odd lines of the file contain a description that is used when displaying a selection menu. The even lines are the actual commands that are executed, either directly or though a batch file. For instance the files `US_SPELL.LST` (on the left) and `NL_SPELL.LST` on the right) may look like this:

```
Dutch                          Nederlands
  nl                             nl
English (American)             Engels (Amerikaans)
  us                             us
English (British)              Engels (Brits)
  uk                             uk
Danish                         Deens
  dk                             dk
French                         Frans
  fr                             fr
German                         Duits
  de                             de
Italian                        Italiaans
  it                             it
Spanish                        Spaans
  es                             es
Swedish                        Zweeds
  se                             se
```

As you can see only the odd lines differ; the even lines are equal. All `.lst` files are set up like this, e.g. `DE_FRM.LST` (on the left) and `PL_FRM.LST` (on the right).

```
LaTeX 2e
  TEX.EXE -progname=latex "&latex"
plain TeX
  TEX.EXE
e-TeX
  ETEX.EXE
PDFTeX
  PDFTEX.EXE
PDFLaTeX
  PDFLATEX.EXE
ConTeXt (English)
  &4DOS context.btm en
ConTeXt (Dutch)
  &4DOS context.btm nl
```

```
LaTeX 2e
  TEX.EXE -progname=latex "&latex"
plain TeX
  TEX.EXE
e-TeX
  ETEX.EXE
PDFTeX
  PDFTEX.EXE
PDFLaTeX
  PDFLATEX.EXE
ConTeXt (angelski)
  &4DOS context.btm en
ConTeXt (niderlandski)
  &4DOS context.btm nl
```

Note that the even lines contain an indentation that is not necessary. It is only used to make the difference between odd and even lines more visible.

Odd lines specify a program with or without parameters and with or without a full path name. Since the executables of the Web2c distribution are all stored in the same directory this directory is added to the `PATH` environment variable during installation. Thus, it is not necessary to add a full path name if you want to start a program from that directory. Programs from the Windows system directory (such as Notepad) can also be specified without a path because the system directory is always part of the `PATH` environment variable.

You can easily add your own programs to a `.lst` file by adding two lines, one line to describe its purpose and one to specify the program. You can also reorder a `.lst` file so that the items you use most often are displayed first.

When you choose another language in which 4TEX shall 'speak' to you it will attempt to restore your current settings using the `.lst` files of the newly chosen language. If the current setting is not available, the top-most will be used. In case an `.lst` file for a specific language is not found at all, the `US` version will be used. Therefore it is wise never to delete the files `US_*.LST`.

Below you will find a list and explanation of all the `.lst` files that 4TEX uses. Some of these have extra functionality or options that will be explained. Of course 'XX' in this list indicates the actual language.

XX_BIBED.LST A list of all the editors that can be used for BibTEX files. There is one special option in this file: instead of typing the program name of your editing program you can use `&editor` and 4TEX will use the editor that is specified in `4TEX.INI` as the BibTEX editing program.

XX_FRM.LST A list of all the compilers and (LA)TEX formats that can be used with 4TEX. Executables or batch files are allowed. In case you want to use 4DOS batch files and you want them to run in a console window controlled by 4TEX (as opposed to a standard DOS-box), you should use the parameter `&4DOS` as in the example `US_FRM.LST` above,

XX_INTRO.LST This file contains the text that is displayed during the 4TEX initialization and hence it doesn't follow the convention of odd and even lines.

XX_PRD.LST A list of all the possible print destinations. Note that there is a special printer destination called 'Print to File'. This option needs as program name the string `file` or a blank line. 4TEX will start a file dialog menu from which you can specify the output file. A print destination can also be specified as a batch file. In that case the first line of the batch file should be a comment listing the device name that output should be sent to. After that arbitrary commands may follow to set up the this device.

XX_PRN.LST A list of all available printer types. Note that using the same printer program with other parameters is a way of specifying another printer type.

XX_SPELL.LST A list of all available languages for spell-checking. In 4TEX we use 4Spell for spell-checking. The odd lines of `XX_SPELL.LST` are used as a parameter for 4Spell. In case you want to use another spell-checker program, you can change the value of `SpellChecker` in `4TEX.INI`.

XX_UTILS.LST This will probably be the first `.lst` file you want to change. It contains many 4TEX screens: these are all specified by a program name with an `&` in front of it. For instance `&bibtex` will start the 4TEX BIBTEX menu. The string `&web2cbin` as part of a program path will be substituted by the path to the directory where the Web2c executables are stored. This feature makes it possible to specify programs without hard-coding the path name. Since you may want to start utilities with the 'Main file' name (or part of the name) or the 'Current file' name the following aliases can be used:

`&mainpath`: the path name of the 'Main file'
`&mainfile`: the file name (without file extension) of the 'Main file'
`&mainext`: the file name extension of the 'Main file'
`&main`: the full 'Main file' name (path + file + extension)
`&currentpath`: the path name of the 'Current file'
`&currentfile`: the file name (without extension) of the 'Current file'
`&currentext`: the file name extension of the 'Current file'
`&current`: the full 'Current file' name (path + file + extension)

So, if for instance your 'Main file' is `c:\texfiles\sample.tex` then the string `&mainpath&mainfile.dvi` will expand to `c:\texfiles\sample.dvi`. This makes it possible to dynamically pass parameters to programs or batch files.

XX_VIEW.LST A list of available DVI previewers. Note that the PostScript previewer and PDF previewer use the same 4DOS batch file with different parameters.

COMPILER.LST This file contains a list of compiler programs that can be used for generating TEX format files (see section 8.5). For each supported compiler the compiler executable name (without extension) is given. On the next line the file name *extension* of the generated format file is given.

HYPHEN.LST This file is not language-specific but nevertheless has the same structure of odd and even lines as described above. This file is used for generating new (LA)TEX formats. It contains a list of all available hyphenation files that can be used when generating a TEX format. Note that the 'comment' line can contain any text you like, but if you intend to use LATEX with the Babel package to switch languages it is essential to make the *first* word of the comment correspond to a Babel language definition file. See section 10.3 for details on managing multi-lingual documents.

MAGSTEPS.LST This file is not language-specific either. It contains a list of standard magnifications for printed output.

GRAPHTYP.LST This file contains a list of file extensions, graphics file type descriptions and graphics viewer programs that is used in the graphics conversion menu.

OWNTREE.LST This file is only used in the initialization phase. It lists all subdirectories that 4TEX should make in the local tree.

Note that if you specify a batch file in a `.lst` file, you can specify it with or without its full path. If you only specify the file name 4TEX will assume that the batch file can be found in the directory specified by the parameter `4TeXBTMDir` in `4TEX.INI`. If you specify a program (with file name extension `.exe` or `.com`) that program should be somewhere on the `PATH`.

It is also possible to specify files that have file name extension other than `.exe`, `.com`, `.bat` or `.btm`. This is actually a standard Windows feature. For instance, specifying `LaTeX2e.hlp` as a 'program' will automatically start the Windows help system displaying this help file. This trick should work with any file type that is *associated* with a specific program by Windows.

12.8 The .scr files

4TEX, 4Spell and 4Project are able to switch to a different language of their user interface. All language-dependent strings are stored in files that are named after a language (`nl` for Dutch, `de` for German, `cz` for Czech, etc.), and their file name extension is `.scr`. These are ASCII files that contain one string per line. The first line identifies the name of the language and the character set (code page) that should be used. The two item should be separated by a + sign. Valid character sets are: `WESTERN`, `EASTEUROPE` and `RUSSIAN`. Instead of these strings you can also specify the character set as a number. Table 12.1 lists the character sets that are supported. The fist lines of `NL.SCR` look like this:

Table 12.1: Character sets

Character set	Number	Description
ANSI_CHARSET	0	ANSI characters.
DEFAULT_CHARSET	1	Font is chosen based solely on Name and Size. If the described font is not available on the system, Windows will substitute another font.
SYMBOL_CHARSET	2	Standard symbol set.
MAC_CHARSET	77	Macintosh characters.
SHIFTJIS_CHARSET	128	Japanese shift-jis characters.
HANGEUL_CHARSET	129	Korean characters (Wansung).
JOHAB_CHARSET	130	Korean characters (Johab).
GB2312_CHARSET	134	Simplified Chinese characters (mainland China).
CHINESEBIG5_CHARSET	136	Traditional Chinese characters (Taiwanese).
GREEK_CHARSET	161	Greek characters.
TURKISH_CHARSET	162	Turkish characters.
VIETNAMESE_CHARSET	163	Vietnamese characters.
HEBREW_CHARSET	177	Hebrew characters.
ARABIC_CHARSET	178	Arabic characters.
BALTIC_CHARSET	186	Baltic characters.
RUSSIAN_CHARSET	204	Cyrillic characters.
THAI_CHARSET	222	Thai characters.
EASTEUROPE_CHARSET	238	Includes diacritical marks for eastern European countries.
OEM_CHARSET	255	Depends on the code page of the operating system.

```
Nederlands + WESTERN
4TeX 5.0
Info
Hoofdbestand
Kies TeX hoofdbestand
Bewerk
Schrijf TeX bestand(en)
Compileer
Compileer
Bekijk
Bekijk hoofdbestand
Afdrukken
```

Note that the strings from 4TEX, 4Spell and 4Project are all combined in one file. You should not change the order of the lines or delete any lines, since the programs expect to find the strings at specific lines.

You can edit these strings while e.g. 4TEX is running. Each time you switch to another language the strings will be read again, so you can see the results of any change you made almost immediately. Naturally you can also add new languages. We recommend that you copy `US.SCR` to a new name and then start translating, while checking the results within the program(s).

12.9 The .4par files

For each 'Main file' that you use some relevant settings are stored so each time you select that TEX file these settings can be restored. The advantage of this approach is that you don't have to remember the TEX format you used, the language to be used for spellchecking, the previewer, printer, etc. This is done be writing a `.4par` file. If the 'Main file' is, say, `c:\texfiles\somebook.tex` then the corresponding `.4par` file would be `c:\texfiles\4texdoc.4PAR` and it would look like this:

```
This file is used by 4TeX, do not delete/change it!
currentfilename=
formatname=latex.exe
format=LaTeX 2e
viewer=windvi.exe
viewertxt=WINDVI (.dvi)
printer=dvips.exe
printertxt=PostScript printer
printerport=lpt1
printerporttxt=lpt1
printeroptions=-t A4 -C 1
spelllanguage=us
spelllanguagetext=English (American)
BibTeXfile=C:\TEXFILES\TEXMF\BibTeX\BIB\bookbib.bib
AlwaysDevirtualize=0
```

Below we will explain the settings:

This file is used by 4TeX, do not delete/change it! Just a reminder that 4TEX uses this file.

currentfilename If you specified a 'Current file' it will be used.

formatname The TEX format that is used.

format The actual name of the TEX format file.

viewer The previewer program.

viewertxt The text to indicate which previewer is selected.

printer The printer program.

printertxt The text to indicate which printer is selected.

printerport The selected printer port or output file name.

printerporttxt The text to indicate which printer port or output file is selected.

printeroptions Printer options.

spelllanguage Language used for spell-checking.

spelllanguagetext The text to indicate which language for spell-checking is selected.

BibTeXfile The selected BibTEX file.

AlwaysDevirtualize The value 1 means: always use DVIcopy to replace virtual fonts in the DVI file by 'real' fonts (devirtualizing). 0 means: do not devirtualise.

12.10 The .opt files

We have already discussed the file `XX_PRN.LST` that is used to select an output device. Many of these devices offer special options that you can select from the 4TEX output menu.

The options supported by each printer type are specified in a `.opt` file. In a `.opt` file many parameters can be set. Leaving a parameter empty implies that the printer doesn't support that feature. Deleting a parameter has the same effect. As an example we show here `DVIPS.OPT`:

```
outputextension=ps
manpage=dvips.ps
tooutputfile=-o
pagerange=-pp
firstpage=-p
lastpage=-l
oddpages=-A
evenpages=-B
collated=-C
copypage=-c
copybody=-b
usetexnumbers==
reverseprint=-r
landscape=-t landscape
manualfeed=-m
printerresolution=-D
xresolution=-X
yresolution=-Y
xyoffset=-O
xoffset=
yoffset=
magnification=-x
dvimagnify=-y
```

```
stripspace=
duplexon=-h duplex.pro
duplexoff=
```

Below we will explain all the options.

outputextension When printing to a file this option specifies the default file extension in the save file menu.

manpage The documentation showing/explaining all the printer options.

tooutputfile Write output to a file.

pagerange The range of pages that should be printed.

firstpage The first page to be printed.

lastpage The last page to be printed.

oddpages Print only odd pages.

evenpages Print only even pages.

collated Create a number of copies by replicating the data in the output. Usually slower than the **copypage** option, but easier on the hands, and faster than resubmitting the same job multiple times.

copypage Generate copies of every page by using PostScript's `#copies` feature.

copybody Generate copies of each page by duplicating the page body. This can be useful in conjunction with a header file setting `bop-hook` to do color separations or other advanced tricks.

usetexnumbers The page ranges specified are the page numbers as generated by TEX, not the page counter of the output.

reverseprint The last page must be printed first.

landscape Output is printed in landscape mode.

manualfeed Wait for manual paper feed for each page.

printerresolution The printer resolution (both horizontal and vertical resolution).

xresolution Horizontal printer resolution.

yresolution Vertical printer resolution.

xyoffset Page margin/offset to move the origin by a certain amount. The offset is a comma-separated pair of dimensions, such as `.1in,-.3cm`. The origin of the page is shifted from the default position (which is one inch down, one inch to the right from the upper left corner of the paper) by this amount.

xoffset Horizontal offset.

yoffset Vertical offset.

magnification Magnify the output by the ratio $x/1000$. Overrides the magnification specified in the DVI file. We recommended that you use standard magnification steps: 1095, 1200, 1440, 1728, 2074, 2488, 2986 and so on.

dvimagnify Magnify the output with this factor, on top of any magnification already specified in the DVI file.

stripspace Some printers expect options to be specified with spaces between parameters and values, others expect no spaces at all. Use `stripspace=y` to ensure that parameters and values are separated by a space.

duplexon Print in duplex mode: on both sides of a sheet.

duplexoff Do not print in duplex mode (sometimes called 'simplex mode'): only print on one sides of a sheet.

12.11 The .pap files

For every printer type 4TEX expects a `.pap` file. This file contains a list of all available paper sizes for the given printer. These files look very similar to `.lst` files. Here are two examples, `DVIPS.PAP` (on the left) and `DVILJ4.PAP` (on the right):

```
A4 (210mm x 297mm)
  -t A4
A3 (297mm x 420mm)
  -t A3
Letter (8.5in x 11in)
  -t letter
Legal (8.5in x 14in)
  -t legal
Ledger (17in x 11in)
  -t ledger
Tabloid (11in x 17in)
  -t tabloid
```

```
A4 (210mm x 297mm)
  -s26
Letter (8.5in x 11in)
  -s2
Legal (8.5in x 14in)
  -s3
Executive (7.25in x 10.5in)
  -s1
Monarch (3.875in x 7.5in)
  -s80
Commercial-10 (4.125in x 9.5in)
  -s81
International DL (110mm x 220mm)
  -s90
International C5 (162mm  x 229mm)
  -s91
```

Note that if you select a different printer type and that printer type doesn't support the currently selected paper size, 4TEX will select the top-most paper type that is supported. If the appropriate `.pap` file can't be found, `DVIPS.PAP` will be used.

12.12 The .for files

Generating TEX formats is usually considered a complex task. In 4TEX we try to make it easy and straightforward. The generation of a format file is controlled primarily by a `.for` file. This file contains essential information that 4TEX uses to generate a format. Here is an example, the file `plain.for` that can be used to generate a Plain TEX format:

```
% format name      : plain TeX
% progname         : tex
% ini options      :
plain
\dump
```

The lines that start with a % sign are comments that 4TEX will use. The line containing the string `format name` specifies the *description* of the format: everything after the colon : is used. The line containing `progname` specifies the `-progname` option for the compiler (see section 13.3.1). The line containing `ini options` specifies any other options that should be used for format generation. The following lines are only used by TEX. Because this file is actually a TEX (or ε-TEX or PDFTEX) input file, 4TEX feeds this file to an iniTEX compiler like this:

```
▷  tex.exe -ini -progname=tex < plain.for
```

But that is not the whole story. At format generation time you can specify which hyphenation patterns you want to install. Plain TEX expects these to be found in a file called `hyphen.tex`. In 4TEX you can make your own choice of languages (or rather, hyphenation patterns) as explained in section 8.5. So, 4TEX will write the file `hyphen.tex` to the current directory just before it starts the compiler. The file could look like this:

```
\language=0
\input ushyphen.tex
\language=1
\input nlhyphen.tex
\language=2
\input dkhyphen.tex
```

In the case of a LATEX format thing works slightly differently. LATEX uses a system called 'Babel' to switch languages. At format generation time LATEX expects to find a file called `language.dat` that contains two columns: one with language 'names' and one with file names of corresponding hyphenation patterns, e.g. like this:

```
% language.dat:
% Babel names and hyphenation patterns
english ushyphen.tex
dutch   nlhyphen.tex
french  frhyphen.tex
```

Note that comments may be added in the usual TEX style. 4TEX will generate this file as well just before it starts the compiler.

With respect to languages CONTEXT takes a different approach. Languages should be specified in a file called `cont-usr.tex`, a default version of which can be found in the CONTEXT input directory `\texmf\tex\context\base`. It contains a section that reads as follows:

```
% \installlanguage [\s!af] [\c!status=\v!start] % afrikaans
% \installlanguage [\s!da] [\c!status=\v!start] % danish
  \installlanguage [\s!de] [\c!status=\v!start] % german
  \installlanguage [\s!en] [\c!status=\v!start] % english
% \installlanguage [\s!fi] [\c!status=\v!start] % finnish
  \installlanguage [\s!fr] [\c!status=\v!start] % french
% \installlanguage [\s!it] [\c!status=\v!start] % italian
  \installlanguage [\s!nl] [\c!status=\v!start] % dutch
% \installlanguage [\s!no] [\c!status=\v!start] % norwegian
  \installlanguage [\s!pl] [\c!status=\v!start] % polish
% \installlanguage [\s!pt] [\c!status=\v!start] % portuguese
  \installlanguage [\s!sp] [\c!status=\v!start] % spanish
% \installlanguage [\s!sv] [\c!status=\v!start] % swedish
% \installlanguage [\s!tr] [\c!status=\v!start] % turkish
```

By removing or adding a % at the beginning of a line you can select or unselect a language. Currently 4TEX doesn't support this kind of language specification through its menu. The `.for` file for the English version of the CONTEXT format is listed below. Note that there is no `\dump` command at the end. CONTEXT takes care of that itself.

```
% format name      : ConTeXt English
% progname         : context
% ini options      :
cont-en.tex
```

If you want to generate an ε-TEX format you need to know about a little trick to switch to extended mode. The trick is that the first character that ini-ε-TEX reads should be a `*`. So here is how the `.for` file for Plain ε-TEX looks like:

```
*% star: extended mode!
% format name      : plain eTeX
% progname         : etex
% ini options      :
etex.src
\dump
```

Note that in extended mode, ini-ε-TEX will write a format file with the extension `.efmt` instead of `.fmt`.

Another difference is that ε-TEX needs yet another way of specifying hyphenation patterns. It expects to find a file called `language.def`, which is similar to LATEX's

language.dat. The first line should identify the ε-TEX version as a comment, then a number of language specifications will follow, one of which *must* be USenglish. The first parameter of the \addlanguage command is a symbolic name for a language; the second one contains the file name of the corresponding hyphenation patterns file; the third should be empty; the fourth specifies the 'left hyphen min' value (the minimal size of the first part of a word broken over two lines); the fifth specifies the 'right hyphen min' value (idem for the second part). The last statement in the file should always be \uselanguage{USenglish}. Here is an example:

```
%% e-TeX V2.1
\addlanguage {USenglish}{ushyphen}{}{2}{3}
\addlanguage {german}{DEHYPHEN.TEX}{}{2}{3}
\addlanguage {polish}{PLHYPHEN.TEX}{}{2}{3}
\uselanguage {USenglish}
```

4TEX generates this file just before it starts the ini-ε-TEX compiler in the usual way. Currently there is no option in the 4TEX menu to specify the values of 'right hyphen min' and 'left hyphen min.' If you want other values, however, you can edit the batch file MFORMAT.BTM.

Note that in all cases a format file called texput.fmt (or texput.efmt in the case of ε-TEX) will be generated. 4TEX will rename it to the same name as the .for file if no such format file exists yet. If it does, it will ask you if you want to overwrite the older version.

After successful generation of a format file 4TEX will add an entry for it to all existing XX_FRM.LST files (see section 12.7 for details). When leaving the format generation menu 4TEX will ask if you would like to regenerate the ls-R files (see section 13.8). If you have generated a new format, you should do so, so the Web2c programs will know where to find it.

12.13 The .4spell files

The spell-checker program 4Spell, written by W. Dol and E. Frambach can be configured through .4spell files. These files can all be modified through from within 4Spell itself, or by hand. There are three kinds of .4spell files:

General:

The file FORMATS.4SPELL lists all input *formats* that are configured for 4Spell. Formats can be, e.g. TEX, BIBTEX, HTML and RTF.

The file CHARSET.4SPELL defines which character set (also called 'codepage') should be assumed for any language. 4Spell support Western, East-European and Russian. If a language is not listed in this file, the Western character set will be assumed.

Language-specific:

For each language (US, PL, NL, DE, etc.) three specific files can be used:

XX_AUTOFIX.4SPELL lists words that are silently replaced by 4Spell. You can use this list to correct common typing errors, or even to automatically expand abbreviations.
XX_USER.4SPELL lists words that 4Spell will assume to be correctly spelled, although they don't appear in the standard dictionary.
XX_SIMILAR.4SPELL lists characters that are 'similar'. These lists are used to find words in the dictionary that look similar to the word you typed.

Format-specific:

There are four kinds of format-specific files: commands, accents, environments and mathematic environments. The file names are constructed as follows. The first part is the name of the format (e.g. TEX), the second part is COMMANDS, ACCENTS, ENVIRONMENTS, MATHENVIRONMENTS, IGNORE or VERBCOMMANDS. The third part is the extension, which is of course .4spell. An example would be TEXACCENTS.4SPELL or HTMLCOMMANDS.4SPELL.
The COMMANDS file contains a list of format-specific commands and parameter delimiters, if any.
The ACCENTS file contains a list of accents coded in the specific format and their ANSI equivalent. 4Spell replaces accent commands by 'real' characters so it can look up words in the dictionary.
The ENVIRONMENTS file contains a list of LaTeX (like) environments that 4Spell will skip.
The MATHENVIRONMENTS file contains a list of commands that indicate mathematical environments that can be skipped while spell-checking.
The IGNORE file contains a list a commands of which the parameter should not be spell-checked.
The VERBCOMMANDS file contains a list of commands used for 'verbatim' printing, which will not be spell-checked.

We advise you to look thoroughly at the default configuration files if you want to change or add anything.

12.14 The .chm files

The programs 4TeX, 4Spell, 4Project and MED all support a new generation of online help files, called HTML HELP. Sooner or later many other programs will follow. HTML HELP files have the extension .chm. They are, as their name suggests, built from HTML sources. You can even 'decompile' such a help file to retrieve the original HTML sources and (GIF or JPG) graphics. Microsoft's HTML Help Workshop was used to prepare the help files. This program is available from Microsoft for free.

HTML HELP files require Microsoft Internet Explorer version 3.02 or higher on your system. You may need to run the update program hhupd.exe to enable HTML

HELP. If your system does not support HTML HELP, 4TEX will display plain HTML files as online help by launching the default World Wide Web browser.

The 4TEX online help file is called `XX_4TEX.CHM`, where 'XX' stands for the language, just like in `.lst` files. If the help in a specific language is not found, 4TEX will default to `US_4TEX.CHM`. The programs 4Spell and 4Project use the same method. Their help files are called `XX_4SPELL.CHM` and `XX_4PROJECT.CHM`, respectively.

Chapter 13

The Web2c TEX system

13.1 Introduction

Web2c is the name of a TEX implementation, originally for Unix, but now also running under MS-DOS, Amiga, Windows and several other operating systems. By TEX implementation, we mean all of the standard programs developed by the Stanford TEX project directed by D. Knuth: METAFONT, DVItype, GFtoDVI, BIBTEX, Tangle, Weave, etc., as well as TEX itself. TEX itself comes in a few variants, such as ε-TEX, PDFTEX and PDF-ε-TEX. Their command syntax, however, is identical, so one description fits all. Several other programs are also included in the Web2c distribution, most notably METAPOST and several DVI drivers.

Web2c derives its name from the fact that it translates Web sources into the C programming language. Actually Web2c takes the Pascal output of a Web file as input and translates it into C. Web2c is not a general Pascal to C convertor, but specifically geared for the (restricted) Pascal code produced from Web sources. All of Web2c is freely available, which means you don't have to pay for it, the source code is available and you can redistribute the package to anyone. However, there are a few restrictions. More information on legal and practical implications can be found in appendix C.

In this chapter we will discuss in full detail most of the programs that come with the Web2c package. A few more programs will be discussed although they are not part of Web2c. These programs are closely linked to other parts of the TEX system so this seems to be the logical place to discuss them.

13.2 General options

All the programs in this implementation accept command line options. Some also have configuration files, and environment variables can be used for configuration. We will

discuss many of them, but not all. For details on compiling the sources you should read the complete Web2c manual by K. Berry and O. Weber (cdrom).

Command line options are specified in a clean and consistent convention:

- You can supply options in any order, interspersed as you wish with non-option arguments.
- You can use – or –– to start an option name.
- You can use any unambiguous abbreviation for an option name.
- You can separate option names and values with either = or one or more spaces.
- You can use file names that would otherwise look like options by putting them after an option ––. This is useful if a file name starts with –.

By convention, non-option arguments, if specified, generally define the name of an input file, as documented for each program.

If a particular option with a value is given more than once, it is the last value that counts. An example should make this formal description clear:

```
▷ tex -verbo -fmt=latex myfile
```

specifies that you want TEX to be `verbose`, to use the `latex` format, and to read `myfile` as input. A format file can also be specified in a more compact way using the ampersand character:

```
▷ tex -verbo &latex myfile
```

The programs have a set of options in common, and some specific options. Common options are:

–help Print a message listing the basic usage and all available options, then exit.

–verbose Print verbose progress reports when running.

–version Print the version number of the program.

Specific options for TEX, METAFONT and METAPOST are explained in sections 13.3.1, 13.4.1 and 13.5.1, respectively.

13.2.1 'Initial' and 'virgin' variants

The TEX, METAFONT and METAPOST programs all have two main variants, called 'initial' and 'virgin'. The initial variant is enabled if:

1. the `-ini` option was specified; or
2. the program name is `initex`, `inimf` or `inimpost`, respectively; or
3. the first line of the main input file reads `%&ini`.

In all other cases the 'virgin' variant is used. The initial variant is used only to create 'memory dumps'. These memory dumps are given the file name extension `.fmt` in the case of TEX, `.base` in the case of METAFONT and `.mem` in the case of METAPOST. Making memory dumps is generally something you do very rarely. A typical TEX invocation to generate the 'Plain TEX' format would be:

```
▷  initex \input plain \dump
```

The file `plain.tex` will be read (which in turn inputs other files, such as a hyphenation table for English). The `\dump` command tells TEX to generate a memory dump, which will be named after the input file: `plain.fmt`.

The virgin variant is the one you use for compiling your documents. This variant reads in a memory dump created by the initial variant. Memory dumps typically contain lots of macros and initializations. Loading a memory dump is much faster than reading those macros and initializations, as the initial variant does. After that it proceeds with reading the input file specified.

Memory dumps for TEX are usually called 'formats'. On a typical TEX system you will find the 'Plain' format and the 'LATEX 2_ε' format. See chapter 15 for more information on the features of these formats. In section 13.3.3 we will explain how the program FMTutil can be used to generate TEX formats.

There is one important difference between initial and virgin TEX that you may want to remember. Only the initial version can *load* hyphenation tables. The virgin variant will be able to *use* the hyphenation tables loaded in the format, but cannot change them or load others. Let us say you want to write in Danish and you want TEX to hyphenate according to Danish rules. Then you will have to use a TEX format file that already contains a Danish hyphenation table. If there is no such format you will have to run `initex` to generate one that loads the appropriate table. In chapter 15 we will explain in more detail how TEX handles multilingual documents.

13.2.2 Editor invocation

Whenever TEX, METAFONT or METAPOST encounters an error during its run, it reports the error and usually stops to asks the user how it should proceed. One of the possible user responses is to type `e` or `E` and press Enter.[1] In this case the program will abort and invoke your editor program. This editor program can be specified by setting the environment variable `TEXEDIT` (for TEX), `MFEDIT` (for METAFONT) and `MPEDIT` (for METAPOST).

Note that these environment variables support a special feature to enable the editor program to jump to the location where the error was reported. If you use the string `%d` in such a variable, it will be replaced by the line number where the error occurred, and the string `%s` will be replaced by the name of the file in which the error occurred. As

1 See section 4.1 for an in-depth discussion of error messages.

an example we will show what the environment variable should look like if the editor program were PFE:[2]

```
set TEXEDIT=pfe32 /g%d %s
```

See section 6.2 for for details on editor programs.

13.3 The TEX program

Mostly TEX will be called simply like this:[3]

```
tex &plain myfile
```

which means the virgin variant is used; it then loads the `plain` format and reads the file `myfile.tex`. TEX's output will be written to `myfile.dvi`, and a log file named `myfile.log` will be created.

If no parameters are specified, TEX will prompt for an input file by showing `**`. You can then enter a file name (e.g. `myfile`) or you can enter a format name (e.g. `&plain`).

You may be able to avoid the awkward `&` sign. Suppose you generated a format file called `myown.fmt`; then you can copy the file `tex.exe` (or `hugetex.exe` if that is what you used to generate the format) to `myown.exe`. Now the command

```
myown myfile
```

will be equivalent to

```
tex &myown myfile
```

The files `latex.exe` and `pdflatex.exe` on the CDROM are actually such shortcuts to `tex.exe &latex` and `pdftex.exe &pdflatex`, respectively. Note that the format files `latex.fmt` and `pdflatex.fmt` are still required.

2 Note that if you use a smart command line interpreter such as 4DOS, you should make sure that `%d` and `%d` are not expanded while specifying an environment variable. You can either use backquotes like this:
`set TEXEDIT=pfe32 '/g%d %s'`
or specify double percent signs like this:
`set TEXEDIT=pfe32 /g%%d %%s`.

3 On Windows NT's standard 'Command prompt' the & has a special meaning: it is used a command separator. Therefore, in this case you must make sure it is not interpreted as such. You can do that by 'escaping' the & character: `tex "&plain myfile`. When using 4DOS you can change the command separator to, e.g., the ^ character. On the Windows 95/98 Dos prompt the & character has no special meaning.

13.3.1 Command line options

The most important TEX command line options are:

–ini Enable the 'initial' variant of the program. This is implicitly set if the program name is `initex`.

–fmt=*dumpname* Use *dumpname* instead of the program name or a `%&` line to determine the name of the format file read. Also set the program name to *dumpname*.

–progname=*string* Set program (and memory dump) name to *string*.

–extend-jobname=*word* Determine how TEX constructs the names of its output files (`.dvi`, `.log` and `.fmt`). The variable *word* can be:

maybe (the default) If the input file name ends in one of the extensions `.drv`, `.dtx`, `.ins`, `.ltx`, `.tex`, `.texi`, `.texinfo` or `.txi`, then TEX strips the extension; thus, `foo.tex` produces `foo.dvi`, and `foo.bar.tex` produces `foo.bar.dvi`.

always Always strip the extension: `foo.bar` produces `foo.dvi`. (This is the behavior of the original TEX program.)

never Never strip the extension: `foo.tex` produces `foo.tex.dvi`.

–shell-escape Enable the `\write18{shell-command}` feature. This is also enabled if the environment variable `SHELL_ESCAPE` or the value of `shell_escape` in the configuration files `texmf.cnf` is set to anything non-null that does not start with `n` or `0` (zero). It is disabled by default to avoid security problems.

–translate-file=*tcxfile* Use *tcxfile* to define which characters are printable and translations between the internal and external character sets. Moreover, *tcxfile* can be explicitly declared in the first line of the main input file like this:

```
%& -translate-file=TCXFILE
```

This is the recommended method for portability reasons. Note that parsing of the first line of a document is disabled if the parameter `parse_first_line` in `TEXMF.CNF` is set to false (`f`). Its default value is true (`t`). See section 13.8.1 for details on `TEXMF.CNF`.

13.3.2 Character translations

Web2c TEX supports 8-bit input directly through the use of `.tcx` files (TEX character translation). These files map an input (keyboard) character code to the internal TEX character code (a superset of ASCII).

A drawback of this approach is that it limits the portability of TEX documents, as some other implementations do not support it (or use a different method for input reencoding). It may also be problematic to determine the encoding to use for a TEX

document of unknown provenance; in the worst case, failure to do so correctly may result in subtle errors in the typeset output.

While `.tcx` files can be used with any format, using them breaks the LATEX `inputenc` package (see section 17.16.2). This is why you should either use a `.tcx` file or `inputenc` in LATEX files, but never both.

You can specify a `.tcx` file to be used for a particular TEX run by specifying the command-line option `-translation-file` or (preferably) by specifying it explicitly in the *first line* of the main document, as described above.

The Web2c distribution comes with at least two `.tcx` files, `il1-t1.tcx` and `il2-t1.tcx`. These support ISO Latin-1 and ISO Latin-2, respectively, with Cork-encoded fonts (a.k.a. the T1 encoding). Moreover, `.tcx` files for Czech, Polish, and Slovak are also provided.

The syntax of `.tcx` files is as follows:

- It is line-oriented. Blank lines are ignored.
- Whitespace is ignored except as a separator.
- Comments start with `%` and continue to the end of the line.
- Otherwise, a line consists of one or two character codes: `SRC [DEST]`
- Each character code may be specified in octal with a leading `0` (zero), hexadecimal with a leading `0x` (zero x), or decimal otherwise. Values must be between 0 and 255, inclusive (decimal).
- If the `DEST` code is not specified, it is taken to be the same as `SRC`.
- If the same `SRC` code is specified more than once, it is the last definition that counts.

Finally, here is what happens: when TEX sees an input character with code `SRC`, it first changes `SRC` to `DEST`; and then makes code the `DEST` 'printable', i.e., printed as-is in diagnostics and the log file.

By default, no characters are translated, and character codes between 32 and 126 inclusive (decimal) are printable. It is not possible to make these (or any) characters unprintable.

Specifying translations for the printable ASCII characters (codes 32–127) will yield unpredictable results. Additionally you should not make the following characters printable: `^^I` (tab), `^^J` (line feed), `^^M` (carriage return), and `^^?` (delete), because TEX uses them in various ways.

Thus, the idea is to specify the input (keyboard) character code for `SRC`, and the output (font) character code for `DEST`. As an example we show here a small part of `cp1250cs.tcx`:

```
%% cp1250cs.tcx: encoding translation table for TeX
%% input:        cp1250 (Windows)
%% internal TeX: CSfont encoding (Czech and Slovak CM fonts)
%% comment: prepared by Staszek Wawrykiewicz <staw@gust.org.pl>
```

```
%%          (thanks to Petr Ol\v{s}ak for suggestions)
%%          (1999) Public domain
0xbc 0xa5 % \v L
0x8a 0xa9 % \v S
0x8d 0xab % \v T
0x8e 0xae % \v Z
0xbe 0xb5 % \v l
0x9a 0xb9 % \v s
0x9d 0xbb % \v t
0x9e 0xbe % \v z
0xc0 0xc0 % \'R
0xc1 0xc1 % \'A
0xc4 0xc4 % \"A
0xc5 0xc5 % \'L
0xc8 0xc8 % \v C
0xc9 0xc9 % \'E
0xcc 0xcc % \v E
0xcd 0xcd % \'I
...
```

13.3.3 Generating formats with FMTutil

The program FMTutil offers a simplified way to generate TEX formats. Its syntax is:

≡ fmtutil [*option...*] *cmd* [*argument*]

Valid options are:

–cnffile *file* Use configuration file *file*.

–fmtdir *directory* Put generated formats in *directory*.

–quiet No output except error messages.

–test Only print what would be done.

–dolinks Link engine to format.

–force Force links even if target exists.

Valid commands are:

–all Recreate all format files.

–missing Create all missing format files.

–byfmt *formatname* (Re)create format for *formatname*.

–byhyphen *hyphenfile* (Re)create formats that depend on *hyphenfile*.

–showhyphen *formatname* Print name of hyphenfile for format *formatname*.

–help Show help message.

The configuration file is called `fmtutil.cnf`, and it is typically stored in the same directory as `TEXMF.CNF`: `/texmf/web2c`. It can contain any number of comment lines (starting with `#`), and for each TEX format a line that lists 4 items: the format *name* (e.g. `latex`), the *engine* (e.g. `pdftex`), the *hyphenation pattern file* (e.g. `language.dat` for LATEX), and the *arguments* that are passed to the TEX engine. All items should be separated by spaces and/or tabs. The last items may contain more than one word. A typical entry for generating a Plain TEX, a LATEX and a Plain PDF-ε-TEX format would be:

```
# Plain TeX format:
tex      tex      -              tex.ini
# LaTeX format:
latex    tex      language.dat   latex.ini

# Plain pdfeTeX, extended mode:
pdfetex  pdfetex  language.def   *pdfetex.ini
```

The file `language.dat` for LATEX typically looks like this:

```
% language.dat:
% Babel names and hyphenation patterns
english ushyphen.tex
dutch   nlhyphen.tex
french  frhyphen.tex
```

The file `language.def` is ε-TEX's way of specifying hyphenation patterns. It typically looks like this:

```
%% e-TeX V2.0;2
\addlanguage {USenglish}{ushyphen}{}{2}{3}
\addlanguage {german}{DEHYPHEN.TEX}{}{2}{3}
\addlanguage {polish}{PLHYPHEN.TEX}{}{2}{3}
\uselanguage {USenglish}
```

The file `tex.ini` could look like this:

```
\input plain
\dump
\endinput
```

Likewise, the file `latex.ini` could look like this:

```
\input latex.ltx
\endinput
```

And the file pdfetex.ini could look like this:

```
\pdfoutput=1
\input etex.src
\dump
\endinput
```

Given the above configuration, FMTutil would run the following commands:

```
tex -ini -fmt=tex -progname=tex tex.ini <nul
tex -ini -fmt=latex -progname=latex latex.ini <nul
pdfetex -ini -efmt=pdfetex -progname=pdfetex *pdfetex.ini <nul
```

13.3.4 Extensions

There are a few extended versions of the TEX programs available on the 4allTEX CDROM and ready to run:

ε-TEX This extended version of TEX adds many primitives and makes typesetting from right to left easy. It supports a compatibility mode in which it runs exactly like a 'normal' TEX program. In section 10.6 we have already described how the extensions of ε-TEX can be enabled. Note that format files for ε-TEX have the file name extension .efmt instead of .fmt.

PDFTEX This extended version of TEX, written by Hàn Thế Thành, can output PDF (Adobe's Portable Document Format) instead of DVI. Section 10.7 explains how to generate PDF from your TEX documents.

PDF-ε-TEX This is PDFTEX and ε-TEX merged into one program.

Running ε-TEX is so similar to running 'normal' TEX that we don't need to explain any particularities here.

PDFTEX needs a little more explanation. The defaults for PDFTEX are defined in the file pdftex.cfg. It can be found in directory texmf\pdftex\config and it looks like this:

```
output_format 1
compress_level 9
decimal_digits 3
page_width 210mm
page_height 297mm
horigin 1in
vorigin 1in
map psfonts.map
map +lw35.map
map +rawfonts.map
map +csfontd.mapfile
```

The configuration file sets default values for the following parameters, all of which can be overridden in the TEX source file:

output_format This integer parameter specifies whether the output format should be DVI or PDF. A positive value means PDF output, otherwise you will get DVI output.

compress_level This integer parameter specifies the level of text and in-line graphics compression. PDFTEX uses *zip* compression. A value of 0 means no compression, 1 means fastest, 9 means best, 2 to 8 means something in between. Just set this value to 9, unless there is a good reason to do otherwise.

decimal_digits This integer specifies the preciseness of real numbers in PDF page descriptions. It gives the maximal number of decimal digits after the decimal point of real numbers. Valid values are in range 0 to 5. A higher value means more precise output, but also results in a much larger file size and more time to display or print. In most cases the optimal value is 2.

image_resolution When PDFTEX is not able to determine the natural dimensions of an image, it assumes a resolution of 72 dots per inch. Use this variable to change this default value.

page_width, page_height These two dimension parameters specify the output medium dimensions (the paper, screen or whatever the page is rendered on).

horigin, vorigin These dimension parameters can be used to set the offset of the TEX output box from the top left corner of the 'paper'.

map This entry specifies the font mapping file, which is similar to those used by many DVI to PostScript drivers. More than one map file can be specified, using multiple map lines. If the name of the map file is prefixed with a +, its values are appended to the existing set, otherwise they replace it. If no map files are given, the default value `psfonts.map` is used, which comes with DVIPS. Note that PDFTEX map files are similar but not equal to the the fontmap files used by DVIPS (see section 13.6.6). The *PDFTEX User Manual* by T. Hàn Thế, S. Rahtz and H. Hagen (cdrom) explains all the details.

Below is part of the file `standard.map`:

```
pcrb8r Courier-Bold <pcrb8a.pfb 8r.enc
pcrbo8r Courier-BoldOblique <pcrbo8a.pfb 8r.enc
pcrr8r Courier <pcrr8a.pfb 8r.enc
pcrro8r Courier-Oblique <pcrro8a.pfb 8r.enc
phvb8r Helvetica-Bold <phvb8a.pfb 8r.enc
phvb8rn Helvetica-Narrow-Bold <phvb8an.pfb 8r.enc
phvbo8r Helvetica-BoldOblique <phvbo8a.pfb 8r.enc
phvbo8rn Helvetica-Narrow-BoldOblique <phvbo8an.pfb 8r.enc
phvr8r Helvetica <phvr8a.pfb 8r.enc
phvr8rn Helvetica-Narrow <phvr8rn.pfb 8r.enc
phvro8r Helvetica-Oblique <phvro8a.pfb 8r.enc
```

```
phvro8rn Helvetica-Narrow-Oblique <phvro8an.pfb 8r.enc
psyr Symbol psyr.pfb
ptmb8r Times-Bold <ptmb8a.pfb 8r.enc
ptmbi8r Times-BoldItalic <ptmbi8a.pfb 8r.enc
ptmr8r Times-Roman <ptmr8a.pfb 8r.enc
ptmri8r Times-Italic <ptmri8a.pfb 8r.enc
pzcmi8r ZapfChancery-MediumItalic <pzcmi8a.pfb  8r.enc
pzdr ZapfDingbats pzdr.pfb
psyro Symbol " .167 SlantFont " psyr.pfb
ptmbo8r Times-Bold " .167 SlantFont  " ptmb8a.pfb 8r.enc
ptmro8r Times-Roman " .167 SlantFont  " ptmr8a.pfb 8r.enc
```

The font definitions above suggest that PDFTEX uses PostScript Type 1 fonts. Type 1 is indeed the preferred format, but PDFTEX is also capable of using TrueType fonts. Before you can use any of the TrueType fonts that are installed on your Windows system, you will have to generate `.tfm` files using the program TTF2AFM (see section 13.7.12). From the generated `.afm` file you can generate a TEX font metrics file in `.tfm` format, using the program AFM2TFM (see section 13.7.13). TrueType fonts can be specified in `pdftex.cfg` just like PostScript fonts.

Suppose that you have a TrueType font called ‘Baskerville’ in a file named `bask.ttf`. You want to make it available to PDFTEX, using the standard 8r encoding. Here is what you should do:

```
ttf2afm -e 8r.enc -o bask.afm bask.ttf
afm2tfm bask.afm -T 8r.enc > myfonts.map
```

Now you should store `bask.afm` in the appropriate directory, e.g. `texmf\fonts\afm\adobe\baskerville`. Store `bask.tfm` also in the appropriate directory, e.g. `texmf\fonts\tfm\adobe\baskerville`. The files `bask.ttf` should be stored in, e.g. `texmf\fonts\truetype\adobe\baskerville`. Make sure that the file `myfonts.map`, created by AFM2TFM, is stored in, e.g. `texmf\pdftex\config` and that it is referenced in `pdftex.cfg` using a ‘map’ command.

13.4 The METAFONT program

METAFONT is a program for generating font shapes. All the ‘Computer Modern’ fonts that are part of any TEX distribution were written in the METAFONT language. D. Knuth’s *The Metafont book* describes the program and the language in detail. For novice users G. Tobin’s *Metafont for Beginners* (cdrom) would be a good starting point.

13.4.1 Command line options

The most important METAFONT command line options are:

–ini Enable the 'initial' variant of the program. This is implicitly set if the program name is `inimf`.

–base=*dumpname* Use *dumpname* instead of the program name or a `%&` line to determine the name of the base file read. Also sets the program name to *dumpname*.

–progname=*string* Set program (and memory dump) name to *string*.

–translate-file=*tcxfile* See section 13.3.2 for details.

Unless you are designing fonts yourself you will probably never invoke the METAFONT program. However, the TEX program or a DVI driver may do so on the fly. A DVI driver may need a rendition of a specific font that is not available at that moment. It may then call METAFONT to render that particular font at a particular size for a particular output device. Provided the METAFONT sources for that font are available, METAFONT will generate a bitmapped rendition as specified and the DVI driver will be able to use it.

This is quite a simplification of the whole process but sufficient for now. In appendix B you can find complete flowcharts of all the programs and files involved.

The command to generate a METAFONT `.base` file typically looks like this:

```
▷  inimf input plain; input modes; dump
```

IniMF will read the file `plain.mf`, then read the file `modes.mf` and write the memory dump `plain.base`. Note that the syntax is slightly different from the TEX program. The file `modes.mf` sets parameters for all output devices that you want METAFONT to support. These parameters define amongst others what resolution the output device supports and how 'black' it prints under certain conditions. See section 13.4.2 if you want to know the details.

The memory dump `plain.base` is the only one commonly used. If you want to use METAFONT for designing fonts you could start by reading G. Tobin's *Metafont for beginners* (cdrom). The reference book on the subject is D. Knuth's *The Metafont book*.

METAFONT has not become very popular as a font design program. Nowadays PostScript ('Type 1') fonts and TrueType fonts are widely used standards. But maybe METAFONT has a second chance through METAPOST, which is discussed in section 13.5.

13.4.2 Printer modes

METAFONT needs to know the characteristics of an output device to be able to produce optimally shaped bitmapped fonts. These characteristics are defined as parameters specified in the file `modes.mf`, in so-called `mode_def` declarations. Below is an explanation taken from the `modes.mf` as maintained by K. Berry.

Technically, a `mode_def` is a METAFONT definition that typically consists of a series of assignments to various variables. These variables include the following (page numbers refer to D. Knuth's *Metafont book*:

aspect_ratio The ratio of the vertical resolution to the horizontal resolution (MFbook page 94).

blacker A correction added to the width of stems and similar features, to account for devices which would otherwise make them too light (MFbook page 93).

fillin A correction factor for diagonals and other features which would otherwise be 'filled in' (page 94). An ideal device would have `fillin=0` (page 94). Negative values for `fillin` typically have either gross effects or none at all, and should be avoided. Positive values lighten a diagonal line, negative values darken it. Changes in the `fillin` value tend to have abrupt non-linear effects on the various design sizes and magnifications of a typeface.

fontmaking If non-zero at the end of the job, METAFONT writes a `.tfm` file (MFbook page 315).

o_correction A correction factor for the 'overshoot' of curves beyond the baseline or x-height. High resolution curves look better with overshoot, so such devices should have `o_correction=1`; but at low resolutions, the overshoot appears to be simply a distortion (MFbook page 93).

pixels_per_inch The horizontal resolution; the METAFONT primitive `hppp` (which is what determines the extension on the `.gf` file name, among other things) is computed from this (MFbook page 94). (An 'inch' is 72.27 pt in the TEX world.)

proofing Says whether to put additional specials in the `.gf` file for use in making proof sheets via, e.g., the utility program GFtoDVI (MFbook page 323–324).

tracingtitles If non-zero, strings that appear as METAFONT statements are typed on the terminal (MFbook page 187).

As an example we show here a portion of `modes.mf`, describing the characteristics of the Canon CX engine (300 dots per inch), used in HP Laserjet II and III and many other laser printers:

```
mode_def cx =
  mode_param (pixels_per_inch, 300);
  mode_param (blacker, 0);
  mode_param (fillin, .2);
  mode_param (o_correction, .6);
  mode_common_setup_;
enddef;
```

Compare this to with the characteristics of an HP LaserJet 5 (600 dots per inch):

```
mode_def ljfive =    % HP LaserJet 5 (600dpi)
  mode_param (pixels_per_inch, 600);
  mode_param (blacker, .75);
  mode_param (fillin, .3);
  mode_param (o_correction, 1);
  mode_common_setup_;
enddef;
```

Naturally METAFONT also supports less advanced printers, such as the Epson FX 9-pin matrix printer, which has a resolution of 240 dots per inch horizontally and 216 vertically:

```
mode_def epson =
  mode_param (pixels_per_inch, 240);
  mode_param (aspect_ratio, 216 / pixels_per_inch);
  mode_param (blacker, 0);
  mode_param (fillin, 0);
  mode_param (o_correction, .2);
  mode_common_setup_;
enddef;
```

The current version of `modes.mf` supports about 150 different printer types. Table A.3 in appendix A lists them all.

13.5 The METAPOST program

METAPOST, a program written by J. Hobby implements a picture-drawing language very much like METAFONT, but unlike METAFONT it outputs PostScript code. This PostScript output can be incorporated into a TEX document or can be used by any other program that supports 'Encapsulated PostScript'. See section 7.5 for more information on PostScript.

13.5.1 Command line options

The most important METAPOST command line options are:

–ini Enable the 'initial' variant of the program. This is implicitly set if the program name is `inimpost`.

–mem=*dumpname* Use *dumpname* instead of the program name or a `%&` line to determine the name of the `.mem` file read. Also set the program name to *dumpname*.

–progname=*string* Set program (and memory dump) name to *string*.

–translate-file=*tcxfile* See section 13.3.2 for details.

The command to generate a METAPOST `.mem` file looks like this:

```
▷  inimp input plain dump
```

Inimpost will read the file `plain.mp` and then write the memory dump `plain.mem`. Note that again the syntax is slightly different from the TEX or METAFONT programs. Typically METAPOST will be invoked like this:[4]

```
▷  mpost &plain myfile
```

METAPOST will write its output to a number of files named `myfile.`*nnn* where *nnn* is the number of the picture as specified in the input. The output files can be fully self-contained EPS files, but may also need extra input from TEX. How exactly METAPOST and TEX and a DVI driver can cooperate is explained in K. Berry's Web2c manual (cdrom). And you will certainly have to read J. Hobby's *Introduction to the MetaPost system* (cdrom) and *A User Manual for MetaPost* (cdrom).

13.5.2 MakeMPX: Support METAPOST labels

In METAPOST, labels can be typeset using any document processor; the Web2c implementation supports TEX and TROFF. MakeMPX, another program written by J. Hobby, translates the labels from the typesetting language back into low-level METAPOST commands in a so-called `mpx` file, so text can be manipulated like other graphic objects. It is invoked automatically by METAPOST. Its syntax is:

```
≡  makempx [–troff] mpfile mpxfile
```

The input comes from *mpfile* (no path searching is done), and the output goes to *mpxfile*. However, if the file *mpxfile* already exists, and is newer than *mpfile*, then nothing is done (presumably the file is up-to-date). Otherwise:

1. The program MPto (see section 13.5.4) is run to extract the label text from the METAPOST source file *mpfile*.
2. The typesetting program itself is run, either TEX or TROFF (see below). If TEX, and the file named by the environment variable `MPTEXPRE` exists (`mptexpre.tex` by default), that file is prepended to the input from the METAPOST file.
3. The typesetter output (a DVI file in the case of TEX) is translated back to METAPOST by DVItoMP (see section 13.5.3).

4 On Windows NT's standard 'Command prompt' the `&` has a special meaning: it it used as command separator. Therefore, in this case you must make sure it is not interpreted as such. You can do that by 'escaping' the `&` character: `tex "&plain myfile`. When using 4DOS you can change the command separator to a less troublesome value such as `^` character. On the Windows 95/98 Dos prompt the `&` character has no special meaning.

If any of the above steps fails, for example, if there was a typesetting mistake in the original `mpfile`, output may be left in files named `mpxerr.log`, `mpxerr.tex` or `mpxerr.dvi` so you can diagnose the problem.

The `.mpx` file created by MakeMPX is a sequence of METAPOST picture expressions, one for every label in the original METAPOST input file.

The names of the commands run by MakeMPX, and the directory added to the shell search `PATH` for the commands' location, are overridden by environment variables. Here is a list:

MAKEMPX_BINDIR The directory added to the environment variable `PATH`. If the METAPOST program is already on the `PATH` and all related programs can be found in the same directory this variable is not needed.

NEWER The command run to determine if `mpxfile` is out of date with respect to `mpfile`; default is `newer`.

MPTOTEX The command run to extract METAPOST labels in TEX format; default is `mpto -tex`.

DVITOMP The command run to convert TEX output back to METAPOST; default is `dvitomp`.

TEX The command run to typeset the labels in TEX; default is `tex`. If you use LATEX, set this to `latex`, and supply an appropriate `verbatimtex` header in the MP source.

13.5.3 DVItoMP: DVI to MPX conversion

DVItoMP, a program written by J. Hobby, converts DVI files into low-level METAPOST commands in a so-called MPX file. This program is generally invoked only by the program MakeMPX (see section 13.5.2). Its syntax is:

≡ dvitomp [*option*...] *dvifile*[.dvi] [*mpxfile*[.mpx]]

If *mpxfile* is not specified, the output goes to the basename of *dvifile* extended with `.mpx` E.g., the command:

```
dvitomp \wherever \foo.dvi
```

will create the file `.\foo.mpx`.

The only options are `-help` and `-version`.

13.5.4 MPto: Extract labels from METAPOST input

MPto extracts the labels from a METAPOST input file; this is the contents of a `btex...etex` and `verbatimtex...etex` section. This program is generally invoked by MakeMPX (see section 13.5.2). Its syntax is:

≡ mpto [*option...*] *mpfile*

The input comes from *mpfile*; no path searching is done. The output goes to standard output. Leading and trailing spaces and tabs are removed, and various predefined typesetter commands are included at the beginning of the file and at the end of the file and of each section.

The program accepts the standard options `-help` and `-version`, as well as:

–tex Surround the METAPOST sections with TEX commands. This is the default.

–troff Surround the METAPOST sections with TROFF commands.

13.5.5 Newer: compare file modification times

The program Newer, written by J. Hobby, compares file modification times. The syntax of the program is:

≡ newer *src dependent*

Newer exits successfully if the file *src* exists and it is older than *dependent*. In other words, if the modification time of *src* is greater than that of *dependent*. This program is used by MakeMPX (see section 13.5.2) but could be used in any batch file.

13.6 DVI drivers

Web2c supports a number of DVI drivers geared for printing on popular laser printers and inkjet printers, both in PCL and in PostScript mode.

The *Web2c* printer drivers should not be confused with *Windows* printer drivers. The Web2c printer drivers are console applications that write their output to `LPT1`, for example. It is up to you to make sure that a valid printer is connected to the `LPT1` port of your computer. Or if you want to print to a network printer it is up to you to redirect `LPT1` to a network printer. In a nutshell: the Web2c printers driver are completely self-contained and they do *not* use any Windows printer drivers you may already have installed.

This may bring back bad memories to you from the pre-Windows time when every word processor came with its own printer drivers. That didn't work well for several reasons. Printer manufacturers didn't want to write printer drivers for dozens of word

processors and other programs. The producers of these word processors and other programs didn't want to write them either. There are just too many printers out there and too few people using one particular printer to make it worth while. A great benefit of Windows is that it unified printer drivers. The result is that you only need to install one Windows printer driver to enable any Windows program to print.

So why doesn't Web2c use the Windows printer drivers? First of all, because of its origin. Web2c runs on dozens of different operating systems, so it needs generic drivers that run on any system. The second reason is that although there are hundreds of different printers on the market today, they can (nearly) all be divided into only three classes ('deskjet', 'laserjet' and 'PostScript'). Each class of printers can be driven by just one printer driver. Remember that one of TEX's aims is to achieve equal output on any device, so in general TEX is not interested in any specific non-standard feature of a device. A third reason is that TEX requires printer drivers to be able to use TEX fonts, which of course Windows printer drivers don't. In general, TEX doesn't use internal printer fonts, only 'downloadable' fonts. The most notable exception here is DVIPS, which can use internal PostScript fonts (e.g. 'Times' and 'Helvetica'). A fourth reason: the quality of Windows printer drivers differs greatly. Many Windows printer drivers produce less accurate output than TEX requires. Moreover, as a rule, output from Windows printer drivers is rather bulky and often inefficient compared with output from Web2c printer drivers. That may not be much of an issue if you have plenty of hard disk space and a fast computer, but if you want to take your PostScript output to a copy shop it is very convenient if it fits on a 3.5 inch diskette.

But even if you disagree with these arguments, you can still play the game. There are three ways to print using regular Windows printer drivers. WINDVI (see section 13.6.8) is a graphic Windows previewer that can also print to any Windows printer; GSview (see section 13.6.9 and 7.5) can interpret PostScript and print to any Windows printer; the program PrintFile (see section 8.17) can send output produced by a TEX DVI driver to any Windows printer.

Now let us take a closer look at the Web2c DVI drivers and a few others. The generic DVI driver for PCL laser printers is called DVILJ, on which a few variants are based. DVILJ2P is a variant for the HP LaserJet IIp; DVILJ4 is a variant for the HP LaserJet 4; DVILJ4L is a variant for the HP LaserJet 4L. Almost all laser printers you can find are compatible with one of these. The DVILJ*xx* drivers were written by G. Neumann.

Inkjet printers are supported by the program DVIHP. It also produces PCL output, but of a somewhat different kind. This driver can drive a Hewlett Packard DeskJet 500 and any other compatible inkjet printer.[5]

Note that all DVILJ*xx* and the DVIHP program do not support virtual fonts. However, they will automatically 'devirtualize' a given DVI file by calling the program DVIcopy (see section 13.7.16 so you don't have to worry about that.

PostScript is supported by the program DVIPS. It produces clean and compact code that should work on any PostScript printer, regardless of brand or resolution. Note that even if you don't have a PostScript printer you may still want to use PostScript as your

5 In 4TEX the program DVIHP is not used to generate output on inkjet printers. We prefer to use Ghostscript.

standard output format. Using the PostScript interpreter 'Ghostscript' you can print such output on almost any type of printer. See section 13.6.9 for details on this strategy.

Output on the screen is supported by WINDVI, a port of the Unix XDVI program to Win32. In many ways (e.g. font generation) it acts just like the other Web2c DVI drivers. Another (non-Web2c) screen previewer discussed here is GSview.

Note that the Web2c DVI drivers that output PCL (for HP Laserjets, Deskjets and compatibles) do not support color. WINDVI does support color reasonably, also when printing. PostScript (produced by DVIPS) is the only output format that fully supports color for both text and graphics.

Many of the parameter settings for the (Web2c) DVI drivers are stored in the global Web2c configuration file `texmf.cnf` (see section 13.8), some settings can be changed during run-time by means of environment variables.

First we will describe the DVI driver program syntax and options (section 13.6.1–13.6.6), then we will explain how to properly set environment variables in section 13.6.10. Finally we will summarize the main features of all DVI drivers so you will be able to determine which one(s) suit your purposes best.

13.6.1 DVILJ

This DVI driver can be used to print on HP LaserJet printers and compatibles. The syntax of the program is:

≡ dvilj [*option...*] *dvifile*[.dvi]

In addition to `--help`, `--version` and `--verbose`, the options are:

–a*X* Set search path leading to bitmap fonts to *X*.

–c*X* Make *X* copies.

–D*X* If $X = 1$: print odd pages only; if $X = 2$: print even pages only.

–e*X* Set output destination to *X*.

–f*X* Don't print pages before page *X*.

–g Do not reset printer at beginning of job.

–h*X* Include header file *X*.

–M*X* Don't generate missing `.pk` files.

–m*X* Set magnification to *X* (`0;h;1;2;3;4;5;#xxxx`).

–p*X* Print no more than *X* pages.

–q Operate quietly.

–r Process pages in reverse order.

–s*X* Set paper size to *X* (see table below).

–t*X* Don't print pages after *X*.

–w Don't print out warnings.

–v Tell user which pixel files are used.

–V*X* Vendor options (Kyocera or Brother).

–x*X* Set x-offset on printout to *X* (in mm)

–y*X* Set y-offset on printout to *X* (in mm)

–X*O* Set x page origin to *O* (in dots, default = 216).

–Y*O* Set y page origin to *O* (in dots, default = 300).

DVILJ supports PCL graphics files. Such files can be included by entering a `\special` command in your TEX file, like this:

```
\special{hpfile=<filename>}
```

where `<filename>` is a valid PCL graphic file. Color is not supported. Paper sizes supported by DVILJ are:

Parameter	Name	Size
–s1	Executive	7.25 inch × 10.5 inch
–s2	Letter	8.5 inch × 11 inch
–s3	Legal	8.5 inch × 14 inch
–s26	A4	210 mm × 297 mm
–s80	Monarch	3.875 inch × 7.5 inch
–s81	Commercial -10	4.125 inch × 9.5 inch
–s90	International DL	110 mm × 220 mm
–s91	International C5	162mm × 229 mm

13.6.2 DVILJ2P

This DVI driver can be used to print on an HP LaserJet IIP laser printer and compatibles. The syntax of the program is:

≡ dvilj2p [*option*...] *dvifile*[.dvi]

DVILJ2P accepts the same options as DVILJ, but there are a few differences and extras:

–d*X* Use duplex mode: *X* = 1: long-edge; *X* = 2: short-edge binding.

–l Print in landscape mode

Graphics and color support are equal to DVILJ (section 13.6.1).

13.6.3 DVILJ4

This DVI driver can be used to print on an HP LaserJet 4 laser printer and compatibles. The syntax of the program is:

≡ dvilj4 [*option*...] *dvifile*[.dvi]

DVILJ4 accepts the same options as DVILJ, but there are a few differences and extras:

–C*X* Set compression mode for raster characters to *X* (can be 0, 2, 3, default = 3).

–d*X* Use duplex mode: *X* = 1: long-edge, *X* = 2: short-edge binding.

–E Print in econo-mode (save ink).

–l Print in landscape mode.

–n Download fonts raw (default: compressed).

–R*X* Set resolution to *X* dpi where *X* = 300 or 600.

–W*X* Set minimum width of compressed characters to *X* (default = 0).

–X*O* Set x page origin to *O* (in dots, default = 432).

–Y*O* Set y page origin to *O* (in dots, default = 600).

Graphics and color support are equal to DVILJ (section 13.6.1).

13.6.4 DVILJ4L

This DVI driver can be used to print on an HP LaserJet 4L laser printer and compatibles. An HP Laserjet 4L is very similar to a standard '4' but its resolution is limited to 300 dpi. The syntax of the program is:

≡ dvilj4l [*option*...] *dvifile*[.dvi]

DVILJ4L accepts the same options as DVILJ, but there are a few differences and extras:

–C*X* Set compression mode for raster characters to *X*. Mode can be 0, 2 or 3; default is 3.

–d*X* Use duplex mode: *X* = 1: long-edge, *X* = 2: short-edge binding.

–E Print in econo-mode (save ink).

–l Print in landscape mode.

–n Download fonts raw (default: compressed).

–W*X* Set minimum width of compressed characters to *X* (default = 0).

Graphics and color support are equal to DVILJ (section 13.6.1).

13.6.5 DVIHP

This DVI driver can be used to print on an HP DeskJet printers and compatibles. The syntax of the program is:

≡ dvihp [*option*...] *dvifile*[.dvi]

Options are recognized from DVIPS (see next section) where possible, other options are passed to the DVILJ program. Supported DVIPS-like options are:

–A Print only odd pages.
–B Print only even pages.
–d# Set debug value to # (see below).
–D# Set resolution to #.
–f Run as a filter.
–l# Don't print pages after #.
–m Manual feed.
–n# Print # pages.
–O#,# Set/change paper offset to #,# mm.
–os Output to *s* instead of spooling.
–p# Don't print pages before #.
–Ps Pass directly to printer.
–v Operate verbosely.
–x# Set magnification to #.

Graphics and color support are equal to DVILJ (section 13.6.1).

13.6.6 DVIPS

This DVI driver, written by T. Rokicki, can be used to print on PostScript printers. The syntax of the program is:

≡ dvips [*option*...] *dvifile*[.dvi]

DVIPS accepts the options `--help`, `--version` and `--verbose`. Below all other options are listed in which **#** = a number; ***f*** = a file; ***s*** = a string; ∗ = suffix: `0` to turn off, `1` to turn on (default); ***c*** = a comma-separated dimension pair (e.g., `3.2in,-32.1cm`). Options can be supplied at the command line or in a configuration file. Options in a configuration file should *not* be preceded by a hyphen!

–a∗ Conserve memory, not time.

–b# Make # page copies, for posters, for example.
–c# Make # uncollated copies.
–d# Set debug value to # (see below).
–e# Set ‘Maxdrift’ to #.
–f∗ Run as a filter.
–h*f* Add header file *f*.
–i∗ Separate file per section.
–j∗ Download fonts partially.
–k∗ Print crop marks.
–l# Don’t print pages after #.
–m∗ Manual feed.
–n# Set maximum number of pages to be printed to #.
–o*f* Set output destination to *f*.
–p# Don’t print pages before #.
–q∗ Run quietly.
–r∗ Print pages in reverse order.
–s∗ Enclose output in save/restore.
–t*s* Set paper format to *s*.
–u*s* Use PS mapfile *s*.
–x# Set magnification to # (1000 = no magnification).
–y# Multiply by DVI magnification by #.
–A Print only odd pages.
–B Print only even pages.
–C# Make # collated copies.
–D# Set resolution to #.
–E∗ Try to create Encapsulated PostScript.
–F∗ Send ‘Control-D’ at end of job.
–K∗ Pull comments from inclusions.
–M∗ Don’t make fonts.
–N∗ No structured comments.
–O*c* Set/change paper offset to *c*.
–P*s* Load file `config.`*s*.
–R Run securely: do not start any program from within DVIPS.
–S# Set maximum section size in pages to #.
–T*c* Set desired page size to *c*.
–U∗ Disable ‘string param’ trick.

–V* Send downloadable PostScript fonts as `.pk` fonts.

–X# Set horizontal resolution to #.

–Y# Set vertical resolution to #.

–Z* Compress bitmap fonts.

Most of the configuration file options are similar to the command line options, but there are a few extra options, and a few options have a *different* meaning when used in a configuration file. Again, many may be turned off by suffixing the letter with a zero (`0`). These options are `a`, `f`, `q`, `r`, `I`, `K`, `N`, `U`, and `Z`.

Within a configuration file, any empty line or line starting with a space, asterisk or equal sign is ignored. When specifying paths in a configuration file a double forward slash (//) may be appended to a path to indicate that you want the program to search also in subdirectories of that path.

Options that can be used in configuration files are:

@ *name hsize vsize* Set the paper size defaults and options for the particular printer this configuration file describes. There are three formats for this option. If the option is specified on a line by itself, with no parameters, it instructs DVIPS to discard all other paper size information (possibly from another configuration files). If *name* is specified that paper size will used. If *hsize* and *vsize* are specified that paper size will be used.

H *path* The (semicolon-separated) path to search for PostScript header files.

I Ignore the environment variable `PRINTER`.

M *mode* Set METAFONT mode to be used when generating bitmap fonts. This is passed along to MKTeXPK and overrides mode derivation from the base resolution.

R *num num ...* Sets up a list of default resolutions to search for bitmap fonts (PK files), if the requested size is not available. The output will then scale the font found using PostScript scaling to the requested size. The resultant output will be ugly, and thus a warning is issued. To turn this off, use a line with just the `R` and no numbers.

S *path* The path to search for special pictures (Encapsulated PostScript files). The environment variable `TEXINPUTS` will override this.

T *path* The path to search for TFM files. The environment variable `TEXFONTS` will override this. This path is used for resident fonts and fonts that can't otherwise be found.

V *path* The path to search for virtual fonts (VF files).

W *string* Writes ***string*** to 'stderr' if a parameter is given; otherwise it cancels any previous message. This is useful in the default configuration file if, for instance, you want the user to specify a printer or if you want to notify the user that the resultant output has special characteristics.

DVIPS can be configured further by setting one or more of the following environment variables:

HOME This variable is automatically set by the shell and is used to replace any occurrences of ~ in a path. There is no default.[6]

MAKETEXPK Sets the command to be executed to create a missing bitmap font. The string `%n` is replaced by the base name of the font to be created (e.g. `cmr10`); the string `%d` is replaced by the resultant horizontal resolution of the font; the string `%b` is replaced by the horizontal resolution at which DVIPS is currently generating output; the string `%o` is replaced with the current METAFONT mode, if any, or default if none is known, and the string `%m` is replaced with a string that METAFONT can use as the right-hand side of an assignment to 'mag' to create the desired font at the proper resolution. If a mode for METAFONT is set in a configuration file and no `%o` is specified in the command, the mode is automatically appended to the command before execution. Note that these substitutions are different from the ones performed on `.pk` paths. Default is: `MakeTeXPK %n %d %b %m`.

DVIPSHEADERS Determines where to search for header files such as `tex.pro`, font files, arguments to the `-h` option, and such files.

PRINTER Determines which default printer configuration file to use. Note that it is the responsibility of the configuration file to send output to the proper print destination (`lpt1`, print queue, etc.), if such functionality is desired.

TEXFONTS Determines where `.tfm` files are searched for. A `.tfm` file only needs to be loaded if the font is a resident (PostScript) font or if for some reason no `.pk` file can be found.

TEXPKS The path on which to search for `.pk` fonts. Certain substitutions are performed if a percent sign is found anywhere in the path. See the description of the `P` configuration file option for more information.

TEXINPUTS Determines where to find PostScript figures when they are included.

TEXCONFIG This environment variable sets the directories to search for configuration files, including the system-wide one. Using this single environment variable and the appropriate configuration files, it is possible to set up the program for any environment. (The other path environment variables may then be redundant.)

VFFONTS Determines where DVIPS looks for virtual fonts. A correct virtual font path is essential if PostScript fonts are to be used.

Debug options (`-d`) supported by DVIPS are listed below:

6 Note that this variable may interfere with the placement of WINDVI's configuration file (see section 7.4).

Debug value	Tracing
1	`\specials`
2	paths
4	fonts
8	pages
16	headers
32	font compression
64	files
128	memory
256	Kpathsearch calls
512	Kpathsearch hash table look ups
1024	Kpathsearch path element expansion
2048	Kpathsearch searches

As you may have guessed, the debug parameter is actually a bitmask. If you want to debug, say, 'fonts' you specify `-d4`. If you want to debug e.g. 'paths', 'pages' and 'headers' you add $2 + 8 + 16 = 26$ and thus specify `-d26`. If you want to debug *all* Kpathsearch items you should specify `-d3650`. Note that DVIPS does *not* write a log file, it writes messages only to the `stderr`. This means that on a standard Windows prompt you can't pipe or redirect output like this:

```
▷  dvips -d255 | more
```

or

```
▷  dvips -d255 > logfile
```

Piping or redirecting would be convenient because the output may well be more than one page on screen. One way to overcome this problem is to use the 4DOS shell. From the 4DOS prompt you *can* pipe or redirect output sent to `stderr`, using the & character:

```
▷  dvips -d255 |& more
```

or

```
▷  dvips -d255 >& logfile
```

As an alternative you could try to press Pause on your keyboard while DVIPS is running but it will be hard. A log file will give you (or someone else) a better chance to analyze the debug information.

Configuring DVIPS is rather complex: many files are involved. The most important files are `config.ps` and `psfonts.map`. We will discuss them below.

config.ps

This file is contains the basic parameter settings for DVIPS. Typically it looks like this:

```
% memory available
m 1000000

% default resolution
D 600
X 600
Y 600

% default printer type
M ljfour

% also look for this list of resolutions
R 300 600 1200
Z
j0

% the printer offsets the output by this much
O 0pt,0pt

% paper sizes
@ A4size 210mm 297mm
@+ %%PaperSize: A4

@ letterSize 8.5in 11in

@ letter 8.5in 11in
@+ %%BeginPaperSize: Letter
@+ letter
@+ %%EndPaperSize

@ legal 8.5in 14in
@+ ! %%DocumentPaperSizes: Legal
@+ %%BeginPaperSize: Legal
@+ legal
@+ %%EndPaperSize

@ ledger 17in 11in
@+ ! %%DocumentPaperSizes: Ledger
@+ %%BeginPaperSize: Ledger
@+ ledger
@+ %%EndPaperSize

@ tabloid 11in 17in
@+ ! %%DocumentPaperSizes: Tabloid
@+ %%BeginPaperSize: Tabloid
```

```
@+ 11x17
@+ %%EndPaperSize

@ A4 210mm 297mm
@+ ! %%DocumentPaperSizes: A4
@+ %%BeginPaperSize: A4
@+ a4
@+ %%EndPaperSize

@ A3 297mm 420mm
@+ ! %%DocumentPaperSizes: A3
@+ %%BeginPaperSize: A3
@+ a3
@+ %%EndPaperSize
```

Most of the parameters listed can also be specified (or overruled) by command line options, but having defaults is much more convenient. Other configuration files, e.g., `config.qms` for QMS 300 dpi printers, can be loaded to overrule some of the defaults:

```
M qms
D 300
```

psfonts.map

This file tells DVIPS what fonts are available and how they should be loaded into the PostScript output file. Here are some examples of statements that could appear in this file:

```
ptmb8r Times-Bold "TeXBase1Encoding ReEncodeFont " <8r.enc
ptmbi8r Times-BoldItalic "TeXBase1Encoding ReEncodeFont " <8r.enc
ptmr8r Times-Roman "TeXBase1Encoding ReEncodeFont " <8r.enc
ptmr8r Times-Roman "TeXBase1Encoding ReEncodeFont " <8r.enc
ptmri8r Times-Italic "TeXBase1Encoding ReEncodeFont " <8r.enc
ma1r8r ArialMT "TeXBase1Encoding ReEncodeFont " <8r.enc <ma1r8a.pfb
psyr Symbol
rptmr8r Times-Roman
mbbbo8x MBemboExpert-Bold " .167 SlantFont " <mbbb8x.pfb
p +cms.map
```

The first word on each line lists the file name of a `.tfm` file; the second word lists the real typeface name, optionally followed by some code to specify a different encoding and/or a 'slant' factor. An encoding vector can be specified e.g. like this: `<8r.enc`. In case the font is one of the 35 standard fonts available on any PostScript printer, no `.pfb` file (PostScript Font Binary) needs to be specified. If it is not an internal font (such as MBemboExpert-Bold) it needs to be available on your system and can be specified as, e.g., `mbbb8x.pfb`. This `.pfb` file will be included in the PostScript output if you used this font in your document.

Instead of making one long listing you can add references to other font mapping file as in the example `p cms.map`, a small part of which is listed below:

```
cmb10 CMB10 <cmb10.pfb
cmbsy10 CMBSY10 <cmbsy10.pfb
cmbx10 CMBX10 <cmbx10.pfb
cmbx12 CMBX12 <cmbx12.pfb
cmbx5 CMBX5 <cmbx5.pfb
cmbx6 CMBX6 <cmbx6.pfb
cmbx7 CMBX7 <cmbx7.pfb
cmbx8 CMBX8 <cmbx8.pfb
cmbx9 CMBX9 <cmbx9.pfb
```

Graphics

DVIPS supports only two kinds of graphics:

- **EPS**: Encapsulated PostScript. `.eps` files can contain almost any legal PostScript code, provided that it represents no more than one page. `.eps` files are (at least partly) ASCII and they typically start with a line that looks like this:

  ```
  %!PS-Adobe-3.0 EPSF-3.0
  ```

  Beware that EPS graphics should *not* contain a so-called 'preview'. A 'preview' is a low-resolution representation of the graphic in TIFF, 'Tagged Image File Format'. Such a preview can easily be detected by looking at the first line of the `.eps` file. If the first line doesn't *start* with `%!PS-Adobe` but instead looks more like this:

  ```
  ÓÓÆNlðs  _8ÅÐ1.FH%!PS-Adobe-3.0 EPSF-3.0
  ```

  you can be sure it contains a preview. You can remove the preview by deleting the characters up to `%!PS-Adobe` in the first line. Then go to the end of the `.eps` file and delete the lines that come *after* the last human-readable line, which usually reads `%%Trailer` or `%%EOF`.
- **PCX**: a format used by almost any graphics program. Note, however, that DVIPS only supports black-and-white PCX graphics. But here is a trick to circumvent this limitation. You can load a color `.pcx` file in a graphics program such as Paint Shop Pro (see section 8.16.2). This program can save the graphics file in EPS format, in which case all color information can be retained. Unfortunately such `.eps` files tend to be very bulky, but on the other hand, if you need to scale the graphics file (e.g. to make it fit into your document) EPS is a much more appropriate format than PCX.

EPS is an extremely versatile graphics format, so we think it is not a bad idea to save any bitmap graphic in EPS format. See section 10.4 for more details on graphic file formats.

DVIPS has many more special features that we will not list here because they are only interesting to very advanced users. T. Rokicki's manual *Dvips: A TEX Driver* (cdrom) is worth reading if you want the full picture.

13.6.7 DVIPDFM

DVIPDFM is a program, written by M. Wick, that can convert DVI output to PDF. The syntax of the program is:

≡ dvipdfm [*option*...] dvifile[.dvi]

The following options ar supported:

–c Ignore color specials (for printing on black and white printers).

–f *filename* Set font map file name (default: `pdffonts.map`).

–o *filename* Set output file name (default: same as input file name with `.pdf`).

–l Landscape mode

–m *number* Set additional magnification.

–p *papersize* Set papersize: `letter`, `legal`, `ledger`, `tabloid`, `a4` and `a3` are supported (default: `letter`).

–r *resolution* Set resolution (in dots per inch) for raster fonts (default: 600).

–s *pages* Select page ranges.

–x *dimension* Set horizontal offset (default: 1.0 inch).

–y *dimension* Set vertical offset (default: 1.0 inch).

–e Disable partial font embedding (by default enabled).

–z *number* Set compression level: 0–9 (default: 9).

–v Be verbose.

–vv Be more verbose.

All dimensions entered on the command line are true TEXdimensions. Page ranges for the `-s` option are physical pages and are separated by commas, e.g., `-s 1-3,5-6`.

13.6.8 WINDVI

WINDVI is F. Popineau's Win32 port of the Unix X-Windows program XDVI. The Windows version differs from the X-Windows version in several respects. Probably the most striking difference is that WINDVI's user interface is more convenient and looks and feels like any other Windows program. The most important features are:

- monochrome or grey-scale bitmaps (anti-aliasing) for fonts

- easy navigation through the DVI file:
  – page by page
  – with different increments (by 5 or 10 pages at a time)
  – go to ‘home’, ‘end’, or any page within the document
- different shrink factors to zoom the page in and out
- a magnifying glass to show the page at the pixel level
- automatic generation of missing `.pk` files even for PostScript fonts
- tracking of DVI file changes, and automatic reopening
- some color support (foreground and background)
- visualization of PostScript inclusions

Unlike all programs described in this chapter so far, WINDVI is a program with a graphic user interface. However, we will not discuss the user interface here, but in section 7.4. The syntax of the WINDVI program is:

≡ windvi [*option*...] [*dvifile*[.dvi]]

Options for the WINDVI program are:

–single Run no more than *one* instance of WINDVI.

–autoscan Automatically reload the DVI file each time you switch to WINDVI if the DVI file was rewritten in the meantime.

+page Specifies the first page to show. If + is given without a number, the last page is assumed; the first page is the default.

–altfont *font* Declares a default font to use if a font cannot be found. Defaults to `cmr10`.

–density *density* Determines the density used when shrinking bitmaps for fonts. A higher value produces a lighter font. The default value is 40. Use this parameter on monochrome displays; for color displays, use **–gamma**. Same as **–S**.

–gamma *value* Controls the interpolation of colors in the grey-scale anti-aliasing color palette. Default value is 1.0. For $0 < gamma < 1$ the fonts will be lighter; for $gamma > 1$ the fonts will be darker.

–keep Keep same position on page when moving to a new page.

–margins *dimen* Specifies the size of both the top margin and side margin. This should be a decimal number optionally followed by *cm* or *in*, giving a measurement in centimeters or inches. It determines the ‘home’ position of the page within the window as follows. If the entire page fits in the window, then the margin settings are ignored. If, even after removing the margins from the left, right, top and bottom, the page still cannot fit in the window, then the page is put in the window such that the top and left margins are hidden, and presumably the upper left-hand corner of

the text on the page will be in the upper left-hand corner of the window. Otherwise, the text is centered in the window. See also **–sidemargin** and **–topmargin**.

–sidemargin *dimen* Specifies the side margin (see **–margins**).

–topmargin *dimen* Specifies the top and bottom margins (see **–margins**).

–mfmode *mode-def* Set the printer mode to *mode-def*. This value is used to search for bitmap fonts. It will also be used to generate bitmap fonts automatically if necessary.

–mgs1 *size* Set the size of the magnifying glass activated by button 1 (left mouse button) to *size*. The *size* can specified as an integer value (in which case the magnifying glass will be square), or as *width*x*height*. Defaults are `200x150`, `400x250` and `700x500` for left, middle and right mouse buttons, respectively.

–mgs2 *size* Set the size of the magnifying glass activated by button 2 (middle mouse button). See **–mgs1** for details.

–mgs3 *size* Set the size of the magnifying glass activated by button 3 (right mouse button). See **–mgs1** for details.

–nogrey Do not apply grey-scale anti-aliasing.

–nomakepk Don't try to generate bitmap fonts if any are missing.

–nopostscript Turns off rendering of PostScript inclusions. Bounding boxes, if known, will be displayed instead.

–noscan Normally, when PostScript is turned on, WINDVI will do a preliminary scan of the DVI file in order to send any necessary header files before sending the PostScript code that requires them. This option turns off such prescanning. It will be automatically turned back on if WINDVI detects any `\specials` that require headers.

–offsets *dimen* Set the size of both the horizontal and vertical offset to *dimen*. This should be a number followed by `mm`, `cm` or `in`. Default is 1.0 inch. See also **–xoffset** and **–yoffset**.

–xoffset *dimen* Set horizontal offset to *dimen*. See **–offsets** for details.

–yoffset *dimen* Set vertical offset to '*dimen*'. See '**–offsets**' for details.

–p *pixels* Set the size of the fonts to be used to *pixels* per inch. Default is 300.

–paper *papertype* Set paper type to *papertype*. It can be specified as *width*x*height* ('inches' are presumed), or *width*x*height*cm, or as a logical name: supported are: *us* (8.5 × 11 inches), *usr* (11 × 8.5 inches), *legal* (8.5 × 14 inches), *foolscap* (13.5 × 17 inches); ISO sizes: *a1 – a7*, *b1 – b7*, *c1 – c7*, *a1r – a7r*, etc. Default is 8.5 × 11 inches.

–rv Reverse video. Background color becomes foreground color and vice versa.

–s *value* Set the initial shrink factor to *value*. Default is 3.

–hush Suppress all suppressible warnings.

–hushchars Suppress warnings about references to characters which are not defined in a font.

–hushchecksums Suppress warnings about checksum mismatches between the DVI file and a font file.

–hushspecials Suppress warnings about `\special` strings that it can't process.

–debug *bitmask* If *bitmask* is non-zero additional information will be typed out. The number is taken as a set of independent bits where:
1 = bitmaps; 2 = DVI translation; 4 = PK reading; 8 = batch operation; 16 = events; 32 = file opening; 64 = PostScript communication; 128 = Kpathsea stat(2) calls; 256 = Kpathsea hash table look ups; 512 = Kpathsea path definitions; 1024 = Kpathsea path expansion; 2048 = Kpathsea searches. To trace everything having to do with file searching and opening, use 4000.

When specifying a *color* you can use `rgb:/rr/gg/bb/`, where `rr`, `gg` and `bb` are the hexadecimal (`00`–`FF`) intensities of the red, green and blue component, or any of the following symbolic names:

```
snow, GhostWhite, WhiteSmoke, gainsboro, FloralWhite, OldLace,
linen, AntiqueWhite, PapayaWhip, BlanchedAlmond, bisque, PeachPuff,
NavajoWhite, moccasin, cornsilk, ivory, LemonChiffon, seashell,
honeydew, MintCream, azure, AliceBlue, lavender, LavenderBlush,
MistyRose, white, black, DarkSlateGray, DarkSlateGrey, DimGray,
DimGrey, SlateGray, SlateGrey, LightSlateGray, LightSlateGrey,
gray, grey, LightGrey, LightGray, MidnightBlue, navy, NavyBlue,
CornflowerBlue, DarkSlateBlue, SlateBlue, MediumSlateBlue,
LightSlateBlue, MediumBlue, RoyalBlue, blue, DodgerBlue, DeepSkyBlue,
SkyBlue, LightSkyBlue, SteelBlue, LightSteelBlue, LightBlue,
PowderBlue, PaleTurquoise, DarkTurquoise, MediumTurquoise,
turquoise, cyan, LightCyan, CadetBlue, MediumAquamarine, aquamarine,
DarkGreen, DarkOliveGreen, DarkSeaGreen, SeaGreen, MediumSeaGreen,
LightSeaGreen, PaleGreen, SpringGreen, LawnGreen, green, chartreuse,
MediumSpringGreen, GreenYellow, LimeGreen, YellowGreen, ForestGreen,
OliveDrab, DarkKhaki, khaki, PaleGoldenrod, LightGoldenrodYellow,
LightYellow, yellow, gold, LightGoldenrod, goldenrod, DarkGoldenrod,
RosyBrown, IndianRed, SaddleBrown, sienna, peru, burlywood, beige,
wheat, SandyBrown, tan, chocolate, firebrick, brown, DarkSalmon,
salmon, LightSalmon, orange, DarkOrange, coral, LightCoral, tomato,
OrangeRed, red, HotPink, DeepPink, pink, LightPink, PaleVioletRed,
maroon, MediumVioletRed, VioletRed, magenta, violet, plum,
orchid, MediumOrchid, DarkOrchid, DarkViolet, BlueViolet, purple,
MediumPurple, thistle, gray0, grey0, DarkGrey, DarkGray, DarkBlue,
DarkCyan, DarkMagenta, DarkRed, LightGreen.
```

If no DVI file is specified WINDVI will present a file menu from which you can select a DVI file.

Just like the other DVI drivers from the Web2c family, WINDVI will call METAFONT (for METAFONT fonts) or GSFtoPK (for PostScript Type 1 fonts) if any font file is not available.

For PostScript Type 1 fonts to be rendered, Ghostscript is required, plus a file called `render.ps`. If Ghostscript is installed properly (which 4TEX does automatically), WINDVI should be able to render any PostScript inclusions referenced in the DVI file.

The current version of WINDVI is not capable of displaying bitmap graphics files, such as `.pcx` or `.gif`. However, it *is* able to render any PostScript inclusions (e.g. `.eps` graphics) by calling Ghostscript (see also section 13.6.9). Note that the magnifying glass can't be used to magnify a PostScript inclusion. Performance would suffer too much.

WINDVI saves a configuration file called `windvi.cnf` in the directory specified as `TEXMFCNF`, or, if that directory is not writable, in the directory specified in the environment variable `HOME`. If that one is not set the file will be stored in `C:\`.

13.6.9 Ghostscript

Ghostscript is a program, written by Aladdin Enterprises, that can interpret PostScript code. This code may be produced by DVIPS (see section 13.6.6), an Windows PostScript printer, a graphics program that outputs EPS (Encapsulated PostScript) or may be hand-written. Ghostscript is available for a large number of operating systems. You can use and/or distribute it free of charge, just like the Web2c suite. It is not part of the Web2c suite.

Ghostscript supports a large number of printer types so you can think of Ghostscript as a PostScript convertor. In fact, it enables you to use the most advanced PostScript tricks and yet print on a very simple printer. Ghostscript is available both as a console application and as a rendering engine, a dynamic link library (DLL). GSview, a Windows PostScript previewer, utilizes Ghostscript to render PostScript output on the screen, or on any Windows printer, provided that Ghostscript supports that particular printer type.

First we will discuss the Ghostscript console application, then we will show how convenient it is to use GSview, written by R. Lang, as a graphical user interface to Ghostscript. The syntax of the Ghostscript program is:

≡ gswin32c [*option*...] [*postscriptfile*...]

Beware that options are case-sensitive. The most frequently used options are:

–help Show program syntax and options.

–dNOPAUSE Do not pause after each page.

–dSAFER Disables file writing and directory modification to provide more security.

–q Run quietly.

–gwidthxheight Set page size in pixels.

–rres Set resolution in pixels per inch.

–sDEVICE=*devname* Select a device (see below).

–dBATCH Exit after last file.

–sOutputFile=*file* Select output file: - for 'stdout', |command for a pipe, embed %d or %ld for page #.

Ghostscript supports the following output devices:

```
mswindll, mswinprn, mswinpr2, epson, eps9high, eps9mid, epsonc,
ibmpro, deskjet djet500, laserjet, ljetplus, ljet2p, ljet3, ljet4,
cdeskjet, cdjcolor, cdjmono cdj550, pj, pjxl, pjxl300, djet500c,
declj250, lj250, jetp3852, r4081, lbp8 uniprint, st800, stcolor,
bj10e, bj200, m8510, necp6, bjc600, bjc800, t4693d2 t4693d4, t4693d8,
tek4696, pcxmono, pcxgray, pcx16, pcx256, pcx24b, pbm, pbmraw pgm,
pgmraw, pgnm, pgnmraw, pnm, pnmraw, ppm, ppmraw, tiffcrle, tiffg3,
tiffg32d tiffg4, tifflzw, tiffpack, bmpmono, bmp16, bmp256, bmp16m,
tiff12nc, tiff24nc psmono, bit, bitrgb, bitcmyk, pngmono, pnggray,
png16, png256, png16m, jpeg jpeggray, pdfwrite, pswrite, epswrite,
pxlmono, pxlcolor, nullpage.
```

Most of the device names are self-descriptive. However, if you need more detailed information you may want to read the technical information in the *.txt files that come with Ghostscript. In these documentation files many more options are explained.

The syntax of the GSview program is:

≡ gsview32 [*option...*] *PostScript file*

Options are:

/D Debug mode: GSview will not delete its temporary files when exiting.

/T*n* By default GSview runs multi-threaded on Windows 95/98 and Windows NT. Specify **/T0** to force single-threaded mode;

/P Print the specified PostScript file.

/S*port* Spool the specified PostScript file to *port*, e.g. LPT3:.

Unfortunately GSview lacks an option to allow only one instance of the program. However, it does recognize an updated input file (e.g. produced by DVIPS). GSview will automatically reload the file when you click anywhere on the display image it is displaying. Note that GSview is also capable of displaying and printing PDF files.

13.6.10 Environment variables to control DVI drivers

The best way to control the DVI drivers is to edit the configuration files texmf.cnf, mktex.cnf and windvi.cnf. We will discuss them in detail in sections 13.8.1 to 13.8.4.

However, if you want to make a temporary change it may be more convenient to set an environment variable, which will overrule the current setting in the configuration files.

Below is a list of environment variables that affect the behavior of DVI drivers. Note that some of them also affect the behavior of other programs, such as the TEX compiler. All variables listed here have been set in `texmf.cnf`, unless indicated otherwise.

`AFMFONTS`
: Adobe font metrics (`.afm`).

`PKFONTS, GFFONTS`
: Font files generated by METAFONT (`.gf, .pk`).

`TEXCONFIG`
: DVIPS configuration files.

`FONTS, GFFONTS, TEXFONTS`
: Generic bitmap fonts (`.gf`).

`TEXPICTS, TEXINPUTS`
: Encapsulated PostScript files (`.eps, .epsi`).

`TEXFONTMAPS`
: Fontmap files (`.map`).

`PKFONTS, TEXPKS, TEXFONTS`
: Packed bitmap fonts (`.pk`).

`TEXPSHEADERS, PSHEADERS`
: Downloadable PostScript (`.pro, .enc`).

`TFMFONTS, TEXFONTS`
: TEX font metrics (`.tfm`).

`T1FONTS, T1INPUTS, TEXPSHEADERS, DVIPSHEADERS`
: PostScript Type 1 fonts (`.pfa, .pfb`).

`T42FONTS`
: PostScript Type42 fonts.

`VFFONTS, TEXFONTS`
: Virtual fonts (`.vf`).

`HOME`
: For DVIPS: replacement text for ~ in configuration files; for WINDVI: directory in which configuration file is read/written (not set in configuration files).

Other programs, such as EMACS, may also use this variable. If there is any interference, try to get rid of any references to `HOME` in Web2c. That should not be hard.

`VARTEXFONTS`
: Directory where bitmap fonts are written.

`MODE`
: METAFONT mode for generating bitmap fonts (see section 13.8.2).

Table 13.1: Main features of DVI drivers

Printer drivers:	**DVILJ*xx* / DVIHP**	**DVIPS**	**Ghostscript**
Windows printer	no	no	yes
graphics support	PCL	PCX, EPS	*
virtual font support	no	yes	*
color support	no	yes	yes
speed	++	+++	−−
Previewers:	**WINDVI**	**GSview**	**Acrobat**
Windows printer	primitive	yes	yes
graphics support	EPS	*	*
virtual font support	yes	*	*
color support	yes	yes	yes
'one instance'	always	no	yes
auto reload	yes	yes	no
'WYSIWIG'	++	+++	++
speed	+	−−	+++

* : not applicable

`BDPI`
: Resolution of the output device (see section 13.8.2).

`MT_FEATURES`
: Additional settings for font generation (see section 13.8.2).

See also table 13.2 in section 13.8.7 for information on using KPSEWHICH to figure out the Web2c configuration.

13.6.11 Summary of main features

In table 13.1 we have listed the main features of all DVI drivers described in this section.

If you study this table carefully you should be able to choose the DVI driver(s) that suit your purposes best. We also hope that you will agree with us that the trio DVIPS–Ghostscript–GSview is an impressively strong combination. This is why we usually advise people to standardize on PostScript output, even if they don't own a PostScript printer.

13.7 Other programs

TEX comes with a large set of programs that allow you to program, edit and enhance fonts. We will explain the meaning of some of them here. We don't intend to give

complete explanations of all the possibilities, but we do feel that you should know of their existence and their place within the whole system. The better you understand the system, the easier it will be to work with TEX. Note that all file types listed here are either plain ASCII or binary. Binary means in this context 'not human readable'. These binary files, however, are *not* operating system specific. They are set up in such a way that they can be used by *any* TEX system on *any* platform.

13.7.1 BIBTEX

The BIBTEX program, written by O. Patashnik, is used to generate TEX code to typeset a bibliography based on bibliographic citations made in your input text. The style of the bibliography and citations is defined by `\bibliographystyle` command in your text. BIBTEX takes the `.aux` file produced by the last TEX run as input and outputs a file with the same file name but the file name extension will be `.bbl`. It also writes a log file with the extension `.blg`. The syntax of the BIBTEX program is:

≡ bibtex [*option*...] *auxfile*[.aux]

Standard options are: `-help` and `-version`. Other options:
`-terse`: suppress program banner and progress report (silent run);
`-min-crossrefs=`*n*: if at least *n* (default is 2) entries refer to another entry *c* via their `crossref` field, include *c* in the output.

O. Patashnik's *BibTEXing* (cdrom) and *Designing BibTEX Styles* (cdrom) make for interesting reading. L. Lamport's *LATEX, a Document Preparation System* explains in detail how to use BIBTEX in combination with LATEX. The principles are also explained in section 8.1.

13.7.2 MakeIndex

MakeIndex, a program written by J. Hobby, is a general purpose hierarchical index generator. It gets its input from output of a TEX job[7] in which index entries are written. MakeIndex can sort the entries and output TEX code to typeset the finished index. It does *not* automate the process of marking words to be indexed in your text. You will have to do that by hand. All the rest is taken care of by MakeIndex. L. Lamport's *MakeIndex: An Index Processor for LATEX* (cdrom) and P. Chen and M. Harrison's *Index Preparation and Processing* (cdrom) provide more information on making indices. In section 17.9 we will show how to use MakeIndex in combination with LATEX. The syntax of the MakeIndex program is:

≡ makeindex [*option*...] [*indexfile*[.idx]...]

7 MakeIndex is not confined to TEX input; it accepts input from TROFF and other programs as well, but for our purposes this is not relevant.

By default input is assumed to be a `.idx` file as produced by LaTeX. Unless specified otherwise output is written to a file with the same file name but the extension `.ind`. Supported options are:

–c Compress intermediate blanks (ignoring leading and trailing blanks and tabs). By default blanks in the index key are retained.

–g Employ German word ordering in the index according to DIN 5007. By default symbols, words starting with special characters (such as accented letters!) will appear *before* 'normal' words. In `-g` mode German TeX commands such as `"u` for 'u umlaut' are also recognized.

–i Read input from standard input (the console or a pipe).

–l Letter ordering. By default word ordering is used.

–o *ind* Write output to *ind*.

–p *num* Set the starting page number of the output index file to *num*.

–q Run quietly. By default messages are written both to the console and to a log file.

–r Disable implicit page range formation: do not generate output like '3–7'.

–s *sty* Employ *sty* as the style file that determines the output format. Note that the environment variable `INDEXSTYLE` should define the path to this file.

–t *log* Write messages to a log file *log*. By default a log file with the same name as the first input file is used, with extension `.ilg`.

13.7.3 GFtoPK

This program, written by T. Rokicki, converts a 'generic font' (GF) to a 'packed font' (PK). The PK format is much more compact than GF and is supported by almost all DVI drivers you can find. That is why METAFONT's output in GF format is always immediately converted to PK. METAFONT will output e.g. the file `cmr10.600gf` (size: 24124 bytes) which GFtoPK will convert to `cmr10.600pk` (size: 10844 bytes). Note that both GF and PK are binary file formats: they are not human-readable. The syntax of the GFtoPK program is simple:

≡ gftopk [*option*...] *gffile.dpi*[gf] [*pkfile*]

so entering `gftopk cmr10.600` would be enough to do the conversion in the example above. Options are: `-verbose`, `-help` and `-version`. See appendix B for an overview of files and programs involved.

13.7.4 PKtoGF

This program, written by T. Rokicki, converts a ‘packed font’ (PK) to ‘generic font’ (GF), so it’s the counterpart of PKtoGF. The syntax is very similar:

≡ pktogf [*option*...] *pkfile.dpi*[pk] [*gffile*]

so entering `pktogf cmr10.600` would be enough to convert `cmr10.600pk` back to `cmr10.600gf`, though you probably never need to convert back to GF format. Options are: `-verbose`, `-help` and `-version`. See appendix B for an overview of files and programs involved.

13.7.5 GFtype

GFtype, written by D. Fuchs, translates a generic font (GF) (as output by METAFONT, for example) to a plain text file that humans can read. It also serves as a GF validating program. If GFtype can read a generic font file, it is correct. The syntax is:

≡ gftype [*option*...] *gfname.dpi*[gf]

The font *gfname* is searched for in the usual places The suffix `gf` is supplied if not already present. This suffix is not an extension; no ‘.’ precedes it: for instance, `cmr10.600gf`. The translation is written to standard output. The program accepts the following options, as well as the standard `-help` and `-version`:

–images Show the characters’ bitmaps using asterisks and spaces.

–mnemonics Translate the commands in the `.gf` file.

As an example of the output, here is the translation of the letter ‘A’ in `cmr5`, as rendered at 300 dpi with the mode `cx` from `modes.mf`, with both `-mnemonics` and `-images` enabled. First we will show the top part of the translation:

```
34: beginning of char 65: 2<=m<=18 0<=n<=13
(initially n=13) paint (7)2
42: newrow 7 (n=12) paint 2
44: newrow 6 (n=11) paint 1(1)2
48: newrow 6 (n=10) paint 1(1)2
52: newrow 6 (n=9) paint 1(1)2
56: newrow 5 (n=8) paint 1(3)2
60: newrow 5 (n=7) paint 1(3)2
64: newrow 4 (n=6) paint 1(5)2
68: newrow 4 (n=5) paint 1(5)2
72: newrow 3 (n=4) paint 10
74: newrow 3 (n=3) paint 1(7)2
78: newrow 3 (n=2) paint 1(7)2
82: newrow 2 (n=1) paint 2(8)2
```

```
86: newrow 0 (n=0) paint 5(5)6
90: eoc
.<--This pixel's lower left corner is at (2,14) in Metafont coordinates
       **
       **
      * **
      * **
      * **
     *   **
     *   **
    *     **
    *     **
   **********
   *       **
   *       **
  **        **
*****     ******
.<--This pixel's upper left corner is at (2,0) in Metafont coordinates

etc.
```

Explanation:

`34:, 42:, 44:, etc.` The byte position in the file where each GF command starts.

`beginning of char 65:` The character code, in decimal notation.

`2<=m<=18 0<=n<=13` The character's bitmap lies between columns 2 and 18 (inclusive) horizontally, and between rows 0 and 13 (inclusive) vertically. Thus, 2 is the left side bearing. The right side bearing is the horizontal escapement (given below) minus the maximum `m`.

`(initially n=13) paint (7)2` The top row of pixels: 7 white pixels, 2 black pixels.

`newrow 6 (n=11) paint 1(1)2` The next row of pixels, with 6 leading white pixels on the row. Then 1 black pixel, 1 white pixel and 2 black pixels.

`eoc` The end of the main character definition.

```
5837: xxx 'fontid=CMR'
5849: xxx 'codingscheme=TeX text without f-ligatures'
5892: xxx 'fontfacebyte'
5906: yyy 15990784 (244)
5911: xxx 'jobname=cmr5'
5925: xxx 'mag=1'
5932: xxx 'mode=cx'
5941: xxx 'pixels_per_inch=300'
5962: xxx 'blacker=0'
5973: xxx 'fillin=0.2'
5985: xxx 'o_correction=0.6'
```

```
Postamble starts at byte 6003, after special info at byte 5837.
design size = 5242880 (5pt)
check sum = -2046583974
hppp = 272046 (4.1511)
vppp = 272046 (4.1511)
min m = -1, max m = 26
min n = -6, max n = 15
Character 0: dx 1114112 (17), width 870915 (17.23885), loc 2714
Character 1: dx 1507328 (23), width 1150541 (22.77376), loc 2757
Character 2: dx 1376256 (21), width 1077722 (21.33238), loc 2810
Character 3: dx 1245184 (19), width 955386 (18.91086), loc 2881
...
Character 65: dx 1310720 (20), width 1028205 (20.35223), loc 34
...
Character 125: dx 917504 (14), width 713626 (14.12547), loc 5488
Character 126: dx 917504 (14), width 713626 (14.12547), loc 5507
Character 127: dx 917504 (14), width 713626 (14.12547), loc 5524
The file had 128 characters altogether.
```

Explanation:

`xxx ...` Comments that list the parameters used by METAFONT to generate the font.

`dx` The device-dependent width, in scaled pixels, i.e., units of horizontal pixels times 2^{16}. The `(20)` is simply the same number rounded. If the vertical escapement is non-zero, it would appear here as a `dy` value.

`width` The device-independent (TFM) width of this character. It is 2^{24} times the ratio of the true width to the font's design size. The `20.35223` is the same number converted to pixels.

`loc` The byte position in the file where this character starts.

13.7.6 TFtoPL

This program takes a (binary) TEX font metric file (`.tfm`) as input and outputs the information stored in the `.tfm` file as a so-called 'property list' (`.pl`). TFM files are usually produced by METAFONT and are not human-readable. `.pl` files are human-readable and can be edited. They can then be converted back to TFM by the program PLtoTF (see next section). The syntax of the TFtoPL program is:

≡ tftopl [*option*...] *tfmfile*[.tfm] [*plfile*[.pl]]

Supported options are: `-verbose`, `-help`, `-version` and `-charcode-format=`*type*. *type* can be either `octal` or `ascii` (which is the default). This type determines the

format in which output character codes are expressed. As an example we show below part of the property list of `cmr10.tfm`:

```
(FAMILY CMR)
(FACE O 352)
(CODINGSCHEME TEX TEXT)
(DESIGNSIZE R 10.0)
(COMMENT DESIGNSIZE IS IN POINTS)
(COMMENT OTHER SIZES ARE MULTIPLES OF DESIGNSIZE)
(CHECKSUM O 11374260171)
(FONTDIMEN
   (SLANT R 0.0)
   (SPACE R 0.333334)
   (STRETCH R 0.166667)
   (SHRINK R 0.111112)
   (XHEIGHT R 0.430555)
   (QUAD R 1.000003)
   (EXTRASPACE R 0.111112)
   )
(LIGTABLE
   (LABEL O 40)
   (KRN C l R -0.277779)
   (KRN C L R -0.319446)
   (STOP)
   (LABEL C f)
   (LIG C i O 14)
   (LIG C f O 13)
   (LIG C l O 15)
   (KRN O 47 R 0.077779)
   (KRN O 77 R 0.077779)
   (KRN O 41 R 0.077779)
   (KRN O 51 R 0.077779)
   (KRN O 135 R 0.077779)
   (STOP)
...
(CHARACTER C f
   (CHARWD R 0.305557)
   (CHARHT R 0.694445)
   (CHARIC R 0.077779)
   (COMMENT
      (LIG C i O 14)
      (LIG C f O 13)
      (LIG C l O 15)
      (KRN O 47 R 0.077779)
      (KRN O 77 R 0.077779)
      (KRN O 41 R 0.077779)
      (KRN O 51 R 0.077779)
      (KRN O 135 R 0.077779)
```

```
        )
    )
...
```

Many of the ‘properties’ are easy to understand from their name, but some may need extra explanation. Anyway, if you feel like tweaking font properties you will need more information than we can give you here. It is beyond the scope of this book. A good start would be to read D. Knuth’s *TEX: The Program*.

13.7.7 PLtoTF

This program is the counterpart of ‘TFtoPL’. It takes a ‘property list’ file (`.pl`) as input and converts it to `.tfm`. The syntax of the PLtoTF program is:

≡ pltotf [*option*...] *plfile*[.pl] [*tfmfile*[.tfm]]

Options are: `-verbose`, `-help` and `-version`.

13.7.8 VPtoVF

Virtual fonts are a difficult subject. We will not discuss the details here but merely show two examples of commonly used applications of virtual fonts. See J. Gibbons’s *What exactly are virtual fonts?* cdrom and D. Knuth’s *Virtual fonts: More fun for grand wizards* cdrom for more information.

Virtual fonts can be used to ‘map’ a certain font onto a specific encoding that is different from the one the font uses itself. TEX has to know what characters are available in a font and their exact position within the font. That information is read from `.tfm` files. But the DVI driver may have to deal with a font that uses a different scheme. If it would simply use the font many or all characters may come out wrong. Therefore it first checks if there is a file available with the *same file name* as the `.tfm` file, but with the extension `.vf`. If so, this ‘virtual font’ is read to determine how to proceed. The virtual font may refer to another another `.tfm` file which holds the actual properties of the font to be used. This method is often used to enable TEX to use PostScript fonts, which hold their characters in different positions than standard TEX fonts. As an alternative, to avoid virtual fonts, you could also change the PostScript font to match any scheme you like, but this is only feasible for very experienced users.

Another application of virtual fonts is combining parts of several fonts in one. Where TEX ‘sees’ just one font, the virtual font read by the DVI driver may refer to any other font for any character. This feature may be used to map both roman characters and small capitals in one TEX font. The DVI driver will read from the virtual font file that some characters should be taken from one actual font file and some others from another.

Don’t worry if you don’t understand the purpose of all this trickery. Unless you are going to tweak fonts yourself you will never have to deal with virtual fonts at all, al-

though you will be using them. It is enough if you understand that they have no meaning to the TEX compiler, but are essential to DVI drivers. If you accidentally delete virtual fonts the DVI driver may not complain, but the output may well be a terrible mess. Remember this if one day characters in your output are wrong or missing.

As an example of a 'virtual property list' we show here part of the file `ptmr7t.vf`, the virtual font used by LATEX to typeset text in the well-known Times font. If you read carefully you will see that this virtual font refers to a font called `ptmr8r`. This font can itself be another virtual font, but this is rare. The DVI driver will try to resolve all references until it has found the basic `.tfm` files. These are sometimes called 'raw' fonts: although they are perfectly valid `.tfm` files they are not supposed to be used by the TEX compiler but only indirectly by the DVI driver through virtual fonts.

```
(VTITLE virtual font ptmr7t created by fontinst v1.6)
(FAMILY UNSPECIFIED)
(FACE F MRR)
(CODINGSCHEME TEX TEXT)
(DESIGNSIZE R 10.0)
(COMMENT DESIGNSIZE IS IN POINTS)
(COMMENT OTHER SIZES ARE MULTIPLES OF DESIGNSIZE)
(CHECKSUM O 614675731)
(FONTDIMEN
   (SLANT R 0.0)
   (SPACE R 0.25)
   (STRETCH R 0.15)
   (SHRINK R 0.0599985)
   (XHEIGHT R 0.45)
   (QUAD R 1.0)
   (EXTRASPACE R 0.0599985)
   )
(MAPFONT D 0
   (FONTNAME ptmr8r)
   (FONTCHECKSUM O 24364160751)
   (FONTAT R 1.0)
   (FONTDSIZE R 10.0)
   )
(LIGTABLE
   (LABEL O 13)
   (LIG C i O 16)
   (LIG C l O 17)
   (KRN O 47 R 0.054993)
   (KRN C i R -0.019995)
   (KRN C f R -0.025)
   (KRN O 20 R -0.05)
   (KRN C a R -0.009998)
   (KRN O 16 R -0.025)
   (KRN O 17 R -0.025)
   (KRN O 13 R -0.025)
   (KRN O 14 R -0.025)
```

```
   (KRN O 15 R -0.025)
   (STOP)
...

(CHARACTER O 13
   (CHARWD R 0.640991)
   (CHARHT R 0.681989)
   (COMMENT
      (LIG C i O 16)
      (LIG C l O 17)
      (KRN O 47 R 0.054993)
      (KRN C i R -0.019995)
      (KRN C f R -0.025)
      (KRN O 20 R -0.05)
      (KRN C a R -0.009998)
      (KRN O 16 R -0.025)
      (KRN O 17 R -0.025)
      (KRN O 13 R -0.025)
      (KRN O 14 R -0.025)
      (KRN O 15 R -0.025)
      )
   (MAP
      (SETCHAR C f)
      (MOVERIGHT R -0.025)
      (SETCHAR C f)
      )
   )
...
(CHARACTER O 23
   (CHARWD R 0.332996)
   (CHARHT R 0.681989)
   (MAP
      (SETCHAR O 264)
      )
   )
...
```

There are several ways to make virtual fonts. A. Jeffrey wrote a TEX program called 'FontInst' to generate `.vpl` files from Adobe font metric files (`.afm`), the equivalents of `.tfm` files for PostScript fonts. Another way is to use the program AFM2TFM (see section 13.7.13) that generates a `.vf` file and two `.tfm` files in one run, using a `.afm` file as input. The syntax of the VPtoVF program is:

≡ vptovf [*option*...] *vplfile*[.vpl] [*vffile*[.vf] [*tfmfile*[.tfm]]]

Options are: `-verbose`, `-help` and `-version`.

13.7.9 VFtoVP

This program takes a `.vf` file and a `.tfm` file as input and generates a 'virtual property list' file (`.vpl`). The syntax of the VFtoVP program is:

≡ vftovp [*option*...] *vfname*[.vf] [`tfmname`[.tfm] [`vplname`[.vpl]]]

Options are: `-verbose`, `-help`, `-version` and `charcode-format=`*`type`*. *type* can be either `octal` or `ascii` (which is the default). This type determines the format in which output character codes are expressed.

13.7.10 PS2PK

PS2PK is a program, written by P. Tutelaers, that takes a PostScript Type 1 font file (`.pfb`) as input to generate a TEX PK font (`.pk`).

≡ ps2pk [*option*...] *type1font* [*pkfile*]

Options are:

–d Debug: show `stat()` calls during recursive path searching.

–v Verbose output.

–e*enc* Use font encoding vector *enc*.

–X*xres* Horizontal resolution is *xres* dpi.

–E*expansion* Expand/compress horizontal spacing by a factor *expansion*.

–S*slant* Slant the font by a factor *slant*.

–O Create old-style checksums (for compatibility with old fonts).

–P*pointsize* Point size is *pointsize* pt.

–Y*yres* Vertical resolution is *yres* dpi.

–a*AFM* Use font metric file *AFM*.

–R*baseres* Base resolution is *baseres*.

13.7.11 GSFtoPK

GSFtoPK, written by P. Vojta is another program that takes a Ghostscript font file (`.gsf`) or a PostScript Type 1 font (`.pfb`) as input to generate a TEX `.pk` font.

≡ gsftopk [*option*...] *fontfile resolution*

Supported options are:

–mapline=*line* Use *line* as the line from the `.map` file.

–mapfile=*file* Use *file* as a `.map` file; default is `psfonts.map`.

–i *GS*, –interpreter=*GS* Use *GS* as Ghostscript interpreter. By default the value of the environment variable `GS_PATH` will be used to call Ghostscript.

–dosnames Generate short PK file name (e.g. `cmr10.pk` instead of `cmr10.600pk`).

–q, –quiet Don't print progress information to standard output.

–debug=*number* Set debugging flags to *number*.

–h, –help Print help message and then exit.

–v, –version Print version number and then exit.

13.7.12 TTF2AFM

This program can be used to generate an Adobe font metric file (`.afm`) from a TrueType font (`.ttf`).

≡ ttf2afm [*option*...] *fontfile*

Supported options are:

–a Print all glyphs.

–c Print encoding tables.

–i Print glyph names in form 'index123'.

–e *encoding* Use encoding from file *encoding*.

–o *output* Write output to file *output* instead of stdout.

13.7.13 AFM2TFM

The program AFM2TFM, written by T. Rokicki, can be used to convert an Adobe font metric file (`.afm`) to a TEX font metric file (`.tfm`). The syntax of the program is:

≡ afm2tfm *afm-file* [*option*...] [*tfm-file*]

As usual the option `--help` will show online help, and `--version` will display the version number. Other important options are:

–e *factor* Widen (extend) characters by *factor*.

–p *encfile* Read/download *encfile* for the PostScript encoding.

–s *factor* Oblique (slant) characters by *factor*.

–t *encfile* Read *encfile* for the encoding of the `.vpl` file.

–T *encfile* Equivalent to `-p` *encfile* `-t` *encfile*.

–u Output only characters from encodings, nothing extra.

–v *file*[*.vpl*] Make a `.vpl` file for conversion to `.vf`.

–V *scfile*[*.vpl*] Like `-v`, but synthesize smallcaps as lowercase.

The program will generate a line that should be appended to the file `psfonts.map`. Because the generated line is written to `stdout` you can easily automate this process like this:

```
afm2tfm bbr >> c:\texfiles\texmf\dvips\base\psfonts.map
```

It is advisable to test the procedure thoroughly before you append lines to `psfonts.map`.

13.7.14 TTF2TFM

The program TTF2TFM, written by F. Loyer and W. Lemberg, can be used to generate a TEX font metric file (`.tfm`) from a TrueType font file (`.ttf`).

The syntax of the program is:

ttf2tfm *ttf-file* [*option*...] [*tfm-file*]

As usual the option `--help` will show online help, and `--version` will display the version number. Other options are:

–c *real* Use *real* for height of small caps made with `-V`.

–e *real* Widen (extend) characters by a factor of *real*.

–E *int* Select *int* as the TTF encoding ID.

–f *int* Select *int* as the font index in a TTC.

–n Use PS names of TrueType font.

–N Use only PS names and no cmap.

–O Use octal for all character codes in the `.vpl` file.

–p *encfile*[*.enc*] Read *encfile* for the TrueType font: raw TEX mapping.

–P *int* Select *int* as the TrueType font platform ID.

–q Suppress informational output.

–r *oldname newname* Replace glyph name *oldname* with *newname*.

–R *rplfile*[*.rpl*] Read *rplfile* containing glyph replacement names.

–s *real* Oblique (slant) characters by *real*.

–t *encfile[.enc]* Read *encfile* for the encoding of the `.vpl` file.

–T *encfile[.enc]* Equivalent to **–p *encfile* –t *encfile***.

–u Output only characters from encodings, nothing extra.

–v *file[.vpl]* Make a `.vpl` file for conversion to `.vf`.

–V *scfile[.vpl]* Like **–*v***, but synthesize smallcaps as lowercase.

13.7.15 TTF2PK

The program TTF2PK, written by F. Loyer and W. Lemberg, can be used to generate a TEX PK file (`.pk`) from a TrueType font file (`.ttf`).

The syntax of the program is:

≡ ttf2pk [*option*...] *ttf-file dpi*

As usual the option `--help` will show online help, and `--version` will display the version number. Other options are:

–q Suppress informational output.

–n Only use `.pk` as file name extension.

–t Test for *ttf-font* (returns 0 on success).

13.7.16 DVIcopy

DVIcopy, written by P. Breitenlohner is a program that takes a DVI file as input and outputs another DVI file in which all references to virtual fonts are replaced by 'real' fonts. The syntax of the DVIcopy program is:

≡ dvicopy [*option*...] *inputdvifile*[.dvi] [*outputdvifile*[.dvi]]

Supported options are:

–help Display help and exit.

–version Display version information and exit.

–magnification=*number* Override existing magnification with *number*.

–max-pages=*number* Process *number* pages; default is one million.

–page-start=*page-spec* Start at *page-spec*, e.g. `2` or `5.*.-2`.

DVIcopy's main purpose is to accommodate DVI drivers that don't support virtual fonts. Most the DVI drivers that come with Web2c need DVIcopy but the 'devirtualization' is done automatically. In other cases, such as with DVIWIN (an older, no longer supported DVI driver that you may be using), devirtualization is not automatic. But 4TEX can be instructed to run DVIcopy after each creation of a DVI file. See section 9 for details.

13.7.17 WEB: Literate programming

The idea behind WEB is to provide a tool for experienced system programmers that makes it easier to cope with two distinct tasks: writing program code and writing documentation. WEB integrates TEX and any programming language (be it C, Pascal, Fortran or whatever) into a system from which either program code or documentation can be derived. Both exist in *one* file and can be maintained simultaneously.

Pieces of program code and documentation are mixed and need not be written in the same sequence as the documentation formatter (TEX) or the programming language requires. The WEB system will take care of linking everything together. This system enables programmers to develop a style of programming that is much clearer to themselves and to anyone who wants to understand the structure of a complex piece of software.

The complete WEB sources of TEX, METAFONT, etc. are included on the 4allTEX CDROM. See the flowcharts in appendix B to get an idea of the files and programs involved in using the WEB system.

In order to use WEB sources you need two programs, Tangle and Weave.

Tangle: From WEB to Pascal

Tangle creates a Pascal file from a WEB source. The program syntax is:

≡ tangle [*option*...] *webfile*[.web] [*changefile*[.ch]]

The Pascal output file will have the same file name base as the WEB file, and its file extension will be `.p`. If the WEB source makes use of the 'string' facility, Tangle will write the string pool to another file with file extension `.pool`. Available options are only `-help` and `-version`.

Weave: From WEB to TEX

Weave creates a TEX document from a WEB source. It assumes various macro definitions in a file called `webmac.tex`. You can find it on the 4allTEX CDROM. The program syntax is:

≡ weave [*option*...] *webfile*[.web] [*changefile*[.ch]]

The TEX output file will have the same file name as the WEB file, and its file extension will `.tex`. Available options are `--help`, `--version` and `--verbose`.

13.7.18 PatGen

This program, written by F. Liang and P. Breitenlohner, can be used to generate hyphenation patterns file that TEX can use when dumping a format file (see section 13.3.1). The syntax of the program is:

≡ patgen *dictionaryfile patternsfile outputfile translationfile*

PatGen reads the *dictionaryfile*, *patternsfile*, and *translationfile*, writing its output to `outputfile`. The *dictionaryfile* should contain a list of hyphenated words.

Below we will show a very trivial example, just to give a taste of the procedure. The *dictionaryfile* should look like this:

```
woor-den-boek
rot-zooi
ezels-oren
```

The *patternsfile* can be a set of previously generated patterns but might as well be empty, so you can also specify `nul`.

The *translationfile* contains translations of characters, so that uppercase and lowercase words are hyphenated similarly. Here is an example (note that there is a *space* at the start of each line!):

```
 a A
 b B
 c C
 d D
 e E
 f F
 g G
 h H
 i I
 j J
 k K
 l L
 m M
 n N
 o O
 p P
 q Q
 r R
 s S
 t T
 u U
 v V
 w W
 x X
```

```
y Y
z Z
```

PatGen writes messages to the screen and expects the user to enter some values interactively. Therefore it is usually more convenient to set up an input file that 'answers' PatGen's questions. At the same time screen output can be redirected to a log file:

```
patgen dict nul patterns translat < patgen.in > patgen.log
```

The file `patgen.in` could look like this:

```
1 5
1 4
1 2 10
1 4
2 1 4
1 5
1 1 1
1 6
3 2 1
1 8
1 1000 1
y
```

And here is an abbreviated version of `patgen.log`:

```
This is PATGEN, Version 2.3 (Web2c 7.3)
left_hyphen_min = 1, right_hyphen_min = 1, 26 letters
0 patterns read in
pattern trie has 256 nodes, trie_max = 256, 0 outputs
hyph_start, hyph_finish: pat_start, pat_finish:
good weight, bad weight, threshold:
processing dictionary with pat_len = 1, pat_dot = 0

0 good, 0 bad, 4 missed
0.00 %, 0.00 %, 100.00 %
...
4 good, 0 bad, 0 missed
100.00 %, 0.00 %, 0.00 %
0 patterns, 256 nodes in count trie, triec_max = 256
0 good and 27 bad patterns added
finding 0 good and 0 bad hyphens
pattern trie has 256 nodes, trie_max = 272, 8 outputs
0 nodes and 6 outputs deleted
total of 0 patterns at hyph_level 5
hyphenate word list? writing pattmp.53
```

```
4 good, 0 bad, 0 missed
100.00 %, 0.00 %, 0.00 %
```

The file `patterns` produced by PatGen will look like this:

```
3b
3d
s3
t3
```

The patterns thus generated can be read by iniTEX for use in hyphenating words. In this case the TEX command to use these patterns would be:

```
\patterns{
3b
3d
s3
t3}
```

For many languages good hyphenation pattern files are already available. Generating your own is far from trivial, so it is better left to specialists. To understand the principles you should definitely first read D. Knuth's *The TEXbook*, appendix H. The hyphenation algorithm is described in-depth in F. Liang's Ph.D. thesis, *Word hy-phen-a-tion by computer*.

13.7.19 LaCheck

The program LaCheck, written by K. Krab Thorup, can be used to check for syntactic and semantic errors in LATEX input files.

≡ lacheck [*filename*[.tex]]

Note that LaCheck's usage is not restricted to LATEX. Many of its tracing functions work just as well on TEX files in other dialects. If LaCheck doesn't find any errors in your file it will generate no output at all. If it does, it will write messages to the console. You may want to capture these messages by redirecting them to a file, like this:

▷ `lacheck myfile > myfile.chk`

Note that LaCheck cannot detect every error you could make, and LaCheck may issue warnings even if your file is free of errors. Nevertheless it can be a very helpful debugging tool. See also the tips and tricks in sections 4.1 and 4.2.

13.8 Configuring Web2c

Configuration of a Web2c TEX system is based on three main principles:

1. The files `texmf.cnf` and `mktex.cnf`: they hold all settings that any Web2c program needs;
2. The environment variable `TEXMFCNF`: it points to the directories in which any Web2c program should look for configuration file(s);
3. The resource database(s) `ls-R`.: this file lists all files in the directory where it resides and all its subdirectories.

If a Web2c program is executed while the environment variable `TEXMFCNF` is not set, the program will try to find the file based on the assumption that the configuration files can be found in their 'natural' place. Suppose that a Web2c program is stored in the directory `d:\texmf\bin\win32` then the program will try to find the configuration files in `d:\texmf\web2c`.

This strategy should work in most cases but we strongly recommend that you specify the variable `TEXMFCNF` to avoid any confusion.

13.8.1 The configuration file texmf.cnf

The file `texmf.cnf` is a plain ASCII file that can be edited using any ASCII editor program. It contains settings for the whole Web2c family of programs, and lots of comments to describe the meaning and usage of all variables.

The Web2c programs will try to find `texmf.cnf` by searching the path specified in the environment variable `TEXMFCNF`. Suppose you enter

```
▷ set TEXMFCNF=c:/texfiles/conf;d:/texmf/web2c
```

Then any Web2c program will try to read `c:/texfiles/conf/texmf.cnf` and then it will attempt to read `d:/texmf/web2c/texmf.cnf`. In fact *all* `texmf.cnf` files that it can find on the path *will* be read. Note that the first assignment of any variable will *not* be overruled by any assignment in a configuration file found further down the path. This means that you can have a large generic configuration file somewhere, and some others that contain only those settings that deviate from the generic, or that are not assigned in the generic configuration file. So if the search path is `.;c:/tex`, then all the assignments in `./texmf.cnf` overrule the assignments in `c:/tex/texmf.cnf`.

Before we analyse `texmf.cnf` in-depth there are some general principles that you should be aware of:

- Blank lines are ignored.
- A \ at the end of a line acts as a continuation character: the next line is appended, including any whitespace at the beginning of that line.

- Comments start with `%` and they continue to the end of the line, just as in TEX.
- Any variable name can be used, but it is recommended to use only the characters `a`–`z`, `A`–`Z` and `_` (the underscore) in variable names.
- A variable is declared as: `variable [. progname] [=] value`
  where the `=` sign and surrounding whitespace are optional. If the string `.progname` is present then the assignment only applies for that particular program. The `value` may contain references to other variables by prepending such a variable with a `$` character. All assignments are read before anything is expanded, so you can use variables before they are defined.
- The *first* assignment read is the one that will used. Any reassignment will be ignored.
- When specifying 'search paths' you should separate them using colons (`:`) or semicolons (`;`). Search paths should use *forward* slashes (`/`) only. A double slash (`//`) at the end of a path means: also recursively search any subdirectory from here. Specifying `!!` at the beginning of a path means that only the file name database `ls-R` file (see next section) should be consulted, never the disk. This is useful on slow static systems such as CDROMs.
- On Windows paths and file names can be specified in lowercase or uppercase because the search mechanism is not case sensitive. However, it is recommended to use the actual names.
- You can use 'brace expansion'. If you specify, e.g.
  `somepath = c:/{thisdir,thatdir}/tex`
  this will be equivalent to
  `somepath = c:/thisdir/tex;c:/thatdir/tex`

Below we have taken the file `texmf.cnf` apart to explain how to configure the Web2c system. The numbers in the left margin represent the line numbers in the real `texmf.cnf` on your run-time system. Note that in the listing below some lines had to be broken to make them fit in this book. In reality they should *not* be broken, or they should be broken using a \ as described above. Please read K. Berry's *Kpathsea manual* (cdrom) for more information.

TEXMF.CNF

original `texmf.cnf` — runtime path configuration file for Kpathsea.

What follows is a super-summary of what this `.cnf` file can contain. Please read the Kpathsea manual for more information.

Any identifier (sticking to `A`–`Z`, `a`–`z` and `_` for names is safest) can be assigned. The `=` (and surrounding spaces) is optional. `$foo` (or `${foo}`) in a value expands to the environment variable or configuration value of `foo`.

Earlier entries (in the same or another file) override later ones, and an environment variable `foo` overrides any `texmf.cnf` definition of `foo`.

All definitions are read before anything is expanded, so you can use variables before they are defined.

If a variable assignment is qualified with `.PROGRAM`, it is ignored unless the current executable is named `PROGRAM`. This `foo.PROGRAM` construct is not recognized on the right-hand side. For environment variables, use `FOO_PROGRAM`.

Which file formats use which paths for searches is described in the various programs' and the kpathsea documentation.

`//` means to search subdirectories (recursively). A leading `!!` means to look only in the `ls-R` database, never on the disk; A leading/trailing/doubled `:` in the paths will be expanded into the compile-time default. Probably not what you want.

You can use *brace notation*, for example: `c:/{mytex;othertex}` expands to `c:/mytex;c:/othertex`. Instead of the path separator (a semicolon `;`) you can use a comma: `c:/{mytex,othertex}` also expands to `c:/mytex;c:/othertex`. However, the use of the comma instead of the path separator is deprecated.

The text above assumes that the path separator is a colon (`:`). Non-Unix systems use different path separators, like the semicolon (`;`).

Part 1: Search paths and directories

You can set an environment variable to override `TEXMF` if you're testing a new TEX tree, without changing anything else.

You may wish to use one of the `$SELFAUTO...` variables here so TEX will find where to look dynamically. See the manual and the definition below of `TEXMFCNF`.

The main tree, which must be mentioned in `$TEXMF`, below:

13. `TEXMFMAIN=D:/texmf`

    A place for local additions to a 'standard' texmf tree. For example:

16. `TEXMFLOCAL=c:/4TEX5.0/texmf`

    User texmf trees can be catered for like this...

19. `HOMETEXMF=$HOME/texmf`

    A place where texconfig stores modifications (instead of the `TEXMFMAIN` tree). teTEX's texconfig relies on the name, so don't change it.

23. `VARTEXMF=$SELFAUTOPARENT/texmf-localconfig`

    Now, list all the texmf trees. If you have multiple trees, use shell brace notation, like this:
    `TEXMF = {$HOMETEXMF:!!$TEXMFLOCAL:!!$TEXMFMAIN}`
    The braces are necessary.

29. `{$HOMETEXMF,$TEXMFLOCAL,!!$VARTEXMF,!!$TEXMFMAIN}`

    The system trees. These are the trees that are shared by all the users.

32. `SYSTEXMF=$TEXMF`

Where generated fonts may be written. This tree is used when the sources were found in a system tree and either that tree wasn't writable, or the `varfonts` feature was enabled in `MT_FEATURES` in `mktex.cnf`:

37. `VARTEXFONTS=C:/TEX5.0/texmf/fonts`

Where to look for `ls-R` files. There need not be an `ls-R` in the directories in this path, but if there is one, Kpathsea will use it:

41. `TEXMFDBS=$TEXMF;$VARTEXFONTS`

It may be convenient to define `TEXMF` like this:
`TEXMF = {$HOMETEXMF:!!$TEXMFLOCAL:!!$TEXMFMAIN:$HOME}`
which allows users to set up entire texmf trees, and tells TEX to look in places like `~/tex` and `~/bibtex`. If you do this, define `TEXMFDBS` like this:
`TEXMFDBS = $HOMETEXMF:$TEXMFLOCAL:$TEXMFMAIN:$VARTEXFONTS`
or MKTEXLSR will generate an `ls-R` file for `$HOME` when called, which is rarely desirable. If you do this you will want to define `SYSTEXMF` like this:
`SYSTEXMF = $TEXMFLOCAL:$TEXMFMAIN`
so that fonts from a user's tree won't escape into the global trees.

On some systems, there will be a system tree which contains all the font files that may be created as well as the formats. For example
`VARTEXMF = /var/lib/texmf`
is used on many Linux systems. In this case, set `VARTEXFONTS` like this
`VARTEXFONTS = $VARTEXMF/fonts`
and do not mention it in `TEXMFDBS` (but *do* mention `VARTEXMF`).

Usually you will not need to edit any of the other variables in part 1

`WEB2C` is for Web2C specific files. The current directory may not be a good place to look for them:

69. `WEB2C=$TEXMF/web2c`

`TEXINPUTS` is for TEX input files — i.e., anything to be found by `\input` or `\openin`, including `.sty`, `.eps`, etc.

LATEX-specific macros are stored in `latex`.

75. `TEXINPUTS.latex=.;$TEXMF/tex/{latex,generic,}//`

76. `TEXINPUTS.hugelatex=.;$TEXMF/tex/{latex,generic,}//`

Fontinst needs to read `.afm` files.

79. `TEXINPUTS.fontinst=.;$TEXMF/{tex{/fontinst,},fonts/afm}//`

Plain TEX. Have the command tex check all directories as a last resort, we may have plain-compatible stuff anywhere.

83. `TEXINPUTS.tex=.;$TEXMF/tex/{plain,generic,}//`

Other plain-based format:

85. `TEXINPUTS.amstex=.;$TEXMF/tex/{amstex,plain,generic,}//`

86. `TEXINPUTS.ftex=.;$TEXMF/tex/{formate,plain,generic,}//`

87. `TEXINPUTS.texinfo=.;$TEXMF/tex/{texinfo,plain,generic,}//`

88. `TEXINPUTS.eplain=.;$TEXMF/tex/{eplain,plain,generic,}//`

```
TEXINPUTS.jadetex=.;$TEXMF/tex/{jadetex,generic,plain,}//
TEXINPUTS.pdfjadetex=.;$TEXMF/{pdftex,tex}/{jadetex,generic,
plain,}//
```

ε-TEX:

```
TEXINPUTS.elatex=.;$TEXMF/{etex,tex}/{latex,generic,}//
TEXINPUTS.etex=.;$TEXMF/{etex,tex}/{generic,plain,}//
```

PDFTEX. This form of the input paths is borrowed from teTeX. A certain variant of TDS is assumed here, unaffected by the build variables.

```
TEXINPUTS.pdftexinfo=.;$TEXMF/{pdftex,tex}/{texinfo,plain,
generic,}//
TEXINPUTS.pdflatex=.;$TEXMF/{pdftex,tex}/{latex,generic,}//
TEXINPUTS.pdftex=.;$TEXMF/{pdftex,tex}/{plain,generic,}//
TEXINPUTS.pdfelatex=.;$TEXMF/{pdfetex,pdftex,etex,tex}/{latex,
generic,}//
TEXINPUTS.pdfetex=.;$TEXMF/{pdfetex,pdftex,etex,tex}/{plain,
generic,}//
```

Omega:

```
TEXINPUTS.lambda=.;$TEXMF/{omega,tex}/{lambda,latex,generic,}//
TEXINPUTS.omega=.;$TEXMF/{omega,tex}/{plain,generic,}//
```

CONTEXT macros by Hans Hagen:

```
TEXINPUTS.context=.;$TEXMF/{pdfetex,pdftex,etex,tex}/{context,
plain,generic,}//
```

CSTEX, from Petr Olsak:

```
TEXINPUTS.cslatex=.;$TEXMF/tex/{cslatex,csplain,latex,generic,}//
TEXINPUTS.csplain=.;$TEXMF/tex/{csplain,plain,generic,}//
TEXINPUTS.pdfcslatex=.;$TEXMF/{pdftex,tex}/{cslatex,csplain,
latex,generic,}//
TEXINPUTS.pdfcsplain=.;$TEXMF/{pdftex,tex}/{csplain,plain,
generic,}//
```

Polish LATEX:

```
TEXINPUTS.platex=.;$TEXMF/tex/{platex,latex,generic,}//
```

French:

```
TEXINPUTS.frtex=.;$TEXMF/{mltex,tex}/{plain,generic,}//
TEXINPUTS.frlatex=.;$TEXMF/{mltex,tex}/{frlatex,latex,generic,}//
```

MLTEX:

```
TEXINPUTS.mltex=.;$TEXMF/{mltex,tex}/{plain,generic,}//
TEXINPUTS.mllatex=.;$TEXMF/{mltex,tex}/{latex,generic,}//
```

Odd formats needing their own paths:

```
129. TEXINPUTS.lollipop=.;$TEXMF/tex/{lollipop,generic,plain,}//
130. TEXINPUTS.lamstex=.;$TEXMF/tex/{lamstex,generic,plain,}//
```

Earlier entries override later ones, so put this last.

```
133. TEXINPUTS=.;$TEXMF/tex/{generic,}//
```

Metafont, MetaPost inputs:

```
135. MFINPUTS=.;$TEXMF/metafont//;{$TEXMF/fonts,$VARTEXFONTS}/source//
136. MPINPUTS=.;$TEXMF/metapost//
```

Dump files (`./fmt/.base/.mem`) for vir{tex,mf,mp} to read (see `web2c/INSTALL`), and string pools (`.pool`) for ini{tex,mf,mp}. It is silly that we have six paths and directories here (they all resolve to a single place by default), but historically...

```
143. TEXFORMATS=.;$TEXMF/web2c
144. MFBASES=.;$TEXMF/web2c
145. MPMEMS=.;$TEXMF/web2c
146. TEXPOOL=.;$TEXMF/web2c
147. MFPOOL=.;$TEXMF/web2c
148. MPPOOL=.;$TEXMF/web2c
```

Device-independent font metric files:

```
151. VFFONTS=.;$TEXMF/fonts/vf//
152. TFMFONTS=.;{$TEXMF/fonts,$VARTEXFONTS}/tfm//
153. VFFONTS.cslatex=.;{$TEXMF/cstex/fonts,$TEXMF/fonts}/vf//
154. VFFONTS.csplain=.;{$TEXMF/cstex/fonts,$TEXMF/fonts}/vf//
155. VFFONTS.pdfcslatex=.;{$TEXMF/cstex/fonts,$TEXMF/fonts}/vf//
156. VFFONTS.pdfcsplain=.;{$TEXMF/cstex/fonts,$TEXMF/fonts}/vf//
157. TFMFONTS.cslatex=.;{$TEXMF/cstex/fonts,$TEXMF/fonts,
     $VARTEXFONTS}/tfm//
158. TFMFONTS.csplain=.;{$TEXMF/cstex/fonts,$TEXMF/fonts,
     $VARTEXFONTS}/tfm//
159. TFMFONTS.pdfcslatex=.;{$TEXMF/cstex/fonts,$TEXMF/fonts,
     $VARTEXFONTS}/tfm//
160. TFMFONTS.pdfcsplain=.;{$TEXMF/cstex/fonts,$TEXMF/fonts,
     $VARTEXFONTS}/tfm//
```

The `$MAKETEX_MODE` below means the drivers will not use a 'cx' font when the mode is 'ricoh'. If no mode is explicitly specified, `kpse_prog_init` sets `MAKETEX_MODE` to `/`, so all subdirectories are searched. See the manual. The modeless part guarantees that bitmaps for PostScript fonts are found.

```
166. PKFONTS=.;{$TEXMF/fonts,$VARTEXFONTS}/pk/{$MAKETEX_MODE,
     modeless}//
```

Similarly for the GF format, which only remains in existence because METAFONT outputs it (and METAFONT isn't going to change).

```
170. GFFONTS=.;$TEXMF/fonts/gf/$MAKETEX_MODE//
```
A backup for PKFONTS and GFFONTS. Not used for anything:
```
173. GLYPHFONTS=.;$TEXMF/fonts
```
For `texfonts.map` and included map files used by MKTeXpk. See `ftp://ftp.tug.org/tex/fontname.tar.gz`
```
177. TEXFONTMAPS=.;$TEXMF/fontname
```
BibTeX bibliographies and style files:
```
180. BIBINPUTS=.;$TEXMF/bibtex/{bib,}//
181. BSTINPUTS=.;$TEXMF/bibtex/{bst,}//
```
PostScript headers, prologues (`.pro`), encodings (`.enc`) and fonts:
```
186. TEXPSHEADERS.pdflatex=.;$TEXMF/{tex,pdftex,dvips,fonts/type1}//
187. TEXPSHEADERS.pdfelatex=.;$TEXMF/{tex,pdftex,dvips,fonts/type1}//
188. TEXPSHEADERS.pdftexinfo=.;$TEXMF/{tex,pdftex,dvips,fonts/type1}//
189. TEXPSHEADERS.pdfcslatex=.;$TEXMF/{tex,pdftex,dvips,fonts/type1}//
190. TEXPSHEADERS.pdfcsplain=.;$TEXMF/{tex,pdftex,dvips,fonts/type1}//
191. TEXPSHEADERS.pdfetex=.;$TEXMF/{tex,pdftex,dvips,fonts/type1}//
192. TEXPSHEADERS.pdfjadetex=.;$TEXMF/{tex,pdftex,dvips,fonts/type1}//
193. TEXPSHEADERS.pdfmex=.;$TEXMF/{tex,pdftex,dvips,fonts/type1}//
194. TEXPSHEADERS.pdftex=.;$TEXMF/{tex,pdftex,dvips,fonts/type1}//
195. TEXPSHEADERS.pdftexinfo=.;$TEXMF/{tex,pdftex,dvips,fonts/type1}//
196. TEXPSHEADERS.cont-de=.;$TEXMF/{tex,pdftex,dvips,fonts/type1}//
197. TEXPSHEADERS.cont-en=.;$TEXMF/{tex,pdftex,dvips,fonts/type1}//
198. TEXPSHEADERS.cont-nl=.;$TEXMF/{tex,pdftex,dvips,fonts/type1}//
199. TEXPSHEADERS.context=.;$TEXMF/{etex,tex,pdftex,dvips,fonts/
     type1}//
200. TEXPSHEADERS=.;$TEXMF/{dvips,fonts/type1,pdftex}//
```
PostScript Type 1 outline fonts:
```
202. T1FONTS=.;$TEXMF/fonts/type1//
```
PostScript AFM metric files:
```
206. AFMFONTS=.;$TEXMF/fonts/afm//
```
TrueType outline fonts:
```
209. TTFONTS=.;$TEXMF/fonts/truetype//
```
Type 42 outline fonts:
```
212. T42FONTS=.;$TEXMF/fonts/type42//
```
Dvips' `config.*` files (this name should not start with `TEX`!).
```
218. TEXCONFIG=.;$TEXMF/dvips//
```

MakeIndex style (`.ist`) files:

```
INDEXSTYLE=.;$TEXMF/makeindex//
```

Used by DMP (ditroff-to-mpx), called by `makempx -troff`.

```
TRFONTS=/usr/lib/font/devpost
```

```
MPSUPPORT=.;$TEXMF/metapost/support
```

For XDvi to find `mime.types` and `.mailcap`, if they do not exist in `$HOME`. These are single directories, not paths. (But the default `mime.types`, at least, may well suffice.)

```
MIMELIBDIR=C:/4TEX5.0/etc
```

```
MAILCAPLIBDIR=C:/4TEX/etc
```

TEX documentation and source files, for use with KPSEWHICH:

```
TEXDOCS=.;$TEXMF/doc//
```

```
TEXSOURCES=.;$TEXMF/source//
```

Omega-related fonts and other files. The odd construction for `OFMFONTS` makes it behave in the face of a definition of `TFMFONTS`. Unfortunately no default substitution would take place for `TFMFONTS`, so an explicit path is retained.

```
OFMFONTS=.;{$TEXMF/fonts,$VARTEXFONTS}/{ofm,tfm}//;$TFMFONTS
```

```
OPLFONTS=.;{$TEXMF/fonts,$VARTEXFONTS}/opl//
```

```
OVFFONTS=.;{$TEXMF/fonts,$VARTEXFONTS}/ovf//
```

```
OVPFONTS=.;{$TEXMF/fonts,$VARTEXFONTS}/ovp//
```

```
OTPINPUTS=.;$TEXMF/omega/otp//
```

```
OCPINPUTS=.;$TEXMF/omega/ocp//
```

TEX4ht utility, sharing files with TEX4ht:

```
T4HTINPUTS=.;$TEXMF/{,tex/latex/}tex4ht//
```

The `mktex*` scripts rely on `KPSE_DOT`. Do not set it in the environment:

```
KPSE_DOT=.
```

This definition isn't used from this `.cnf` file itself (that would be paradoxical), but the compile-time default in paths.h is built from it. The `SELFAUTO*` variables are set automatically from the location of `argv[0]`, in `kpse_set_progname`.

About the `/.` construction:
1) if the variable is undefined, we'd otherwise have an empty path element in the compile-time path. This is not meaningful.
2) if we used `/$VARIABLE`, we'd end up with `//` if `VARIABLE` is defined, which would search the entire world.

The `TETEXDIR` stuff isn't likely to be relevant unless you're using teTeX, but it doesn't hurt.

```
TEXMFCNF=.;{$SELFAUTOLOC,$SELFAUTODIR,$SELFAUTOPARENT}{,{/share,
}/texmf{.local,}/web2c};C:/4TEX5.0/texmf/web2c
```

Part 2: Non-path options

Write `.log`, `.dvi`, etc. files here, if the current directory is unwritable.
`TEXMFOUTPUT = /tmp`

If a dynamic file creation fails, log the command to this file, in either the current directory or `TEXMFOUTPUT`. Set to the empty string or `0` to avoid logging.

281. `MISSFONT_LOG=missfont.log`

Set to a colon-separated list of words specifying warnings to suppress. To suppress everything, use `TEX_HUSH = all`; this is equivalent to
`TEX_HUSH = checksum:lostchar:readable:special`

286. `TEX_HUSH=none`

Enable system commands via `\write18{...}`? If set to `t` it is enabled; if set to `t` it is disabled. The latter is the default to avoid security problems.

289. `shell_escape=f`

Allow TEX `\openin`, `openout` or `\input` on file names starting with '.' (e.g., `.rhosts`)?
`a` (any) : any file can be opened
`r` (restricted): disallow opening 'dotfiles'. `p` (paranoid): as `r` and disallow going to parent directories, and restrict absolute paths to be under `$TEXMFOUTPUT`.

296. `openout_any=p`

297. `openin_any=a`

Allow TEX, Metafont, and MetaPost to parse the first line of an input file for the `%&format` construct. If set to true, the compiler will attempt to load the specified format file (if any) and respect a `translate-file` specification (if any):

301. `parse_first_line=t`

Enable the `mktex...` scripts by default? These must be set to `0` or `1`. Particular programs can and do override these settings, for example DVIPS' `-M` option. Your first chance to specify whether the scripts are invoked by default is at configure time.

These values are ignored if the script names are changed; e.g., if you set `DVIPSMAKEPK` to `foo`, what counts is the value of the environment variable/config value `FOO`, not the `MKTEXPK` value.
`MKTEXTEX = 0`
`MKTEXPK = 0`
`MKTEXMF = 0`
`MKTEXTFM = 0`
`MKOCP = 0`
`MKOFM = 0`

What METAPOST runs to make `.mpx` files. This is passed an option `-troff` if METAPOST is in TROFF mode. Set to `0` to disable this feature.

321. `MPXCOMMAND=makempx`

Part 3: Array and other sizes for TEX (and Metafont and MetaPost)

If you want to change some of these sizes only for a certain TEX variant, the usual dot notation works, e.g.,
`main_memory.hugetex = 20000000`

If a change here appears to be ignored, try redumping the format file.

Memory. Must be less than 8,000,000 total.

`main_memory` is relevant only to iniTEX, `extra_mem_*` only to non-ini. Thus, have to redump the `.fmt` file after changing `main_memory`; to add to existing `.fmt` files, increase `extra_mem_*`. (To get an idea of how much, try `\tracingstats=2` in your TEX source file.

To increase space for boxes (as might be needed by, e.g., PiCTEX), increase `extra_mem_bot`.

For some xy-pic samples, you may need as much as 700,000 words of memory. For the vast majority of documents, 60,000 or less will do.

Words of inimemory available; also applies to inimf and inimp:

346. `main_memory=263000`

Extra high memory for chars, tokens, etc.:

347. `extra_mem_top=0`

Extra low memory for boxes, glue, breakpoints, etc.:

348. `extra_mem_bot=0`

Words of font info for TEX (total size of all `.tfm` files, approximately):

351. `font_mem_size=100000`

Total number of fonts. Must be between 50 and 2000 (without `tex.ch` changes):

354. `font_max=1000`

Extra space for the hash table of control sequences (which allows 10K names as distributed):

358. `hash_extra=0`

Maximum number of characters in all strings, including all error messages, help texts, font names, control sequences. These values apply to TEX and MetaPost:

363. `pool_size=125000`

Minimum pool space after TEX/METAPOST's own strings; must be at least 25,000 less than `pool_size`, but doesn't need to be nearly that large:

367. `string_vacancies=25000`

Maximum number of strings:

368. `max_strings=15000`

Minimum pool space left after loading `.fmt`:

369. `pool_free=5000`

Hyphenation trie. As distributed, the maximum is 65,535; this should work unless 'unsigned short' is not supported or is smaller than 16 bits. This value should suffice for UK English, US English, French, and German (for example). To increase, you must

change `ssup_trie_opcode` and `ssup_trie_size` in `tex.ch` (and rebuild TEX); the trie will then consume four bytes per entry, instead of two.

US English, German, and Portuguese: 30,000.
German: 14,000.
US English: 10,000.

382. `trie_size=64000`

Buffer size. TEX uses the buffer to contain input lines, but macro expansion works by writing material into the buffer and reparsing the line. As a consequence, certain constructs require the buffer to be very large. As distributed, the size is 50,000; most documents can be handled within a tenth of this size.

389. `buf_size=50000`

These are Omega-specific:
Character buffers for ocp filters:

392. `ocp_buf_size=20000`

Stacks for ocp computations:

393. `ocp_stack_size=10000`

Control for multiple ocp's:

394. `ocp_list_size=1000`

These work best if they are the same as the I/O buffer size, but it doesn't matter much. Must be a multiple of 8.
TEX:

398. `dvi_buf_size=16384`

Metafont:

399. `gf_buf_size=16384`

It's probably inadvisable to change these. At any rate, we must have:
45 < `error_line` < 255;
30 < `half_error_line` < `error_line` – 15;
60 <= `max_print_line`;
These apply to Metafont and Metapost as well.

406. `error_line=79`

407. `half_error_line=50`

408. `max_print_line=79`

Simultaneous input sources:

409. `stack_size=300`

For saving values outside current group:

410. `save_size=4000`

Simultaneous macro parameters:

411. `param_size=500`

Simultaneous input files and error insertions:

412. `max_in_open=15`

Number of hyphenation exceptions, > 610 and < 32767:

413. `hyph_size=1000`

Simultaneous semantic levels (e.g., groups):

414. `nest_size=100`

Simultaneous input files and error insertions:

459. `max_in_open=15`

ConTEXt-specific:

417. `main_memory.context=1100000`
418. `hash_extra.context=25000`
419. `pool_size.context=750000`
420. `string_vacancies.context=45000`
421. `max_strings.context=55000`
422. `pool_free.context=47500`
423. `nest_size.context=500`
424. `param_size.context=1500`
425. `save_size.context=5000`
426. `stack_size.context=1500`

Huge TEX-specific:

428. `main_memory.hugetex=1100000`
429. `param_size.hugetex=1500`
430. `stack_size.hugetex=1500`
431. `hash_extra.hugetex=15000`
432. `string_vacancies.hugetex=45000`
433. `pool_free.hugetex=47500`
434. `nest_size.hugetex=500`
435. `save_size.hugetex=5000`
436. `pool_size.hugetex=500000`
437. `max_strings.hugetex=55000`

Huge LATEX-specific:

439. `main_memory.hugelatex=1100000`
440. `param_size.hugelatex=1500`
441. `stack_size.hugelatex=1500`
442. `hash_extra.hugelatex=15000`
443. `string_vacancies.hugelatex=45000`
444. `pool_free.hugelatex=47500`

445. nest_size.hugelatex=500
446. save_size.hugelatex=5000
447. pool_size.hugelatex=500000
448. max_strings.hugelatex=55000

JadeTeX-specific:

450. main_memory.jadetex=1500000
451. param_size.jadetex=1500
452. stack_size.jadetex=1500
453. hash_extra.jadetex=50000
454. string_vacancies.jadetex=45000
455. pool_free.jadetex=47500
456. nest_size.jadetex=500
457. save_size.jadetex=5000
458. pool_size.jadetex=500000
459. max_strings.jadetex=55000

PDFJadeTeX-specific:

461. main_memory.pdfjadetex=2500000
462. param_size.pdfjadetex=1500
463. stack_size.pdfjadetex=1500
464. hash_extra.pdfjadetex=50000
465. string_vacancies.pdfjadetex=45000
466. pool_free.pdfjadetex=47500
467. nest_size.pdfjadetex=500
468. save_size.pdfjadetex=5000
469. pool_size.pdfjadetex=500000
470. max_strings.pdfjadetex=55000

PDFLaTeX-specific:

472. main_memory.pdflatex=1500000
473. param_size.pdflatex=1500
474. stack_size.pdflatex=1500
475. hash_extra.pdflatex=15000
476. string_vacancies.pdflatex=45000
477. pool_free.pdflatex=47500
478. nest_size.pdflatex=500
479. pool_size.pdflatex=500000

```
save_size.pdflatex=5000
max_strings.pdflatex=55000
PDF-e-LaTeX-specific:
main_memory.pdfelatex=1500000
param_size.pdfelatex=1500
stack_size.pdfelatex=1500
hash_extra.pdfelatex=15000
string_vacancies.pdfelatex=45000
pool_free.pdfelatex=47500
nest_size.pdfelatex=500
pool_size.pdfelatex=500000
save_size.pdfelatex=5000
max_strings.pdfelatex=55000
```

13.8.2 The configuration file mktex.cnf

This configuration specifies a few settings for Web2c DVI drivers. The DVI drivers need to know what kind of output device they are addressing. The output device determines where a DVI driver should look for bitmapped fonts. And of course a DVI driver needs to know at which resolution an output device works. This information is also needed in case bitmapped fonts have to be generated. Bitmapped fonts can be generated from METAFONT sources or from PostScript Type 1 sources. Typically `mktex.cnf` looks like this:

```
: ${MODE=ljfour}
: ${BDPI=600}
: ${ps_to_pk=gsftopk} # some prefer ps2pk
: ${MT_FEATURES=appendonlydir:dosnames}
```

In case METAFONT sources are used the `MODE` variable from `mktex.cnf` is used, unless it is overruled by an environment variable with the same name. Likewise, the `BDPI` variable determines the resolution.

Type 1 fonts can be rendered either by the program GSFtoPK or PS2PK. We recommend that you use GSFtoPK. Note that GSFtoPK requires Ghostscript to do its job. This means that the environment variables `GS_LIB` and `GS_PATH` need to be set correctly. GSFtoPK also expects to find the file `render.ps` in the TDS.

Web2c DVI drivers will try to put bitmapped fonts in their 'correct' TDS trees if you have more than one. This is guessed by locating the corresponding `.tfm` file. In case this method fails, bitmapped fonts will be written to the directory specified in `texmf.cnf` as `VARTEXFONTS` or the corresponding environment variable.

The parameter `MT_FEATURES` can be used to specify some additional features. The following keywords are supported:

appendonlydir This tells the system to create directories (if necessary) with a 'sticky bit' set. This features is silently ignored on non-Unix platforms (such as Windows 95, 98 and NT) because they don't support similar functionality.

dosnames Use '8.3' names; e.g., `dpi600/cmr10.pk` instead of `cmr10.600pdi`. Note that this feature only affects file names that would otherwise clash with other TEX-related file names. Nothing is done about file names that exceed the 8.3 limits but remain unique when truncated (by the operating system) to these limits; nor will anything be done about possible clashes with files that aren't related to TEX.

fontmap Instead of deriving the location of a font in the destination tree from the location of the sources, the aliases and directory names from the 'Fontname' distribution are used.

nomode Omit the directory level for the mode name. This is fine as long as you generate fonts for only one mode.

stripsupplier Omit the font supplier name directory level.

striptypeface Omit the font typeface name directory level.

strip Omit the font supplier *and* typeface name directory levels. This feature is discouraged in favor of 'stripsupplier' and 'striptypeface'.

varfonts When this feature is enabled, fonts that would otherwise be written in the system texmf tree will go to the directory specified as `VARTEXFONTS` in `texmf.cnf`. This option can be overridden by the environment variable `USE_VARTEXFONTS`: if set to 0 the feature is disabled, if set to 1 the feature is enabled.

13.8.3 Path searching and ls-R

All Web2c programs use a common search mechanism to find the resources they need. This mechanism (not a program) is called 'kpathsearch' ('k' for K. Berry, who implemented the routines).

Kpathsearch can do regular subdirectory searching by querying the file system, but it can also utilize a precompiled database, in order to minimize disk access. This database contains an index of all the files present in a certain directory and below. The database's file name is `ls-R` which is derived from the Unix `ls` command that can produce such an index directly.[8] The Win32 implementation comes with a program called MKTEXLSR that does exactly the same. The syntax of the program is:

≡ mktexlsr [*directory* ...]

8 Actually the Unix command is `ls -LAR ./ >ls-R`, entered from the root of a texmf tree.

So you can simply enter:

```
▷   mktexlsr c:\tex\texmf
```

to generate an `ls-R` in the directory `c:\tex\texmf`, which will recursively list all the files in that directory and all its subdirectories.

If you run MKTEXLSR without parameters it will (re)generate `ls-R` indexes in all directories specified in the variable `TEXMFDBS` (see section 13.8.1).

Here is a piece of a typical `ls-R` database:

```
% ls-R -- filename database for kpathsea; do not change this line.
/dev/null

./:
ALIASES
ETEX
FONTS
ls-R
makeindex
METAFONT
METAPOST
PDFTEX
TEX
web2c

./ETEX:
EPLAIN
GENERIC

./ETEX/EPLAIN:
CONFIG

./ETEX/EPLAIN/CONFIG:
HYPHEN.TEX

./ETEX/GENERIC:
ETEXDEFS.LIB

etc.
```

As you can see from the example the database contains lines in plain ASCII. Blank lines in it are ignored. If a line begins with `/` or `./` and ends with a colon, it is assumed to be a directory name. All other lines represent files residing in the most recently listed directory.

In the same directory as this database you can have a second database file called `aliases`. As its name suggests, this database contains aliases for file names. Here is an example:

```
% real-filename alias
blackdvi.tex blackdvi.sty
colordvi.tex colordvi.sty
czhyph2e.tex lang-cz.pat
dehyphn.tex gnhyph01.tex
dehyphn.tex lang-de.pat
dehypht.tex ghyph31.tex
dehypht.tex lang-deo.pat
dkhyphen.tex lang-da.pat
epsf.tex epsf.sty
frhyph.tex lang-fr.pat
ithyph.tex lang-it.pat
letterspacing.tex letterspace.sty
```

Now, if a TEX document requests the input file `fancyhea.sty` (e.g on an MS-DOS machine) but you have `fancyheadings.sty`, TEX will automatically find it.

The format of the aliases database is very straightforward. On each line there are two file names separated by one or more spaces. The left name is the real name, the right name is the alias. Blank lines and lines starting with `%` or `#` are ignored.

Note that again these databases are strictly ASCII and therefore human-readable. However, editing `ls-R` is not recommended. It is much safer to simply regenerate it if you think there is something wrong.

13.8.4 The configuration file windvi.cnf

Unlike `texmf.cnf` and `mktex.cnf` the configuration file `windvi.cnf` is actually a Windows `.ini` file in disguise. Windows `.ini` files are usually stored either in the Windows system directory (indicated by the environment variable `windir`) or in the directory where the corresponding program is stored.

WINDVI, however, saves its configuration file in the directory specified as `TEXMFCNF`, or, if that directory is not writeable, in the directory specified in the environment variable `HOME`. If that one is not set, the file will be stored in `C:\`. Note that unlike the other `.cnf` files this file is rewritten every time WINDVI runs; it is not static.

Like all Windows `.ini` files `windvi.cnf` contains a section header (only one, though: `[Display]`) and a lot of variable names with (optionally) a value. Below is a listing of the content of `windvi.cnf`:

```
[Display]
shrinkFactor=7
densityPercent=40
grey=true
gamma=1.000000
```

```
grid1Color=
grid2Color=
grid3Color=
pixelsPerInch=600
Margin=
sideMargin=
topMargin=
Offset=
xOffset=
yOffset=
paper=a4
altFont=cmr10
makePk=true
mfMode=ljfour
listFonts=true
reverseVideo=false
magnifierSize1=200x150
magnifierSize2=400x250
magnifierSize3=700x500
magnifierSize4=
magnifierSize5=
warnSpecials=false
Hush=true
hushLostChars=true
hushChecksums=true
safer=false
borderWidth=2
foreground=black
background=white
geometry=480x566+8+2
keepPosition=false
postscript=true
prescan=false
allowShell=false
ghostscript=true
gsSafer=true
gsAlpha=false
interpreter=gsdll32.dll
palette=Color
underLink=true
wwwBrowser=
urlBase=
singleInstance=true
autoScan=true
inMemory=false
[Last Used Files]
numFiles=3
lastFile0=C:\texfiles\latexsample.dvi
```

```
lastFile1=C:\4texwin\doc.tex
lastFile2=C:\test\Mathtest.dvi
```

Many of the variables listed can be set through command line parameters (see section 13.6.8). Some of them may also be set within WINDVI using its 'Options' screens. Then there are a few variables left that can only be changed by editing the configuration file. Some of these are not documented, so good luck experimenting.

13.8.5 Fontmap files

If a bitmap font or metric is not found with the original name, Kpathsea looks through any existing *fontmap* files for an *alias* for the original font name. These files are named `texfonts.map` and searched for along the path in the environment variable `TEXFONTMAPS` or `texmf.cnf` variable. All fontmap files found are read; earlier definitions override later ones. Fontmap files can be helpful in two cases:

- An alias name is limited in length only by available memory, not by your file system. Therefore, if you want to ask for `Times-Roman` instead of `ptmr`, you can.
- A few fonts have historically had multiple names: specifically, LaTeX's 'circle font' has variously been known as `circle10`, `lcircle10` `lcircle1` and `lcirc10`. Aliases can make all these names equivalent, so that it no longer matters what the actual name of the installed font is; TeX documents will find their favorite name.

The format of the fontmap files is straightforward:

- Comments start with a `%` and they continue to the end of the line.
- Blank lines are ignored.
- Each non-blank line is broken up into a series of *words*: a sequence of non-whitespace characters. If the first word is `include`, then the second word is used as a file name, and it is searched for and read. Otherwise, the first word on each line is the true file name; the second word is the alias; subsequent words are ignored.

If an alias has an extension, it matches only those files with that extension. Otherwise, it matches anything with the same root, regardless of extension. For example, an alias `foo.tfm` matches only when `foo.tfm` is being searched for, but an alias `foo` matches `foo.vf`, `foo.600pk`, etc.

As an example, here is an excerpt from the file `texfonts.map` on the CDROM.

```
circle10        lcircle10
circle10        lcirc10
lcircle10       circle10
lcircle10       lcirc10
lcirc10         circle10
lcirc10         lcircle10
...
```

```
% Allow people to use PostScript font names.
include adobe.map
include apple.map
...
```

And here is an excerpt from the file `adobe.map`:

```
ptmr8r   Times-Roman
ptmri8r  Times-Italic
ptmb8r   Times-Bold
ptmbi8r  Times-BoldItalic
...
```

You can check this with the program KPSEWHICH. If you enter e.g.:

```
▷  kpsewhich Times-Roman.tfm
```

KPSEWHICH will respond like this:

```
D:/TEXMF/TEX/fonts/tfm/adobe/times/ptmr8r.tfm
```

13.8.6 Logging

Kpathsea can record the time and file name found for each successful search. This may be useful in finding good candidates for deletion when your filesystem is full, or in discovering usage patterns at your site.

To do this, define the environment variable `TEXMFLOG`. Its value should be the name of a file to which output will be appended. If the file doesn't exist yet, it will be created automatically.

Each successful search turns into one line in the log: two words separated by a space. The first word is the time of the search, as the integer number of seconds since UTC (Coordinated Universal Time) midnight 1 January 1970. The second word is the file name. Here is an example of a log file after compiling a short LATEX document:

```
902003004 D:/TEXMF/ls-R
902003004 c:/texfiles/texmf/fonts/ls-R
902003006 D:/TEXMF/aliases
902003006 D:/TEXMF/web2c/latex.fmt
902003008 D:/TEXMF/tex/latex/base/article.cls
902003008 D:/TEXMF/tex/latex/base/article.cls
902003009 D:/TEXMF/tex/latex/base/size10.clo
902003009 D:/TEXMF/tex/latex/base/size10.clo
902003010 D:/TEXMF/fonts/tfm/public/cm/cmbx10.tfm
902003010 D:/TEXMF/fonts/tfm/public/cm/cmtt10.tfm
```

Only file names that are absolute are recorded, to preserve some semblance of privacy.

13.8.7 KPSEWHICH

As explained before, a TEX system uses many many files, which are kept in a complex directory tree. Nevertheless, as a rule, you don't specify *where* a certain resource can be found (e.g. a LATEX style file), you only refer to its *file name*. All TEX and related programs 'know' how to search the directory tree for the files they need.

But it may happen that a search fails and, for example, the TEX compiler halts. In that case the source may be nonexistent or misplaced. Or you may find that the wrong version of a certain source is being used.

In such cases you will want to analyze what went wrong, and KPSEWHICH may be helpful. It uses the Web2c configuration file(s) and `ls-R` file(s) to locate resources, just like all the Web2c programs.

The syntax of the KPSEWHICH program is:

≡ kpsewhich [*option...*] [*file name*]

Apart from the usual `--help` and `--version`, options are:

–debug=*num* Set debugging flags to *num*. See table 13.3 for details on debugging options.

–D, –dpi=*num* Use a base resolution for bitmap fonts of *num*; default is 600.

–expand-braces=*string* Output variable and brace expansion of *string*.

–expand-path=*string* Output complete path expansion of *string*.

–expand-var=*string* Output variable expansion of *string*.

–format=*name* Use file type format *name* (see table 13.2).

–interactive Ask for additional file names to look up.

–mktex=*fmt* Enable mktex*ftm* generation where *fmt* = `pk`, `mf`, `tex` or `tfm`.

–no-mktex=*fmt* Disable mktex*fmt* generation where *fmt* = `pk`, `mf`, `tex` or `tfm`.

–mode=*string* Set device name for `$MAKETEX_MODE` to ***string***; no default.

–must-exist Search the disk as well as `ls-R` if necessary.

–path=*string* Search in the path *string* instead of guessing the search path from the file name.

–progname=*string* Set program name to *string*.

–show-path=*name* Output search path for file type *name* (see list below).

Table 13.2 lists the most important recognized file type format names with short descriptions, corresponding environment variables and file name extensions.

KPSEWHICH can tell you which file the TEX compiler will use if you ask for, say, the LATEX class file `article`. From the command line you could enter

▷ `kpsewhich article.cls`

Table 13.2: File type formats for KPSEWHICH

Format name	Description	Environment variables	Extensions
afm	Adobe font metrics	AFMFONTS	.afm
base	METAFONT memory dumps	MFBASES	.base
bib	BIBTEX bibliography sources	BIBINPUTS, TEXBIB	.bib
bst	BIBTEX style files	BSTINPUTS	.bst
bitmap font	font files by METAFONT	PKFONTS, GFFONTS	
cnf	runtime configuration files	TEXMFCNF	.cnf
dvips config	DVIPS configuration files	TEXCONFIG	
fmt	TEX memory dumps	TEXFORMATS	.fmt, .efmt
gf	generic bitmap fonts	FONTS, GFFONTS, TEXFONTS	.gf
graphic/figure	Encapsulated PostScript files	TEXPICTS, TEXINPUTS	.eps, .epsi
ist	MakeIndex style files	TEXINDEXSTYLE	.ist
ls-R	file name databases	TEXMFDBS	ls-R
map	fontmaps	TEXFONTMAPS	.map
mem	METAPOST memory dumps	MPEMS	.mem
mf	METAFONT sources	MFINPUTS	.mf
mfpool	METAFONT program strings	MFPOOL	.pool
mp	METAPOST sources	MPINPUTS	.mp
mppool	METAPOST program strings	MPPOOL	.pool
pk	packed bitmap fonts	PKFONTS, TEXPKS, TEXFONTS	.pk
PostScript header	downloadable PostScript	TEXPSHEADERS, PSHEADERS	.pro, .enc
tex	TEX sources	TEXINPUTS	.tex
texpool	TEX program strings	TEXPOOL	.pool
TeX system documentation		TEXDOCS	
TeX system sources		TEXSOURCES	
tfm	TEX font metrics	TFMFONTS, TEXFONTS	.tfm
type1 fonts	PostScript Type 1 fonts	T1FONTS, T1INPUTS, TEXPSHEADERS, DVIPSHEADERS	.pfa, .pfb
type42 fonts	PostScript Type42 fonts	T42FONTS	
vf	virtual fonts	VFFONTS, TEXFONTS	.vf
web2c files	Web2c support files	WEB2C	

and KPSEWHICH will respond like this:

```
c:/texmf/tex/latex/base/article.cls
```

Note that you have to specify the file name *extension* because `.cls` is default for LATEX but KPSEWHICH can't possibly know that you are searching for a LATEX file. If KPSEWHICH doesn't write any output then apparently it didn't find the file you wanted to locate.

If you want to know which directories are scanned to find DVIPS configuration files you could enter

```
▷ kpsewhich -show-path="dvips config"
```

You need the double quotes here because the name contains spaces. KPSEWHICH will respond with, e.g.

```
.;!!c:/texfiles/texmf/dvips//;!!D:/TEXMF/dvips//
```

Troubleshooting tips

If you think something is really wrong with you Web2c installation you should use KPSEWHICH to check your configuration. We will assume you have installed the system in `c:/tex`, and give you a few tips:

- Enter the command

  ```
  ▷ kpsewhich -expand-path=$SELFAUTOPARENT
  ```

  The output should read something like `c:/texfiles`
- Enter the command

  ```
  ▷ kpsewhich -expand-path=$TEXMF
  ```

  The output should read: `c:/texfiles/texmf`
- Enter the command

  ```
  ▷ kpsewhich -expand-path=$TEXMFCNF
  ```

  The output should read:
  `c:/texfiles/texmf/web2c;d:/texmf/web2c;`
- Enter the command

  ```
  ▷ kpsewhich -expand-var-$TEXINPUTS
  ```

  The output should read:
  `.;c:/texfiles/texmf/tex//;d:/texmf/tex//`
- Check if you have other TEX-related variables set in your environment that may unintentionally affect your TEX system's behavior. Remember that environment variables always override the settings in `texmf.cnf`.
- Check the values of some common TEX files:

  ```
  ▷ kpsewhich cmr10.tfm
  ```

Table 13.3: Kpathsea's debug values

Value	Meaning
−1	Report *all* debugging information.
1	Report on file searches: if `ls-R` databases are up to date you should get no output.
2	Report on *all* file look ups, including `aliases`, font aliases and configuration file values.
4	Report on file openings and closings.
8	Report general path information on each file Kpathsea is asked to search for: useful to determine how a particular path was found (`texmf.cnf`, `config.ps` or an environment variable, etc.)
16	Report the directory list corresponding to each path element that Kpathsea searches.
32	Report on each file search: the *name* of the file searched for, the *path* searched in, whether the file must exist or or not (e.g. `.vf` files need not exist), and whether we are collecting all occurrences of the file in the paths or not.
64	Report the value of each variable that Kpathsea looks up.
128	Activate debugging output for the GSFtoPK program.
256	Let any `mktex*` program write debugging information.
512	Let any `mktex*` program write debugging information on its internal functions

should return: `d:/texmf/fonts/tfm/public/cm/cmr10.tfm;`

▷ `kpsewhich latex.fmt`

should return: `d:/texmf/web2c/latex.fmt`

- If everything so far checks out things may get a little tougher. You may need to play with KPSEWHICH's debugging options. See table 13.3 for details on debugging options.

You can use debug values like this:

▷ `latex -kpathsea-debug=2 myfile`

or

▷ `kpsewhich -debug=-1 cmr10.tfm`

Though unlikely, it is not entirely impossible that you have run into a bug in one of the Web2c programs. But before you send in a bug report you may want to consult

other Web2c users who subscribed to the Web2c and Kpathsea mailing list `tex-k`. See chapter 5 for details.

If you feel certain you have discovered a bug, you should write a bug report. The general principle is that a good bug report includes all the information necessary for reproduction. Therefore, to enable investigation, your report should include the following:

- The version number(s) of the programs involved and of Kpathsea itself. You can get the former by giving the sole option `--version` to the program, and the latter by running `kpsewhich --version`.
- The hardware and operating system you are using.
- The log of all debugging output, if the bug is in path searching. You can get this by setting the environment variable `KPATHSEA_DEBUG` to `-1` before running the program. Please look at the log yourself to make sure the behavior is really a bug before reporting it. Perhaps 'old' environment variables are causing files not to be found, for example.
- The content of any input files necessary to reproduce the bug. For bugs in DVI-reading programs, e.g., this generally means a DVI file (and any EPS or other files it uses). TEX source files are helpful but the DVI file is essential because that is the actual program input.
- Any additional information that may be helpful in reproducing, diagnosing or fixing the problem.

A convenient and efficient way of packaging (possibly binary, non-ASCII) files for electronic mail is 'zip'. It is available on the CDROM in directory `\TOOLS`.

Bug reports should be sent via electronic mail to `tex-k@mail.tug.org`, or by postal mail to 135 Center Hill Road / Plymouth, MA 02360 / USA.

If you feel your files are too big to send by email, you can 'FTP' them to `ftp://ftp.tug.org/incoming` (that directory is writable, but not readable to clients). Please make sure you put together an understandable package. Also make sure your name and address, preferably email, are included.

13.9 The TEX Directory Structure

TEX is by nature an exceptionally easily portable system: it runs on virtually all operating systems. One unfortunate side effect of this flexibility is that there has been no single 'right' way to install it. This has resulted in many sites having different installed arrangements.

So what, you might think. The answer is: if we were to have a standardized arrangement it would make maintenance and exchange of files much easier. As you know by now many of the files in a TEX system are implementation-independent, so no matter on what kind of computer a TEX system is maintained, the arrangement should be useable

on any other system. Provided the other system makes the same assumptions on the location of files...

This is where the TEX Directory Structure (TDS) kicks in. It defines a directory hierarchy for macros, fonts, and the many other implementation-independent files. Adhering to TDS enables the TEX world to maintain a single reference system that can be used by any (TDS-compliant) TEX system, regardless of operating system or implementation. It goes without saying that Web2c, a widely used implementation on several operating system, cooperates seamlessly with TDS.

The TDS is an initiative of the TWG-TDS (*TEX Users Group Working Group on the TEX Directory Structure*). For details on the ideas behind this structure you should read the TWG-TDS's report (cdrom).

As a *user* you will probably notice very little of the TDS. But if one day you want to expand or compress your system you will find that TDS makes it much easier to identify the parts involved. You could even update your system simply by inserting a new TEX CDROM, even if that CDROM was originally made for Unix systems. Simply because it is TDS-compliant.

Chapter 14

Managing and tuning the installation

14.1 TEX on Microsoft Windows

Thanks to F. Popineau we can use the superb Web2c TEX implementation on the 'Win32' platform. This platform currently consists of three operating systems: Windows 95, Windows 98 and Windows NT. Soon Windows 2000 will appear, which will be the successor of all three.

Technically speaking 'Win32' is a programming environment with true 32-bit flat memory addressing, multitasking and multithreading. On top of that a graphic user interface is supported that allows you to give programs the typical Windows look and feel.

Most Win32 programs use the graphic user interface but that doesn't mean that non-graphic programs are not 32-bit. Most probably some Windows programs that you are using are 16-bit (like in the old days of Windows 3.x). 16-bit programs are not necessarily inferior to 32-bit programs, but they will lack a few features only available to 32-bit programs. Probably the most noticeable difference is that only 32-bit programs can handle files using 'long file names'.

There are a number of 32-bit programs around that do not use the graphic user interface. They are called 'console applications'. A typical Windows 95/98 system e.g. will contain a program called `xcopy32.exe`. This program is a true Win32 console application. On Windows NT this program is called simply `xcopy.exe`.

16-bit Windows programs usually write their configuration parameters in a so-called `.ini` file which is often stored in the Windows system directory. 32-bit Win32 programs have the option to write an `.ini` file or to use the Windows *registry*, which is what Microsoft recommends.

An important feature of all Windows programs, graphic or console, 16-bit or 32-bit, is the ability to share executable program code in so-called 'dynamic link libraries', DLL for short. This allows for more efficient memory usage because DLL's are loaded only once, but can serve many programs simultaneously.

14.1.1 File systems and file names

On computers running any of the operating systems Windows 95, 98, or NT multiple file systems are supported:

MS-DOS FAT File names are restricted to 8 characters; file name extensions to 3 characters ('8.3'); all names are in uppercase. This file system is very inefficient on large disk partitions because the cluster size (the minimal disk spaces that even the smallest file will occupy) can easily become as big as 16, 32 or even 64 kilobytes. TEX systems typically contain many small files, so you can easily get in trouble here.

Protected mode FAT Long file names are possible, and they are case-insensitive.

FAT32 Similar to protected mode FAT, but with support for much smaller cluster sizes, thus more efficient on large hard disks.

NTFS Stands for 'NT File System' Windows NT's native file system. Long file names are possible and they are case-sensitive. Very efficient on large disks.

ISO-9660 CDROM '8.3' file names in uppercase, much like MS-DOS FAT, but even more restricted: file names can only consist of the characters `A`–`Z`, the digits `0`–`9` and the underscore `_`. File name extensions cannot contain digits. Directories cannot have any extension. Cluster size is usually quite small, comparable to NTFS.

CDROM with 'Joliet' extensions Similar to protected mode FAT. The 4allTEX CDROM supports this file system. Note that to operating systems that do not support 'Joliet' extensions (most notably MS-DOS and Windows NT prior to version 4.0) such a CDROM will look like any ordinary ISO-9660 CDROM.

The Web2c TEX system consists entirely of Win32 programs. All of them are console applications, except for the DVI driver WINDVI, which has a graphic user interface.

4TEX, 4Spell and 4Project are all graphic Win32 programs. Most of the program that are executed by 4TEX are also Win32 programs, but a few older MS-DOS and 16-bit Windows programs are also among them. In general, this mixture should cause no problems. Beware, though, that, e.g., long file names can play funny tricks on non-Win32 programs.

Although Win32 programs support long file names, we advise to be very careful in choosing file names. As we will explain later, some characters in TEX documents

have a special meaning. Therefore, it is a bad idea to use such characters in file names. We strongly recommend that you follow these rules:

- never use spaces within file names or paths
- make sure there is exactly one dot in a file name, no more, no less
- Use only the characters `a`–`z`, `A`–`Z`, `0`–`9`, `-` (hyphen) and `_` (underscore).

14.1.2 Installation issues

In some important respects the Web2c TEX system installation differs from most other Windows programs:

- None of the Web2c programs write an `.ini` file — not in their own directory, certainly not in the Windows system directory. Only WINDVI writes a configuration file, but you can specify its location yourself.
- None of them use the Windows *registry* in any way.
- No dynamic link libraries (DLL files) are added to the Windows system directory. All necessary DLL files are read from the Web2c program directory.
- No existing dynamic link libraries are replaced.
- No system font files are added or replaced.
- When running, no write-access is required in any sensitive area of the system.
- Rebooting your computer is not required.

Doesn't that sound like heaven? Of course the system does use configuration files, but they can be installed anywhere. The system will have its default configuration file which can be overruled by a configuration file in a user's private directory. This is more in the tradition of Unix applications. And frankly, we believe it is far superior to the strategy of common Windows programs.

Other programs that come with 4TEX can and will use the Windows registry, write more `.ini` files, install `.dll`'s, make changes in `autoexec.bat`, etc.

14.2 4TEX installation

During installation of 4TEX some changes will be made to your system. Some of them are necessary to make 4TEX run, others to make the Web2c system run, and a few more to make related programs run. Below we will describe what happens during installation.

- The 'Tahoma' fonts (`tahoma.ttf` and `tahomabd.ttf`) will be installed. These fonts usually come with the Microsoft Office package, so they may already be present on your system. They are not essential, but they are the recommended fonts for the 4TEX menus.

- A few `.ini` files (`gsview32.ini` and `latexcad.ini`) will be copied to the Windows system directory, because otherwise the corresponding programs simply won't work (well).
- The file `4TEX.INI` is written to either the directory where 4TEX is installed or the Windows system directory.
- A TEX directory tree is built (see section 3.2).
- A few example TEX files may be copied to the user's TEX directory.
- Some `.lst` and `.4spell` files are copied to the user's TEX directory, so he/she can personalize some aspects of the 4TEX system.
- An entry in the Windows registry is made to indicate that 4TEX is installed. This entry is not essential for running 4TEX, though. It only makes it easier to deinstall the program, and it prevents the installation program from starting every time you insert the 4allTEX CDROM. The entry can be found in `HKEY_LOCAL_MACHINE\Software\4TeX\5.0`, where two keys are made: `InstallFrom` and `InstallTo`. They contain the name of the folder *from* which the program was installed (usually the 4allTEX CDROM) and the name of the folder on the hard disk *to* which files were copied.
- Files with extension `.tex` may be associated with 4TEX.
- Shortcuts to 4TEX may be added to the Desktop, to the Start menu and to Programs.

Note that 4TEX can 'speak' several languages. It offers a menu from which you can select a language (see section 9. The language specific strings are built into the program, so only the authors of the program can change them, or add other languages. When selecting a language, 4TEX will generate a list based on the files `*_UTILS.LST` that it can find. If you don't need, say, Polish, you can delete `PL_*.LST` from the system.

You should never delete any `.lst` files that start with `US_`. These are used as defaults in case a language-specific file is not available.

14.3 Network usage

4TEX was designed to run as a network application. It can run perfectly in a completely (well, almost completely) read-only environment. Any user who has access to 4TEX which is installed on a network can start using it, without any need to store lots of files on his/her own system or network drive.

Below we will give you some hints on setting up 4TEX on a network in which any number of users can work with the program.

4TEX.INI

In a network installation you may want to set up a read-only version of `4TEX.INI`. This file can be installed in the same directory as the 4TEX program itself. 4TEX will al-

ways first check if the Windows system directory (indicated by the environment variable `%windir%`) contains `4TEX.INI`. If not, it will check its own program directory.

In case 4TEX uses the read-only `4TEX.INI`, it will of course not be able to save any changes that any user may have made during his or her 4TEX session. By default 4TEX will issue a warning message if writing to this file failed. You can suppress this message by setting the value of the parameter `ShowINIErrors` to 0 in `4TEX.INI`:

```
[System]
ShowINIErrors=0
```

Warning messages that are meaningless can thus be suppressed.

Environment variables

4TEX sets a few environment variables that you should be aware of. If these variables are not set or set incorrectly, things may not work properly. 4TEX does not need any global variables, instead it will set variables as specified in `4TEX.INI` (see section 12.3).

The environment variable `HOME` can be troublesome. The DVI previewer WINDVI and the print program DVIPS use this variable, but other programs such as EMACS (not part of 4TEX) use it too, for other purposes. Do not set this variables unless you really need it. See also section 13.6.10 and table 13.2 (page 250) for a list of environment variables that are meaningful to Web2c programs.

.INI files

4TEX will read its initialization file `4TEX.INI` from the Windows directory (specified by the environment variable `windir`) or in the directory where `4tex.exe` itself resides. If the file cannot be found, 4TEX will start in initialization mode, which is probably not what you want.

A few programs that 4TEX uses will require or write their own `.ini` file in the Windows directory. LaTeXcad and GSview are examples. Other programs may want write-access to their own directory (e.g. BibEdit), which may cause problems. There are no standard procedures to circumvent these problem, you will have to be creative if such problems occur.

Limiting access

The basic functions that can be configured from the 4TEX menus are quite harmless. The advanced options (see section 8.11), however, is less harmless. Check out its functions and decide whether you want your users to have access to these functions. If not, delete the entry from the `*_UTILS.LST` files. Likewise, if you feel that any of the other items in the utilities list should not be there on your network, simply delete it. Naturally you can also add any of your own utilities.

Network printers

Network printers can be configured in the `*_PRD.LST` files. In section 12.2, page 146, you can find detailed information on setting up network printers for 4TEX.

MED configuration

The editor program MED (see section 6.2.2) can be run by multiple users from the network. However, any changes to the editor's configuration are not saved. Each time the program is launched, it will use its default settings. If users need to store their own configuration, they will have to copy the whole MED directory (`bin\win32\med`) and subdirectories to a directory where they have read-write access. Naturally the path to the MED editor should then be changed accordingly in `4TEX.INI`

4Spell

If you want to allow users to set up their own dictionaries, lists of own words, autofix lists, etc., you should install all `*.4spell` and `*_SPELL.LST` files in a users' read-write directory. Make sure that you set the variable `UserSpellDir` in `4TEX.INI` (section `[4Spell]`) accordingly.

14.4 Trouble shooting

While working with 4TEX you may find that some program, e.g. the TEX compiler complains about a file that apparently it cannot find. Or another program may refuse to work. Or Windows does not behave as you expect it to.

A number of things can go wrong in a complex system such as 4TEX in which so many programs interact. Many of them have their own peculiarities when it comes to configuration or usage. We have tried to shield you as much as possible from weird error messages and warnings. However, problems may still occur, so it seems worthwhile to give you a few more tips on handling problems. See also section 4.1 for hints on dealing with TEX error messages and warnings.

Trouble finding input files

If the TEX system seems to have trouble finding input files, there could be something wrong with your configuration. Here are a few methods for finding out how your system is configured. These may give you clues to locate the problem. Open a DOS-box and then:

- Check the current *environment* settings by entering:
  `set > env.set`
  Then check the contents of the file `env.set` for any variables that are used by Web2c programs.

- Enter the command:
  `kpsewhich texmf.cnf`
  This will tell you which configuration files the Web2c programs are actually using.
- Enter the command:
  `kpsewhich --expand-var=$TEXMF`
  This will tell you which directory trees Web2c programs will searched for locating any resources.
- Enter the command:
  `kpsewhich --expand-var=$TEXMFMAIN`
  This will tell you what the main TEX tree is.
- Enter the command:
  `kpsewhich --expand-var=$TEXMFCNF`
  This will tell you in which directories Web2c programs will look for configuration files.

If you still have no clue why an input file can't be found but you are sure that it exists, then try the following two commands:

```
kpsewhich yourfile
kpsewhich --must-exist yourfile
```

If the fist command returns nothing, but the second one does, then most probably an `ls-R` index file needs updating. You can check which `ls-R` indexes are read by entering:

```
kpsewhich --expand-var=$TEXMFDBF
```

Use the MKTEXLSR program to regenerate indexes.

Windows randomly searches the floppy drive

You may experience that Windows accesses the floppy drive (or drives) every time an application is launched. This problem is not confined to any TEX related program or to the use of Explorer. Rebooting does not fix the problem. Possible solutions are listed below:

- Clear the documents menu.
- Clear unwanted entries from the Start menu's Run command.
- Check for any viruses on your system.
- Search the hard disk for all `.pif` files that point to programs on a floppy drive.
- Check `HKEY_CLASSES_ROOT/CLSID` in the Windows *registry* for any references to `.ocx`'s or `.dll`'s referenced on the floppy.
- If you are using Norton Navigator: clear Norton Navigator's run history (or disable the run history all together). If you are noticing this behavior with only a particular

application, you should clear the history list for that application. Maybe you should download a bug fix from `ftp://ftp.symantec.com`.

- Disable (temporarily) your anti-virus program. If that helps, get an update or bug fix from your dealer.
- Disable the FastFind feature from the Microsoft Office package.

PART IV

The many roads to TEX

Chapter 15

What we mean by TEX

In the next three chapters we will describe Plain TEX in general, and then move on to two general-purpose dialects: LATEX and CONTEXT.

But first we would like to say a few words about learning TEX in general. We must warn you that learning TEX is not something you can do in one afternoon — unless you're a genius. Most likely it will take days or weeks before you really get the spirit and understand how TEX works. You will get plenty of error messages that will seem completely incomprehensible. The learning curve is quite steep which means you will have to learn a lot in the initial phase in order to get anything working. Nevertheless, once you have seen many of them you will begin to see a pattern. It is all very logical.

Our advise is: go with the flow, don't let the errors and mistakes get you down, and soon you will see the light.

15.1 Writing in TEX

As explained before, when we talk about 'TEX' we could be talking about a typesetting system, a set of computer programs or a programming language geared for typesetting.

In this chapter we will focus on the programming language. We will discuss three macro packages that define many commands at the user level that you will be entering from your keyboard in order to typeset your own texts.

Most of the macro packages that are available world wide are based on the original macro package called 'Plain TEX' written by D. Knuth. We will call such macro packages 'dialects'.

A typical TEX document contains many unexpected characters such as backslashes, curly braces and square brackets. This is where TEX differs from most common word processors. What you type is not only what you want printed, you also supply *markup* which is essentially an instruction for the compiler.

Therefore you need to distinguish between 'real' text and 'markup'. Because the TEX language is so rich many characters have a 'reserved' function, just like e.g. `WHILE` is a reserved word in the Pascal programming language. In *TEX speak* we call such characters *active* characters. Typical examples are: `\ { } # $ & _ ^ %`. Be warned: it is all too easy to write `100%` instead of `100\%`. But you will soon get used to this, and of course we will explain all of these special cases.

15.2 Learning a 'dialect'

Whatever dialect you choose, you will find that they all have a common base. Most dialects, including the ones we discuss here, rely heavily on 'Plain TEX'. Almost all commands defined in Plain TEX can be used in those dialects as well. Therefore you may well have to refer to the Plain TEX introduction even if you decide to use LATEX or CONTEXT. Basic knowledge of Plain TEX is always required.

Plain TEX is a very basic yet extremely powerful set of macros that will definitely appeal to those who want to understand TEX in full detail or those who want to do everything themselves. Its power is well demonstrated by e.g. *The TEXbook* by D. Knuth. That book was typeset with Plain TEX and only a handful (well, about 700 lines of code) of macros. If you like, you can follow in Knuth's footsteps using his `manmac` macros. They will take you very far.

LATEX is currently the most popular and most widely used TEX macro package. It builds on Plain TEX and adds many features to make your life as a TEX user more comfortable. Using LATEX you find yourself doing a lot less programming yourself. So if you would rather use already available macros than writing them yourself LATEX could be a good choice. For almost any reasonable feature you might need someone has already written LATEX macros that you can use.

CONTEXT is a fairly new package in the global TEX world but has been in use by a small group for several years. From a functional point of view it is similar to LATEX. A very unusual feature of CONTEXT is its support for English, German and Dutch interfaces.

So what is the difference between LATEX and CONTEXT? Whereas LATEX is designed as a relatively small kernel that can load so-called 'packages' in order to provide the necessary functionality, CONTEXT is much more monolithic. CONTEXT has most of the functionality that LATEX supports through 'packages' already built in. This strategy has both advantages and disadvantages. The major advantage is that everything in CONTEXT is very consistent. In LATEX any package can define its own command syntax. An even bigger problem with packages is that they may interfere with each other. In CONTEXT such interference is much less likely. A disadvantage of the CONTEXT approach could be that the system is more demanding with respect to memory. But you should not fear for problems here. 8 megabytes of memory will still suffice, which is peanuts compared to any modern word processor. You may notice that CONTEXT also runs a bit slower than LATEX or Plain TEX. That is the price you pay for integration at this level.

In combination with PDF, ConTeXt is probably one of the most powerful systems on earth. It allows you to generate interactive documents with hypertext links, Javascript and more... Creating good looking text books and manuals with ConTeXt is surprisingly easy compared with LaTeX, but LaTeX can handle highly demanding scientific articles a little better, and it runs faster. A comparison of LaTeX and ConTeXt is given by T. Hoekwater (cdrom).

The trio Plain TeX, LaTeX and ConTeXt is by no means an exhaustive listing of all currently available dialects. On the CDROM you will find other dialects such as blue, eplain, texinfo, lollipop and lamstex, to name but a few. Some dialects are popular in certain areas, others in other areas; some are widely used, some are only rarely used (any more), if at all. We don't want to overwhelm you with details on all of them, so we picked only three of them which we think will cover the needs of a wide range of users.

15.3 Summary of features and characteristics

In table 15.1 we tried to summarize features and characteristics that we think might help you in determining how to proceed when choosing a dialect. Note once more that you should think in terms of the *job* that you are trying to accomplish. So you may even decide to use LaTeX for a multi-author science project and ConTeXt for interactive brochures. Or maybe you choose Plain TeX to obtain a better understanding of the basics, before you start writing professional documentation. Take your pick, but do take one at a time.

If you already know you want to use LaTeX or ConTeXt, not Plain TeX, as your basic language, you could study the Plain TeX more superficially and perhaps skip parts of it. After reading the introduction to the language of your choice you may return to the Plain TeX introduction for those things that were not covered or that are unclear.

15.4 The future

We live in an ever-changing world. And TeX is changing along with it. Because of TeX's intrinsic ability to adapt to almost any new development TeX's future seems secure. Or does it?

We have seen TeX integrate with the World Wide Web in quite astonishing ways. In fact, TeX (LaTeX to be exact) documents could be automatically converted to HTML long before any of the current word processors learned how to do that.

When Adobe launched PDF, TeX was again the first that could automatically generate PDF documents with all kinds of cross-links, hyperlinked indexes and more. Currently TeX (or rather PDFTeX) is one of the very few systems that can generate PDF without any assistance from commercial software.

Table 15.1: Comparing Plain TEX, LATEX and CONTEXT

	Plain TEX	**LATEX**	**CONTEXT**
learning curve	very steep	steep	less steep
error recovery	hardly	poor	reasonable
documentation:			
– online	poor	good	excellent
– books	good	good	poor
consistency	good	poor	high
tweakability	high	difficult	high
extendibility	very good, but not easy	very good	little need for
command language	English	English	English, Dutch, German
number of users	a lot	majority	few
memory usage	very low	moderate	high
speed	very fast	fast	slow
files loaded during runtime	few	many	few
structure	bare bones	modular	monolithic
PDF-support	no	yes	excellent
maintenance and support	none, it is 'frozen'	LATEX-team does kernel, 'world' the rest	only by author + task force
license fee	none	none	non-commercial: yes; others pay 'decently'

But is this convincing enough to make you 'invest' in TEX? Let us have a closer look at current developments and try to predict TEX's role in the near future.

Currently there are developments both at the TEX *language* level and at the TEX *program* level. At the language level we can see new macro packages and formats appearing. They build on many years of experience, but they also take new approaches and take advantage of developments in other realms. The appearance of PDF was such a god-sent gift. It gave TEX the opportunity to generate interactive documents that go far beyond anything that anyone could have imagined 10 years ago.

Macro packages such as CONTEXT realized this exciting new area and took advantage of it. For LATEX packages were developed that made LATEX hyperlink-aware as well. Obviously the time was right for this, because the World Wide Web was growing exponentially at the same time. Convertors from TEX to HTML soon appeared, which is no surprise if you take into account how similar the two are or at least can be. When XML, the successor of HTML, takes over, convertors to XML will be ready. At the same time

interfaces are being build that enable TEX to *read* HTML, XML or SGML, which opens even more perspectives.

At the TEX *program* level time has not stood still either. Currently there are at least three possible successors to the current TEX program.

We have already mentioned PDFTEX, an extended version of the TEX program that can generate PDF output. Another extended version of TEX is called ε-TEX. This version is completely compatible to current TEX, but it adds several new commands and it extends several existing commands. These extensions make the life of TEX programmers somewhat easier, and some extension may also help improve the typographical quality of the output. A very recent development is the integration of ε-TEX and PDFTEX into one program called PDF-ε-TEX.

Then there is OMEGA. This is an extended version of TEX that does away with all current intrinsic memory limitations. In the current TEX program you can have, for instance, no more than 255 counters because the counter index is only 8-bits wide. In OMEGA all such 8-bits constants have been widened to 16-bits, which means that you can have, for instance, over 65,000 counters. OMEGA uses Unicode natively, which makes typesetting of (arbitrary combinations of) radically different languages relatively easy. Another new feature in Omega is its ability to typeset in any direction, meaning from left to right, right to left, top to bottom or bottom to top, as required by certain languages.

Both OMEGA and ε-TEX use the original TEX sources and add their changes own changes to it. NTS takes a different approach. This program is currently being implemented from scratch in the programming language Java. The first version will behave identical to standard TEX. The difference with the standard TEX program will be that adding features or changing current features will be much easier. The object-oriented approach of the program and the availability of Java on every operating system should guarantee this. New features that could be added are, for instance, different typesetting routines, multiple (new) output formats, extra routines that make complex TEX macros obsolete, a different input routine that reads Unicode or other special formats. The sky is the limit.

Exciting as this may sound, its also poses a threat to NTS. Somehow we have to make sure that NTS doesn't become a moving target that gets out of control. An important feature of standard TEX is that it is *standard*. You can almost blindly rely on it to do its job as it always did. When NTS takes off, this could change. But at the same time, change is needed to keep up with modern requirements...

It is up to the whole TEX community to preserve the high standard of typeset output, stability and reliability of TEX in any of its successors. In the past 20 years the community has evolved into a well-organized world-wide network of enthusiastic people that share a common sense for what is good and what is bad. This doesn't mean that the whole TEX community agrees on any particular issue, but we do feel confident that the right decisions will be made, following humbly in the footsteps of Don Knuth, TEX's father, but at the same time exploring new directions, to boldly go where no typesetting system has gone before.

Chapter 16

Plain TEX: Knuth's approach

'Plain TEX' is a complex set of macros and other constructions that can be used to typeset documents. Many of these constructions are fairly 'low level', meaning that they are not supposed to be used at the level of writing a book, but rather at the level of designing a *markup language* for writing a book. LATEX and CONTEXT (see chapter 17 and 18) are such 'higher level' languages based on Plain TEX.

Nevertheless, no one will stop you from using any Plain TEX command anywhere in your documents. In fact, if your demands are not extremely high you can write beautiful documents using only a few of the features provided by Plain TEX. In this chapter we will focus on a small subset of Plain TEX's features that can be used to do just that. Writing a 'higher level' language based on Plain TEX is beyond the scope of this book. Nevertheless, we will give a brief overview of the basic TEX programming tools.

For further reading we suggest that you read M. Doob's *A Gentle Introduction to TEX, A Manual for Self-study* (cdrom), V. Eijkhout's *TEX by Topic*, or R. Seroul and S. Levy's *A Beginner's Book of TEX*. Many interesting technicalities for programmers can be found in D. Salomon's *The Advanced TEXbook*. The ultimate reference is still D. Knuth's *The TEXbook*.

16.1 General concepts

In Plain TEX all *markup* is done using commands that start with a backslash \. In Plain TEX there are about 900 of them. About 300 of these are called 'primitive' because they can't be decomposed into simpler functions. All other commands are ultimately defined in terms of primitives.

Note that some actions are not (necessarily) triggered by an explicit command but by implicit markup. E.g., the end of a paragraph can be indicated by an empty line (implicit) or by the command `\par`.

Some commands take parameters, others don't. Some take one parameter and some take more than one. In some cases the parameters are special keywords or specific objects. Sometimes you are supposed to supply parameters surrounded by curly braces `{}`, sometimes no such delimiters are required or allowed. In other words: you need to know the syntax of any particular command if you want to use it.

16.1.1 Control sequences

The reason for this seemingly chaotic scheme is that although all control sequences start with a backslash, there are significant differences. We need to be more strict in terminology in order to prevent confusion. In TEX the following control sequences have to be clearly distinguished:

macros Also called 'definitions'. They often take one or more parameters that usually need braces. Macros usually perform an action.

token lists These don't perform an action like macros do, they are used rather to store lists of 'tokens' (which can be control sequences) that can be used in macros.

dimensions These hold a size value that can be used to specify e.g. the size of a page or the amount of white space between two paragraphs.

counters These hold a numerical (integer) value that can be used e.g. to store chapter or page numbers. You can do simple calculations with them.

Actually, there are a few more types of control sequences, but for now this is more than enough.

Plain TEX defines lots of control sequences, but you can also define your own. In section 16.15 we will show some examples. For now it suffices to understand the distinction between them. In the course of this chapter you will find examples of each of them.

Now that you understand what control sequences are it is time to introduce a few rules that you need to be aware of:

- The name of a control sequence consists of the characters a–z and A–Z. Punctuation characters, spaces, numerals, etc. are not allowed.[1]
- The end of a control sequence name is marked by any character that can't possibly be part of the name (see previous item). In practice this means that a space, new line, brace, numeral or backslash (starting another control sequence) will mark the end.

1 In fact this is not entirely true: you *can* have control sequences containing any kind of character, but this is better left to experienced TEX programmers.

- Control sequences are case sensitive. So `\BYE` is not the same as `\bye`. This feature can be used to define a lower and an uppercase version of a control sequence. In Plain TEX e.g. Greek letters use this intuitive method: `\Omega` produces Ω; `\omega` produces ω.

16.1.2 Grouping

Just like the backslash, the characters `{}` (curly braces) have a special meaning in TEX. They are used to indicate the begin and end of a so-called *group*. The most important cases in which groups are used are:

- To make *local* changes in a document. You could, e.g. start a group, choose a large bold font, typeset a few words, and then end the group. All settings from before starting the group will be restored.
- To delimit the parameter(s) of a control sequence. If a control sequence expects a parameter it usually accepts the first token it finds and nothing more. Suppose you want to use a macro called `\mymac` that expects two parameters. If you write `\mymac{one}{two}` all will be fine. However, if you write `\mymac one two` something completely different will happen. The parameters that `\mymac` finds are `o` and `n` respectively. The rest (`e two`) will probably be typeset as normal text.
- Spaces that appear after a control sequence are always gobbled up. Therefore braces can also be used to stop TEX from absorbing spaces. The example below will show how this can be achieved.

TEXTEX! TEX! TEX ! TEX ! TEX ! TEX !

```
\TeX \TeX     !
\TeX   !
\TeX{} !
\TeX      {}  !
{\TeX} !
\TeX\  !
```

As you can see the space after 'TEX' can also be enforced by a command that produces a 'visible space'. Because white ink on white paper doesn't work very well we usually show such required spaces differently in this manual: `\␣`

16.2 Document structure

A good way to learn a language is by studying examples. Therefore we will show many examples in this introduction. They all look like this: on the right you can see the actual input, on the left the output that it will produce. Here is how to write the unavoidable 'Hello world!' in Plain TEX:

Hello World!

```
Hello World! \bye
```

In this example we only needed the control sequence

```
\bye
```

which tells TEX that this is the end of the document. You may wonder why TEX needed this explicit end statement at all. Well, it actually can be very handy to end a document somewhere *before* the end of the file. All text *after* the `\bye` command will be ignored.

Although there is no limit to the size of files that TEX can handle, it is a good idea to keep files relatively small because that is more practical. Plain TEX provides the command

```
\input file
```

to include the file *file* within the current file. If no file extension is specified, `.tex` will be assumed. If no path is specified the current directory will be assumed. If the file is not there, the usual input path will be searched. Note that if you specify a path you should always use *forward* slashes `/`, not backslashes `\`. Also note that *no* curly braces should be used here. The included file can in turn include other files if you wish. A book such as the one you are reading now could be set up like this:

```
\input bookmacros
\input booklayout
\input cover
\input preface
\input intro
\input part1
\input part2
\input part3
\input part4
\input appendix
\input bibliography
\input index
\bye
```

Included files don't need an explicit end statement such as `\bye` in the main file. However, you can explicitly end an included file anywhere by using the command

```
\endinput
```

All text after that command will be ignored.

You may want to structure your document using sections. Plain TEX provides the command

```
\beginsection title
```

Beware that the end of the title is marked by an empty line.

The example below shows the effect of some correct and incorrect methods to use this command, although all of them are syntactically correct.

```
\beginsection My first attempt
Not so good.

\beginsection Try again

Better. Note that the second empty line
caused indentation.

\beginsection
Getting better and better

Right!
```

My first attempt Not so good.

Try again

Better. Note that the second empty line caused indentation.

Getting better and better

Right!

As you can see TEX takes care of spacing and fonts. Appendix E of D. Knuth's *The TEXbook* shows how you could define similar commands for starting chapters, subsections, etc. The 'BLUe' format (by K. van der Laan) and the 'eplain' format (by K. Berry and S. Smith) are based on Plain TEX and provide such commands and many more. Documentation on both can be found on the cdrom.

16.3 Characters, words and paragraphs

Now that we have established the basic construction of a document we can focus on the content. We have already seen that in TEX all characters are not equal. The backslash is a special case because it flags the start of a control sequence. In other words, it will not be typeset as a backslash.

There are a few more characters that have a special meaning. In *TEX speak* we say that they have a different *category code*. We will explain the concept of category codes in section 16.15.4. For now, just remember that if you need such a character in a plain text paragraph (i.e., not in a mathematical formula) you should type it as shown below.

```
\& \# \$ \% \_
$\backslash$ $<$ $>$ $\{$ $\}$
```

& # $ % _ \ < > { }

As you can deduce from the example the $ also has a special meaning. It is used to start and to end a different *mode* in which different rules apply. This mode is called *math mode* because it is mainly used for typesetting mathematical formulae. We will describe the ins and outs of math mode in section 16.16. Note also that the space after \&, \#, \$, \% and _ is *not* gobbled up, so there is no need for extra braces here.

Characters make up words, and words together with spaces make up paragraphs. Spaces are somewhat special in TEX. In normal text one space is as good as any number of spaces: to TEX they only signal 'space between words', the size of which will be

determined by TEX, not by the number of spaces you typed. The same goes for 'Carriage Return': it is equivalent to a space, so it makes no difference where or how many of them you supply within a paragraph.

```
This   is
    just         an example
to show you how spaces
work.
```

This is just an example to show you how spaces work.

In section 16.6 we will show how to create horizontal and vertical blank space.

A paragraph is ended implicitly by an empty line, or explicitly with the command

```
\par
```

In the example below we show both techniques. Remember that multiple `\par` commands and multiple empty lines count as only one.

```
This is the first paragraph. Remember that
multiple empty lines count as only one.


And this is the second paragraph.
\par
So this must be the third paragraph
because of the paragraph command between
this one and the previous paragraph.
```

This is the first paragraph. Remember that multiple empty lines count as only one.

And this is the second paragraph.

So this must be the third paragraph because of the paragraph command between this one and the previous paragraph.

16.4 Page layout

Now let us see how we can set up pages that contain all these characters, words and paragraphs.

The width of a page is determined by the value of the dimension `\hsize`. The height of a page is determined by the value of `\vsize`. With these you actually define the area accessible to any normal text. Headlines and footlines are not included. We will discuss them in a moment. Here is an example of setting up page dimensions:

```
\hsize=150mm
\vsize=210mm
```

Dimensions can be specified in a variety of units. Table 16.1 gives an overview. Note how the = sign is used in this example. In fact it is optional so you could write `\hsize150mm` but we recommend that you use the = sign always to avoid confusion with macros. If you mistook `\hsize` for a *macro*, you might accidentally write `\hsize{150mm}` but this is illegal.

Table 16.1: TEX dimension units

Name	Unit	Size
point	pt	72.27 pt = 1 in
pica	pc	1 pc = 12 pt
inch	in	1 in = 25.4 mm
big point	bp	72 bp = 1 in
centimeter	cm	1 cm = 10 mm
millimeter	mm	1 mm = 0.001 m
didot point	dd	1157 dd = 1238 pt
cicero	cc	1 cc = 12 dd
scaled point	sp	65536 sp = 1 pt
x-height	ex	height of letter 'x' in current font
M-width	em	width of letter 'M' in current font

The position of the page on the physical page can be set with the following commands:

`\voffset` = *dimen*

This command defines the vertical offset (displacement) relative to the origin of the physical page, which TEX defines as one inch (2.54 cm) down from and one inch to the right of the upper left corner.

`\hoffset` = *dimen*

This command defines the horizontal offset relative to the origin.

Above the page body as defined by `\hsize` and `\vsize` you can place a headline using the command

`\headline` ={*tokens*}

Likewise you can place a footer underneath the page body using

`\footline` ={*tokens*}

These commands are examples of *token lists*. The tokens can be anything, text and/or control sequences. An obvious candidate to put in a header is the current page number. This number is maintained by TEX in a *counter* called `\pageno`. The number can be accessed using the command `\the` as follows:

```
\headline={\hfil page \the\pageno}
```

The command \hfil is used to flush the text that follows to the right. By default the contents of a headline is spread over the full width of the text column. See section 16.6 for details on \hfil and related commands.

By default Plain TEX typesets the current page number at the center of the footline. If you don't want that, you can suppress it with

```
\nopagenumbers
```

This command is actually an abbreviation of \footline={\hfil}.

Footnotes can be produced with

\footnote *symbol*{*text*}

The *symbol* (which can be a number, a star, a dagger or whatever) will be typeset at current position in the text. The *text* will be placed at the bottom of the current page, preceded by the *symbol*. If there is insufficient space on the current page, part of the text may flow to the bottom of the next page. Here is an example:

```
This is important\footnote * {Not really} to know, if you
want to learn.\footnote{213}{Just kidding.}
```

If you want to start a new page at a certain point, even if the current page is not full yet (e.g. to start a new chapter), you can use the command

```
\eject
```

Even better would be to write

```
\vfill\eject
```

The \vfill command will make sure the rest of the remaining page will be blank. If you only write \eject, TEX may start stretching whatever is left on the current page, which is probably not what you want.

```
\raggedbottom
```

This command tells TEX to permit a small amount of variability in the bottom margins of different pages, in order to keep the spacing within each page uniform.

16.5 Typesetting paragraphs

TEX supports a large number of commands to tweak the layout of paragraphs. Nearly all variations you can think of can be implemented by simply (or tediously) adjusting a few of the paragraph parameter values.

The amount of indentation is one of these values. It can be set with the command

`\parindent` = *dimen*

You can set `\parindent=0mm` if you want no indentation at all. If you only want to suppress indentation of the next paragraph you should use the command

`\noindent`

Beware that TEX always uses the parindent value that is in effect at the *start* of the paragraph. In the example below we will demonstrate this:

```
\parindent=10mm
Starting with ‘parindent’ at 10mm;
\parindent=30mm
then change it to 50mm;
\parindent=50mm
and change it again to 20mm;
\parindent=20mm
and see what happens here.

Guess what happens now?
```

Starting with ‘parindent’ at 10mm; then change it to 50mm; and change it again to 20mm; and see what happens here.

Guess what happens now?

Of course you can make changes local by *grouping*:

```
{\parindent=0mm
I want no indentation only for this
particular paragraph, so let’s set
it to 0 millimeters locally.}

This should be ‘normal’ again.
```

I want no indentation only for this particular paragraph, so let’s set it to 0 millimeters locally.

This should be ‘normal’ again.

The amount of blank space between paragraphs is defined by

`\parskip` = *dimen* `plus` *dimen* `minus` *dimen*

This command takes up to five parameters but only the first one is mandatory. The underlined words `plus` and `minus` are keywords that should be used if you want to specify the stretchability and/or shrinkability of this amount.

TEX tries to optimize the distribution of blank space on each page. It may e.g. stretch or shrink the blank space that appears before and after mathematical equations in order to make them fit better on the page. The amount of stretchability can be set with the dimension after `plus`, the shrinkability with the dimension after `minus`.

If you want the blank space between paragraphs to have somewhat rubber like features you could write, e.g.: `\parskip 12pt plus 3pt minus 1pt`. In *TEX speak* such values are called ‘glue’. Beware that whenever you specify a glue value, TEX will

scan for the keywords `plus` and `minus`, which may interfere with the text you are writing. You can tell TEX to relax by using the command

`\relax`

immediately after the assignment, e.g. like this:

```
\parskip=2.5mm\relax
plus or minus won't harm now.
```

The distance between lines within a paragraph can be set with

`\baselineskip` = *dimen*

Note that this is actually the distance from the baseline of one line to the baseline of the next line, *not* the space in between lines. This size usually depends strongly on the font you are using (see also section 16.8 on fonts).

The alignment of paragraphs is determined by the setting of

`\leftskip` = *dimen* `plus` *dimen* `minus` *dimen*

and

`\rightskip` = *dimen* `plus` *dimen* `minus` *dimen*

The first one defines the horizontal stretchability of the left margin, the second defines the stretchability of the right margin.
Using Plain TEX's default settings a paragraph would look like this:

A good way to learn a language is by studying examples. Therefore we will show many examples in this introduction. They all look like this: on the right you can see the actual input, on the left the output that it will produce.

```
A good way to learn a language is by
studying examples. Therefore we will
show many examples in this introduction.
They all look like this: on the right you
can see the actual input, on the left the
output that it will produce.
```

Now let us tweak the value of `\leftskip`. By setting `\rightskip` at a fixed value you can make the text column narrower:

A good way to learn a language is by studying examples. Therefore we will show many examples in this introduction. They all look like this: on the right you can see the actual input, on the left the output that it will produce.

```
\rightskip=15mm
A good way to learn a language is by
studying examples. Therefore we will
show many examples in this introduction.
They all look like this: on the right you
can see the actual input, on the left the
output that it will produce.
```

By making both `\leftskip` and `\rightskip` stretchable you can get a centering effect:

A good way to learn a language is by studying examples. Therefore we will show many examples in this introduction. They all look like this: on the right you can see the actual input, on the left the output that it will produce.

```
\leftskip=10mm plus 20mm minus 10mm
\rightskip=10mm plus 20mm minus 10mm
A good way to learn a language is by
studying examples. Therefore we will
show many examples in this introduction.
They all look like this: on the right you
can see the actual input, on the left the
output that it will produce.
```

A ragged left margin can be obtained like this:

A good way to learn a language is by studying examples. Therefore we will show many examples in this introduction. They all look like this: on the right you can see the actual input, on the left the output that it will produce.

```
\leftskip=0mm plus 20mm
A good way to learn a language is by
studying examples. Therefore we will
show many examples in this introduction.
They all look like this: on the right you
can see the actual input, on the left the
output that it will produce.
```

Note that the last line of the paragraph was flushed to the left. This is because of the value of

`\parfillskip` = *glue*

which defines the amount of blank space at the end of a paragraph. In the next example we will set it to zero.

A good way to learn a language is by studying examples. Therefore we will show many examples in this introduction. They all look like this: on the right you can see the actual input, on the left the output that it will produce.

```
\leftskip=0mm plus 20mm
\parfillskip=0mm
A good way to learn a language is by
studying examples. Therefore we will
show many examples in this introduction.
They all look like this: on the right you
can see the actual input, on the left the
output that it will produce.
```

Did you notice how *all* line breaks changed? TEX optimizes a whole paragraph, not line by line.

Plain TEX provides the command

`\raggedright`

to make a ragged right margin:

A good way to learn a language is by studying examples. Therefore we will show many examples in this introduction. They all look like this: on the right you can see the actual input, on the left the output that it will produce.

```
\raggedright
A good way to learn a language is by
studying examples. Therefore we will
show many examples in this introduction.
They all look like this: on the right you
can see the actual input, on the left the
output that it will produce.
```

As you can probably guess this command is actually a macro that assigns appropriate values to `\leftskip` and `\rightskip`. And it does a little more: it uses the command

`\spaceskip` = *glue*

The word '*glue*' is used here as an alias for '*dimen* `plus` *dimen* `minus` *dimen*', like we have seen before. This command can be used to override the default width of a space in the current font and/or its stretchability and/or its shrinkability.

Two other control sequences to manipulate the indentation of a paragraph are

`\hangafter` = *number*

and

`\hangindent` = *dimen*

These commands can be used to change the indentation of a specific number of lines. `\hangindent` specifies how much the lines shall be indented; `\hangafter` specifies for how many lines this indentation scheme will be in effect. The indentation is on the left side if the value of `\hangindent` is positive, or on the right side if it is negative. If `\hangafter` is greater or equal to zero then the indentation scheme will affect all lines of the paragraph except the first *number*. If `\hangafter` is negative then it will effect only the first *number* of lines of the paragraph.

A good way to learn a language is by studying examples. Therefore we will show many examples in this introduction. They all look like this: on the right you can see the actual input, on the left the output that it will produce.

```
\noindent
\hangindent=3cm \hangafter=-3
A good way to learn a language is by
studying examples. Therefore we will
show many examples in this introduction.
They all look like this: on the right you
can see the actual input, on the left the
output that it will produce.
```

A good way to learn a language is by studying examples. Therefore we will show many examples in this introduction. They all look like this: on the right you can see the actual input, on the left the output that it will produce.

```
\noindent
\hangindent=1cm \hangafter=2
A good way to learn a language is by
studying examples. Therefore we will
show many examples in this introduction.
They all look like this: on the right you
can see the actual input, on the left the
output that it will produce.
```

At the end of the paragraph `\hangindent` and `\hangafter` are automatically reset to `0pt` and `1` respectively.

Plain TEX also provides two simple commands to typeset items, much like the `\hangindent` examples we have just seen.

`\item` *label*

This command is used to type set one item. The item ends implicitly at the end of the paragraph or at the next item (which of course starts a new paragraph).

`\itemitem` *label*

This command is used to typeset a subitem if you are already typesetting an item.

TEX is great because:

1. It's powerful
   * It's programmable
   * It's reliable
2. It's fun

```
\noindent
\TeX\ is great because:
\item {1.}
  It's powerful
  \itemitem *
    It's programmable
  \itemitem *
    It's reliable
\item {2.} It's fun
```

Note that if the label of an item is more than one character you should put it in a group, i.e, put curly braces around it. Note also that the indentation in the example is not required. This is done only to keep the source more human-readable. If the whole example were written on one line, it would make no difference.

More exotic paragraph shapes can be obtained by using the command

`\parshape` = *number* $i_1\ l_1\ i_2\ l_2\ \ldots\ i_n\ l_n$

The *number* specifies how many lines you will specify the indentation (i) and length (l) for.

A
good way to learn
a language is by studying ex-
amples. Therefore we will show many
examples in this introduction. They all look
like this: on the right you can see the
actual input, on the left the
output that it will
produce.

```
\noindent
\parshape=9 0.45\hsize 0.01\hsize
            0.30\hsize 0.4\hsize
            0.20\hsize 0.6\hsize
            0.10\hsize 0.8\hsize
            0mm        1.0\hsize
            0.10\hsize 0.8\hsize
            0.20\hsize 0.6\hsize
            0.30\hsize 0.4\hsize
            0.40\hsize 0.2\hsize
A good way to learn a language is by
studying examples. Therefore we will
show many examples in this introduction.
They all look like this: on the right you
can see the actual input, on the left the
output that it will produce.
```

If the paragraph has fewer lines than you specified, the superfluous specifications will be ignored. If the paragraph has more lines than you specified, the specification for the last line will be repeated until the end of the paragraph. At the end of a paragraph the specifications will be automatically reset to `\parshape=0`.

16.6 Blank space

We have already seen how the spacing within paragraphs can be tweaked. Now we will focus on spacing between paragraphs and special cases within paragraphs.

`\vskip` *glue*

This is the most generic command to insert vertical blank space.

To fill the rest of the current page with blank space you can use the command

`\vfill`

This command is essentially equivalent to `\vskip 0pt plus 1fill`. The `fill` keyword needs some explanation: it indicates 'infinity'. As you can deduce from this example, this keyword requires a multiplication factor. In stead of giving an exhaustive formal description of its usage we will give some examples of valid expressions:

```
\vskip 0pt plus 0.5fill
\vskip 10mm minus 10mm plus 1fill
\vskip 0in plus -1fill
```

You can experiment with it to get a good feeling of what it does.

In Plain TEX there are a few predefined amounts of vertical blank space:

```
\smallskip
```

inserts `\smallskipamount` of space. This amount defaults to `3pt plus 1pt minus 1pt`. Of course you can change this amount like this

```
\smallskipamount=5pt plus 1pt
```

It is a good idea to use such symbolic space values instead of `\vskip` because you can change the values globally and thus ensure consistent spacing throughout your document.

```
\medskip
```

Inserts `\medskipamount` of space. This amount defaults to `6pt plus 2t minus 2pt`.

```
\bigskip
```

Inserts `\bigskipamount` of space. This amount defaults to `12pt plus 4pt minus 4pt`.

Blank space in horizontal mode works very similar to vertical mode. The command

```
\hfill
```

can be used to flush whatever follows to the right margin. As an alternative there is also

```
\dotfill
```

which fills the space with dots.

This to the left that to the right.

```
\noindent
This to the left \hfill that to the right.
```

Since `\hfill` and `\dotfill` are stretchable you can use more of them on one line to spread items equally.

Left.....Center.....Right

```
\noindent
Left\dotfill Center\dotfill Right
```

This command can also be used to end a line in the middle of a paragraph. You will also need the `\break` command:

You can break a line anywhere using the 'break' command but chances are that it looks bad.

A line break
looks better if you do it
like this.

```
You can break a line anywhere using
the 'break' command but\break
chances are that it looks bad.

A line break\hfill\break
looks better if you do it\hfill\break
like this.
```

Of course there is the equivalent of \vskip for horizontal blank space:

\hskip *glue*

Predefined commands for inserting horizontal blank space are:

\quad

which is defined as \hskip 1em\relax.

\qquad

which is defined as \hskip 2em\relax.

\kern *dimen*

is mostly used to fine tune spacing. A kern is not stretchable or shrinkable. Beware that TEX will *never* break (hyphenate) a word at a kern, whereas \hskip *is* breakable.

16.7 Boxes

In certain cases you want to prevent TEX from breaking a sequence of words over multiple lines or pages. This can be achieved by putting material in *boxes*. TEX defines a number of boxes as primitives, and lots of other commands use these internally.

Horizontal boxes can be used to keep material on the same line. The most basic command that you can use for that is \hbox. Anything you supply as a parameter will not be broken. So if you write \hbox{My Name}, TEX will put 'My' and 'Name' on the same line, no matter what.

The \hbox command is actually a lot more powerful. You can also use it like this:

\hbox to *dimen*{...}

Instead of allowing TEX to calculate the optimal size for a box you can specify it explicitly.

Tell me your name: .....
Tell me your address: .....

```
\noindent
\hbox to 40mm{Tell me \hfil your name:}
\hbox to 25mm{\dotfill}
\par\noindent
\hbox to 40mm{Tell me \hfil your address:}
\hbox to 25mm{\dotfill}
```

Note that you can't put vertical material (\vskip, \par, etc.) inside an \hbox.

Plain TEX defines the command

```
\centerline{...}
```

to center text within the text column.

```
\line
```

creates an \hbox of width \hsize.

To keep vertical material together TEX defines two commands that work very similar to \hbox. The command

```
\vbox to dimen{...}
```

makes a box that will align at its bottom line; the command

```
\vtop to dimen{...}
```

makes a box that aligns at its top line.

Now look
at
this.

this.
at
And then have a look

```
\parindent=0mm
Now \vtop{
    look  \par
    at    \par
    this.}

And then have a \vbox{
    this. \par
    at    \par
    look}
```

Both horizontal and vertical boxes can be moved up or down with the command

```
\lower dimen \somebox
```

What goes up must come down.

```
What goes \lower -2mm\hbox{up}
must come \lower 2mm\hbox{down}.
```

16.8 Fonts

Plain TEX is set up to use fonts of the Computer Modern family, but of course you can use other fonts instead if you prefer. Sixteen basic fonts are used, which are listed in table 16.2.

Table 16.2: Plain TEX's sixteen basic fonts

Name	Full font name	Point size
cmr10	Computer Modern Roman	10
cmr7	Computer Modern Roman	7
cmr5	Computer Modern Roman	5
cmbx10	Computer Modern Bold Extended	10
cmbx7	Computer Modern Bold Extended	7
cmbx5	Computer Modern Bold Extended	5
cmsl10	Computer Modern Slanted Roman	10
cmti10	Computer Modern Text Italic	10
cmtt10	Computer Modern Typewriter Type	10
cmmi10	Computer Modern Math Italic	10
cmmi7	Computer Modern Math Italic	7
cmmi5	Computer Modern Math Italic	5
cmsy10	Computer Modern Math Symbols	10
cmsy7	Computer Modern Math Symbol	7
cmsy5	Computer Modern Math Symbol	5
cmex10	Computer Modern Math Extension	10

Fonts are declared with the command

`\font` *\somename* = *fontmetric* `at` *dimen*

One of the predefined fonts in Plain TEX is `\tenrm`. It is defined as follows:

```
\font\tenrm=cmr10 at 10pt
```

Likewise there is `\sevenrm` (cmr7 at 7pt) and `\fiverm` (cmr5 at 5pt) for roman type. For bold type there is `\tenbf` (cmbx10 at 10pt), `\sevenbf` (cmbx7 at 7pt) and `\fivebf` (cmbx5 at 5pt). For italic there is `\tenit` (cmti at 10pt), for slanted there is `\tensl` (cmsl10 at 10pt), and for bold there is `\tenbf` (cmbx10 at 10pt). Here is an example:

TenRoman *TenSlanted* *TenItalic* **TenBold**
SevenRoman FiveRoman **SevenBold** **FiveBold**

```
\tenrm   TenRoman   \tensl   TenSlanted
\tenit   TenItalic  \tenbf   TenBold
\sevenrm SevenRoman \fiverm  FiveRoman
\sevenbf SevenBold  \fivebf  FiveBold
```

These commands use the CM fonts at their natural size. However, You can also use them at other sizes. If you declare a font like this

```
\font\myfont=cmr5 at 10pt
```

You will get a font that behaves like a 10pt font but which was designed for a much smaller size. The difference to e.g. cmr10 at 10pt is obvious in the next example:

This is cmr10 at 10pt.
This is cmr5 at 10pt.

```
\font\tenfive=cmr5 at 10pt
\tenrm This is cmr10 at 10pt. \par
\tenfive This is cmr5 at 10pt.
```

To make 'magnification' of fonts easier you can use the command

`\magstep` *number*

where *number* is 0, 1, 2, 3, 4 or 5. Magsteps are in fact powers of 1.2: `\magstep0` $= 1.2^0 = 1$; `\magstep1` $= 1.2^1 = 1.2$; `\magstep2` $= 1.2^2 = 1.44$; `\magstep3` $= 1.2^3 = 1.728$; `\magstep4` $= 1.2^4 = 2.074$; `\magstep5` $= 1.2^5 = 2.488$. An intermediate value between `\magstep0` and `\magstep1` is provided by the command `\magstephalf`: it is $1.2^{0.5} = \sqrt{1.2} = 1.095$.

These magnification commands can be used in conjunction with the *scaled* keyword in font declarations, e.g. like this:

```
\font\myfont=cmr12 scaled \magstep4
```

In this example the font 'myfont' is declared at 14.4 points. Actually the parameter that follows the 'scaled' keyword is just a number: '1000' represents the natural size for that font (e.g. 17 points for cmr17 or 5 points for cmr5). So, `cmr5 scaled \magstep2` is equivalent to `cmr5 at 7.2pt` and `cmr7 scaled \magstep0` is equivalent to `cmr7 at 7pt`.

As there are seven predefined 'magsteps' and the CM roman font is available in eight design size, a total of $7 \times 8 = 56$ sizes are readily available in Plain TEX. Table 16.3 shows all the variations.

In case you want to 'magnify' a complete document uniformly without touching any of the font declarations of page size parameters you can use the command

`\magnification` = *number*

where *number* can be, e.g. `\magstep1` or a number, where '1000' stands for 'normal size'.

When changing fonts it is also necessary to change the line spacing. TEX takes care of this automatically but you may want to adjust the value yourself. You can do this with the command `\baselineskip`. The following example will show the effect:

A good way to learn a language is by studying examples. Therefore we will show many examples in this introduction.

A good way to learn a language is by studying examples. Therefore we will show many examples in this introduction.

```
A good way to learn a language is by
studying examples. Therefore we will
show many examples in this introduction.

\baselineskip=17pt
A good way to learn a language is by
studying examples. Therefore we will
show many examples in this introduction.
```

Table 16.3: The CM roman font in all point sizes and magnifications

	Magnification					
0	**half**	**1**	**2**	**3**	**4**	**5**
5	5.48	6.00	7.20	8.64	10.37	12.44
6	6.57	7.20	8.64	10.37	12.44	14.93
7	7.67	8.40	10.08	12.10	14.52	17.42
8	8.76	9.60	11.50	13.82	16.59	19.91
9	9.86	10.80	12.96	15.55	18.66	22.39
10	10.95	12.00	14.40	17.28	20.74	24.88
12	13.15	14.40	17.28	20.74	24.88	29.86
17	18.62	20.40	24.48	29.38	35.25	42.30

Plain TEX defines a few easy commands for selecting a different font *shape*. The command

```
\bf
```

selects a bold shape;

```
\rm
```

selects a roman shape;

```
\it
```

selects an italic shape;

```
\sl
```

selects a slanted shape;

```
\tt
```

selects a monospaced typewriter shape. The example below demonstrates all of them.

Normal **Bold** Roman *Slanted* *Italic* `Typewriter`

```
Normal \bf Bold \rm Roman \sl Slanted
\it Italic \tt Typewriter
```

Note that the effect of these commands is global, so if you want to keep the effect local you must use 'grouping'.

16.9 Accents and ‘foreign’ characters

The table below gives an overview of all accents that Plain TEX supports. Such accents can be used in combination with arbitrary characters. In ‘math mode’ accents are handled a little differently. See section 16.16.7 for details on accents in math mode.

Input	Output	Input	Output
\‘o	ò	\’o	ó
\^o	ô	\~o	õ
\=o	ō	\.o	ȯ
\u o	ŏ	\v o	ǒ
\H o	ő	{\"o}	ö
\c o	o̧	\d o	ọ
\b o	o̲	\t oo	o͡o

A number of ‘foreign’ characters are also available. The table below gives an overview.

Input	Output	Input	Output
\oe	œ	\OE	Œ
\ae	æ	\AE	Æ
\aa	å	\AA	Å
{\"o}	ö	\O	Ø
\l	ł	\L	Ł
\i	ı	\j	ȷ
!‘	¡	?‘	¿
\ss	ß	\dots	…
\dag	†	\ddag	‡
$\backslash$	\	\copyright	©
\S	§	\P	¶
\#	#	\$	$
\&	&	\%	%

Note that the \backslash command needs to be surrounded by $ signs. These indicate ‘math mode’, which will be explained in section 16.16. Other commands that require a ‘math’ environment are \dagger and \ddagger. Do not confuse these with \dag and \ddag.

Instead of using the TEX commands to specify accented letters and ‘foreign’ characters, you could use one of TEX’s input translation tables, which are explained in section 13.3.2). In that case you can enter text in a more readable style.

16.10 Floating insertions

Objects such as tables and figures can be placed at the top of a page, even if such a table appears 'midpage' in your TEX document. TEX can 'float' such an object to e.g. the top of the current page, or it can store it until it finds a reasonable place to typeset it.

An object that you want to be placed at the top of a page, be it the current page or the next, should be preceded by the command

```
\topinsert
```

Then the object can be specified, followed by the command

```
\endinsert
```

The material in such an insertion can be anything. Here is an example:

```
\topinsert
\centerline{Table 1: A large table}
A large table
that may fload
to the top of
the next page
\endinsert
```

In case you prefer to have the insertion typeset at the current position on the page, you can use the command

```
\midinsert
```

instead of `\topinsert`. If there is not enough room on the current page the insertion will effectively be converted to a `\topinsert`. For material that occupies so much space that it is better to put it on a separate page anyway, there is the command

```
\pageinsert
```

The material between `\pageinsert` and `\endinsert` will be typeset at the next page.

Note that TEX will *not* change the order in which insertions will be typeset, so even if a very small `\topinsert` follows a `\midinsert` that is too big for the current page, the `\topinsert` will also move to another page.

To make sure that all insertions that TEX may still be storing are typeset at a given point, you can use the command

```
\supereject
```

or rather `\vfill\supereject`. This command will typically appear at the end of a chapter. All insertions will be flushed out with blank space filling the the bottom of incomplete pages.

16.11 Graphics

The most versatile kind of graphics are Encapsulated PostScript graphics. An easy way to include such graphics in your document is provided by the EPSF macros written by T. Rokicki. You can load these macros using the following command:

```
\input epsf
```

Now you are ready to include EPS pictures using the command

`\epsfbox[`*options*`]{`*EPS picture file*`}`

This command can be used to include (very small) pictures in the middle of a paragraph, but you could just as well use it in a command, within an or even within a displayed equation environment (`$$`).

By default a picture will be printed at its 'natural' size. This size is determined by the so-called 'BoundingBox' in the EPS file. Unfortunately some programs that generate EPS output specify no BoundingBox at all, or the BoundingBox is incorrect. In such cases you should use the optional parameter to supply the BoundingBox yourself like this:

```
\epsfbox[0 0 480 220]{mypicture.eps}
```

You can specify the exact size that you want the picture to have in your output with the commands

`\epsfxsize` = *dimen*

for setting the width, and

`\epsfysize` = *dimen*

for setting the height. Both width and height are restored to zero after each use, so `\epsfxsize` or `\epsfysize` (not both) must be specified before *each* use of `\epsfbox`. Here is an example:

Nice pictures

```
\centerline
 {\hfil \epsfysize=15mm
  \epsfbox{hass03g.eps}
  \hfil \epsfysize=25mm
  \epsfbox{hass07g.eps}
  \hfil}
\centerline{Nice pictures}
```

If you want TEX to report the size of the figure (as a message on your terminal when it processes each figure), add the command `\epsfverbosetrue`.

16.12 Color support

If you input the file `colordvi.tex` in your document you are ready to use colors. The colors supported by 'colordvi' are listed below:

```
GreenYellow, Yellow, Goldenrod, Dandelion, Apricot, Peach, Melon,
YellowOrange, Orange, BurntOrange, Bittersweet, RedOrange, Mahogany,
Maroon, BrickRed, Red, OrangeRed, RubineRed, WildStrawberry, Salmon,
CarnationPink, Magenta, VioletRed, Rhodamine, Mulberry, RedViolet,
Fuchsia, Lavender, Thistle, Orchid, DarkOrchid, Purple, Plum, Violet,
RoyalPurple, BlueViolet, Periwinkle, CadetBlue, CornflowerBlue,
MidnightBlue, NavyBlue, RoyalBlue, Blue, Cerulean, Cyan, ProcessBlue,
SkyBlue, Turquoise, TealBlue, Aquamarine, BlueGreen, Emerald,
JungleGreen, SeaGreen, Green, ForestGreen, PineGreen, LimeGreen,
YellowGreen, SpringGreen, OliveGreen, RawSienna, Sepia, Brown, Tan,
Gray, Black, White.
```

All these colors actually represent TEX commands that expect one parameter: the text that you want to write in that particular color. Here are some examples:

This is 'normal'. This is Goldenrod, PineGreen, Gray, Lavender and Red.

```
This is 'normal'.
\Goldenrod{This is Goldenrod},
\PineGreen{PineGreen}, \Gray{Gray},
\Lavender{Lavender} and \Red{Red}.
```

If you want to typeset larger parts or a whole document in a specific color, you can use similar commands of the format `\text`*Color*. The background of a whole page and all subsequent page can be set with the command `\background`. Here is an example:

Yellow on Mahogany.

```
\background{Mahogany}
\textYellow
\bf Yellow on Mahogany.
```

You can define your own colors with the commands

`\Color{`*CMYK values*`}{`*text*`}`

or

`\textColor{`*CMYK values*`}`

The 'CMYK values' are four values between 0 and 1 that represent the relative intensities of the colors cyan, magenta, yellow and black. They must be separated by blanks.

Normal black text, then a special color and then grayish.

```
Normal black text, then a
\Color{0.8 0.3 0.4 0.1}{special color}
and \textColor{0 0 0 .5} then grayish.
```

Remember that TEX was not designed with color in mind, and producing colors requires a lot of help from the DVI driver. Thus, depending on the driver, some or all features of the color macros may not always work properly.

In case the color macros cause serious problems, you can turn them off globally by inputting the file `blackdvi.tex` instead of `colordvi.tex`. You will get straight black and white without having to ferret out all the color macros.

16.13 Lining things up

Plain TEX basically provides two commands for typesetting material that has to line up vertically, as in tables. The first is based on setting *tabs* just like on old-fashioned typewriters, the second is based on logical column sizes that TEX will calculate itself.

Remember that tables typeset with 'tabs' are line-oriented, which means that such a table can run over more than one page without problems. The second type of table ('halign') can never be broken in the middle. We will first discuss 'tabbing'.

The command

`\settabs` *number* `\columns`

can be used to set up columns:

	English	Dutch	German	French
1	one	een	ein	un
2	two	twee	zwei	deux
3	three	drie	drei	trois

```
{\hsize=70mm \settabs 5 \columns
\+    & \bf English & \bf Dutch
      & \bf German  & \bf French \cr
\+ \cr % empty line
\+ 1 & one   & een  & ein  & un    \cr
\+ 2 & two   & twee & zwei & deux  \cr
\+ 3 & three & drie & drei & trois \cr}
```

In the example we set the total width of the table to 70 mm. the command `\+` tells TEX that this line should be aligned according to the current tab settings. The character `&` is used to separate columns. A tab aligned line always ends with `\cr`.

In the previous example all 4 columns were equally wide, regardless of the contents of any cell. But you can also use the `\settabs` command like this:

```
\settabs \+ \cr
```

The next example will show how this works out.

1	2	3	4
an overfull	column	will look	bad
but	this one	looks	fine

```
\settabs
\+ this line & determines & the widths of
 & all columns \cr
\+ 1 & 2 & 3 & 4 \cr
\+ an overfull & column & will look & bad \cr
\+ but & this one & looks & fine \cr
```

The first line sets the tabs according to the width of column content. Then this line is discarded.

Note that the tab settings of a `\settabs` command remain in effect until the next `\settabs` occurs. You can even change the tab settings while typesetting a table:

```
\settabs \+ \hskip15mm & \hskip15mm &
           \hskip15mm & \hskip15mm \cr
\+ one  & two  & three & four  \cr
\+ five & six  & seven & eight \cr
{\settabs 3 \columns
\+ one  & two  & three \cr
\+ four & five & six   \cr
} % back to old tab settings
\+ one  & two  & three & four  \cr
\+ five & six  & seven & eight \cr
```

```
one       two       three     four
five      six       seven     eight
one             two             three
four            five              six
one       two       three     four
five      six       seven     eight
```

`\cleartabs`

This command clears any previous `\settabs`.

`\halign{` *table material* `}`

This command is used to typeset tables for which you want TEX to adjust the column widths according to the content of the columns.

Table material consists of two parts: a preamble and the actual table content. The preamble specifies a sequence of templates, one for each column. Each template must contain one `#` that indicates where TEX should insert column material. Templates are separated by `&` and each row is ended with `\cr`.

The format of the actual table content is very similar to the tabbing environment described earlier, except that there is no `\+` command at the beginning of a row.

```
\halign{ # & # & # \cr
  1    & 2     & 33333333 \cr
  1111 & 22    & 333      \cr
  11   & 22222 & 3        \cr }
```

```
1     2      33333333
1111  22     333
11    22222  3
```

Note that the width of each column was automatically determined by TEX. The command

`\tabskip` = *glue*

can be used to set the amount of white space between columns. Note also that by default columns are left-aligned.

The preamble of an \halign environment can be much more complex, as we will demonstrate in the next example. At the same time we will introduce the command \noalign which can be used to do some special things within the table.

First	Second	...Third...
12	9	...today...
1888	800	...tomorrow...

```
\halign{ \hfil \it # &
         \hfil #
         \hfil &
          ...#... \cr
First & Second & Third\cr
\noalign{\smallskip
         \hrule
         \smallskip}
12   & 9   & today\cr
1888 & 800 & tomorrow\cr
\noalign{\smallskip \hrule}}
```

In this example we used \hrule to insert a horizontal rule.

\hrule width *dimen* height *dimen* depth *dimen*

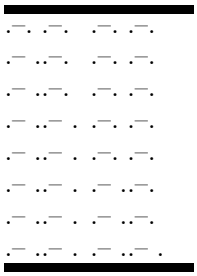
All parameters are optional, so if you only want to specify the width or the height you can do so.

In case you need to typeset a particular cell in a way that differs from the template, you can use the \omit command. In the next example we will also introduce the command \multispan which can be used to typeset table material over multiple columns.

	Holland	**Germany**	**Belgium**
milk:	12 l/day	17 l/day	10 l/day
beer:	3 l/day	9 l/day	7 l/day
wine:	2 l/day	2 l/day	6 l/day
water:	(classified information)		

```
\tabskip=3mm
\halign{ # &
         #l/day\hfil &
         #l/day\hfil &
         #l/day\hfil \cr
& \omit \bf Holland
& \omit \bf Germany
& \omit \bf Belgium \cr
milk:  & 12 & 17 & 10 \cr
beer:  & 3  & 9  & 7  \cr
wine:  & 2  & 2  & 6  \cr
water: & \multispan 3
(classified information)\hfil \cr }
```

Note that spaces *do* matter. You will get different results if you add or remove spaces in the preamble or the table content.



```
\halign{.#.&.#. & .#.&.#.\cr
\noalign{\hrule height 3pt width 25mm
         \smallskip}
--&--&--&--\cr
-- &--&--&--\cr
-- & --&--&--\cr
-- & -- &--&--\cr
-- & -- & --&--\cr
-- & -- & -- &--\cr
-- & -- & -- & --\cr
-- & -- & -- & -- \cr
\noalign{\smallskip
         \hrule height 3pt width 25mm}}
```

In case you want a rule between every row, you could use the command `\everycr` to specify this:

October

		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

```
{\everycr={\noalign{\smallskip
                    \hrule
                    \smallskip}}
\noindent {\bf October} \par
\halign{# & # & # & # & # & # & # \cr
   &    &  1 &  2 &  3 &  4 &  5\cr
 6 &  7 &  8 &  9 & 10 & 11 & 12\cr
13 & 14 & 15 & 16 & 17 & 18 & 19\cr
20 & 21 & 22 & 23 & 24 & 25 & 26\cr
27 & 28 & 29 & 30 & 31\cr }}
```

Note that the value of `\everycr` remains in effect until a new value is assigned to it. Therefore it is usually wise to assign its value locally, as in the example above.

Vertical rules can be added in a manner much similar to horizontal rules. They can be used both in the preamble and in the table content.

`\vrule` `width` *dimen* `height` *dimen* `depth` *dimen*

```
\halign{\vrule\strut
        #&\ # &\ # &\ #\
        &#\vrule\cr
\noalign{\hrule}
&\multispan 3\hfil\bf License fee\hfil&\cr
\noalign{\hrule}
& 10 user: \vrule
& 20 users: \vrule
& many users:&\cr
& \$ 100 \vrule
& \$ 150 \vrule
& \$ 300&\cr
\noalign{\hrule}}
```

License fee		
10 user:	20 users:	many users:
$ 100	$ 150	$ 300

Note that the spaces in the table above were stretched. You should use `\hfil` commands for proper alignment.

The `\strut` commands in the example above ensure that the height of a cell is always sufficient for any character in the current font. Nevertheless, it can be difficult to get proper result with vertical rules. The command `\offinterlineskip` can be helpful. It ensures that TEX will not insert any interline glue which could cause gaps in vertical rules. The next example shows such gaps. Note how we used the optional parameters of `\vrule` and `\hrule` here to change the sizes of the rules.

```
%\offinterlineskip
\halign{\strut\vrule
        #\ &#&#&#&\
        \vrule #\cr
\noalign{\hrule height1mm}
depth13mm&\multispan 3\hfil\bf head line\hfil
&\cr
\noalign{\hrule}
height 6mm depth 2mm
& 11111 \vrule &\ 2     \vrule &\ 3&\cr
height 6mm depth 2mm
& 1     \vrule &\ 22222 \vrule &\ 33333&\cr
\noalign{\hrule height1mm}}
```

head line		
11111	2	3
1	22222	33333

In the next example we will show how you could set up a table that has the width of the current column, and in which all columns are stretched so as to optimally fill the space. The stretchability is obtained by changing the value of `\tabskip` in the preamble.

```
\halign to \hsize{\tabskip=0mm
                  \vrule#\strut
                  \tabskip=2mm plus 1fil&
                  \hfil#\hfil&
                  \vrule#&
                  \hfil#\hfil&
                  \vrule#&
                  \hfil#\hfil&
                  \tabskip=0mm
                  #\vrule\cr
\noalign{\hrule}
& 111 && 2     && 3&\cr
& 1   && 22222 && 33333&\cr
\noalign{\hrule}}
```

111	2	3
1	22222	33333

The control sequence `\valign` is very similar `\halign`, but now columns and rows are reversed. Or you could say, the table is rotated by 90 degrees.

```
{\hsize=20mm
\valign {\strut#&\strut#&\strut#&\strut#\cr
11111 & 2     &    3  & 4444\cr
1     & 22222 & 33333 & 444\cr
111   & 222   &   33  & 4\cr }}
```

11111	1	111
2	22222	222
3	33333	33
4444	444	4

When using the `\halign` command it is very easy to add another *row* without touching the preamble. When using the `\valign` command it is very easy to add another *column* without touching the preamble.

16.14 Fine tuning

16.14.1 Optimizing paragraphs

TEX assigns a numerical value called 'badness' to each line it typesets. This 'badness' is used to find the optimal spacing and line breaks of a paragraph. If a line can be typeset without having to stretch or shrink spaces its badness will be zero; if a line had to be stretched or shrunk, or it became 'overfull', its badness can be up to 10000.

By default Plain TEX tries to keep the badness below 200. This value is set by the command

`\tolerance` = *number*

By setting e.g. `\tolerance=2000` TEX will tolerate a much higher badness, but chances are that it will complain about 'underfull' boxes. It is up to you to determine whether this is acceptable or even better than other settings. TEX treats `\tolerance=10000` as 'infinite' tolerance, which would allow for arbitrarily wide space.

TEX will report all events where a certain badness threshold is exceeded. This threshold is set to 1000 by default, but you can change that with

`\hbadness` = *number*

In case 'overfull' lines appear in the output, TEX will mark them visually by appending a small rule to them. By default a 5 points thick rule is used, but you can change it with

`\overfullrule` = *dimen*

In final copy you will probably want to set it to 0pt, even if some lines may still be overfull. Lines that stick out less than 1 point are usually no esthetic problem. You can use the command

`\hfuzz` = *dimen*

to specify how much overfullness in horizontal boxes is still acceptable, before TEX will issue warning messages. By default this value is 0.1pt.

`\vfuzz` = *dimen*

With this command you can specify how much overfullness in vertical boxes is still acceptable, before TEX will issue warning messages. By default this is 0.1pt.

`\vbadness` = *number*

Similarly to `\hbadness`, this command sets the threshold for warnings about overfull or underfull *vertical* boxes.

16.14.2 Optimizing hyphenations

TEX uses a very powerful mechanism to decide how words can be broken if necessary. However, in some cases this mechanism will not give optimal results. With a little tweaking you can make things perfect.

`\hyphenation{`*hyp-he-nat-ed ex-cep-tions dontbreakthis*`}`

In case the general mechanism for hyphenating words makes a mistake you can tell the system the exact breakpoints. Put a hyphen at each position at which hyphenation is allowed. Putting no hyphen at all in a word means that it should never be broken. All following occurrences of the words you specified will be hyphenated according to your specifications. Note that the hyphenation mechanism is not case sensitive, so you only

need to specify one version of any word. If you only want to make one exception at a particular point you can use:

`\-`

If you write `micro\-bio\-logy`, TEX will only hyphenate this words (if necessary) at the the points where you put a `\-` command.

Actually the `\-` command is just a special (or rather, the most trivial) case of using the command

`\discretionary{`*prebreak text*`}{`*postbreak text*`}{`*nobreak text*`}`

This command takes three parameters: the first two specify how the two parts of a word should look if hyphenated; the third specifies how the word should look if *not* hyphenated. Such a complex structure will make sense if you understand that in some cases the spelling of a word actually *changes* if it is hyphenated. Here are three examples:

```
\discretionary{back-}{ken}{backen}
\discretionary{opa-}{tje}{opaatje}
\discretionary{eight-}{teen}{eighteen}
```

The German word 'backen' should repeat the letter 'k' if hyphenated; the Dutch word 'opaatje' should lose an 'a' if hyphenated; the English word 'eighteen' should repeat the letter 't' if hyphenated. The command `\-` is defined as `\discretionary{-}{}{}`.

Note that if a hyphen occurs in a word, e.g. 'path-finder', the word will only be hyphenated at the position of the hyphen, no where else. If you want TEX to consider more break points you will have to add `\-` commands, e.g. `path-find\-er`, which defines two possible breaks.

`\lefthyphenmin` = *number*

This command can be used to specify what the minimum length of the left part of any hyphenated word should be.

`\righthyphenmin` = *number*

specifies the minimum length of the right part.

Different languages have different hyphenation rules, so you should inform TEX in what language you are writing. You can switch to another language with the command

`\language` = *number*

Your TEX system may have several sets of hyphenation rules preloaded. Each of them was assigned a number to which you must refer if you want to select a specific set.

There are several ways to switch hyphenation off. Each of them has its own merits and demerits:

- Choose a `\language` for which you deliberately loaded empty hyphenation patterns when generating a format file (see section 8.5). Note that `\discretionary`, `\hyphenation` and `\-` commands can still cause hyphenation to occur.
- Set `\righthyphenmin` and `\lefthyphenmin` to, say, 100. Now `\hyphenation` will have *no* effect anymore, but `\discretionary` and `\-` will.
- Set `\hyphenpenalty` to 10000. Now `\hyphenation`, `\discretionary` and `\-` will have no effect anymore. That means: no hyphenation will ever occur.

You may choose any of these methods, depending on what exactly you want to achieve. In order to avoid many overfull and underfull lines you may have to make interword more stretchable. You could establish that by changing the the value of `\spaceskip` to, say, `3.3pt plus 10pt minus 1pt`.

In case you are not sure if TEX is hyphenating correctly, possibly because it is using the wrong hyphenation pattern, you can test a few words using the command

`\showhyphens{`*words*`}`

The results of this test will only be written to the console and to the log file, not the output file.

```
\language=0
\showhyphens{hyphenation confuses me terribly}
\language=1
\showhyphens{hyphenation confuses me terribly}
```

If language 0 represents US English and language 1 represents Dutch, then the results could be:

```
hy-phen-ation con-fuses me ter-ri-bly
hyp-he-na-tion con-fu-ses me ter-ribly
```

16.14.3 Optimizing at the character level

A completely different kind of fine tuning that any well written document requires is related to much finer details. Although anyone will understand that '-1' means 'minus one', it would look better like this: −1. TEX distinguishes the following hyphen-like shapes:

hyphen This character is used when breaking a word at the end of a line, or as a connector between words. Use the 'minus' key to enter such a character. Hyphens are used in compound words such as 'daughter-in-law'.

minus You automatically get a minus sign when in mathematical mode: `$-1$` will result in -1.

en-dash This is a small rule with the length equal to the width of the character 'n'. It look like this: — and can be entered as `--` (two consecutive hyphens). The en-dash is mostly used in for number ranges such as 'pages 12–15'.

em-dash This is a small rule with the length equal to the width of the character 'm'. It look like this: — and can be entered as `---` (three consecutive hyphens). Em-dashes are used for punctuation in sentences—we usually call them dashes.

Quotes are another important matter. The first rule here is: do *not* use the double quote key on your keyboard. Instead, use the ` key to enter a opening quote and the ' key to enter a closing quote. Use two of them if that is more appropriate, like this: ``` ``like this'' ``` will result in "like this".

Some combinations of characters are treated as a unit. Such units are called *ligatures*. In the many fonts several ligatures are available, and they are automatically used by TEX. However, in some cases it is considered better not to use ligatures. E.g., the combination 'fl' is usually a ligature but the word 'shelfful' looks better without ligatures: 'shelfful'. You can prevent TEX from using ligatures by writing `shelf{}ful`. Even better would be `shelf\/ful`: the command `\/` is an 'italic correction' (see below) that makes the word look like this: 'shelfful'. The advantage of this method is that it will prevent TEX from reinserting the ff ligature after attempting to hyphenate `shelf{}ful`. The table below demonstrates the ligatures available in the Computer Modern fonts. Note that TEX will automatically choose an appropriate ligature whenever possible, even if a word is hyphenated.

Ligature	Non-ligature
ff	ff
fi	fi
fl	fl
ffl	ffl
ffi	ffi

An italic correction `\/` can be necessary if you mix roman and italic text. When switching from an italic typeface to a roman typeface in the middle of a sentence the space between them may seem too small. This can be corrected by inserting an italic correction like this: `{\it good\/} idea`.

16.15 Programming

Programming your own macros in TEX can increase your productivity considerably. By using macros instead of repeated commands you can ensure consistency, you can make changes without having to go through each line of your text, and your text becomes more readable.

Be warned, though, that serious TEX programming is not a topic you can master on a Saturday afternoon. TEX is quite different from the usual procedural, predicative or object oriented programming languages, so you should expect some weird results now and then, simply because you don't yet fully understand how TEX works.

When programming you will require a good reference book such as D. Knuth's *The TEXbook* and/or V. Eijkhout's *TEX by Topic*, a lot of patience and a lot of practicing. Fortunately the reward for this is high: the TEX language is very exciting, not to say addictive, especially if the concepts of macro programming are new to you. And it is a big challenge.

This introduction to programming in TEX is no more than an introduction. It demonstrates some important concepts that are of general use, but it deliberately doesn't tell the whole story, simply because that is beyond the scope of this book. The examples given here should be regarded as educational, not as ready-made elements that you can apply in your own documents. Nevertheless, we hope this will be sufficient to get you started writing your own macros.

16.15.1 The basics: definitions and assignments

The most basic command for defining a macro is

```
\def \macroname  parameters {something}
```

In its simplest form you only supply a *group* in which you write some commands that you want executed when you invoke this macro. E.g., you could write a simple macro to jump to the next page, or a macro that typesets the closing of a letter:

```
\def\newpage
  {\vfill\eject}

\def\closing
  {\bigskip
   \noindent
   Sincerily yours,
   \smallskip
   \noindent
   Ben Lee User}
```

The contents of a macro is not limited to any size, but it is advisable to keep it small so you don't loose track. A macro can call any other macro, so you could make modules, building blocks that can be called by several other macros.

Macros can also take up to 9 parameters. Parameters are specified as templates right after the command name. Here is a simple example that takes two parameters:

```
\def\chapter#1#2
  {\vfill\supereject
   \noindent
   Chapter #1
```

```
   \medskip\noindent
   #2
   \bigskip
   \noindent}
% usage:
\chapter{number}{title}
```

If you wish, you can define delimiters for each parameter. Here is an example of three macros that can typeset a simple table:

```
\def\tablehead
  #1,#2,#3,#4,#5,#6.
  {\halign\bgroup
     \vrule\ \strut
     ##\hfil&##\hfil&##\hfil&%
     ##\hfil&##\hfil&## \hfil\vrule\cr
     \noalign{\hrule}
     \bf#1 & \bf#2 & \bf#3 &
     \bf#4 & \bf#5 & \bf#6\cr
     \noalign{\hrule}}
\def\tablerow
  #1,#2,#3,#4,#5,#6.
  {\strut#1 & #2 & #3 & #4 & #5 & #6\cr}
\def\tabletail
  {\noalign{\hrule}
   \egroup}
% usage:
\tablehead  first col,2nd,3rd,%
            4th,5th,last col.
\tablerow   q,w,e,r,t,y.
\tablerow   a,s,d,f,g,h.
\tablerow   z,x,c,v,b,n.
\tabletail
```

first col	2nd	3rd	4th	5th	last col
q	w	e	r	t	y
a	s	d	f	g	h
z	x	c	v	b	n

In this example we used a few new tricks:

- We used the commands `\bgroup` and `\egroup` instead of braces. The reason is that a group is started in one macro (`\tablehead`) and ended in another one (`\tabletail`). If we had used braces, TEX would have complained about a mismatch.
- We wrote `##` signs in the `\halign` templates. This is the way to tell TEX that you want a `#`, not a reference to a parameter.

In the macros we have defined so far we used TEX's ability to evaluate the content at the time the macro is *executed*. But sometimes you want to define a macro that completely *expands* its content. This could be necessary if you want to store, e.g., the current

page number for later referencing. TEX provides the command \edef exactly for such purposes. The example below shows two macros with the same content.

```
\def \mypageref
  {\the\pageno}
\edef\Mypageref
  {\the\pageno}

\vfill\eject

\mypageref, \Mypageref
```

Now \mypageref will always (dynamically) yield the current page number, \Mypageref will (statically) yield the number of the page at which it was defined.

Another important trick for macro writers is the \let command. This command can be used to make an exact copy of another macro under a new name. It can be used to change a macro temporarily and restore its old meaning afterwards. It can also be used to add functionality to an existing macro.

```
\let\oldtablehead=\tablehead
\def\tablehead{%
  \topinsert
  \centerline{Table \the\tablecounter}
  \smallskip
  \oldtablehead}
\let\oldtabletail=\tabletail
\def\tabletail{%
  \oldtabletail
  \endinsert}
```

The reason for *not* redefining \tablehead and \tabletail completely is that those macros may be defined somewhere else by someone else. Using \let the macros will work, even if e.g. the number of parameters that \tablehead expects changes. Note that we did't specify any parameters here. We simply call \oldtablehead at the end of the macro so it will automatically find its parameters.

The counter \tablecounter in the example is a counter that we must declare and maintain. A counter is declared with the command

\newcount *countername*

Counters are set to zero when declared. Assigning a value to it is very straightforward: \tablecounter=3 is perfectly valid. However, in many cases you will want to do calculations with counters. TEX provides commands for this purposes, such as

\advance *something* by *amount*

where *something* can be a counter, dimension or glue. Naturally the *amount* must be compatible: you can't add glue to a counter, or vice versa. In the example above

a statement such as `\newcount\tablecount` and, inside the `\tablehead` macro, `\advance\tablecount by 1` would make it nearly perfect. Here are a few more valid examples of `\advance`:

```
\advance \hsize by 10mm
\advance \spaceskip \spaceskip % value & stretch doubled
\advance \pageno by -10
```

Note that the keyword 'by' is optional. Note also that *amount* can be negative. This is how subtraction can be implemented. Multiplication is also possible, using the command:

`\multiply` *something* by *factor*

Likewise there is

`\divide` *something* by *factor*

Beware that these arithmetics are limited to *integers*. E.g., if the counter `\pageno` has 2, then after `\divide\pageno by 3` the counter's value will be 0.

A macro definition or an assignment to a counter or dimension can be made local. If it appears inside a group, the new value will not be accessible outside that group. This way grouping can be used to make temporary changes.

In some cases a definition you are inside a group but you want to make a *global* assignment of definition. In that case you should prepend the command `\global` to `\def`, `\advance`, etc.

If you need additional token lists, you can declare your own. All the usual rules for tokens lists will apply.

`\newtoks` *tokenlistname*

Similarly you can define your own dimensions, i.e., in TEXnical sense. You can use it as any other TEX dimension.

`\newdimen` *dimensionname*

16.15.2 Branching and looping

More advanced macros often use some kind logic to determine what should be done. Therefore you need constructions like 'if *some condition* then *some action* else *some other action*'.

TEX provides a number of commands to test conditions. A test typically looks like this:

```
\if... some condition
  some action
\else
  some other action
\fi
```

There are several kinds of tests, some of which we will demonstrate shortly. The `\else` clause is always optional; any 'action' can be empty. Note that there is no 'then' keyword: the 'then' clause immediately follows the conditional clause.

There is no such thing as an 'and' operator for multiple conditions, but of course you can nest `\if` statements to achieve the same result. Similarly, there is no 'not' operator to negate a condition: you will simply have to use the `\else` clause for that purpose.

`\ifodd` *counter something* `\fi`

This command test if the following counter is odd or even. This can be useful, e.g. in a macro that is supposed to start a new chapter on an odd page:

```
\def\newpage        % jump to new page
  {\vfill\eject}
\def\newOddpage     % jump to next odd page
  {\newpage
  \ifodd\pageno     % is current page number odd?
                    % OK, no action
  \else
    \strut          % insert another empty page
    \newpage        % i.e., visually empty
  \fi}
```

Note that how put several comments in the TEX code. We recommend that you document clearly what each statement means. Your code may soon become too complex to understand at first sight. Which will lead to mistakes...

`\ifnum` *counter1 relation counter2 something* `\fi`

This command can be used to compare two counters. Three kinds of relations are possible: equal (=), greater than (>), smaller than (<). There is no 'greater or equal' or 'smaller or equal'. The next example shows a definition for `\headline` that will display a page number in roman numerals on pages with negative page numbers, and arabic numerals on other pages. Naturally this definition only works well if you set the page number to appropriate values, e.g. negative in a preface of a document, positive in the rest of the document.

```
\headline =
  {\hfil --
   \ifnum \pageno < 0
     \romannumeral -\pageno   % -- ix --
   \else
     \number \pageno          % -- 9 --
   \fi
   \ -- \hfil}
```

Note that we introduced the commands \romannumeral and \number here. Both commands take a counter as parameter and output its value in roman and arabic format respectively. As you can see you can negate a counter value by simply prepending a minus sign. Note also that if \pageno is negative, TEX will automatically increment it with −1 at each new page. This is most convenient.

\ifdim *dim1 relation dim2 something* \fi

This command is very similar to \ifnum. As its name suggests, it can be used to compare dimensions. In the next example we define a macro that typesets a thick bar of any width. We compare that width to the current column width and issue a warning in case it is larger.

```
\def\thickbar#1
  {\ifdim #1 > \hsize
     \message{Warning: thickbar width exceeds column width!
              (width=#1, hsize=\the\hsize)}
   \fi
   \hrule width #1 height 5mm\relax}
\thickbar 200pt
```

In the example above we introduced the command \message. This command can be used write messages to both the console and the log file.

\ifcase *something* \or *something* \fi

This command can be used to implement 'cases'. The commands takes a counter as parameter, which should have a non negative value that indicates a specific state. The example below is a simple program to test if TEX will survive into the next millennium.

```
\newcount\YtwoK
\YtwoK = 2000
\advance \YtwoK by -\year
\def\countdown
  {It's \the\year:                  % display the current year.
   \ifnum \YtwoK > 0                % before 2000:
     \ifcase \YtwoK                 % 1999
       Time to panic!
     \or                            % 1998
       Relax, still so much time.
     \or                            % 1997
       Millennium?
     \else                          % before 1997
       Don't worry, be happy.
     \fi
   \else                            % 2000 or later:
```

```
      \ifcase -\YtwoK                % 2000
        You've survived!
      \or                            % 2001
        Business as usual.
      \else                          % after 2001
        When's the next deadline?
      \fi
    \fi}
\countdown
```

Actually, only the value of TEX's counter `\year` determines Y2K-compliancy. This is a value that TEX gets from the operating system, so if the operating system is Y2K-compliant, so is TEX. You can run this program, then change the date on your computer and rerun the program to see what happens. See E. Frambach's *Is TEX Y2K-compliant?* for details on TEX's Y2K-compliancy.

`\ifmmode` *something* `\fi`

This command tests for mathematic mode. If you are currently in a mathematic environment, you may wish to do something differently than outside. In the next example we will show a definition that makes it more easy to write '$f'(x)$', regardless of the current environment.

```
\def\fp#1
  {\ifmmode
     f'(#1)      % no $ $ needed
   \else
     $f'(#1)$    % add $ $
   \fi}
Now both \fp x and $\fp y = 12x$ are valid.
```

In many programming languages you can declare your own conditionals ('boolean variables'). TEX is no exception:

`\newif \if`*name*

The first part `if` is required, the rest of the name you can choose freely. The reason for this is that when declaring a conditional actually three commands will be defined. If you declare, say, `\ifleapyear` then the commands `\leapyeartrue` and `\leapyearfalse` are also automatically defined. A new conditional is initially false. Writing `\leapyeartrue` changes the value of the conditional `\ifleapyear` to true; likewise `\leapyearfalse` changes it to false.

In the example below we will demonstrate (part of) an algorithm to test for a leap year. We divide the current year by 4 and then multiply it again by 4. As explained before, divisions and multiplications work on integers. So, if the result of these calcula-

tions differs from the current year, it proofs that the current year cannot be divided by 4, so it it is not a leap year.[2]

```
\newif\ifleapyear
\newcount\testleapyear
\testleapyear=\year
\divide\testleapyear by 4     % integer division!
\multiply\testleapyear by 4
\ifnum \testleapyear = \year  % year can be divided by 4
  \leapyeartrue
\else                         % year can't be divided by 4
  \leapyearfalse
\fi
\ifleapyear
  This is a leap year.
\else
  This is not a leap year.
\fi
```

A completely different kind of comparison can be implemented using the following command:

`\ifx` *token1token2* `\fi`

This command takes two parameters (or rather, tokens) that it compares. Note that you should *not* write an equal sign between the two tokens.

```
\def\this
  {something}
\def\that
  {something else}
\ifx \this \that
  they are equal
\else
  they are different
\fi
```

The command `\ifx` can also be used to test if a certain command is already defined. The trick is to compare a command name with another command name of which you are sure it is not defined. You could use such a construction e.g. to test if a set of macros is already loaded. Suppose your macros are input from another file, then that file could contain a test that prevents multiple loading.

```
% macro file

\ifx \MyMacrosLoaded \undefined
```

2 The actual algorithm for determining leap years is a little more complicated. The example merely demonstrates a technique.

```
  \def\MyMacrosLoaded {loaded!}
\else
  \message {Macros already loaded!}
  \endinput
\fi

\newcount\x
\newcount\y
\def\...
...
```

Multiple loading could cause serious problems, e.g. if you define new counters. For each new counter TEX allocates memory, even if a counter with that name already exists. Since the number of counters you can have is limited it makes sense to take precautions.

`\loop` *pre-condition stuff* `\if...` *post-condition stuff* `\repeat`

This is the general set-up for a loop. Note that pre-condition stuff can be anything, including nothing. Note also that the `\if...` clause is *not* terminated with `\fi`. As soon as the condition returns 'false' the loop will stop.

```
\newcount\linecounter             % declare a line counter
\def\writepunishmentlines#1 #2    % #1 is a number, terminated by a space
  {\linecounter=0                 % #2 is the text
   \loop                          % start the loop
   \ifnum \linecounter < #1       % test if enough lines written
     \advance \linecounter by 1   % increment line counter
     \the\linecounter.            % write the line counter (1..#1)
     #2\hfill\break               % and then the line itself
   \repeat}                       % repeat the loop

\writepunishmentlines 1000
  {I shall write properly documented programs.}
```

Note that inside a loop you cannot use `\par`. A related 'problem' you may run into is that parameters of a macro usually cannot contain more than one paragraph. In the example below we define a macro that takes one parameter. The first definition will cause an error message: `Runaway argument? ! Paragraph ended before \emphasize was complete.` The second definition will work perfectly because of the command `\long`:

```
\def\emphasize#1{{\it #1}}
\emphasize
{Be careful!

This is a pitfall.}
```

```
\long\def\emphasize#1{{\it #1}}
\emphasize
{Congratulations!

You managed to avoid this pitfall.}
```

16.15.3 Reading and writing files

We have already explained the \input command. Now we will have a closer look at some other commands that can be used to read and write files.

Before you can read anything from a file we should declare a 'handle' to it:

\newread *filehandle*

Now you can use this handle to 'open' a specific file for reading with the command

\openin *filehandle* = *file name*

Reading is done one line by line. However, more lines can be read at once if this is required to match braces. This is necessary because the contents of a line (or more) is assigned to a macro:

\read *filehandle* to *macroname*

In many cases you can't know in advance how many lines a file contains. You can use the \ifeof command to test if you haven reached the end of the file. Once you are done reading a file you should close it. This is important because TEX can 'only' have 16 files open for input simultaneously.

\closein *filehandle*

In the example below we will demonstrate how you could read lines from a file, searching for John and finding his address on the next line.

```
\newread\mydata
\newif\ifnotfound
\def\getaddress#1{%
  \openin\mydata=friends.dat          % contains my addresses
  \notfoundtrue
  \def\searchfor{#1 }                 % need a space here!
  \def\someaddress{unknown}           % default answer
  \loop
    \ifeof\mydata \notfoundfalse \fi % stop at end of file
    \ifnotfound
      \read\mydata to \someone        % read name
      \message{\someone}              % just to check
      \ifx \someone\searchfor         % found the person?
        \read\mydata to \someaddress % read person's address
        \notfoundfalse                % stop reading
```

```
        \fi
    \repeat
    \closein\mydata
    \hisaddress}

\getaddress{John}
```

The file `friends.dat` could look like this:

```
Vincent
Time Square 16, New York

John
Rembrandtsplein 9, Amsterdam

Peter
Kurf\"urstendamm 222, Berlin
```

in which case `\getaddress{John}` would yield ‘Rembrandtsplein 9, Amsterdam’.

Just like reading *from* a file you can also write *to* a file. The fist thing you will have to do is declare an output file with the command

`\newwrite` *\filehandle*

Now you can use this handle to ‘open’ a specific file for writing with the command

`\openout` *\filehandle = file name*

Note that with this command erases the content of the file if it already existed. You cannot simply append to the end of the file.

`\write` *\filehandle{anything}*

Every `\write` statement writes one line to the output file. TEX’s read and write system is only meant for managing ASCII files that can be read/written line by line. Other kinds of files will require more complex programming.

When you are done writing a file you can close the file with the command

`\closeout` *\filehandle*

TEX can only have 16 files open for output simultaneously. TEX will automatically close files at the end of a job.

A special feature of the `\write` command is that by default it is deferred until the current page is written to the output file. The reason for this is that only at that moment it is certain if a particular piece of a document will appear on the current page or perhaps somewhere else. If your application requires a write command to be executed immediately, you can prepend the command

`\immediate`

before the \write command.

In the next example we will demonstrate a few simple macros that can automatically typeset a table of contents.

```
\newwrite\toc                  % declare TOC file
\openout\toc=\jobname.toc      % \jobname yields file name of TeX job
\newcount\chaptercount         % automatical chapter counter
\def\chapter#1
  {\vfill\supereject
   \advance\chaptercount by 1
   \write\toc {Chapter \the\chaptercount: #1
               \noexpand\dotfill \the\pageno
               \vskip 2mm\noindent}
   \noindent
   Chapter \the\chaptercount: #1
   \medskip}
\def\tableofcontents
  {\closeout\toc
   \vfill\eject\noindent
   {\bf Table of Contents}
   \bigskip\noindent
   \input \jobname.toc}

\chapter{Let's start here}
My first chapter.

\chapter{And another one}
I feel lucky today.

\tableofcontents
```

In the example above we introduced the command

\noexpand *token*

which tells TEX not to 'expand' the command that follows. In this case we want TEX to write the *string* \dotfill to the table of contents file, not the actual *meaning* of that macro. The reason for this is that we want to postpone the typesetting of the table of contents. The page size may be different later, or even the meaning of \dotfill may be changed.

On the other hand, the commands \the\chaptercount and \the\pageno *need* to be expanded: we want the *current* value to be written to the table of contents file. The commands \vskip and \noindent are TEX primitives, so they will be written 'as is' to the table of contents file. The result is a table of contents file that looks like this:

```
Chapter 1: Let's start here \dotfill 1\vskip 2mm\noindent
Chapter 2: And another one \dotfill 2\vskip 2mm\noindent
```

The table of contents can only be known to TEX at the end of a job. So how can we get a table of contents at the beginning of a document? By running the TEX job twice. In the second run we can input the table that was generated in the first run.

In the next example we define a macro that tests if a table of contents is already available (actually it only tests if the file is not empty). If so, it will input the table, otherwise it will write a message instead.

```
\newread\testfile
\def\tableofcontents
  {\openin\testfile=\jobname.toc
   \ifeof\testfile
     No TOC available yet.
     Run this job once more.
   \else
     \input \jobname.toc
   \fi
   \closein\testfile}
\tableofcontents
```

Note that you must be very careful when reading and writing the same file. Opening a file for writing is equivalent to erasing it, so correct timing of opening and closing of input and output files is essential here.

16.15.4 Category codes

When TEX reads a file, it knows that some characters have a special meaning. E.g., we have already seen that the backslash, percent sign, curly braces and dollar are special. In fact, every character that TEX may encounter is classified into one of 16 categories. An overview is given in table 16.4.

You can change the category code of any character at any time. Beware, though, that many existing TEX files assume that at least the characters \ { } $ & # ^ _ % and ~ are of the category code as defined in Plain TEX.

Nevertheless, it can be very convenient to change the category codes of some characters for special purposes.

`\catcode` ‘*character* = *number*

is the command to change the category code of a character. E.g., if you have a text in which the percent sign is used a lot, and you don’t want to replace all instances of % with \%, you could make the percent sign a letter:

```
\catcode‘\%=11
```

Table 16.4: Category codes in Plain TEX

Category	Meaning	Example
0	escape character	\
1	beginning of group	{
2	end of group	}
3	math shift	$
4	alignment tab	&
5	end of line	
6	parameter	#
7	superscript	^
8	subscript	_
9	ignored character	
10	space	␣
11	letter	A–Z and a–z
12	other character	the rest
13	active character	~
14	comment character	%
15	invalid character	

Now every percent sign will be printed as such. Beware that as long as this change is in effect, you can no longer add comments to your document. But of course you can make such changes locally:

```
This is 100% good % but not quite
{\catcode`\%=11
This is 100% better}
Back to normal % don't print
```

As you can deduce from table 16.4, only the standard ASCII characters are of category 'letter'. So what about accented characters? By default the character `\"e` is an invalid character. If it appears in your document then TEX will complain about it and ignore it. By changing its category code to 'active', you can change a character's meaning to anything you want. In the next example we will define `\"e` as `\"e` and `\"\i` as `\"\i`:

```
\catcode`\"=13 % active
\catcode`\"\i=13 % active
\def\"e{\"e}
\def\"\i{\"\i}
Now you can write `ge\"eerd' and `ge\"\ind'.
```

Now you can write 'geëerd' and 'geïnd'.

The macros that define the meaning of active character can be as complex as you like. Another example is shown below. As you know TEX will not hyphenate a Dutch word such a `be\"\invloed`. Furthermore, if TEX were to hyphenate the word after 'be', then the 'trema' on the 'i' on the next line should become a single dot. Instead of spelling out all the obtrusive but necessary code, you could use an active character and keep your text much more legible.

```
\catcode`\|=13
\def|#1{%
   \ifx i#1% i is special
     \discretionary{-}{i}{\"\i}%
   \else   % all others
     \discretionary{-}{#1}{\"{#1}}%
   \fi}

\noindent
be|invloed ge|eerd

\hsize=1mm
be|invloed ge|eerd
```

beïnvloed geëerd
be-
invloed
ge-
eerd

In case you are not sure about the category code of a certain character, ask TEX:

```
\def\tellcatcode#1{%
   #1 has catcode \the\catcode`#1.\par}
\tellcatcode A
\tellcatcode /
\tellcatcode \$
\tellcatcode \%
```

A has catcode 11.
/ has catcode 12.
$ has catcode 3.
% has catcode 14.

16.15.5 Debugging

In this section about programming we have already shown a few tricks that can help you debug your macros (e.g., `\if...`, `\the` and `\showhyphens`).

The log file that TEX produces is another helpful tool. You can write debugging information to the log file yourself. The simplest command for doing that is

`\message{`*your message*`}`

This command can be used to display something, e.g. at the start of a macro. The message gets written both to the console and the log file.

`\meaning` *token*

If you want to know the current meaning of a (possibly active) character or a macro, this command will prove very helpful. If you write, e.g.:

```
\message{TeX? \meaning\TeX}
```

Then in the log file you would find:

```
macro:->T\kern -.1667em\lower .5ex\hbox {E}\kern -.125emX
```

`\showthe` *token*

can be used to display the fully expanded value of a number or dimension.

```
\showthe\hsize
```

This command would write

```
> 469.75499pt.
l.1 \showthe\hsize
```

to the console and to the log file. Note that the TEX compiler will pause after each `\showthe` command and wait for you to press Enter. This pause does *not* indicate an error, it only gives you a better chance to read the message than, e.g.,

```
\message{hsize = \the\hsize}
```

Likewise, there is the command

`\show` *token*

that can be used to display the current meaning of a macro. If you want to know how, say, the macro `\item` is defined, you could write:

```
\show\item
```

TEX would answer

```
> \item=macro:
->\par \hang \textindent .
```

Another method for detecting errors can be enabled by the command

`\pausing` = *number*

If this counter is set to 1, TEX will pause after each line it reads. Press Enter to continue, or enter 'i' to insert a TEX statement.

TEX can write a lot more information about its internal processes to the log file. These 'tracing' commands are explained below.

`\tracingmacros` = *number*

If this counter is set to a positive value, everything that happens while processing macros is written to the log file.

`\tracingcommands` = *number*

If this counter is set to a 1, TEX will show every all commands before it executes them. If set to 2, TEX will show all conditional and unconditional commands that it executes.

`\tracinglostchars` = *number*

This switch can be used to trace character that are not available in the current font.

`\tracingparagraphs` = *number*

TEX's line-breaking calculations can be traced if this counter is set to a positive value.

`\tracingpages` = *number*

The page-breaking calculations can be traced if this counter is set to a positive value.

`\tracingrestores` = *number*

When a *group* ends, TEX restores the values of anything that was changed inside the group. Setting `\tracingrestores` to a positive value allows you to trace these restores.

`\tracingstats` = *number*

TEX's internal memory usage can be traced if this counter is set to a positive value.

`\tracingonline` = *number*

All tracing information to appear on the console as well as in the log file if this counter is set to a positive value.

`\tracingall` = *number*

If this counter has a positive value, all possible tracing information is written to both the console and the log file.

Note that all these tracing commands can generate enormous amounts of information, especially `\tracingall`. Therefore it is advisable to use them very locally for debugging small pieces of TEX code.

`\errorcontextlines` = *number*

In case the TEX compiler detects an error, but the information about context of the error is not sufficient, you may want to increase the value of this variable. The default value is 5.

16.16 Typesetting mathematics

TEX is able to typeset even the most complex mathematical expressions. Nevertheless the syntax of formulas in TEX is surprisingly simple. The idea is that even the most complex formula is composed of smaller parts that are not so hard to express. Once you get the idea, you will be able to write math of any level of complexity. You start from simple elements, combine them into more complex elements, etc., etc., until the expression is complete. It doesn't matter from which element you start, as long as you keep the general structure of the expression in mind.

There are several ways to enter formulas in TEX, but before we describe them we should explain that TEX distinguishes two kinds of formulas:

- *inline*: formulas appearing in the middle of a paragraph. Within a paragraph there is little room for fractions, exponents, etc. so TEX tries to keep formulas as compact as possible.
- *displayed*: formulas that appear as paragraphs by themselves. In this case formulas can be centered and numbered if required. Fractions and other elements that require more vertical space can easily be accommodated.

Inline formulas can be started with a *single* dollar sign `$`.[3] The end of the formula is indicated with another single dollar sign.

Solve the equation $x + 1 = 3x + 5$.

```
Solve the equation $x + 1 = 3 x + 5$.
```

Displayed formulas can be started with *double* dollar signs `$$`. The end of the formula is also indicated with another pair of dollar signs.

Consider the conditions

$$f(y) + f'(y) > 0, \quad 0 < y < \infty$$

and proof that there is no solution.

```
\noindent
Consider the conditions
$$
f(y) + f'(y) > 0, \quad 0 < y < \infty
$$
and proof that there is no solution.
```

3 Remember that if you want to typeset a dollar sign you should write `\$`.

Within these mathematic environments somewhat different rules apply that you should be aware of:

- Spaces have no meaning. TEX applies rules that determine the *meaning* of everything that appears in a math environment.
- The characters a–z are usually set in an italic font and they may look different from similar text in a text paragraph. Especially the spacing between characters may be different.
- In math environments several commands are available that cannot appear outside math environments, e.g. all Greek letters.
- Some characters have a special meaning, e.g. to specify superscripts or subscripts.
- All characters that have no keyboard equivalent, e.g. ∞, γ, $\approx$, $\in$, can be entered as TEX commands.

The next example shows how the characters `^` and `_` can be used to typeset superscripts and subscripts.

x^2
x_3
x_2^3
x_2y_3
x_{2y_3}
$((x^3)^4)^5)$
$\left((x^3)^4\right)^5$

```
$ x^2 $              \par
$ x_3 $              \par
$ x_2^3 $            \par
$ x_2y_3 $           \par
$ x_{2y_3} $         \par
$ ((x^3)^4)^5) $     \par
$ {({(x^3)}^4)}^5 $ \par
```

16.16.1 Mathematical elements

TEX classifies all elements it encounters in a math environment. The most important classes it distinguishes are listed below. The tables we refer to can be found at the end of this chapter. You don't need to look them up right now.

- Ordinary characters: e.g. /, `\forall` ($\forall$) and `\partial` (∂). Table 16.5 gives a more complete listing.
- Large operators: e.g. `\sum` ($\sum$). See table 16.7.
- Binary operators: e.g. $+$. See table 16.6.
- Relational operators: e.g. $=$. See table 16.8. Negated relations, such as $\notin$, can easily be created by prepending the `\not` command to a relational operator. See also table 16.9.
- Openings: e.g. (. See table 16.10.
- Closings: e.g.). See table 16.10.
- Punctuation: e.g. , (comma) and ; (semicolon).

In the example below we will demonstrate all the elements:

$$\sum_{0}^{\infty}(x+\alpha)^2 = -10, \qquad \alpha > 0$$

```
$$ % start displayed equation
 \sum_0^{\infty}  % large operator
 (                % opening
   x              % ordinary character
   +              % binary operator
   \alpha         % ordinary character
 )                % closing
 ^2
   =              % relational operator
   -              % not binary: monadic
   10             % ordinary characters
 ,                % punctuation
 \qquad
 \alpha           % ordinary character
 >                % relational operator
 0
$$ % end displayed equation
```

Note that you could have written the same equation much more compact, if you are more familiar with the way TEX works:

$$\sum_{0}^{\infty}(x+\alpha)^2 = -10, \qquad \alpha > 0$$

```
$$\sum_0^{\infty}(x+\alpha)
^2=-10,\qquad\alpha>0$$
```

However, we recommend that you write in a more readable style, so you can more easily identify mistakes. In case TEX finds an error it only reports the line number on which it got stuck. If your code is split over multiple lines it may be easier to debug. Note also that in many cases symbols can be specified in more than one way. E.g., the 'not equal' symbol can be written as `\neq`, `\not=` or `\ne`. You can choose the one that you feel most comfortable with. Table 16.11 shows some more examples.

Greek letters are often used in mathematic formulas. Of course Plain TEX supports them. Table 16.12 lists all Greek letters that are available.

Plain TEX also defines several frequently used symbols such as 'lim', 'sin', 'log' and 'max'. These symbols are usually set in an upright roman font. Table 16.13 list them all.

A large collection of arrows, ranging from small single-pointed to hooked, long, double or double-pointed, is readily available in Plain TEX. See table 16.14 for an overview.

Of course it is also possible to underline, overline or put a tilde or brace over or under any element in a formula. Naturally roots and fractions are also supported. Table 16.15 shows some of the possibilities.

16.16.2 Arranging things horizontally and vertically

Writing mathematical formulas you will often need to align things horizontally and/or vertically. Plain TEX provides several to commands for that purpose.

`\over`

This is an unusual command because it requires an argument *before* and *after*. The next example will show how it works.

$$\frac{\frac{1}{1-x}}{2-x}$$

```
$$
{1 \over {1 - x}} \over {2 - x}
$$
```

Note that when using `\over` grouping (curly braces) is essential to indicate which is which. In case of doubt, add another pair of curly braces around the fraction that you want to express.

Related commands are

`\atop`

and

`\choose`

which are demonstrated in the next example.

$$\begin{matrix} x \\ x-1 \end{matrix} \binom{n}{k}$$

```
$$
{x \atop {x-1}}
{n \choose k}
$$
```

In some case it is more convenient to use Plain TEX's generalized version of `\over` and `\atop`:

`\above` *dimension*

The dimension represents the thickness of the line between the two parts of a fraction.

$$\frac{\frac{1}{2}}{\frac{1}{4}}$$

```
$$
{1 \over 2} \above2pt {1 \over 4}
$$
```

Matrices can be expressed in a way that may remind you of the `\halign` command. The main difference is that now you don't need to specify a template for the matrix columns.

`\matrix{` *matrix* `}`

In many case you will want braces around a matrix. A variant of the `\matrix` command does just that.

`\pmatrix{` *matrix* `}`

The example will show how easy these command are.

```
$$
\matrix{
1 & 2 & 3 & 4 \cr
5 & 6 & 7 & 8 \cr
9 & 10 & 11 & 12 \cr
}
$$
$$
A =
\pmatrix{
1 & 2 & 3 & 4 \cr
5 & 6 & 7 & 8 \cr
9 & 10 & 11 & 12 \cr
}.
$$
```

$$\matrix{
1 & 2 & 3 & 4 \cr
5 & 6 & 7 & 8 \cr
9 & 10 & 11 & 12 \cr
}$$

$$A = \pmatrix{
1 & 2 & 3 & 4 \cr
5 & 6 & 7 & 8 \cr
9 & 10 & 11 & 12 \cr
}.$$

Another kind of vertical alignment can be achieved with the command

`\cases{` *cases* `}`

which works similarly to `\matrix`.

```
$$
|x| = \cases{
      x,  & if $x \ge 0$;\cr
      -x, & otherwise.\cr}
$$
```

$$|x| = \cases{
x, & if $x \ge 0$;\cr
-x, & otherwise.\cr}$$

Study this example carefully, and you will see that there are a few unusual things about `\cases`. The first column is set in mathematical mode, whereas the second column is set in text mode. Therefore we had to add another pair of dollars around `x \ge 0`. Naturally you can have as many lines as you like in a cases statement.

Sometimes you need to typeset formulas that are too long to fit on one line. Therefore you need to break it somewhere, preferably in a logical way that retains or even enhances the structure of the formula. The command

`\eqalign{` *lines*`}`

can be used in such cases. This is again a command that resembles `\halign`.

$$\begin{aligned}(x+y)(x-y) &= x^2 - xy + yx - y^2\\ &= x^2 - y^2;\\ (x+y)^2 &= x^2 + 2xy + y^2.\end{aligned}$$

```
$$
\eqalign{
  (x + y)(x - y) &= x^2 - xy + yx - y^2 \cr
                 &= x^2 - y^2;\cr
  (x + y)^2      &= x^2 + 2xy + y^2.\cr}
$$
```

For other types of displays you may want to use a command that typesets any number of formulas without any alignment.

`\displaylines{` *formulas* `}`

Beware that within this command equations cannot be numbered using the commands explained in section 16.16.5.

$$\begin{aligned}f(x) &= x^3\\ f'(x) &= 3x^2\\ f''(x) &= 6x\end{aligned}$$

```
$$
\displaylines{
   f(x) = x^3 \cr
   f'(x) = 3x^2 \cr
   f''(x) = 6x \cr}
$$
```

16.16.3 Stretchable symbols

In certain cases you need symbols to have a size dependent on the size of (part of) an equation. In the example below we show a large fraction with square brackets around it of exactly the right size, without ever specifying that size; TEX will calculate it for us.

$$\left[\frac{\frac{1}{1-x}}{2-x}\right)$$

```
$$
\left[
{{1 \over {1 - x}} \over {2 - x}}
\right)
$$
```

The commands `\left` and `\right` were used to specify that the following symbol (a square bracket on the left, a brace on the right) should be extended as necessary to fit around the material in between. The symbols `[]{}()|` can all be stretched with this technique. Beware that curly braces must be written as `\{` and `\}`.

Note that in case you want such a delimiter only on one side, you will still have to supply both `\left` and `\right`. For the symbol that you want to omit you can specify a dot.

$$f(x) = 2^x \Longrightarrow \left\{ \begin{matrix} 1 & 2 & 4 & \dots & 256 \\ 2 & 4 & 8 & & 512 \\ \vdots & & & \ddots & \vdots \\ 16 & 32 & 64 & \dots & 4096 \end{matrix} \right.$$

```
$$
f(x) = 2^x \Longrightarrow
\left\{
\matrix{
1      & 2      & 4      & \ldots & 256\cr
2      & 4      & 8      &        & 512\cr
\vdots &        &        & \ddots & \vdots\cr
16     & 32     & 64     & \ldots & 4096\cr}
\right.
$$
```

In case you need a delimiter that just a little bigger than usual you could use the commands `\bigl` and `\bigr`.

$$\bigl\{ \{ x^3 \mid f(x) \in \{ -1,0,+1\} \} \bigr\}$$

```
$$
\bigl\{
\{ x^3 \mid f(x) \in \{ -1,0,+1\}
\bigr\}
$$
```

16.16.4 Changing sizes

TEX understands four general sizes of equations:

```
\displaystyle
```

This style is used by default in displayed equations. Inline equations can be forced to be typeset in display mode with this command. By default inline equations are typeset in

```
\textstyle
```

You can use this command in displayed equation to force TEX to typeset it as were it an inline equation.

```
\scriptstyle
```

This style is used for subscripts and superscripts. You can use it to enforce a very small style.

```
\scriptscriptstyle
```

This style is used for subsubscripts and supersuperscript, which are naturally rather small.

In the example below we will show the effect of these commands.

$$\sum_{x=2}^{\infty} \sqrt{\frac{x^2}{x-1}} \qquad \sum_{x=2}^{\infty} \sqrt{\frac{x^2}{x-1}}$$

$$\sum_{x=2}^{\infty} \sqrt{\frac{x^2}{x-1}} \qquad \sum_{x=2}^{\infty} \sqrt{\frac{x^2}{x-1}}$$

```
$$ { \displaystyle
\sum_{x=2}^\infty \sqrt{{x^2} \over {x-1}}
   \qquad} {\textstyle
\sum_{x=2}^\infty \sqrt{{x^2} \over {x-1}}}
$$
$$ { \scriptstyle
\sum_{x=2}^\infty \sqrt{{x^2} \over {x-1}}
   \qquad} { \scriptscriptstyle
\sum_{x=2}^\infty \sqrt{{x^2} \over {x-1}}}
$$
```

16.16.5 Numbering equations

Equations are often numbered. A very common style is to put a number at the right-hand margin. The Plain TEX command to achieve this is

`\eqno` *text*

If you want the equation number to appear on the left side of the equation you can use the command

`\leqno` *text*

The text can be anything, as we will demonstrate in the next example. Note that this command should be the last element within the equation: everything that follows one of these commands, up to the '$$' that ends the equation, is the equation number.

$$\tan x = \frac{\sin x}{\cos x} \qquad (1)$$

```
$$
\tan x = {{\sin x} \over {\cos x}} \eqno(1)
$$
```

(2) $$\tan x = \frac{\sin x}{\cos x}$$

```
$$
\tan x = {{\sin x} \over {\cos x}} \leqno(2)
$$
```

For equations typeset with `\eqalign` there are two variants that produce the same output with an equation number. `\eqnalignno` produces an equation number at the right margin; `\leqnalignno` produces an equation the left margin.

$$\begin{aligned}(x+y)(x-y) &= x^2 - xy + yx - y^2\\ &= x^2 - y^2; && (1)\\ (x+y)^2 &= x^2 + 2xy + y^2 && (2)\end{aligned}$$

```
$$
\eqalignno{
  (x + y)(x - y) &= x^2 - xy + yx - y^2\cr
                 &= x^2 - y^2; &(1)\cr
  (x + y)^2      &= x^2 + 2xy + y^2 &(2)\cr}
$$
```

Note that you need to specify an optional *third* column for the equation number. If you don't, that particular line will not be numbered.

16.16.6 Specifying types of elements

As explained in section 16.16.1 TEX assigns a meaning to each character it encounters in an equation. You can overrule these assignments if you need to. With the commands explained below you can make any character or sequence of character behave just like the ones already provided by Plain TEX.

`\mathord{`*character(s)*`}`

This command allows you to define your own ordinary characters in math mode. It is mostly used to redefine the meaning of characters that already were assigned as special meaning in Plain TEX. You could e.g. instruct TEX to regard `\frown` (a relational operator) as an ordinary character instead

$x + y \frown z, \qquad x + y \mathord{\frown} z$

```
$$
x + y \frown z, \qquad
x + y \mathord{\frown} z
$$
```

`\mathop{`*character(s)*`}`

This command can be used to force an element to be typeset as a large operator.

`\mathbin{`*character(s)*`}`

This command can be used to force an element to be typeset as a binary operator.

`\mathrel{`*character(s)*`}`

This command can be used to force an element to be typeset as a relational operator.

`\mathopen{`*character(s)*`}`

This command can be used to force an element to be typeset as an opening.

`\mathclose{`*character(s)*`}`

This command can be used to force an element to be typeset as a closing.

`\mathpunct{`*character(s)*`}`

This command can be used to force an element to be typeset as punctuation.

Note that all these commands do *not* redefine the meaning of any character globally or even locally, they only force TEX to typeset one instance of a character (or set of characters) to be typeset differently.

A few other commands that can be useful to allow for elements that Plain TEX does not provide are described below.

`\limits`

This commands can be used to make any set of characters to accept 'superscript' and/or 'subscript' that will be typeset similarly to e.g. an integral. I.e., the superscript and subscript are not placed *after* the characters, but above and below, respectively. Note that `\limits` should always follow a math *operator*.

$$\int_{-\infty}^{\infty} x^2, \qquad \mathop{\hbox{Integral}}\limits_{-\infty}^{\infty} x^2$$

```
$$
\int_{-\infty}^\infty x^2, \qquad
\mathop{\hbox{Integral}}
\limits_{-\infty}^\infty x^2
$$
```

In some case you may want to do the opposite: force superscript and subscript to be typeset as such in a situation where TEX would normally tread them as limits.

`\nolimits`

This command will suppress 'limits' and typeset scripts instead.

$$\sum_{x=0}^{\infty} \frac{1}{1+x}, \qquad \sum\nolimits_{x=0}^{\infty} \frac{1}{1+x}$$

```
$$
\sum_{x=0}^\infty {{1} \over {1+x}}, \qquad
\sum\nolimits_{x=0}^\infty {{1} \over {1+x}}
$$
```

We have already explained that TEX will provide appropriate spacing in math mode automatically. However, in some cases you will need to add or adjust the spacing a bit.

`\>`

This command inserts a medium sized space.

`\;`

This command inserts a thick space.

`\,`

This command inserts a small space.

```
\!
```

This command inserts a small negative space. Of course the commands `\quad` and `\qquad` can also be used insert some horizontal white space between parts of an equation.

16.16.7 Tables

This section consists of a number a tables that can be used as a reference for all kinds of mathematical elements that we have explained in the previous sections.

Table 16.5: Mathematic symbols of type 'ordinary'

$\aleph$	`\aleph`	$\prime$	`\prime`	$\forall$	`\forall`
$\hbar$	`\hbar`	$\emptyset$	`\emptyset`	$\exists$	`\exists`
$\imath$	`\imath`	∇	`\nabla`	$\neg$	`\neg`
$\jmath$	`\jmath`	$\surd$	`\surd`	$\flat$	`\flat`
ℓ	`\ell`	$\top$	`\top`	$\natural$	`\natural`
$\wp$	`\wp`	$\bot$	`\bot`	$\sharp$	`\sharp`
$\Re$	`\Re`	$\|$	`|`	$\clubsuit$	`\clubsuit`
$\Im$	`\Im`	$\angle$	`\angle`	$\diamondsuit$	`\diamondsuit`
∂	`\partial`	$\triangle$	`\triangle`	$\heartsuit$	`\heartsuit`
∞	`\infty`	$\backslash$	`\backslash`	$\spadesuit$	`\spadesuit`

Table 16.6: Mathematic binary operators

$+$	`+`	$-$	`-`		
$\pm$	`\pm`	$\cap$	`\cap`	$\vee$	`\vee`
$\mp$	`\mp`	$\cup$	`\cup`	$\wedge$	`\wedge`
$\setminus$	`\setminus`	$\uplus$	`\uplus`	$\oplus$	`\oplus`
$\cdot$	`\cdot`	$\sqcap$	`\sqcap`	$\ominus$	`\ominus`
$\times$	`\times`	$\sqcup$	`\sqcup`	$\otimes$	`\otimes`
$*$	`\ast`	$\triangleleft$	`\triangleleft`	$\oslash$	`\oslash`
$\star$	`\star`	$\triangleright$	`\triangleright`	$\odot$	`\odot`
$\diamond$	`\diamond`	$\wr$	`\wr`	$\dagger$	`\dagger`
$\circ$	`\circ`	$\bigcirc$	`\bigcirc`	$\ddagger$	`\ddagger`
$\bullet$	`\bullet`	$\bigtriangleup$	`\bigtriangleup`	$\amalg$	`\amalg`
$\div$	`\div`	$\bigtriangledown$	`\bigtriangledown`		

Table 16.7: Mathematic large operators

$\sum$	\sum	$\bigcap$	\bigcap	$\bigodot$	\bigodot
$\prod$	\prod	$\bigcup$	\bigcup	$\bigotimes$	\bigotimes
$\coprod$	\coprod	$\bigsqcup$	\bigsqcup	$\bigoplus$	\bigoplus
$\int$	\int	$\bigvee$	\bigvee	$\biguplus$	\biguplus
$\oint$	\oint	$\bigwedge$	\bigwedge		

Table 16.8: Mathematic relational operators

$<$	<	$>$	>	$=$	=
$\leq$	\leq	$\geq$	\geq	$\equiv$	\equiv
$\prec$	\prec	$\succ$	\succ	$\sim$	\sim
$\preceq$	\preceq	$\succeq$	\succeq	$\simeq$	\simeq
$\ll$	\ll	$\gg$	\gg	$\asymp$	\asymp
$\subset$	\subset	$\supset$	\supset	$\approx$	\approx
$\subseteq$	\subseteq	$\supseteq$	\supseteq	$\cong$	\cong
$\sqsubseteq$	\sqsubseteq*	$\sqsupseteq$	\sqsupseteq*	$\bowtie$	\bowtie
$\in$	\in	$\ni$	\ni	$\propto$	\propto
$\vdash$	\vdash	$\dashv$	\dashv	$\models$	\models
$\smile$	\smile	$\mid$	\mid	$\doteq$	\doteq
$\frown$	\frown	$\parallel$	\parallel	$\perp$	\perp

Table 16.9: Mathematic negations

$\not<$	\not<	$\not>$	\not>	$\neq$	\not=
$\not\leq$	\not\leq	$\not\geq$	\not\geq	$\not\equiv$	\not\equiv
$\not\prec$	\not\prec	$\not\succ$	\not\succ	$\not\sim$	\not\sim
$\not\preceq$	\not\preceq	$\not\succeq$	\not\succeq	$\not\simeq$	\not\simeq
$\not\subset$	\not\subset	$\not\supset$	\not\supset	$\not\approx$	\not\approx
$\not\subseteq$	\not\subseteq	$\not\supseteq$	\not\supseteq	$\not\cong$	\not\cong
$\not\sqsubseteq$	\not\sqsubseteq	$\not\sqsupseteq$	\not\sqsupseteq	$\not\asymp$	\not\asymp

Table 16.10: Mathematic openings and closings

(	(	[	[	{	\{
)	)	]	]	}	\}
[	\lbrack	{	\lbrace	⟨	\langle
]	\rbrack	}	\rbrace	⟩	\rangle
⌊	\lfloor	⌈	\lceil		
⌋	\rfloor	⌉	\rceil		

Table 16.11: Synonyms in math environments

≠	\ne	or \not= or \neq
≤	\le	or \leq
≥	\ge	or \geq
{	\{	or \lbrace
}	\}	or \rbrace
→	\to	or \rightarrow
←	\gets	or \leftarrow
∋	\owns	or \ni
∧	\land	or \wedge
∨	\lor	or \vee
¬	\lnot	or \neg
|	\vert	or |
‖	\Vert	or \\|

Table 16.12: Mathematic Greek letters

α	\alpha	ι	\iota	ϱ	\varrho
β	\beta	κ	\kappa	σ	\sigma
γ	\gamma	λ	\lambda	ς	\varsigma
δ	\delta	μ	\mu	τ	\tau
ϵ	\epsilon	ν	\nu	υ	\upsilon
ε	\varepsilon	ξ	\xi	ϕ	\phi
ζ	\zeta	o	o	φ	\varphi
η	\eta	π	\pi	χ	\chi
θ	\theta	ϖ	\varpi	ψ	\psi
ϑ	\vartheta	ρ	\rho	ω	\omega
Γ	\Gamma	Ξ	\Xi	Φ	\Phi
Δ	\Delta	Π	\Pi	Ψ	\Psi
Θ	\Theta	Σ	\Sigma	Ω	\Omega
Λ	\Lambda	Υ	\Upsilon		

Table 16.13: Mathematic symbols

\arccos	\cos	\csc	\exp	\ker	\limsup	\min	\sinh
\arcsin	\cosh	\deg	\gcd	\lg	\ln	\Pr	\sup
\arctan	\cot	\det	\hom	\lim	\log	\sec	\tan
\arg	\coth	\dim	\inf	\liminf	\max	\sin	\tanh

Table 16.14: Mathematic arrows

$\leftarrow$	\leftarrow	$\longleftarrow$	\longleftarrow	$\uparrow$	\uparrow
$\Leftarrow$	\Leftarrow	$\Longleftarrow$	\Longleftarrow	$\Uparrow$	\Uparrow
$\rightarrow$	\rightarrow	$\longrightarrow$	\longrightarrow	$\downarrow$	\downarrow
$\Rightarrow$	\Rightarrow	$\Longrightarrow$	\Longrightarrow	$\Downarrow$	\Downarrow
$\leftrightarrow$	\leftrightarrow	$\longleftrightarrow$	\longleftrightarrow	$\updownarrow$	\updownarrow
$\Leftrightarrow$	\Leftrightarrow	$\Longleftrightarrow$	\Longleftrightarrow	$\Updownarrow$	\Updownarrow
$\mapsto$	\mapsto	$\longmapsto$	\longmapsto	$\nearrow$	\nearrow
$\hookleftarrow$	\hookleftarrow	$\hookrightarrow$	\hookrightarrow	$\searrow$	\searrow
$\leftharpoonup$	\leftharpoonup	$\rightharpoonup$	\rightharpoonup	$\swarrow$	\swarrow
$\leftharpoondown$	\leftharpoondown	$\rightharpoondown$	\rightharpoondown	$\nwarrow$	\nwarrow
$\rightleftharpoons$	\rightleftharpoons				

Table 16.15: Special constructions in math environments

$\widetilde{abc}$	`\widetilde{abc}`	$\widehat{abc}$	`\widehat{abc}`
$\overleftarrow{abc}$	`\overleftarrow{abc}`	$\overrightarrow{abc}$	`\overrightarrow{abc}`
$\overline{abc}$	`\overline{abc}`	$\underline{abc}$	`\underline{abc}`
$\overbrace{abc}$	`\overbrace{abc}`	$\underbrace{abc}$	`\underbrace{abc}`
$\sqrt{abc}$	`\sqrt{abc}`	$\sqrt[n]{abc}$	`\sqrt[n]{abc}`
f'	`f'`	$\frac{abc}{xyz}$	`{abc \over xyz}`

Table 16.16: Mathematical accents

`\hat a`	$\hat{a}$	`\check a`	$\check{a}$
`\tilde a`	$\tilde{a}$	`\acute a`	$\acute{a}$
`\grave a`	$\grave{a}$	`\dot a`	$\dot{a}$
`\ddot a`	$\ddot{a}$	`\breve a`	$\breve{a}$
`\bar a`	$\bar{a}$	`\vec a`	$\vec{a}$

Chapter 17

LaTeX: Lamport's approach

This introduction to LaTeX is largely based on *The Not So Short Introduction to LaTeX 2ε* by T. Oetiker (cdrom), which in turn is based on the *LaTeX 2ε-Kurzbeschreibung* by T. Knappen, H. Partl, E. Schlegl, and I. Hyna (cdrom). Parts of this introduction are based on P. van Oostrum's *Handleiding LaTeX* (cdrom).

Reference books and public domain documents on LaTeX are: *LaTeX — A Document Preparation System* by L. Lamport, *The LaTeX Companion* by M. Goossens, F. Mittelbach and A. Samarin, *The LaTeX Graphics Companion* by M. Goossens, S. Rahtz and F. Mittelbach, *LaTeX 2ε for Authors* by the LaTeX3 Project Team (cdrom) and *LaTeX 2ε for Class and Package Writers* by the LaTeX3 Project Team (cdrom). The *LaTeX Command Summary* by C. Biemesderfer (cdrom) should be very helpful when you start using LaTeX. The *TeX Reference Card* by J. Silverman (cdrom) is also very handy. On the 4allTeX CDROM you will find many other documents on LaTeX.

17.1 The structure of a LaTeX document

A typical LaTeX input file looks like this:

```
\documentclass[options]{class}
preamble stuff
\begin{document}
the real stuff
\end{document}
```

The `\documentclass` command defines what *class* of LaTeX document this is. Options can be supplied within square brackets.

The actual content of the document should always follow within the so-called document *environment*, which is started by entering `\begin{document}` and ended by entering `\end{document}`. The part between `\documentclass` and `\begin{document}` is called the *preamble*.

Let us first look at document *classes*. A document class can be one of the standard classes that are distributed with LaTeX or one that you or someone else wrote. The most important standard classes are:

article A standard for scientific articles.

book For writing books.

report For writing reports and manuals.

letter For writing letters.

slides For making 'slides' (transparencies, sheets for overhead projection).

proc For making conference proceedings (based on the **article** class).

minimal This class is the bare minimum (3 lines) that is needed in a LaTeX class file. It just sets the text width and height, and defines `\normalsize`. It is principally intended for debugging and testing LaTeX code in situations where you do not need to load a 'full' class such as **article**. If, however, you are designing a completely new class that is aimed for documents with structure radically different from the structure supplied by the article class, then it may make sense to use this as a base and add to it code implementing the required structure, rather than starting from **article** and modifying the code there.

Here are a few non-standard classes that you may find useful:

artikel1, artikel2, artikel3 These classes are functionally equivalent to the standard LaTeX class **article** but the layout adheres more to Dutch traditions. Most notably headers (sections, subsection, etc.) are much more sober and white space is treated differently.

rapport1, rapport3 These classes are functionally equivalent to the standard LaTeX class **report** but the layout adheres more to Dutch traditions, in line with the **artikel***x* classes.

boek This class is functionally equivalent to the standard LaTeX class **book** but the layout adheres more to Dutch traditions, in line with the **artikel***x* classes.

Possible 'options' are listed below. Note that in case you use a non standard LaTeX class the options may be different.

10pt Sets the size of the main font for the document. If no option is specified, `10pt` is assumed. Other size options are **11pt** and **12pt**

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

fleqn Typesets displayed formulae left-aligned instead of centered.

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

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

twocolumn Instructs LaTeX to typeset the document in two columns.

twoside, oneside Specifies whether double or single sided output should be generated. The classes `article` and `report` are *single sided* and the `book` class is *double sided* by default.

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

You can supply multiple options if you separate them by commas, e.g. like this:

```
\documentclass[11pt,twoside,fleqn]{report}
```

The 'preamble stuff' can be anything as long as it doesn't produce any output. Usually the preamble consists of definitions (macros) that you need in your document, and commands to load some LaTeX 'packages'. We will discuss packages in detail in section 17.20.

17.2 Big projects

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

`\include{`*file name*`}`

and with

`\input{`*file name*`}`

you can insert the contents of another file.

```
\documentclass[a4paper,twoside,11pt]{article}
\usepackage{times}
\author{H. Partl}
\title{Minimalism}
\begin{document}
\maketitle
\tableofcontents
\include{Preface}
\section{Start}
Well and here begins my lovely article.
\section{End}
\ldots{} and here it ends.
\end{document}
```

Figure 17.1: Example of a realistic journal article

`\input` can be used in the preamble whereas `\include` cannot. The `\include` command allows you to instruct LaTeX to only input some of the `\included` files. If you write in the preamble:

`\includeonly{`*somepart,anotherpart*`}`

then LaTeX will only execute the include statements `\include{somepart}` and `\include{anotherpart}`. All other `\include` statements will be ignored. You can specify any number of file names with the `\includeonly` command. Note that file names must be separated by commas and that there should be no spaces in the argument.

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

17.3 LaTeX input files

Now that we have established the basic structure of a LaTeX input file, let us have a look at the smallest details: the text as you type it.

In figure 17.1 we have set up a more or less realistic example of a LaTeX document.

In this example LaTeX is instructed to typeset the document as an *article* with a base font size of *eleven points* and to produce a layout suitable for *double sided* printing on *A4 paper*, and the article is to be typeset in Times fonts.

The interpretation of the rest of the example is left to the reader as an exercise. The commands should look familiar by now.

17.3.1 Spaces and paragraphs

'White space' characters such as blank or tab are treated uniformly as 'space' by LaTeX. *Several consecutive* whitespace characters are treated as *one* 'space'. Whitespace at the start of a line is generally ignored and a single linebreak is treated like a 'space'.

A special kind of space is the tilde: ~. It is defined as a 'non breakable' space. It means that this space looks like any ordinary space, but the two words that it connect will never be broken over two lines. Here are a few cases in which it is useful to keep words together:

```
Mr.~White
See Fig.~24
chapter~8
```

To get a straight right margin in the output, LaTeX inserts varying amounts of space between the words. At the end of a sentence it inserts slightly more space, as this makes the text more readable. LaTeX assumes that sentences end with periods, question marks or exclamation marks. If a period follows an uppercase letter this is not taken as a sentence ending since periods after uppercase letters are normally for abbreviations.

Any exception from these assumptions has to be specified by the author. A backslash in front of a space generates a space which will not be enlarged. The command

```
\@
```

in front of a period specifies, that this period terminates a sentence even when it follows a uppercase letter. Here is an example:

```
I like BASIC\@. And you?
```

The additional space after periods can be disabled with the command

```
\frenchspacing
```

which tells LaTeX *not* to insert any more space after a period than after ordinary character. This is very common in non-English languages. If you use `\frenchspacing`, the command `\@` is not necessary.

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

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

An empty line starts a new
paragraph.
```

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

An empty line starts a new paragraph.

In the example above you could see that an empty line starts a new paragraph. In this case the output show an empty line between the two paragraphs. Note that this is not necessarily so. It is usually the document class (or perhaps a LaTeX package) that determines what should happen at the start of a new paragraph. E.g., instead of white space between paragraphs you could have no space at all, but you could indent the first line of the second paragraph.

You can set the indentation (`\parindent`) and the space between paragraphs (`\parskip`) as follows:

```
\setlength{\parindent}{10mm}
\setlength{\parskip}{0mm}
```

This is typically something you would write in the preamble of your LaTeX document.

17.3.2 Vertical space

The space between paragraphs, sections, subsections, ... is determined automatically by LaTeX. If necessary, additional vertical space *between two paragraphs* can be added with the command

`\vspace{`*length*`}`

This command should normally be used between two empty lines. If the space should be preserved at the top or at the bottom of a page, use the starred version of the command `\vspace*` instead of `\vspace`.

The `\stretch` command in connection with `\pagebreak` can be used to typeset text on the last line of a page, or to center text vertically on a page.

```
Some text \ldots

\vspace{\stretch{1}}
This goes onto the last line of the page.\pagebreak
```

Additional space between two lines of *the same* paragraph or within a table is specified with the command

`\\[`*length*`]`

In the example above we used the

`\pagebreak[`*strength*`]`

command. You can encourage LaTeX to break a page at a certain point. The *strength* is a number from 0 to 4. The higher the number, the stronger the encouragement. Likewise there is

`\nopagebreak[`*strength*`]`

to do the opposite: discourage pagebreaks. In case you find no satisfactory pagebreaks the command

```
\enlargethispage{length}
```

may be helpful. With this command you can can enlarge the current page (or shrink it: *length* may be negative).

Depending on the class file, LaTeX may stretch blank space on pages (e.g. before the start of a new section) to make the last line of all pages appear at the same height. This is accomplished with the command

```
\flushbottom
```

The counterpart of `\flushbottom` is

```
\raggedbottom
```

The spacing of words may also cause problems. If LaTeX cannot find a good breakpoint a line may stick out into the right-hand margin. The command

```
\sloppy
```

tells LaTeX that you would rather have more space between words if there is no other way to make words fit. It is advisable to use this command only if there is no alternative. The counterpart of this command is

```
\fussy
```

You should use these commands preferably locally.

17.3.3 Reserved characters

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

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

As you will see, these characters can be used in your documents all the same by adding a prefix backslash: `\$ \& \% \# \_ \{ \{`.

The other symbols and many more can be printed with special commands in mathematical formulae or as accents. The backslash character `\` can *not* be entered by adding another backslash in front of it (`\\`) as this sequence is used for linebreaking. Try `$\backslash$` or `\textbackslash` instead to produce \.

17.3.4 Comments

When LaTeX encounters a % character while processing an input file, it ignores the rest of the present line. This is useful for adding notes to the input file, which will not show up in the printed version. Below we will give an example. On the right you can see the actual input, on the left the output that LaTeX will produce:

This is an example. But this is 100I think

```
This is an % stupid
% Better: instructive <----
example.
But this is 100% wrong
I think
```

17.3.5 Quotation marks

For quotation marks you should *not* use the " as on a typewriter. In publishing there are special opening and closing quotation marks. In LaTeX, use two ‘s on for opening quotation marks and two ’s for closing quotation marks.

‘Please press the ‘x’ key.’

```
‘Please press the ‘x’ key.’
```

17.3.6 Dashes and hyphens

LaTeX knows four kinds of dashes. You can access three of these with different numbers of consecutive dashes. The fourth sign is actually no dash at all: it is the mathematical minus sign. The differences are subtle but significant:

daughter-in-law, X-rated
pages 13–67
yes—or no?
0, 1 and −1

```
daughter-in-law, X-rated\\
pages 13--67\\
yes---or no? \\
$0$, $1$ and $-1$
```

The names for these dashes are: hyphen, en-dash, em-dash and minus sign.

17.3.7 Ellipsis

On a traditional typewriter a comma or a period takes the same amount of space as any other letter. In book printing these characters occupy only a little space and are set very close to the preceding letter. Therefore you cannot enter ‘ellipsis’ by just typing three dots, as the spacing would be wrong. Besides that there is a special command for these dots. It is called `\ldots` and here is an example:

Not like that ... but like that:
New York, Tokyo, Budapest, . . .

```
Not like that ... but like that:\\
New York, Tokyo, Budapest, \ldots
```

Table 17.1: Accents and special characters

Input	Output	Input	Output
`ò`	ò	`ó`	ó
`ô`	ô	`\~o`	õ
`\=o`	ō	`\.o`	ȯ
`\u o`	ŏ	`\v o`	ǒ
`\H o`	ő	`ö`	ö
`\c o`	o̧	`\d o`	ọ
`\b o`	o̲	`\t oo`	o͡o
`\oe`	œ	`\OE`	Œ
`æ`	æ	`Æ`	Æ
`\aa`	å	`\AA`	Å
`ö`	ö	`\O`	Ø
`\l`	ł	`\L`	Ł
`\i`	ı	`\j`	ȷ
`!‘`	¡	`?‘`	¿

17.3.8 Accents and special characters

LaTeX supports the use of accents and special characters from many languages. Table 17.1 shows all sorts of accents being applied to the letter o. Naturally other letters work too.

To place an accent on top of an i or a j, their dots have to be removed. This is accomplished by typing `\i` and `\j`.

Hôtel, naïve, élève,
smørrebrød, ¡Señorita!,
Schönbrunner Schloß Straße

```
Hôtel, naï ve, élève,\\
smörrebröd, !‘Señorita!,\\
Schönbrunner Schlo\ss{}
Stra\ss e
```

You can also type in accented characters directly, provided that you use the `inputenc` package. See section 17.16.2 for details on this method.

17.3.9 Emphasized words

In manuscripts produced on traditional typewriters, important words get underlined. In printed books these words are *emphasized*. The command to switch to an *emphasized* font is called

`\emph{`*text*`}`

Its argument is the text to be emphasized.

```
\emph{If you use
  \emph{emphasizing} in an
  already emphasized text, then
  \LaTeX{} uses an
  \emph{upright} font for
  emphasizing.}
```

If you use emphasizing *in an already emphasized text, then LaTeX uses an* upright *font for emphasizing.*

17.3.10 Font families and shapes

In normal text (as opposed to mathematic environments) you can select different font *shapes* using the following commands:

`\textrm`	`\rmfamily`	normal (roman)
`\textsf`	`\sffamily`	sans serif
`\texttt`	`\ttfamily`	`typewriter`
`\textbf`	`\bfseries`	**bold**
`\textup`	`\upshape`	upright
`\textit`	`\itshape`	*italic*
`\textsl`	`\slshape`	*slanted*
`\textsc`	`\scshape`	CAPS AND SMALL CAPS

Note that the commands in the first column all take one argument:

`\textxx{`*text*`}`

and that the argument cannot contain more than one paragraph.

The commands in the second column do *not* take any argument: all text following such a command will be affected. But of course you can keep any change local like this:

```
This is normal text, {\itshape now some
italic}, and back to normal with some
{\bfseries bold} text. If you prefer to write
\textit{italic} that is fine. You can even
write \texttt{\textsl{slanted typewriter}}
if you like.
```

This is normal text, *now some italic*, and back to normal with some **bold** text. If you prefer to write *italic* that is fine. You can even write ***slanted typewriter*** if you like.

LaTeX provides the following commands to change the font *size*:

`\tiny`	Just an example
`\scriptsize`	Just an example
`\footnotesize`	Just an example

\small Just an example
\normalsize Just an example
\large Just an example
\Large Just an example
\LARGE Just an example
\huge Just an example
\Huge Just an example

These commands also do *not* take any argument, so you will often make them act locally as in the example given above.

17.4 Titles, chapters, and sections

To help the reader find his or her way through your work, you should divide it into chapters, sections, and subsections. LaTeX supports this with special commands which take the section title as their argument. It is up to you to use them in the correct order.

For the `article` class the following sectioning commands are available:

```
\section{name}          \paragraph{name}
\subsection{name}       \subparagraph{name}
\subsubsection{name}    \appendix
```

For the `report` and the `book` class you can use two additional sectioning commands:

```
\part{name}
```

and

```
\chapter{name}
```

As the `article` class does not know about chapters, it is quite easy to add articles as chapters to a book. The spacing between sections, the numbering and the font size of the titles will be set automatically by LaTeX. Two of the sectioning commands are a bit special:

- The `\part` command does not influence the numbering sequence of chapters.
- The `\appendix` command does not take an argument. It just changes the chapter numbering to letters.[1]

1 For the article style it changes the section numbering.

LaTeX creates a table of contents by taking the section headings and page numbers from the previous run of the document. The command

```
\tableofcontents
```

expands to a table of contents at the place where it is issued. A new document has to be processed twice to get a correct table of contents. In some circumstances it might be necessary to compile the document a third time. LaTeX will tell you when this is necessary.

All sectioning commands listed above also exist as 'starred' versions. A 'starred' version of a command is built by adding a star * after the command name. They generate section headings which will not show up in the table of contents and which will not get numbered. The command `\section{Help}` for example would become `\section*{Help}`.

Normally the section headings show up in the table of contents exactly as they were entered in the text. Sometimes this is not possible, because the heading is too long to fit into the table of contents. The entry for the table of contents can therefore be specified as an optional argument (within square brackets) before the actual heading.

```
\chapter[Read it! It is Exciting]{This
   is a very long
   and especially boring title}
```

The *title* of the whole document is generated by issuing a

```
\maketitle
```

command. The contents of the title has to be defined by the commands

```
\title{name}
```

```
\author{name}
```

and optionally

```
\date[date]
```

before calling `\maketitle`. In the argument of `\author` you can supply several names separated by `\and` commands. Here is an example of some of the above mentioned commands:

```
\documentclass{article}

\title{A simple sample}
\author{My Name\thanks{My Company, address, etc.}
        \and
        The Other Author's Name\thanks{Her Company, etc.}
        \and Some Other VIP}
\date{May 1998}
```

```
\begin{document}
\maketitle
\section{...}
...
\end{document}
```

As you can see we slipped in two more commands:

```
\thanks{...}
```

that produces a footnote with e.g. your credentials, and

```
\and
```

which provides for a nice layout in case of multiple authors.

Apart from the sectioning commands explained above, three additional commands for use with the `book` class are available:

```
\frontmatter
```

```
\mainmatter
```

```
\backmatter
```

They are useful for dividing your publication. The commands alter chapter headings, and page numbering to work as you would expect it in a book.

17.5 Environments

To typeset special purpose text, LaTeX defines many different `environments` for all sorts of formatting:

```
\begin{name}  text  \end{name}
```

where *name* is the name of the environment. Environments can be called several times within each other as long as the calling order is maintained.

```
\begin{aaa}... \begin{bbb}... \end{bbb}... \end{aaa}
```

You can also define your own environments. In section 17.15 we will show you how it is done.

In the following sections the most important environments are explained.

17.5.1 Itemize, enumerate, and description

For simple lists The `itemize` environment is suitable for simple lists, the `enumerate` environment for enumerated lists, and the `description` environment for descriptions.

1. You can mix the list environments to your taste:
   - But it might start to look silly.
   – With a dash.
2. Therefore remember:

   Stupid things will not become smart because they are in a list.

   Smart things though, can be presented beautifully in a list.

```
\begin{enumerate}
\item You can mix the list
   environments to your taste:
   \begin{itemize}
   \uselistsetting{compact}
     \item But it might start to
      look silly.
     \item[--] With a dash.
   \end{itemize}
\item Therefore remember:
   \begin{description}
   \item[Stupid] things will not
      become smart because they are
      in a list.
   \item[Smart] things though, can be
      presented beautifully in a list.
   \end{description}
\end{enumerate}
```

17.5.2 Flushleft, flushright, and center

The environments `flushleft` and `flushright` generate paragraphs which are either left or *right aligned*.

This text is
left aligned. LaTeX is not trying to make each line the same length.

This text is right
aligned. LaTeX is not trying to make each line the same length.

```
\begin{flushleft}
This text is\\ left aligned.
\LaTeX{} is not trying to make
each line the same length.
\end{flushleft}

\begin{flushright}
This text is right\\ aligned.
\LaTeX{} is not trying to make
each line the same length.
\end{flushright}
```

The `center` environment generates centered text. If you do not issue `\\` to specify linebreaks, LaTeX will automatically determine linebreaks.

At the centre
of the
earth

```
\begin{center}
At the centre\\of the\\ earth
\end{center}
```

17.5.3 Quote, quotation, and verse

The `quote` environment is useful for quotes, important phrases and examples.

A typographical rule of thumb for the line length is:

> No line should contain more than 66 characters.
>
> This is why LaTeX pages have such large borders by default.

That is why multicolumn print is often used in newspapers.

```
A typographical rule of thumb
for the line length is:
\begin{quote}
No line should contain more than
66~characters.

This is why \LaTeX{} pages have
such large borders by default.
\end{quote}
That is why multicolumn print is
often used in newspapers.
```

There are two similar environments: the `quotation` and the `verse` environments. The `quotation` environment is useful for longer quotes going over several paragraphs, because it does indent paragraphs. The `verse` environment is useful for poems where the linebreaks are important. The lines are separated by issuing a `\\` at the end of a line and a empty line after each verse.

I know only one English poem by heart. It is about Humpty Dumpty.

> Humpty Dumpty sat on a wall:
> Humpty Dumpty had a great fall.
> All the King's horses and all the King's men
> Couldn't put Humpty together again.

```
I know only one English poem by
heart. It is about Humpty Dumpty.
\begin{verse}
Humpty Dumpty sat on a wall:\\
Humpty Dumpty had a great fall.\\
All the King's horses and all the King's men\\
Couldn't put Humpty together again.
\end{verse}
```

17.5.4 Tabular

The `tabular` environment can be used to typeset beautiful tables with optional horizontal and vertical lines. LaTeX determines the width of the columns automatically.

The *table spec* argument of the

`\begin{tabular}`(*table spec*)

command defines the format of the table. Use an `l` for a column of left aligned text, `r` for right aligned text and `c` for centered text, `p{`*width*`}` for a column containing justified text with linebreaks, and `|` for a vertical line.

Within a `tabular` environment the character `&` jumps to the next column, `\\` starts a new line and the command `\hline` inserts a horizontal line.

7C0	hexadecimal
3700	octal
11111000000	binary
1984	decimal

```
\begin{tabular}{|r|l|}
\hline
7C0 & hexadecimal \\
3700 & octal \\
11111000000 & binary \\
\hline \hline
1984 & decimal \\
\hline
\end{tabular}
```

Welcome to Boxy's paragraph. We sincerely hope you will all enjoy the show.

```
\begin{tabular}{|p{4.7cm}|}
\hline
Welcome to Boxy's paragraph.
We sincerely hope you will
all enjoy the show.\\
\hline
\end{tabular}
```

With the `@{...}` construct it is possible to specify the column separator. This command kills the intercolumn space and replaces it with whatever is included in the curly braces. One common use for this command is explained below in the decimal alignment problem. Another possible usage is to suppress leading space in a table with `@{}`.

no leading space

```
\begin{tabular}{@{} l @{}}
\hline
no leading space\\
\hline
\end{tabular}
```

leading space left and right

```
\begin{tabular}{l}
\hline
leading space left and right\\
\hline
\end{tabular}
```

Since there is no built-in way to align numeric columns on a decimal point[2] we can 'cheat' and do it by using two columns: a right-aligned integer and a left-aligned fraction. The `@{.}` command in the `\begin{tabular}` line replaces the normal intercolumn spacing with just a '.', giving the appearance of a single, decimal-point-justified

2 The `dcolumn` package provides alignment on arbitrary characters.

column. Don't forget to replace the decimal point in your numbers with a column separator (&)! A column label can be placed above our numeric 'column' by using the \multicolumn command.

Pi expression	Value
π	3.1416
π^{π}	36.46
$(\pi^{\pi})^{\pi}$	80662.7

```
\begin{tabular}{c r @{.} l}
Pi expression       &
\multicolumn{2}{c}{Value} \\
\hline
$\pi$               & 3&1416  \\
$\pi^{\pi}$         & 36&46   \\
$(\pi^{\pi})^{\pi}$ & 80662&7 \\
\end{tabular}
```

17.5.5 Tabbing

The tabbing environment allows you to use tab stops much like on an old-fashioned typewriter.

The command

```
\=
```

defines a tab position. Tab positions are usually set on the first line of the table, but you can also (re)define them within the table, e.g. if you need different alignments in certain parts of the table. Tab positions can also be set to align with text that will not appear on the first line (or not at all). The command

```
\kill
```

will accomplish this. Extra space can be generated with the

```
\hspace{length}
```

command, e.g. like this: \hspace{3cm}. Note that space at the beginning of a line is ignored. If you do want space there you should use the 'starred' version: \hspace*{3cm}. See table 16.1 in chapter 16 for an overview of supported length units. The command

```
\>
```

generates a jump to the next tab position. Note that unlike a typewriter LaTeX will jump to the exact tab position that you specified, even if you have already 'passed' that position, which can result in text being overprinted. The command

```
\\
```

is a 'Carriage Return'.

```
\begin{tabbing}
was eens\hspace{5mm} \=
 middenstuk\hspace{5mm} \= \kill
links \> middenstuk \> rechts\\
Er \\
was eens \> maar nu
 \> niet meer,\\
een, \>  \> gerepareerde\\
 \>  \> teddybeer
\end{tabbing}
```

links middenstuk rechts
Er
was eens maar nu niet meer,
een, gerepareerde
teddybeer

Less commonly used commands in the `tabbing` environment are:

`\+`

Make all following lines jump one tab position extra so you will need to specify one less `\>`.

`\-`

Cancel the `\+` command for the following lines.

`\<`

Cancel the `\+` command only for the current line. Can only be used at the start of a line.

`\'`

Make the preceding text align to the right (the space between columns is defined by `\tabbingsep`.

`\`

Move the rest of the current line to the right. The line must be ended with `\\`.

Beware that the commands for putting accents on characters (`\=`, `\'` and `\`) have a different meaning in a `tabbing` environment. You can still specify accents with the `\a` command like this: `\a=`, `\a'`, `\a`.

```
\begin{tabbing}
Als \= het vriest \+\\
en \= het regent \+\\
dan hebben we ijzel \\
én we glijden uit \\
\< of \> het sneeuwt \\
dan wordt alles wit\-\-\\
anders is alles gewoon
\end{tabbing}
```

Als het vriest
    en het regent
        dan hebben we ijzel
    én we glijden uit
    of het sneeuwt
        dan wordt alles wit
anders is alles gewoon

Note that the tabbing environment cannot be used within certain other environments, and it cannot be nested within another tabbing environment.

17.5.6 Printing verbatim

Text which is enclosed in a verbatim environment

```
\begin{verbatim}
```

will be directly printed, as if it was typed on a typewriter, with all linebreaks and spaces, without any LaTeX command being executed.

```
10 PRINT "HELLO \TeX\ WORLD ";
20 GOTO 10
```

```
\begin{verbatim}
10 PRINT "HELLO \TeX\ WORLD ";
20 GOTO 10
\end{verbatim}
```

The verbatim* environment is a special case of print verbatim: all spaces are made 'visible':

```
the␣starred␣version␣of
the␣␣␣␣␣␣␣verbatim
environment␣emphasizes
the␣spaces␣␣␣␣in␣the␣text
```

```
\begin{verbatim*}
the starred version of
the       verbatim
environment emphasizes
the spaces    in the text
\end{verbatim*}
```

Within a paragraph, similar functionality can be accessed with \verb+*text*+. The + is just an example of a delimiter character. You can use any character except letters, * or blank. Many LaTeX examples in this chapter are typeset with this command.

The \ldots command ...

```
The \verb|\ldots| command \ldots
```

The \verb command can be used in a similar fashion with a star:

```
like␣␣␣this␣:-)␣
```

```
\verb*|like   this :-) |
```

The verbatim environment and the \verb command may not be used within parameters of other commands.

17.5.7 Letters

The document class `letter` offers a special environment for writing letters. One such document may contain any number of letters, each specified within a `letter` environment. Below is an example of such a document.

```
\documentclass[12pt]{letter}
\address{Faraway 111\\
         Sometown\\
         Acountry}
\signature{My Name}
\begin{document}
\begin{letter}{My Friend\\
               Memorylane 432\\
               Sametown}
\opening{Dear Friend,}
Thanks for....

..... See you later.
\closing{Love,}
\end{letter}
\begin{letter}{Another Friend\\
               Propylane 98\\
               Anothertown}
\opening{Dear Friend,}
How are you today....
..... Will call you soon.
\closing{Greetings,}
\end{letter}
\end{document}
```

The commands

```
\signature{...}
```

and

```
\address{...}
```

define the name and address of the sender.

```
\begin{letter}{...}
```

The second argument of the `\begin{letter}{...}` command defines to whom this letter will be sent. You can specify a full address using `\\` to start new lines.

```
\opening{...}
```

needs no explanation, other than that LaTeX will insert some white space between the opening and the text of the letter.

```
\closing{...}
```

defines, together with the `\signature` command, the ending of the letter.

17.5.8 Slides

To produce transparencies ('slides') for use on overhead projectors the document class `slides` can be used. This document class provides the environment

```
\begin{slide}
```

Note that the `slides` document class does not provide a `figure` or `table` environment. Floating bodies would not make sense because with slides you want to determine very precisely what should go on each slide. The commands `\section`, `\subsection`, etc. are also not supported.

Below is an example of a document containing two simple slides.

```
\documentclass{slides}
\begin{document}

\begin{slide}
\begin{center}
\Large\textbf{Contents}
\end{center}
\begin{enumerate}
\item The structure of a document
\item Big projects
\item \LaTeX\ input files
\end{enumerate}
\end{slide}

\begin{slide}
\begin{center}
\Large\textbf{1. The structure of a document}
\end{center}
...
\end{slide}
\end{document}
```

LaTeX will automatically use much bigger fonts than you would get with, e.g. the `article` class. By default sans serif fonts are used.

The slides class offers a few more advanced features for producing slides. The environment

```
\begin{overlay}
```

can be used to define 'overlays': slides that are meant to be put on top of the previous slide. This environment behaves just like a normal slide environment, except that the page number will be the same as the previous slide, with '-a' appended to it. Very useful commands in slide/overlay combinations are

```
\invisible
```

and

```
\visible
```

They can be used to make parts of a slide 'invisible', which parts can be filled in by the following overlay(s). Here is an example:

```
\begin{slide}
A {\invisible \emph{slide} environment}
can be used for making slides.
\end{slide}
\begin{overlay}
\invisible
A {\visible \emph{slide} environment}
can be used for making slides.
\end{overlay}
```

If you want to write notes between your slides you can use the environment

```
\begin{note}
```

Notes will be typeset on separate pages. You can suppress notes in the output with the command

```
\onlyslides{numbers}
```

The *numbers* parameter should contain the numbers of the slides you want in the output. The numbers can be specified comma delimited: `\onlyslides{1,3,4,9,10}` and/or as ranges: `\onlyslides{1-9,12-13,19,20}` (corresponding overlays, if any, will also be output). This command should be given in the preamble of the document. Likewise there is

```
\onlynotes{numbers}
```

if you want to print just the notes. The syntax is the same.

The document class option `clock` can be used to plan your presentation precisely. Right before or after a slide you can put a

```
\addtime{seconds}
```

command that specifies how much time you intend to spend here. The total time taken so far will be printed at the bottom of each note. You can reset the clock with the command

```
\settime{seconds}
```

Do not use any time commands inside slide, overlay or note environments.

17.6 Page styles

LaTeX supports three predefined header/footer combinations — so-called 'page styles'.

```
\pagestyle{style}
```

The *style* parameter defines which one to use. Here is a list of the predefined page styles:

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

headings prints the current chapter heading and the page number in the header on each page while the footer remains empty.

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

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

```
\thispagestyle{style}
```

In *The LaTeX Companion* by M. Goossens, F. Mittelbach and A. Samarin you can find a description of how to create your own headers and footers. A probably more convenient way to do that is by using the `fancyhdr` package.

The `fancyhdr` package provides a few simple commands which allow you to customize the header and footer lines of your document. Figure 17.2 shows how you could use LaTeX commands and commands provided by the `fancyhdr` package to set up headers and footers.

The tricky problem when customizing headers and footers is to get things like running section and chapter names in there. LaTeX accomplishes this with a two stage approach. In the header and footer definition you use the commands `\rightmark` and `\leftmark` to represent the current chapter and section heading respectively. The values of these two commands are overwritten whenever a chapter or section command is processed.

For ultimate flexibility, the `\chapter` command and its friends do not redefine `\rightmark` and `\leftmark` themselves, they call yet another command called

```
\documentclass{book}
\usepackage{fancyhdr}
\pagestyle{fancy}
% with this we ensure that the chapter and section
% headings are in lowercase:
\renewcommand{\chaptermark}[1]{\markboth{#1}{}}
\renewcommand{\sectionmark}[1]{\markright{\thesection\ #1}}
% delete current setting for header and footer:
\fancyhf{}
% on the left side of even pages and
% on the right side of odd pages we want a page number:
\fancyhead[LE,RO]{\bfseries\thepage}
% on the left side of odd pages we want section info:
\fancyhead[LO]{\bfseries\rightmark}
% on the right side of even pages we want chapter info:
\fancyhead[RE]{\bfseries\leftmark}
% we want a rule below the header:
\renewcommand{\headrulewidth}{0.5pt}
% we want no rule above the footer:
\renewcommand{\footrulewidth}{0pt}
% make space for the rule (or LaTeX will complain):
\addtolength{\headheight}{0.5pt} % make space for the rule
% define the style for 'fancy' pages (first page of chapters):
\fancypagestyle{plain}{%
   % no headers here:
   \fancyhead{}
   % and no headrule:
   \renewcommand{\headrulewidth}{0pt}}
```

Figure 17.2: Example `fancyhdr` set-up

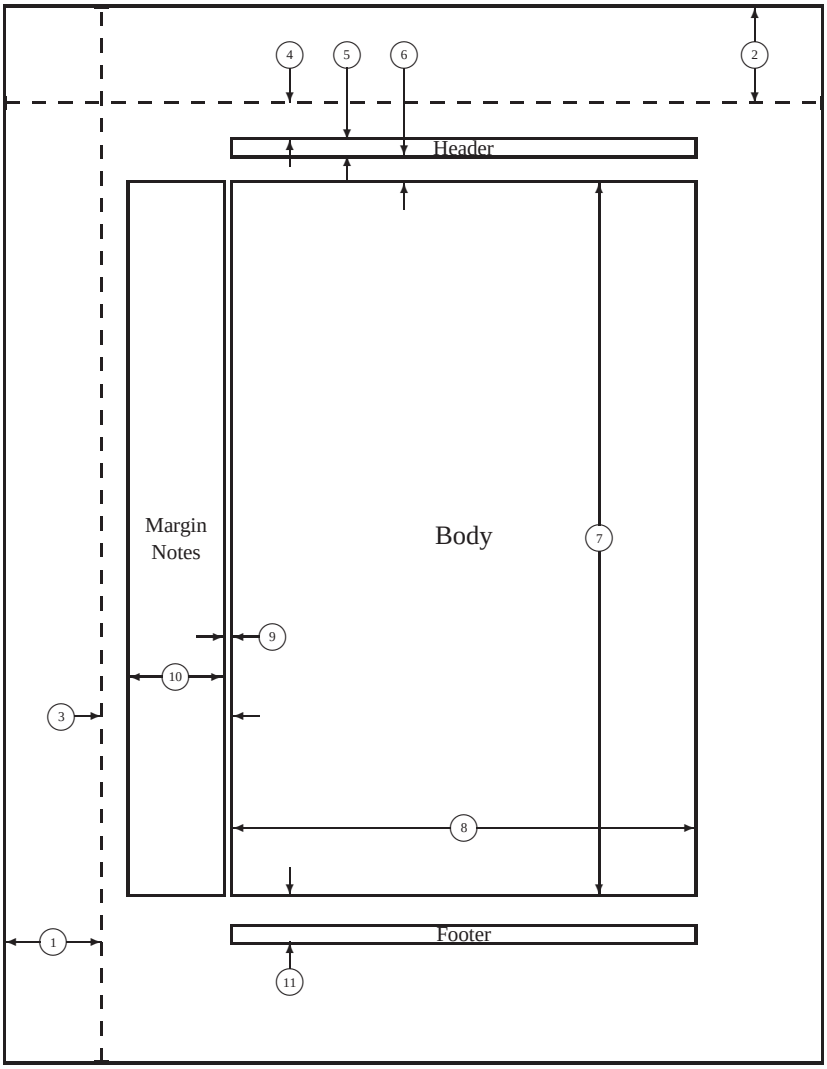
`\chaptermark`, `\sectionmark` and `\subsectionmark` which is then responsible for redefining `\rightmark` and `\markleft`.

So, if you wanted to change the look of the chapter name in the header line, you simply have to 'renew' the `\chaptermark` command.

Though this may all sound perfectly clear and simple, we advise you to read the manual that comes with the `fancyhdr` package. Things can be tricky.

17.7 Page layout

LaTeX allows you to specify the paper size in the `\documentclass` command. It then automatically picks the right text margins. But sometimes, you may not be happy with the predefined values. Naturally, you can change them. Figure 17.3 shows all the parameters which can be changed.




```
 1   one inch + \hoffset          2   one inch + \voffset
 3   \evensidemargin = 99pt       4   \topmargin = 28pt
 5   \headheight = 12pt           6   \headsep = 20pt
 7   \textheight = 536pt          8   \textwidth = 348pt
 9   \marginparsep = 7pt         10   \marginparwidth = 71pt
11   \footskip = 36pt                 \marginparpush = 5pt (not shown)
     \hoffset = 0pt                   \voffset = 0pt
     \paperwidth = 614pt              \paperheight = 794pt
```

Figure 17.3: Page layout parameters

LaTeX provides two commands to change these parameters, one of which we have already seen an application of. They are both usually used in the document preamble.

The first command assigns a fixed value to any of the parameters:

```
\setlength{parameter}{length}
```

The second command adds a length to any of the parameters.

```
\addtolength{parameter}{length}
```

This second command is actually more useful than the `\setlength` command, because you can now work relative to the existing settings. To add one centimeter to the overall text width, you can put the following commands into the document preamble:

```
\addtolength{\hoffset}{-0.5cm}
\addtolength{\textwidth}{1cm}
```

17.8 Bibliographies

Bibliographic references can be included in a document by using the

```
\cite{key}
```

command in which *key* represents a name much like in a `\label` or `\ref` command. You can specify multiple keys separated by commas (again, no spaces allowed here). The command

```
\nocite{key}
```

can be used to generate an entry in the bibliography without generating any output in the text. This can be useful if you want to format the citation yourself. Instead of a *key* you can also specify `*`. This is equivalent to citing all references there are in the bibliographies that you specify. It is a trick to generate a complete nicely formatted listing of your bibliographical database(s).

The bibliography (or rather: the list of cited author, articles, etc.) can be printed using the `thebibliography` environment.

```
\begin{thebibliography}{label}
```

The *label* parameter is actually not a label but a piece of text that determines the indentation of the list — it is never printed. The `thebibliography` environment closely resembles the `enumerate` environment (see section 17.5.1) but instead of `\item` commands you should use

```
\bibitem{key}
```

The generation of the bibliography can be completely automated if you use the BIBTEX program. In that case you will not use the `thebibliography` environment but you will use two other commands:

`\bibliography{`*name*`}`

`\bibliographystyle{`*style*`}`

See section 8.1 for details on BIBTEX, or read O. Patashnik's *BibTEXing* (cdrom).

17.9 Indexing

A very useful feature of many books is their index. With LATEX and the support program MakeIndex indexes can be generated quite easily. In this introduction, only the basic index generation commands will be explained. For a more in depth view please refer to *The LATEX Companion* by M. Goossens, F. Mittelbach and A. Samarin.

To enable the indexing feature of LATEX the `makeidx` package must be loaded in the preamble and initialized:

```
\usepackage{makeidx}
\makeindex
```

The content of the index is specified with

`\index{`*key*`}`

commands, where *key* is the index entry. You enter the index commands at the points in the text where you want the final index entries to point to. The table below explains the syntax of the *key* argument with several examples.

Example	Index entry	Comment
`\index{hello}`	hello, 1	Plain entry
`\index{hello!Peter}`	Peter, 3	Subentry under 'hello'
`\index{Sam@\textsl{Sam}}`	*Sam*, 2	Formatted entry
`\index{Lin@\textbf{Lin}}`	**Lin**, 7	Same as above
`\index{Jenny|textbf}`	Jenny, **3**	Formatted page number
`\index{Joe|textit}`	Joe, *5*	Same as above

When the input file is processed with LATEX, each `\index` command writes an appropriate index entry together with the current page number to a special file. The file has the same name as the LATEX input file, but a different extension (`.idx`). This `.idx` file can then be processed with the `makeindex` program.

▷ `makeindex` *filename*

The MakeIndex program generates a sorted index with the same base file name, but this time with the extension `.ind`. If now the LaTeX input file is processed again, this sorted index gets included into the document at the point where LaTeX finds

```
\printindex
```

The `showidx` package which comes with LaTeX 2_ε prints out all index entries in the left margin of the text. This is quite useful for proofreading a document and verifying the index.

17.10 Typesetting mathematics

LaTeX has a special mode for typesetting mathematics. Mathematical text within a paragraph is entered between `\(` and `\)`, between `$` and `$` or between `\begin{math}` and `\end{math}`.

```
Add $a$ squared and $b$ squared to get
$c$ squared. Or using a more mathematical
approach: $c^{2}=a^{2}+b^{2}$
\TeX{} is pronounced as
\begin{math}
\tau   \epsilon \chi
\end{math}. 100~m$^{3}$ of water is
not the same as $100 m^3$ of water.

This comes from my $\heartsuit$.
```

Add a squared and b squared to get c squared. Or using a more mathematical approach: $c^2 = a^2 + b^2$ TEX is pronounced as $\tau\epsilon\chi$. 100 m^3 of water is not the same as $100m^3$ of water.

This comes from my ♡.

It is preferable to display larger mathematical equations or formulae, that is to typeset them on separate lines. Therefore you enclose them between `\[` and `\]` or using the

```
\begin{displaymath}
```

environment. This produces formulae which are not numbered. If you want LaTeX to number them, you can use the equation environment.

```
\begin{equation}
```

```
Add $a$ squared and $b$ squared
to get $c$ squared. Or using
a more mathematical approach:
\begin{displaymath}
c^{2}=a^{2}+b^{2}
\end{displaymath}
And just one more line.
```

Add a squared and b squared to get c squared. Or using a more mathematical approach:

$$c^2 = a^2 + b^2$$

And just one more line.

With `\label` and `\ref` you can reference an equation within the text.

$$\epsilon > 0 \tag{1}$$

From (1) we gather ...

```
\begin{equation} \label{eq:eps}
\epsilon > 0
\end{equation}
From (\ref{eq:eps}) we gather
\ldots
```

Note that expressions will be typeset in a different style if displayed:

$\lim_{n\to\infty} \sum_{k=1}^{n} \frac{1}{k^2} = \frac{\pi^2}{6}$

$$\lim_{n\to\infty} \sum_{k=1}^{n} \frac{1}{k^2} = \frac{\pi^2}{6}$$

```
$\lim_{n \to \infty}
\sum_{k=1}^n \frac{1}{k^2}
= \frac{\pi^2}{6}$

\begin{displaymath}
\lim_{n \to \infty}
\sum_{k=1}^n \frac{1}{k^2}
= \frac{\pi^2}{6}
\end{displaymath}
```

There are differences between *math mode* and *text mode*. For example, in *math mode*:

1. Most spaces and linebreaks do not have any significance, as all spaces are either derived logically from the mathematical expressions or have to be specified using special commands such as `\{`, `\quad` or `\qquad`.
2. Empty lines are not allowed. Only one paragraph per formula.
3. Each letter is considered to be the name of a variable and will be typeset as such. If you want to typeset normal text within a formula (normal upright font and normal spacing) then you have to enter the text using the `\textrm{...}` commands.

$$\forall x \in \mathbf{R}: \qquad x^2 \geq 0 \tag{1}$$

$$x^2 \geq 0 \qquad \text{for all } x \in \mathbf{R} \tag{2}$$

```
\begin{equation}
\forall x \in \mathbf{R}:
\qquad x^{2} \geq 0
\end{equation}

\begin{equation}
x^{2} \geq 0\qquad
\textrm{for all }x\in\mathbf{R}
\end{equation}
```

Mathematicians can be very fussy about which symbols are used: it would be conventional here to use 'blackboard bold' which is obtained by `\mathbb` from the package `amsfonts` or `amssymb`.

17.10.1 Grouping in math mode

Most math mode commands act only on the next character. So if you want several characters affected by a command you have to group them together using curly braces: `{...}`.

$$a^x + y \neq a^{x+y} \qquad (1)$$

```
\begin{equation}
a^x+y \neq a^{x+y}
\end{equation}
```

17.10.2 Building blocks of a mathematical formula

In this section the most important commands used in mathematical typesetting will be described. For a list of all symbols available take a look at the tables in section 16.16.7.

Lowercase Greek letters are entered as `\alpha`, `\beta`, `\gamma`, ..., uppercase letters[3] are entered as `\Gamma`, `\Delta`, ...

$$\alpha, \beta, \delta, \gamma, \ldots \Delta, \Gamma, \Lambda, \Omega$$

```
$ \alpha,\beta,\delta,\gamma,\ldots
  \Delta,\Gamma,\Lambda,\Omega $
```

Exponents and subscripts can be specified using the `^` and the `_` characters.

$$a_1 \qquad x^2 \qquad e^{-\alpha t} \qquad a^3_{ij}$$
$$e^{x^2} \neq e^{x2}$$

```
$a_{1}$ \qquad $x^{2}$ \qquad
$e^{-\alpha t}$ \qquad
$a^{3}_{ij}$\\
$e^{x^2} \neq {e^x}^2$
```

The *square root* is entered as `\sqrt`, the n^{th} root is generated with `\sqrt[`*n*`]`. The size of the root sign is determined automatically. If just the sign is needed use `\surd`.

$$\sqrt{x} \qquad \sqrt{x^2+\sqrt{y}} \qquad \sqrt[3]{2}$$
$$\surd[x^2+y^2]$$

```
$\sqrt{x}$ \qquad
$\sqrt{ x^{2}+\sqrt{y} }$
\qquad $\sqrt[3]{2}$\\[3pt]
$\surd[x^2 + y^2]$
```

The commands `\overline` and `\underline` create *horizontal lines* directly over or under an expression.

$$\overline{m+n}$$

```
$\overline{m+n}$
```

The commands `\overbrace` and `\underbrace` create long *horizontal braces* over or under an expression.

$$\underbrace{a+b+\cdots+z}_{26}$$

```
$\underbrace{ a+b+\cdots+z }_{26}$
```

3 There is no uppercase Alpha defined in LaTeX because it looks the same as a normal roman character A.

To add mathematical accents such as small arrows or tilde signs to variables you can use the commands given in table 16.16 in section 16.16.7. Wide hats and tildes, covering several characters are generated with `\widetilde` and `\widehat`. The `'` symbol gives a prime.

$\widetilde{o\dot{p}}$ $\widehat{AB}y = x^2$ $y' = 2x$ $y'' = 2$

```
\begin{displaymath}
\widetilde{o\dot p} \qquad \widehat{AB}
y=x^{2}\qquad y'=2x\qquad y''=2
\end{displaymath}
```

Often *vectors* are specified by adding small arrow symbols on top of a variable. This is done with the `\vec` command. To denote the vector from A to B the two commands `\overrightarrow` and `\overleftarrow` are useful.

$\vec{a}$ $\overrightarrow{AB}$

```
\begin{displaymath}
\vec a\quad\overrightarrow{AB}
\end{displaymath}
```

Names of log-like functions are often typeset in an upright font and not italic as variables. Therefore the following commands are supplied to typeset the most important function names:

```
\arccos  \cos   \csc  \exp  \ker    \limsup  \min  \sinh
\arcsin  \cosh  \deg  \gcd  \lg     \ln       \Pr   \sup
\arctan  \cot   \det  \hom  \lim    \log      \sec  \tan
\arg     \coth  \dim  \inf  \liminf \max      \sin  \tanh
```

$$\lim_{x \rightarrow 0} \frac{\sin x}{x} = 1$$

```
\[\lim_{x \rightarrow 0}
\frac{\sin x}{x}=1\]
```

For the modulo function there are two commands: `\bmod` for the binary operator '$a \bmod b$' and `\pmod` for expressions such as '$x \equiv a \pmod{b}$.'

A built-up *fraction* is typeset with the command

```
\frac{...}{...}
```

Often the slashed form 1/2 is preferable, because it looks better for small amounts of 'fraction material.'

$1\frac{1}{2}$ hours

$\frac{x^2}{k+1}$ $x^{\frac{2}{k+1}}$ $x^{1/2}$

```
$1\frac{1}{2}$~hours
\begin{displaymath}
\frac{ x^{2} }{ k+1 }\qquad
x^{ \frac{2}{k+1} }\qquad
x^{ 1/2 }
\end{displaymath}
```

To typeset binomial coefficients or similar structures you can use either the command `{... \choose ...}` or `{... \atop ...}`. The second command produces the same output as the first one, but without braces.

$$\binom{n}{k} \qquad \begin{matrix} x \\ y+2 \end{matrix}$$

```
\begin{displaymath}
{n \choose k}\qquad {x \atop y+2}
\end{displaymath}
```

The *integral operator* is generated with `\int`, the **sum operator** with `\sum`. The upper and lower limits are specified with `^` and `_` as with subscripts and superscripts.

$$\sum_{i=1}^{n} \qquad \int_{0}^{\frac{\pi}{2}}$$

```
\begin{displaymath}
\sum_{i=1}^{n} \qquad
\int_{0}^{\frac{\pi}{2}} \qquad
\end{displaymath}
```

For *braces* and other delimiters there exist all types of symbols in TEX (e.g. [⟨ ‖ ↕). Round and square braces can be entered with the corresponding keys, curly braces with `\{`, all other delimiters are generated with special commands (e.g. `\updownarrow`). For a list of all delimiters available, check table 16.10.

$$a, b, c \neq \{a, b, c\}$$

```
\begin{displaymath}
{a,b,c}\neq\{a,b,c\}
\end{displaymath}
```

If you put the command `\left` in front of an opening delimiter or `\right` in front of a closing delimiter, TEX will automatically determine the correct size of the delimiter.

$$1 + \left(\frac{1}{1-x^2} \right)^3$$

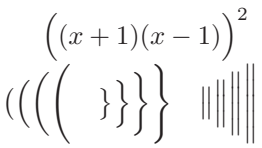
```
\begin{displaymath}
1 + \left( \frac{1}{ 1-x^{2} }
    \right) ^3
\end{displaymath}
```

Note that you must close every `\left` with a corresponding `\right` command. If you don't want anything on the right, use the invisible `\right.`! Just make sure you have exactly as many `\left` commands as `\right` commands, no matter what the delimiters are.

$$\left(\sqrt{\frac{\Omega}{1-\Omega}} + \Omega \right] \dots \lim_i^{\infty} x^i \big] \dots \left\{ \lambda_{\epsilon_\chi}^{\alpha^\tau}$$

```
$\left( \sqrt{\frac{\Omega}{1-\Omega}}
 + \Omega \right] \dots
 \left. \lim_i^{\infty} x^i \right] \dots
 \left\{
 \lambda^{\alpha^\tau}_{\epsilon_\chi}
 \right.$
```

In some cases it is necessary to specify the correct size of a mathematical delimiter by hand, therefore you can use the commands `\big`, `\Big`, `\bigg` and `\Bigg` as prefixes to most delimiter commands.[4]



```
$\Big( (x+1) (x-1) \Big) ^{2}$\\
$\big(\Big(\bigg(\Bigg($\quad
$\big\}\Big\}\bigg\}\Bigg\}$\quad
$\big\|\Big\|\bigg\|\Bigg\|$
```

To enter *three dots* into a formula you can use several commands. `\ldots` typesets the dots on the baseline, `\cdots` sets them centered. Beside that there are the commands `\vdots` for vertical and `\ddots` for diagonal dots.

$$x_1,\ldots,x_n \qquad x_1+\cdots+x_n$$

```
\begin{displaymath}
x_{1},\ldots,x_{n} \qquad
x_{1}+\cdots+x_{n}
\end{displaymath}
```

17.10.3 Math spacing

If the spaces within formulae chosen by TEX are not satisfactory, they can be adjusted by inserting special spacing commands. There are some commands for small spaces: `\,` `\:` and `\;`. The escaped space character `\␣` generates a medium sized space and `\quad` and `\qquad` produce large spaces. The `\!` command produces a negative space.

$$\iint_D g(x,y)\,\mathrm{d}x\,\mathrm{d}y$$

$$\int\!\!\int_D g(x,y)\mathrm{d}x\mathrm{d}y$$

```
\begin{displaymath}
\int\!\!\!\int_{D} g(x,y)
  \, \mathrm{d} x\, \mathrm{d} y
\end{displaymath}
\begin{displaymath}
\int\int_{D} g(x,y)
  \mathrm{d} x \mathrm{d} y
\end{displaymath}
```

Note that ‘d’ in the differential is set in roman type. Many mathematicians seem to prefer this style.

4 These commands do not work as expected if a size changing command has been used, or the `11pt` or `12pt` option has been specified. Use the `exscale` or `amsmath` packages to correct this behavior.

17.10.4 Vertically aligned material

To typeset *arrays*, use the `array` environment. It works somewhat similar to the `tabular` environment. The `\\` command is used to break the lines.

$$\mathbf{X} = \left(\begin{array}{ccc} x_{11} & x_{12} & \ldots \\ x_{21} & x_{22} & \ldots \\ \vdots & \vdots & \ddots \end{array} \right)$$

```
\begin{displaymath}
\mathbf{X} =
\left( \begin{array}{ccc}
x_{11} & x_{12} & \ldots \\
x_{21} & x_{22} & \ldots \\
\vdots & \vdots & \ddots
\end{array} \right)
\end{displaymath}
```

The `array` environment can also be used to typeset expressions which have one big delimiter by using a `.` as a invisible `\right` delimiter:

$$y = \left\{ \begin{array}{ll} a & \text{if } d > c \\ b + x & \text{in the morning} \\ l & \text{all day long} \end{array} \right.$$

```
\begin{displaymath}
y = \left\{ \begin{array}{ll}
 a & \textrm{if $d>c$}\\
 b+x & \textrm{in the morning}\\
 l & \textrm{all day long}
  \end{array} \right.
\end{displaymath}
```

For formulae running over several lines or for equation systems you can use the environments `eqnarray` and `eqnarray*` instead of `equation`. In `eqnarray` each line gets an equation number. In the `eqnarray*` no line numbers are produced.

The `eqnarray` and the `eqnarray*` environments work like a 3-column table of the form `{rcl}`, where the middle column can be used for the equal sign or the not-equal sign. Or any other sign you see fit. The `\\` command breaks the lines.

$$\begin{aligned} f(x) &= \cos x && (1) \\ f'(x) &= -\sin x && (2) \\ \int_0^x f(y)dy &= \sin x && (3) \end{aligned}$$

```
\begin{eqnarray}
f(x) & = & \cos x      \\
f'(x) & = & -\sin x    \\
\int_{0}^{x} f(y)dy &
 = & \sin x
\end{eqnarray}
```

Notice that there is too much space each side of the middle column, the equal signs. This can be reduced by setting `\setlength\arraycolsep{2pt}` as in the next example.

Long equations will not be automatically divided into neat bits. The author has to specify where to break them and how much to indent. The following two methods are the most common ones used to achieve this.

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \\ - \frac{x^7}{7!} + \cdots \qquad (1)$$

```
{\setlength\arraycolsep{2pt}
\begin{eqnarray}
\sin x & = & x -\frac{x^{3}}{3!}
     +\frac{x^{5}}{5!}-{}
                        \nonumber\\
 & & {}-\frac{x^{7}}{7!}+{}\cdots
\end{eqnarray}}
```

$$\cos x = 1 - \frac{x^2}{2!} + \\ + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots \qquad (1)$$

```
\begin{eqnarray}
\lefteqn{ \cos x = 1
     -\frac{x^{2}}{2!} +{} }
                        \nonumber\\
 & & {}+\frac{x^{4}}{4!}
     -\frac{x^{6}}{6!}+{}\cdots
\end{eqnarray}
```

The `\nonumber` command causes LaTeX not to generate a number for this equation.

It can be difficult to get vertically aligned equations to look right with these methods; the package `amsmath` provides a more powerful set of alternatives.

17.10.5 Math font size

In math mode TeX selects the font size according to context. Superscripts for example get typeset in a smaller font. If you want to add roman text to an equation and use the `\textrm` command, the font size switching mechanism will not work, as `\textrm` temporarily escapes to text mode. Use `\mathrm` instead to keep the size switching mechanism active. But pay attention, `\mathrm` will only work well on short items. Spaces are still not active and accented characters do not work.

$$2^{\mathrm{nd}} \quad 2^{\mathrm{nd}} \qquad (1)$$

```
\begin{equation}
2^\textrm{nd} \quad
2^\mathrm{nd}
\end{equation}
```

Pieces of plain text within math-mode can be typeset in a different font using the following commands:

ABCdef	`\mathrm{ABCdef}`	*ABCdef*	`\mathit{ABCdef}`
$ABCdef$	`\mathnormal{ABCdef}`	**ABCdef**	`\mathbf{ABCdef}`
ABCdef	`\mathsf{ABCdef}`	ABCdef	`\mathtt{ABCdef}`
$\mathcal{ABC}$	`\mathcal{ABC}` (only capitals)		

Note that `\mathcal` style is only available for capitals.

Although (LA)TEX usually automatically chooses the right font sizes you sometimes need to specify them yourself. In math mode the font size is set with the four commands:

`\displaystyle` (123), `\textstyle` (123), `\scriptstyle` (123) and `\scriptscriptstyle` (123).

Changing styles also affects the way limits are displayed. The following commands can be used to change the font size within formulas. By default formulas are printed in `\textstyle` and displayed formulas in `\displaystyle`. These commands are declarations and are effective only within the part of the formula in which they occur.

`\displaystyle`	$\displaystyle\sum_{i=1}^{n} x^i$
`\textstyle`	$\textstyle\sum_{i=1}^{n} x^i$
`\scriptstyle`	$\scriptstyle\sum_{i=1}^{n} x^i$
`\scriptscriptstyle`	$\scriptscriptstyle\sum_{i=1}^{n} x^i$

17.11 Theorems, laws, etc.

When writing mathematical documents, you probably need a way to typeset 'Lemmas', 'Definitions', 'Axioms' and similar structures. LaTeX supports this with the command

`\newtheorem{`*name*`}[`*counter*`]{`*text*`}[`*section*`]`

The *name* argument is a short keyword used to identify the 'theorem'. With the *text* argument you define the actual name of the 'theorem' which will be printed in the final document.

The arguments in square brackets are optional. They are both used to specify the numbering used on the 'theorem'. With the *counter* argument you can specify the *name* of a previously declared 'theorem'. The new 'theorem' will then be numbered in the same sequence. The *section* argument allows you to specify the sectional unit within which you want your 'theorem' to be numbered.

After executing the \newtheorem command in the preamble of your document, you can use the following command within the document.

\begin{*name*}[*text*]
This is my interesting theorem
\end{*name*}

Here are a few examples that will hopefully make it clear how the \newtheorem environment works in practice.

Law 1 *Don't hide in the witness box*

Jury 2 (The Twelve) *It could be you! So beware and see law 1*

Law 3 *No, No, No*

```
% definitions for the document
% preamble
\newtheorem{law}{Law}
\newtheorem{jury}[law]{Jury}
%in the document
\begin{law} \label{law:box}
Don't hide in the witness box
\end{law}
\begin{jury}[The Twelve]
It could be you! So beware and
see law \ref{law:box}\end{jury}
\begin{law}No, No, No\end{law}
```

The 'Jury' theorem uses the same counter as the 'Law' theorem. Therefore, it gets a number which is in sequence with the other 'Laws'. The argument in square brackets is used to specify a title or something similar for the theorem.

Murphy 0.1 *If there are two or more ways to do something, and one of those ways can result in a catastrophe, then someone will do it.*

```
\newtheorem{mur}{Murphy}[section]
\begin{mur}
If there are two or more
ways to do something, and
one of those ways can result
in a catastrophe, then
someone will do it.\end{mur}
```

The 'Murphy' theorem gets a number which is linked to the number of the current chapter. You could also use another unit, like section or subsection.

17.12 Cross references

In books, reports and articles there are often cross references to figures, tables and special segments of text. LaTeX provides the following commands for cross referencing

\label{*marker*}

`\ref{`*marker*`}`

`\pageref{`*marker*`}`

where *marker* is an identifier chosen by the user. LaTeX replaces `\ref` by the number of the section, subsection, figure, table, equation or theorem where the corresponding `\label` command was issued. `\pageref` prints the page number of the corresponding `\label` command. Here also the numbers from the previous run are used.

```
Everything will be explained in
section~\ref{sec:answers} on
page~\pageref{sec:answers}.

\section{Answers}
\label{sec:answers}
```

Everything will be explained in section 1 on page 1.

1 Answers

17.13 Footnotes

With the command

`\footnote{`*footnote text*`}`

a footnote will be printed at the foot of the current page.

```
Footnotes\footnote{This
  is a footnote} are often used
by people using \LaTeX.
```

17.14 Floating bodies

Today most publications contain a lot of figures and tables. These elements need special treatment because they cannot be broken across pages. One method would be to start a new page every time a figure or a table is too large to fit on the present page. This approach would leave pages partially empty which looks very bad.

The solution to this problem is to 'float' any figure or table which does not fit on the current page to a later page while filling the current page with body text. LaTeX offers two environments for floating bodies: one for tables and one for figures. To take full advantage of these two environments it is important to understand approximately how LaTeX handles floats internally. Otherwise floats may become a major source of frustration because LaTeX never puts them where you want them to be.

Let us first have a look at the commands LaTeX supplies for floats: any material enclosed in a `figure` or `table` environment will be treated as floating matter.

`\begin{figure}[`*placement specifier*`]`

`\begin{table}[`*placement specifier*`]`

Both float environments support an optional *placement specifier*. This parameter is used to tell LaTeX about the locations the float is allowed to be moved to. A *placement specifier* is constructed by building a string of *float placing permissions*.

Placement specification	Permission to place the float ...
h	*here* at the very place in the text where it occurred. This is useful mainly for small floats.
t	at the *top* of a page
b	at the *bottom* of a page
p	on a special *page* containing only floats.
!	without considering most of the internal parameters (such as the maximum number of floats allowed on one page) which could stop this float from being placed.

A table could be started with the following line e.g.

```
\begin{table}[!hbp]
```

The placement specifier `[!hbp]` allows LaTeX to place the table right here (`h`) or at the bottom (`b`) of some page or on a special floats page (`p`) and all that even if it does not look that good (`!`). If no placement specifier is given, the standard classes assume `[tbp]`.

LaTeX will place every float it encounters, according to the placement specifier supplied by the author. If a float cannot be placed on the current page it is deferred to either the *figures* or the *tables* queue. These are fifo ('first in first out') queues. When a new page is started, LaTeX first checks if it is possible to fill a special 'float' page with floats from the queues. If this is not possible, the first float on each queue is treated as if they had just occurred in the text: LaTeX tries again to place them according to their respective placement specifiers (except 'h' which is no longer possible). Any new floats occurring in the text get placed into the appropriate queues. LaTeX strictly maintains the original order of appearance for each type of float. That is why a figure which cannot be placed, pushes all the further figures to the end of the document. Therefore:

If LaTeX is not placing the floats as you expected, it is often only one float jamming one of the two float queues.

Having explained the difficult bit, there are some more things to mention about the `table` and `figure` environments. With the

```
\caption{caption text}
```

command you can define a caption for the float. A running number and the string 'Figure' or 'Table' will be added by LaTeX.

The two commands

```
\listoffigures
```

and

```
\listoftables
```

operate analogously to the `\tableofcontents` command, printing a list of figures or tables respectively. In these lists, the whole caption will be repeated. If you tend to use long captions, you must have a shorter version of the caption going into the lists. This is accomplished by entering the short version in brackets after the `\caption` command.

```
\caption[Short]{LLLLLooooooonnnnggggg}
```

With `\label` and `\ref` you can create a reference to a float within your text.

A `\label` command should *always* be placed *after* the `\caption` command. Placing it *before* the `\caption` command is not illegal but it will reference the current *section* instead of the the current *figure* or *table* — an error that may take days to find.

The following example draws a square and inserts it into the document. You could use this if you wanted to reserve space for images you are going to paste into the finished document.

```
Figure~\ref{white} is an example of Pop-Art.
\begin{figure}[!hbp]
\makebox[\textwidth]{\framebox[5cm]{\rule{0pt}{5cm}}}
\caption{Five by five in centimeters} \label{white}
\end{figure}
```

In the example above (assuming the figure queue is empty) LaTeX will try *really hard* (`!`) to place the figure right *here* (`h`). If this is not possible, it tries to place the figure at the *bottom* (`b`) of the page. Failing to place the figure on the current page, it determines if it is possible to create a float page containing this figure and maybe some tables from the tables queue. If there is not enough material for a special float page, LaTeX starts a new page and once more treats the figure as if it had just occurred in the text.

Under certain circumstances it might be necessary to use one of the following commands:

```
\clearpage
```

```
\cleardoublepage
```

They order LaTeX to immediately place all floats remaining in the queues and then start a new page. `\cleardoublepage` goes to a new odd numbered page on top of that.

17.15 Defining your own commands and environments

Undoubtedly you have noticed that in any TeX document there are lots of TeX commands which you can easily recognize because they start with a \. TeX commands usually have a name that consists only of letters. Commands are terminated by a space, a number or any 'non-letter'. Commands can also consist of a backslash and exactly one special character, as we have seen in the previous section. Commands are case-sensitive, so `\something` is not the same as `\Something`. LaTeX (or rather, TeX) ignores whitespace after commands. If you want to get a space after a command, you have to put either `{}` or `\␣` after the command name. The `{}` stops LaTeX from eating up all the space after the command name.

Some commands need a parameter which has to be given between curly braces `{}` after the command name. Some commands support optional parameters which are added after the command name in square brackets `[ ]`. We have already seen examples of this.

Commands or macros (we use both terms to refer to the same object) serve several purposes:

- To define the structure of a document, e.g. using `\documentclass`, `\chapter` and `\section`.
- To enter symbols that are not available on the keyboard, e.g. the `\copyright` for ©.
- As abbreviations of complex constructions, e.g. `\LaTeX` for LaTeX.
- To make it easy to specify complex structures and on top of that ensure that they are typeset with absolute consistency.
- To change the layout of (part of) a document, e.g. using `\Large` or `\textbf` to select a different typeface.

New commands can be defined with

```
\newcommand{commandname}[number of parameters]{definition}
```

An simple example will make this clear:

Who needs a *What You See Is What You Get* interface?

```
\newcommand{\wysiwig}
  {\textit{What You See Is What You Get}}
Who needs a \wysiwig\ interface?
```

There is really almost no limit to how complex a command can be, but remember that a command can never have more than 9 parameters. Here is a rather silly example using 4 parameters:

Look: $\begin{array}{|ccc} \mathbf{11} & \ldots & \underline{23} \\ \vdots & \frac{11}{89} & \vdots \\ 41 & \ldots & \mathbf{89} \end{array}$ or $\begin{array}{|ccc} \mathbf{300} & \ldots & \underline{2223} \\ \vdots & \frac{300}{809} & \vdots \\ 54 & \ldots & \mathbf{809} \end{array}$

```
\newcommand{\mymatrix}[4]
{\begin{array}{|ccc}
 \mathbf{#1} & \ldots & \underline{#2} \\
 \vdots      & \frac{#1}{#4} & \vdots\\
 #3          & \ldots & \mathbf{#4}
 \end{array}}
Look: $\mymatrix{11}{23}{41}{89}$ or
      $\mymatrix{300}{2223}{54}{809}$
```

Note that if you attempt to define a command that already exists LaTeX will not allow it. However, if you are sure that your redefinition will not be disastrous you can use the command

`\renewcommand{`*commandname*`}[`*number of parameters*`]{`*definition*`}`

LaTeX environments are a special case of commands. They change the way LaTeX works locally, or they provide special functions, but again only locally. At the end of the environment the original state is restored. We have already seen several examples of predefined LaTeX environments. You can make your own using the command:

`\newenvironment{`*envname*`}{`*opening commands*`}{`*ending commands*`}`

Here is an example:

! ! Don't read this ! !

Why don't you listen to me?

```
\newenvironment{teaser}
  {\vspace{2mm}
   \hrule
   \vspace{2mm}
   \centerline{{\Large
     \textbf{! ! Don't read this ! !}}}
   \vspace{5mm}
   \sffamily}
  {\vspace{2mm}
   \hrule}
\begin{teaser}
Why don't you listen to me?
\end{teaser}
```

Analogous to `\renewcommand` you should use

`\renewenvironment{`*envname*`}{`*opening commands*`}{`*ending commands*`}`

in case you want to redefine an existing LaTeX environment.

17.16 International language support

If you need to write documents in languages other than English, LaTeX must apply different hyphenation rules in order to produce correct output. See section 10.3 for details on hyphenation rules.

17.16.1 The babel package

For many languages, these changes can be accomplished by using the `babel` package.

If your system is already appropriately configured, you can activate the Babel package by adding the command

```
\usepackage[language1,language2,...]{babel}
```

to the preamble of your document. Note that if your document contains text in multiple languages you should list all of them. The first language you specify will be the default at the start of your document. You can switch to another language using the command

`\selectlanguage{`*language*`}`

Switching to a different language involves a lot more than merely using different hyphenation rules. Babel also redefines many LaTeX keywords such as 'Table of Contents', 'Chapter' and 'Table'.

For some languages Babel also specifies new commands, which simplify the input of special characters. The German language for example, contains a lot of umlauts (äöü). Using Babel you can enter an ö by typing `"o` instead of ö.

17.16.2 The inputenc package

Some computer systems allow you to input special characters directly from the keyboard. LaTeX can handle such characters if you use `inputenc` package. When using this package you should consider that other people might not be able to display your input files on their computer, because they use a different encoding. For example, the German umlaut ä on a standard PC running standard MS-DOS is encoded as 132 and on some Unix systems using ISO-Latin 1 it is encoded as 228. Therefore, use this feature with care.

The LaTeX `inputenc` package support many different encodings. For Windows users the following encoding are most important:

ascii This is the standard ASCII encoding that defines the characters 32–127.

latin1 This encoding is based on ISO 8859-1.

latin2 This encoding is based on ISO 8859-2.

latin3 This encoding can be used for general purpose applications in typical office environments in Afrikaans, Catalan, English, Esperanto, French, Galician, German, Italian, Maltese and Turkish. It is based on ISO 8859-3 encoding.

latin5 This encoding is based on ISO 8859-5 and is used for Turkish.

cp1252 This is the standard Windows 3.1 ANSI encoding (Western Europe), which is based on ISO-Latin 1, but has important additions in the 128–159 range. It was designed for Danish, Dutch, English, Finnish, French, German, Icelandic, Italian, Norwegian, Portuguese, Spanish and Swedish. A synonym for `cp1252` is `ansinew`.

cp1250 This is the Windows encoding for Central and Eastern Europe.

You can load a specific encoding as follows:

```
\usepackage[codepage]{inputenc}
```

where *codepage* is one of the items listed above.

Hôtel, naïve, élève, Française,
smørrebrød, Señorita, wunderschön

```
Hôtel, naïve, élève, Française,\\
smørrebrød, Señorita, wunderschön
```

You can also switch to another encoding in the middle of a document using the command

`\inputencoding{`*codepage*`}`

See the documentation on the `inputenc` package for a complete list of available encodings.

Instead of using the `inputenc` package, you could use one of TeX's input translation tables, which are explained in section 13.3.2.

You should either use TeX's translation feature *or* the `inputenc` package in LaTeX files, but never both.

17.17 Hyphenation

LaTeX hyphenates words whenever necessary. If the hyphenation algorithm does not find the correct hyphenation points you can remedy the situation by using the following commands, to tell TeX about the exception. The command

`\hyphenation{`*word list*`}`

causes the words listed in the argument to be hyphenated only at the points marked by -. This command should be given in the preamble of the input file and should only contain words built from normal letters. The case of the letters is ignored. The example below will allow 'hyphenation' to be hyphenated as well as 'Hyphenation' and it prevents 'FORTRAN', 'Fortran' and 'fortran' from being hyphenated at all. No special characters or symbols are allowed in the argument. Example:

```
\hyphenation{FORTRAN Hy-phen-a-tion}
```

The command \- inserts a discretionary hyphen into a word. If TEX encounters a word that contains discretionary hyphens it will only consider hyphenating at those positions, even if other positions were possible. This command is especially useful for words containing special characters (e.g. accented characters), because LATEX does not automatically hyphenate words containing special characters.

```
I think it is su\-per\-cal\-%
i\-frag\-i\-lis\-tic\-ex\-pi\-%
al\-i\-do\-cious to hyphenate
the Dutch word wijs\-heid
```

Several words can be kept together on one line with the command

\mbox{*text*}

It causes its argument be kept together under all circumstances: no linebreak, no hyphenation.

```
My phone number will change soon.
It will be \mbox{0116 291 2319}.

The parameter
\mbox{\emph{file name}} should
contain the name of the file.
```

You can switch hyphenation off completely like this:

```
\pretolerance=10000
\hyphenpenalty=10000
```

TEX provides many hooks to the hyphenation algorithm but it is beyond the scope of this book to discuss them. They are explained in detail in D. Knuth's *The TEXbook*.

17.18 Illustrations

For making relatively simple illustrations consisting of lines, arrows, circles and text LaTeX provides the `picture` environment.

`\begin{picture}(`*x-size,y-size*`)`

Usually you will put a picture inside a `figure`, `center` or `quote` environment but it can be included in the middle of a sentence. If you keep it small like this  it may work out well.

The size of the picture and the size of any element in it is not given in exact measures (e.g. 5mm) but in relative measures. The exact size is determined by the value of

`\unitlength`

By assigning an appropriate value to `\unitlength` you can scale a complete picture, e.g. like this:

```
\setlength{\unitlength}{1pt}
```

By default `\unitlength` is 1 point but packages that you load (or you yourself!) may change the value, so it is good practice to specify the required value with each picture.

The `picture` environment provides a system in which you can position elements relative to the origin of the picture. The origin, designated as point $(0, 0)$, is the lower left corner of the picture (see the small picture above).

The most common command inside a `picture` environment is

`\put(`*x-coordinate,y-coordinate*`){...}`

These are the most basic elements that can be used inside a `picture` environment:

`\circle{`*size*`}`

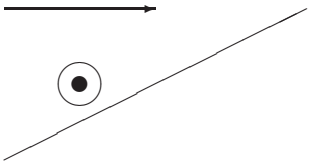
prints a circle of diameter (*size* × `\unitlength`). The maximum is 40 pt. Use the 'starred' version for filled circles (e.g. `\circle*{4}`). The maximum diameter is 15 pt.

`\line(`*x-direction,y-direction*`){`*size*`}`

`\vector(`*x direction,y direction*`){`*size*`}`

prints a line or vector (arrow). The x and y-direction together determine the slope of a line or vector. $(0, 1)$ would be a vertical line; $(1, 0)$ would be a horizontal line; $(-1, 0)$ would also be a horizontal line, going left instead of right; $(1, 1)$ would be line with 45° slope. Other slopes are possible but very restricted. Direction values must be integer values between -6 and 6 and the values of x and y-direction must be undividable (except for trivial cases such as $(1, 1)$ and $(2, 1)$). So, $(2, 5)$ and $(-3, -1)$ are legal,

but (1, 7) and (−2, −4) are not. Vectors are even more restricted: their range of x and y-directions is limited to −4 and 4. Here is a small example:



```
\setlength{\unitlength}{1mm}
\begin{picture}(40,20)
\put(10,10){\circle{6}}
\put(10,10){\circle*{2}}
\put(0,0){\line(2,1){40}}
\put(0,20){\vector(1,0){20}}
\end{picture}
```

Other useful commands available in the `picture` environment are:

`\makebox(`*x*`,`*y*`)[`*position*`]{`*text*`}`

prints *text* in an invisible rectangle of size (x, y). The *position* parameter determines the alignment within the rectangle: `l` (left aligned), `r` (right aligned), `t` (top aligned), `b` (bottom aligned), `rb` (right bottom aligned), et cetera.

`\framebox(`*x*`,`*y*`)[`*position*`]{`*text*`}`

works just like `\makebox` but prints the rectangle as a frame. A simplified version of this command is

`\fbox{`*anything*`}`

which simply 'frames' its argument. It can also be used outside of the `picture` environment. The thickness of the frame can be set with `\fboxrule`. Another variant is

`\frame{`*anything*`}`

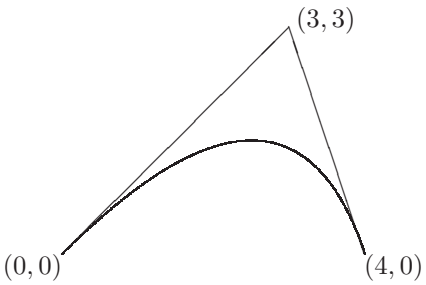
which is equivalent to `\fbox` except that it adds no space around the object.

`\dashbox{`*dashlength*`}(`*x*`,`*y*`)[`*position*`]{`*text*`}`

works just like `\framebox` but now the frame is dashed. The *dashlength* parameter determines the size of the dashes.

`\qbezier[`*n*`]`$(x_1, y_1)(x_2, y_2)(x_3, y_3)$

draws a Bezier curve from (x_1, y_1) to (x_3, y_3) using (x_2, y_2) as guiding point. The guiding point itself is not on the curve but represents the intersection of the tangents of the end points. The figure below should make this construction more clear.




```
\setlength\unitlength{1cm}
\begin{picture}(6,3.5)(-1,-0.5)
  \put(0,0){\makebox(0,0)[tr]{$(0,0)$}}
  \put(4,0){\makebox(0,0)[tl]{$(4,0)$}}
  \put(3.1,3){$(3,3)$}
  \put(0,0){\line(1,1){3}}
  \put(4,0){\line(-1,3){1}}
  \qbezier(0,0)(3,3)(4,0)
\end{picture}
```

If the optional parameter [*n*] is specified it determines the number of points that will be drawn. Otherwise, LaTeX will decide by itself, with a maximum of `\qbeziermax` (you can use `\renewcommand` to change this value).

The thickness on lines can be set using the command

```
\linethickness{size}
```

that takes a *real* size as parameter, e.g. `\linethickness{0.8mm}`. Predefined thicknesses are

```
\thinlines
```

which is the default, and

```
\thicklines
```

Note that there is no support for shading or filling shapes with patterns or colors. However, clever TeXies will find that it is not impossible, but it is beyond the scope of this introduction to show you such tricks.

Although the `picture` has its limitations it can be very useful.[5] A big advantage over other solutions is that it is guaranteed to work on *any* LaTeX system.[6] Another advantage is that all text inside a `picture` environment is in the same typeface as the rest of the document. Note also that any legal TeX command (e.g. formulae!) can be included.

Some graphics programs such as GNUplot (see section 8.16.6) and LaTeXcad (see section 8.16.5) can output a `\picture` environment. They can do things you would never write by hand, e.g. drawing a complex curve by `\put`ing by dividing it into lots of small line fragments. Figure 17.4 is an example of the output of a `picture` environment generated by GNUplot.

5 All diagrams in appendix B are actually LaTeX pictures.

6 At the same time this can be a disadvantage because *only* LaTeX can make any sense of it. As far as we know there are no tools for converting LaTeX pictures to more common formats such as JPG, PCX, GIF or EPS — other than a complete TeX system.

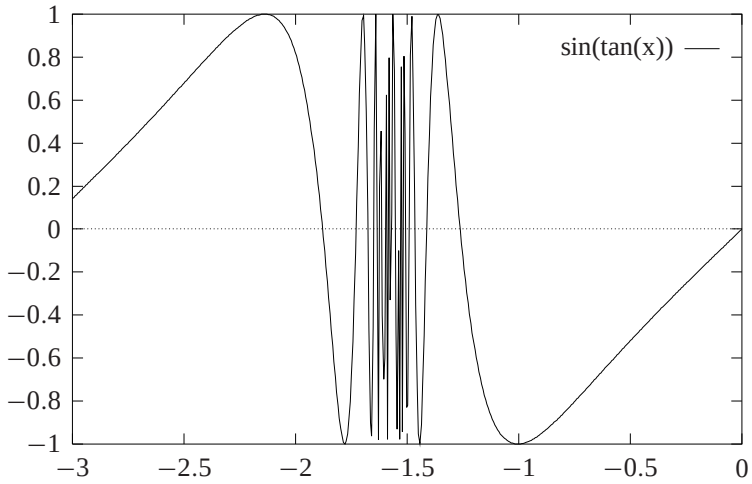


Figure 17.4: A complex curve in a `picture` environment

There are a few extensions to the `picture` environment available. A well known and supported extension is called *epic*. It adds several higher-level commands for picture construction, so you can easily draw solid, dotted and dashed lines with arbitrary slopes, ellipses and a lot more. See S. Podar's documentation on *epic* (cdrom) for more details. The extension called *eepic* extends *epic* even further. C. Kwok's documentation on *eepic* is also available (cdrom).

17.18.1 Packages for inserting graphic illustrations

Though we haven't discussed *packages* yet — we will in section 17.20 — this is a good time to introduce a few LaTeX packages that you may need if you want to include non-TeX graphics.

In a nutshell, a *package* is a LaTeX file that adds new commands and features that you can use in your document. The command `\usepackage{`*package*`}` is used in the preamble to load a package.

The following packages are specifically geared for graphics applications:

epic This package makes it easy to specify all kinds of dashed or dotted lines in a `picture` environment ('epic' is an abbreviation of 'extended picture'). It also supports commands for drawing shapes by simply specifying a list of points. The connecting lines will be drawn automatically. If the slope of a line does not obey the rules described on page 17.18 the line will be approximated by using small line pieces, which may look less attractive.

eepic This is an extension of the `epic` package that eliminates the limitations just described. Therefore it is usually better to load both packages: `\usepackage{epic,eepic}`.

epsfig This package enables the inclusion of EPS pictures in any TeX document.

xypic This package allows you to make diagrams in a relatively easy way, though it may take some time to master its complex syntax.

graphics / graphicx These packages support a completely different way of including graphics. 'graphics' is the basic package, 'graphicx' is an extension of it. We will discuss these package in detail below.

The graphics and graphicx package

With the `figure` and the `table` environment LaTeX provides the basic facilities to work with floating bodies such as images or graphics.

There are also several possibilities to generate the actual graphics with basic LaTeX or a LaTeX extension package. Unfortunately, most users find them quite difficult to understand. Therefore this will not be explained any further in this manual. For more information on that subject please refer to *The LaTeX Companion* by M. Goossens, F. Mittelbach and A. Samarin, and *The LaTeX Graphics Companion* by M. Goossens, S. Rahtz and F. Mittelbach.

A much easier way to get graphics into a document, is to generate them with a specialized software package and then include the finished graphics into the document. Here again, LaTeX packages offer many ways to do that. In this introduction, only the use of Encapsulated PostScript (EPS) graphics will be discussed, because it is quite easy to do and widely used. In order to use pictures in the EPS format, you must have a PostScript printer.[7]

A good set of commands for inclusion of graphics is provided in the `graphicx` package. It is part of a whole family of packages called the 'graphics' bundle.

Assuming you are working on a system with a PostScript printer available for output and with the `graphicx` package installed, you can use the following step by step guide to include a picture into your document:

1. Export the picture from your graphics program in EPS format.
2. Load the `graphicx` package in the preamble of the input file with
   `\usepackage[`*driver*`]{graphicx}`
   where *driver* is the name of your DVI to PostScript program. The most widely used program is DVIPS. The name of the driver is required because there is no standard on how graphics are included in TeX. Knowing the name of the *driver*, the `graphicx` package can choose the correct method to insert information about the graphics into the DVI file so that the printer understands it and can correctly include the `.eps` file.

7 Another possibility to output PostScript is the Ghostscript program. See section 13.6.9.

3. Use the command

   `\includegraphics[`*key*`=`*value*`,...]{`*file*`}`

   to include *file* into your document. The optional parameter accepts a comma-separated list of *keys* and associated *values*.

The `\includegraphics` command accepts the following options:

height The height of the graphic (in any of the accepted TEX units).

totalheight The total height of the graphic.

width The width of the graphic.

scale The scale factor for the graphic. Specifying `scale=2` makes the graphic twice as large as its natural size.

angle Specifies the angle of rotation in degrees, counter-clockwise.

origin Specifies what point to use for the rotation origin. By default the object is rotated about its reference point. E.g., `\rotatebox` command in `origin=c` rotates the graphic about its center

The following options are especially useful for EPS graphics:

viewport Specifies what portion of the graphic to view. Like a BoundingBox, the area is specified by four numbers which are the coordinates of the lower-left corner and upper-right corner. The coordinates are relative to lower-left corner of the BoundingBox. For example, if the graphic's BoundingBox parameters are 50 50 410 302, `viewport=50 50 122 122` displays the 1-inch square from the lower left of the graphic, and `viewport=338 230 410 302` displays the 1-inch square from the upper right of the graphic. The `clip` option must be used to prevent the portion of the graphic outside the viewport from being displayed.

trim An alternate method for specifying what portion of the graphic to view. The four numbers specify the amount to remove from the left, bottom, right, and top side, respectively. Positive numbers trim from a side, negative numbers add to a side. For example, `trim=1 2 3 4` trims the graphic by 1 bp on the left, 2 bp on the bottom, 3 bp on the right, 4 bp on the top. The `clip` option must be used to prevent the trimmed portion from being displayed.

noclip (default) The entire graphic appears, even if portions appear outside the viewing area.

clip Any graphics outside of the viewing area are clipped and will not appear.

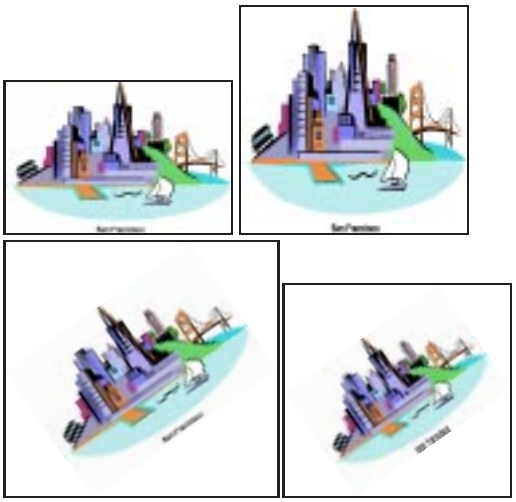
draft The graphic's BoundingBox and file name are displayed in place of the graphic, making it faster to display and print the document Specifying

`usepackage[draft]{graphicx}` will cause all graphics in a document to be inserted in draft mode.

final (default) Graphics will be displayed. It can be used to override a global `draft` option.

keepaspectratio If not specified, specifying both `width` and either `height` or `totalheight` causes the graphic to be scaled anamorphically to fit both the specified height and width; if specified, specifying both `width` and either `height` or `totalheight` makes the graphic as large as possible such that its aspect ratio remains the same and the graphic exceeds neither the specified height nor width.

The following example code will hopefully make things clear:



```
\frame{%
  \includegraphics
    [width=3cm]
    {artwork.eps}}
\frame{%
  \includegraphics
    [width=3cm,height=3cm]
    {artwork.eps}}

\vspace{2pt}

\frame{%
  \includegraphics
    [width=3cm,angle=35]
    {artwork.eps}}
\frame{%
  \includegraphics
    [angle=35,width=3cm]
    {artwork.eps}}
```

The first box show the picture at a width of 3 centimeters; the height is not specified. The second box shows the effect of specifying both width and height. The third and fourth box show how rotation works, and how the order in which the options are given effects the output.

In the example we specified the graphics file's extension (`.eps` in this case). Usually this is not necessary. The *driver* that you specified with the `\usepackage` statement defines which file name extensions are valid. It will try to use the most appropriate file that it can find. E.g., if the *driver* is `dvips`, valid extensions are `.eps`, `.ps`, `.pcx`, `.bmp`, `.msp`, `.pict` and `.pntg`. If the *driver* is `pdftex`, then valid extensions are `.png`, `.pdf`, `.jpg` and `.mps`.

17.18.2 PostScript

PostScript is a language designed for describing all visual aspects of a printed page. It is widely used to drive printers and is a very popular format among drawing programs. A PostScript picture can contain not only graphic elements such as lines and shapes, it can also contain text.

PostScript is much more flexible and powerful than (LA)TEX when it comes to drawing pictures. In PostScript you can, amongst many other things:

- draw lines in arbitrary directions and thickness;
- draw curved lines;
- rotate text;
- shade or color shapes;
- include *bitmaps* (exact representations of graphic screens).

On the other hand PostScript has less 'knowledge' of typography and text layout than (LA)TEX. In PostScript you cannot make text wrap automatically over multiple lines, or pages.

So, for text oriented projects (LA)TEX is much better suited than PostScript, but for graphic oriented projects PostScript may be a better choice. And fortunately they can cooperate so you can get the best of both worlds.

In principle a PostScript file is an ordinary ASCII file, so you can browse or edit it with any editing program or text processor. But this is really only for those who understand to some extend the PostScript language. Usually you will only let other programs *generate* PostScript code for you. In that case there is no need for you to understand or read any PostScript file.

17.18.3 Including PostScript files in LATEX documents

PostScript pictures can easily be included in LATEX documents. Note however that the PostScript file is not actually *included* in the DVI file. It is the *DVI driver* that finds a *reference* to the PostScript file in the DVI file. It will attempt to send the PostScript file to the selected output device, probably a printer or your computer screen.

This means that the DVI file is not completely *device independent* any more. However, this is usually no problem because PostScript is a worldwide standard that is supported on almost any TEX system. Besides, now that an excellent PostScript interpreter called Ghostscript is commonly available you even don't need an (expensive) PostScript printer to get output on *any* kind of printer.

As a rule PostScript pictures are written as so-called *Encapsulated PostScript*, which represents a subset of the full blown PostScript language.

When LATEX tries to include an EPS file it only tries to find the exact measures of the pictures. These measures are stored in the EPS file and usually look like this:

```
%%BoundingBox: 75 435 523 743
```

The *BoundingBox* specifies a rectangle that completely includes all elements of the picture. The first two numbers represent the coordinates of the lower left corner; the other two represent the coordinates of the top right corner. The numbers are specified as *points*. One point is about $\frac{1}{3}$mm.

Any well written EPS file should specify a BoundingBox, either near the top or near the end of the file.

See section 17.18.1 for a detailed explanation of ways to include (EPS) graphics in your document.

17.19 Color support

A fairly easy way to specify colors in your LaTeX document is by using the `color` package. This package provides for a set of commands to typeset text in any color, provided that the DVI driver you use supports colors.

The first thing you will have to do is load the package in the preamble of your document, specifying your DVI driver at the same time:

```
\usepackage[dvips]{color}
```

A number of colors are already defined and ready for use:

```
GreenYellow, Yellow, Goldenrod, Dandelion, Apricot, Peach, Melon,
YellowOrange Orange, BurntOrange, Bittersweet, RedOrange, Mahogany,
Maroon, BrickRed, Red OrangeRed, RubineRed, WildStrawberry, Salmon,
CarnationPink, Magenta VioletRed, Rhodamine, Mulberry, RedViolet,
Fuchsia, Lavender, Thistle, Orchid DarkOrchid, Purple, Plum, Violet,
RoyalPurple, BlueViolet, Periwinkle CadetBlue, CornflowerBlue,
Cornfl, MidnightBlue, NavyBlue, RoyalBlue, Blue Cerulean, Cyan,
ProcessBlue, SkyBlue, Turquoise, TealBlue, Aquamarine BlueGreen,
Emerald, JungleGreen, SeaGreen, Green, ForestGreen, PineGreen
LimeGreen, YellowGreen, SpringGreen, OliveGreen, RawSienna, Sepia,
Brown, Tan Gray, Black, White.
```

You can define you own colors with the command

`\definecolor{`*colorname*`}{`*colormodel*`}{`*color specification*`}`

The *name* could be e.g. 'LovelyGreen'. The *model* defines which color model you use. It should one of the following:

rgb (red, green, blue) The color specification should be three values between 0 and 1, separated by commas.

cmyk (cyan, magenta, yellow, black) The color specification should be four values between 0 and 1, comma separated.

gray The color specification should be one values between 0 and 1.

named The color must be one of the predefined names listed earlier.

If you use DVIPS as your DVI driver, the color models ‘rgb’ and ‘named’ will work nicely. Below is an example in which all commands are demonstrated. Note that on black and white output devices the colors will appear as gray shades.

Once a color is defined you can activate it with the command

`\color[`*colormodel*`]{`*colorname*`}`

All text after this command will appear in the color you specified. If you want to restrict this color to just a part of your document you can use braces like this:

```
{\color{PrettyPink} Text in pink} text
in ‘normal’ color.
{\color{Yellow} Text {\color{PineGreen} in} yellow}
{\color{Fuchsia} except for ‘{\color{PineGreen}in}’
which was in {\color{PineGreen}pine-green}.}
```

Another way to color just a few words is

`\textcolor{`*colorname*`}{`*text*`}`

in which case only *text* will be colored, e.g. like this:

```
This is your \textcolor{RedOrange}{last} warning.
```

`\pagecolor[`*colormodel*`]{`*colorname*`}`

can be used to select the background color of the whole page and all subsequent pages.

If you only want to change the background of a part of the page you can use the command

`\colorbox[`*colormodel*`]{`*colorname*`}{`*text*`}`

which will generate a colored box with *text* in it. Even fancier effects can be obtained with the command

`\fcolorbox{`*framecolor*`}{`*backgroundcolor*`}{`*text*`}`

which allows you to specify both the background color of the box, and the color of the frame around it. This command is very similar to the `\fbox` command (see page 383).

```
\definecolor{Trial}{rgb}{0.3, 0.6, 0.6}
\definecolor{Background}{gray}{0.8}
\definecolor{PrettyPink}{named}{CarnationPink}
\definecolor{verywhite}{named}{White}

\bf Normal {\color{Trial} Trial}
\textcolor{verywhite}{very pretty}
\colorbox{PrettyPink}{Try this}

\fboxrule=2mm
\fcolorbox{verywhite}{Trial}{Try}
\fboxrule=4mm
\fcolorbox{PrettyPink}{Trial}{this}
\pagecolor{Background}
```

Remember that TeX was not designed with color in mind, and producing colors requires a lot of help from the DVI driver. Thus, depending on the driver, some or all features of the color package may not always work properly. See D. Carlisle's *Packages in the 'graphics' bundle* (cdrom) for more details.

17.20 Packages

While writing your document, you will probably find that there are some areas where basic LaTeX cannot solve your problem. In that case loading one or more LaTeX packages may solve your problem. Packages are used for three different purposes:

- To change the layout of a document. E.g., the package `A4` sets certain page size parameters.
- To extend the capabilities of existing LaTeX commands. E.g., the `array` and `theorem` packages enhance the `array` and `theorem` environments respectively.
- To add functionality, usually through new commands. E.g., the `graphics` package adds support for including pictures.

You can activate a package with the `\usepackage` command.

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

package is the name of the package and *options* is a (list of) keyword(s) that trigger(s) special features in the package. Some packages come with the LaTeX base distribution, others are provided separately.

The standard packages distributed with LaTeX are:

alltt This package provides the `alltt` environment, which is like the `verbatim` environment except that `\`, `{`, and `}` have their usual meanings. It is described in `alltt.dtx` and in *LaTeX — A Document Preparation System* by L. Lamport.

doc This is the basic package for typesetting the documentation of LaTeX programs. It is described in `doc.dtx` and in *The LaTeX Companion* by M. Goossens, F. Mittelbach and A. Samarin.

exscale This provides scaled versions of the math extension font. It is described in `exscale.dtx` and *The LaTeX Companion* by M. Goossens, F. Mittelbach and A. Samarin.

fontenc This is used to specify which font encoding LaTeX should use. It is described in `ltoutenc.dtx`. See also section 10.1.

ifthen Provides commands of the form 'if... then do... otherwise do...'. It is described in `ifthen.dtx` and *The LaTeX Companion* by M. Goossens, F. Mittelbach and A. Samarin.

inputenc Allows the specification of an input encoding such as ASCII, ISO Latin-1, ISO Latin-2, 437/850 IBM code pages, Apple Macintosh, Next, ANSI-Windows or user-defined one. It is described in `inputenc.dtx`. See also section 10.1.

latexsym LaTeX 2_ε doesn't load the LaTeX symbol font by default. To access it, you should use the `latexsym` package. It is described in `latexsym.dtx` and in *The LaTeX Companion* by M. Goossens, F. Mittelbach and A. Samarin.

makeidx This provides commands for producing indices. It is described in *LaTeX — A Document Preparation System* by L. Lamport and in *The LaTeX Companion* by M. Goossens, F. Mittelbach and A. Samarin.

newlfont This is used to emulate the font commands of LaTeX 2.09 with the New Font Selection Scheme. It is described in *The LaTeX Companion* by M. Goossens, F. Mittelbach and A. Samarin. Since LaTeX 2.09 is obsolete you may never need this.

oldlfont This is used to emulate the font commands of LaTeX 2.09. It is described in *The LaTeX Companion* by M. Goossens, F. Mittelbach and A. Samarin. Since LaTeX 2.09 is obsolete you may never need this.

showidx This causes the argument of each `\index` command to be printed on the page where it occurs. It is described in *LaTeX — A Document Preparation System* by L. Lamport.

syntonly This is used to process a document without typesetting it. It is described in `syntonly.dtx` and in *The LaTeX Companion* by M. Goossens, F. Mittelbach and A. Samarin.

tracefnt This allows you to control how much information about LaTeX's font loading is displayed. It is described in *The LaTeX Companion* by M. Goossens, F. Mittelbach and A. Samarin.

But that is not all. You need at least the **graphics** and **tools** collections in order to have all the files described in *LaTeX — A Document Preparation System* by L. Lamport. The **amsmath** package (part of **amslatex**) and **babel** are also mentioned in the list of 'standard packages' in section C.5.2 of that book.

amslatex Advanced mathematical typesetting from the American Mathematical Society. This includes the `amsmath` package; it provides many commands for typesetting mathematical formulas of higher complexity. It is produced and supported by the American Mathematical Society and it is described in *The LaTeX Companion* by M. Goossens, F. Mittelbach and A. Samarin.

babel This package and related files support typesetting in many languages. It is described in *The LaTeX Companion* by M. Goossens, F. Mittelbach and A. Samarin.

graphics This includes the `graphics` package which provides support for the inclusion and transformation of graphics, including files produced by other software. Also included is the `color` package which provides support for typesetting in color. See also section 17.18.1.

mfnfss Everything you need (except the fonts themselves) for typesetting with a large range of bitmap (METAFONT) fonts.

psnfss Everything you need (except the fonts themselves) for typesetting with a large range of Type 1 (PostScript) fonts.

tools Miscellaneous packages written by the LaTeX3 project team.

These packages come with documentation and each of them is also described in at least one of the books *The LaTeX Companion* by M. Goossens, F. Mittelbach and A. Samarin and *LaTeX — A Document Preparation System* by L. Lamport.

17.20.1 Tools

This collection of packages includes, at least, the following (some files may have slightly different names on certain systems):

array Extended versions of the environments `array`, `tabular` and `tabular*`, with many extra features.

dcolumn Alignment on 'decimal points' in tabular entries. Requires the `array` package.

delarray Adds 'large delimiters' around arrays. Requires `array`.

hhline Finer control over horizontal rules in tables. Requires `array`.

longtable Multi-page tables. (Does not require `array`, but it uses the extended features if both are loaded.)

tabularx Defines a `tabularx` environment that is similar to `tabular*` but it modifies the column widths, rather than the inter-column space, to achieve the desired table width.

afterpage Place text after the current page.

bm Access bold math symbols.

enumerate Extended version of the `enumerate` environment.

fontsmpl Package and test file for producing 'font samples'.

ftnright Place all footnotes in the right-hand column in two-column mode.

indentfirst Indent the first paragraph of sections, etc.

layout Show the page layout defined by the current document class.

multicol Typeset text in columns, with the length of the columns 'balanced'.

rawfonts Preload fonts using the old internal font names of LaTeX 2.09.

somedefs Selective handling of package options. (Used by the rawfonts package.)

showkeys Prints the 'keys' used by `\label`, `\ref`, `\cite`, etc.; useful while drafting.

theorem Flexible declaration of 'theorem-like' environments.

varioref 'Smart' handling of page references.

verbatim Flexible extension of the verbatim environment.

xr Cross reference other 'external' documents.

xspace 'Smart space' command that helps you to avoid the common mistake of missing spaces after command names.

17.20.2 Other useful packages

Name	Description
algorithms	Environments for typesetting algorithms
amsmath	AMSLaTeX: extensions by the American Mathematical Society (mostly for typesetting mathematics)
amssymb	Extra symbols defined by the AMS
bibunits	Bibliographies per section
calc	Support for floating point calculations
caption	Adaptation of the figure and table captions

... continued on next page

Name	Description
caption2	Improved version of **caption**
changebar	Support for 'changebars': vertical lines indicating text that was changed
chapterbib	Bibliographies per chapter
color	Support for colored text
colortab	Color support in tables
deleq	Flexible numbering of equations
eepic	Extension of the `picture` environment
epsfig	Include PostScript pictures
eqnarray	Generalization of the `eqnarray` environment
euler	'Euler' fonts in math mode
fancybox	Fancy boxes, e.g. boxed paragraphs
fancyhdr	Easy set-up of page headers and footers
float	Adaptation of the `figure` and `table` environment, make new 'floating' environments
floatflt	Make text float around pictures (like `picins`)
formlett	For producing series of letters
geometry	Define non-standard page formats
harvard	Harvard bibliography style family
indentfirst	Also indent the first paragraph of a section
index	Define more than one index in a document
lastpage	Refer to the total number of pages in a document
longtable	Make tables run over multiple pages
lscape	Switch between portrait and landscape orientation
ltxtable	Combination of **longtable** and **tabularx**
minitoc	Table of contents per chapter
moreverb	Extension of `verbatim` environment
overcite	Cite using superscripts
picins	Make text float around pictures
pstricks	Use PostScript constructions for fancy tricks
rotating	Rotation of anything
seminar	Make overhead sheets
showlabels	Print label commands in the margin
subeqnarray	Equation array with subequation numbering
subfigure	Divide a figure in subfigures
supertab	Make tables run over multiple pages
wrapfig	Pictures within paragraphs, text wrapping around it

17.21 Auxiliary files

During a LaTeX compilation a number of auxiliary files may appear in your working directory. You can remove these files if you want because you can always regenerate

them by rerunning LaTeX, BibTeX and/or MakeIndex. Nevertheless, we recommend that you don't delete them because you may need to run LaTeX and related programs several times before your final output is again complete.

File type	Written by	Meaning
`.aux`	LaTeX	cross-references, counter info, etc.
`.lof`	LaTeX	list of figures
`.lot`	LaTeX	list of tables
`.toc`	LaTeX	table of contents
`.idx`	LaTeX	raw document index
`.ind`	MakeIndex	sorted document index
`.bbl`	BibTeX	document bibliography

Other files specific to LaTeX are listed below. Usually they will read from another directory than your document. Except for `.dtx` and `.ins` files you should not delete them.

File type	Meaning
`.cls`	class file
`.cfg`	configuration file
`.def`	input encoding definition, graphics driver, ...
`.dtx`	package documentation
`.ins`	installation package
`.fd`	font definition
`.ldf`	Babel language definition
`.sty`	style file or package

17.22 The LaTeX3 Project

The LaTeX3 Project is a long-term research project into developing the next version of the LaTeX typesetting system. Some of the reports of the project's work are available by anonymous FTP from CTAN (see section 5.7).

Until LaTeX3 is delivered, the project team are committed to the active maintenance of The New Standard LaTeX (LaTeX 2_ε). The experience gained from the production and maintenance of The New Standard LaTeX will have a major influence on the design of LaTeX3.

The LaTeX3 Project is a volunteer project funded by donations. If you make use of LaTeX and are interested in its welfare then please consider making a donation. This can either be financial, via the TeX Users Group, or by helping with one of the volunteer groups.

The LaTeX3 Project Team personnel are: J. Braams, D. Carlisle, A. Jeffrey, F. Mittelbach, C. Rowley and R. Schöpf. If you would like to contact the LaTeX3 Project Team personally (for example for speaking at conferences, volunteering effort, offering con-

sultancy work, etc.) then please follow the instructions for submitting a bug report. See also appendix D.

Chapter 18

CONTEXT: Hagen's approach

This chapter describes the capabilities of CONTEXT and the available commands and their functionality. It is based on *ConTEXt, an excursion* by H. Hagen and T. Otten (cdrom). The pictures and drawings in this chapter were made by J. Jonker.

CONTEXT was developed for practical applications: the typesetting and production of documents ranging from simple straightforward books up to very complex and advanced technical manuals and study books, either in a paper or an electronic version. This introduction describes the CONTEXT basics. This should suffice to make manuals, articles or books. But CONTEXT has a lot more in store. With CONTEXT you can make interactive electronic documents with hyperlinks, multiple indexes, to name but a few of the more advanced options. Users who want to explore these fascinating areas should be aware that much more extensive documentation is available (cdrom).

Another interesting feature of CONTEXT is its multi-lingual interface. It enables users to work with CONTEXT in their own language. Currently CONTEXT supports English, German and Dutch commands. In this chapter we will describe the English version. The official CONTEXT documentation is therefore also available in Dutch (cdrom), German (cdrom) and English. CONTEXT is published in the public domain with the help of the Dutch language oriented TEX Users Group (NTG). All CONTEXT products and information can be obtained from Pragma's WWW server `http://www.pragma-ade.nl` and from NTG's WWW server (`http://www.ntg.nl/context/`).

On the CDROM you can find several other documents on CONTEXT, e.g. complete overviews of all available commands and keywords in the English, Dutch and German versions.

18.1 Command syntax

CONTEXT was developed for non-technical users in the WYSIWYG era. Therefore a user-friendly interface and easy commands are primary goals. Cryptic commands, programming and logical reasoning is avoided as much as possible. Nevertheless, getting to know and understand CONTEXT may take some time. And you still have to learn some commands and how they are used.

The syntax of CONTEXT commands differs significantly from dialects such as Plain TEX or LATEX. Therefore it is essential that we explain the general syntax before we delve any deeper into CONTEXT.

You will notice that in CONTEXT commands you will often specify parameters to commands in square brackets [ ]. In case a command requires a parameter that will produce typeset text, it should be specified in curly braces { }.

In formal command descriptions we will write *required* parameters like this `[]`, or slanted *`[]`* if they are optional. Parameters in curly braces are always *required* parameters, whereas parameters in brackets are sometimes *options* that are not required.

If you want to specify *options* in CONTEXT commands, you should remember that there are three kinds of options:

- Keywords: these are predefined words that have a special meaning to CONTEXT. Beware that if you supply an unknown keyword (perhaps a misspelled keyword), CONTEXT will simply ignore without warning. An example is the keyword `packed` that can be used in itemizations.
- Assignments: these always look like this: *name*=*value*. In this case *name* is another keyword, and *value* could also be a keyword but not necessarily. Here are two examples: `align=left` (both *name* and *value* are keywords), `width=2.3cm` (the *name* is a keyword, the *value* is not).
- Implicit options: in many cases CONTEXT is intelligent enough to understand the meaning of an option without explicit specifications. We have already seen an example of this: the header commands accept an option in brackets that defines a label that you can refer to.

Note that in many cases multiple keywords and/or multiple assignments can be specified. Lists of keywords or assignments are always separated by commas. The order in which such list items are given is not significant. Here is an example:

```
\startitemize[a,packed][width=10mm,textstyle=bold]
```

Here are a few imaginary commands to make this clear:

```
\acommand[option1,option2]
\anothercommand[required]
\yetanothercommand[required][option1=x,option2=y]
\nextcommand[option]{required}
```

The command `\acommand` can be used without parameters, whereas the command `anothercommand` requires a parameter in brackets. The command `\yetanothercommand` requires a parameter in brackets and it *optionally* accepts another parameter, also in brackets. The command `\nextcommand` requires a parameter in curly braces, optionally preceded by a parameter in brackets.

Having read all this you may wonder if this is supposed to be a user-friendly interface, as we promised. The answer is both yes and no. If your demands are not very high, you will need very few commands and options. The complexity of the commands that ConTeXt supports will not be needed in your documents. However, if your demands are high, you will need to understand how ConTeXt deals with commands, parameters, options and more. But this is simply because complex documents just can't be described in one or two lines. We are still a long way from the ultimate command `\MakeaBeautifulDocument{myfile}` and be done with it. Nevertheless, you will be pleased to find that, even if your demands are high, almost any option you can think of is available in ConTeXt, or it can easily be added. The internal consistency of ConTeXt will help you a lot in setting up complex documents.

18.2 Document structure

Because ConTeXt is fully compatible with Plain TeX, the most trivial ConTeXt document could look like this:

Hello world!

```
Hello world! \bye
```

This will work, but a typical ConTeXt document will look somewhat different. As a rule, a document always starts with the command

```
\starttext
```

and it ends with

```
\endtext
```

The area before `\starttext` is called the set-up area. It is used for defining new commands and for setting up the layout of your document. You may, for instance, want to specify the font size:

```
\setupbodyfont[dimension]
```

So, a very simple document could look like this:

```
\setupbodyfont[12pt]
\starttext
This is a one line document.
\stoptext
```

Within the \starttext ... \stoptext a document can be divided in four parts:

- front matter
- body matter
- back matter
- appendices

The parts are defined with:

\startfrontmatter ... \stopfrontmatter

This part is mostly used for the table of contents, the list of figures and tables, the preface, the acknowledgements, etc. In the front matter as well as back matter the command \chapter produces a non-numbered header in the table of contents.

\startbodymatter ... \stopbodymatter

In the body matter \chapter produces a numbered header.

\startbackmatter ... \stopbackmatter

This part has much in common with the front matter.

\startappendices ... \stopappendices

In this part you could include appendices. Note that the layout of each part can be specified individually. In section 18.2.2 we will describe commands to set up the layout of your document.

18.2.1 Headings

The structure of a document is also determined by its headings. Headings are created with the following commands:

\chapter*[options]*{*title*}

\section*[options]*{*title*}

\subsection*[options]*{*title*}

\title*[options]*{*title*}

\subject[*options*]{*title*}

\subsubject[*options*]{*title*}

These commands will produce a number and a heading in a predefined font size and font type with an amount of vertical spacing before and after the heading. The heading commands have two appearances. For instance:

```
\chapter[hasselt-by-night]{Hasselt by night}
```

and

```
\chapter{Hasselt by night}
```

The optional parameter is used for internal references. If you want to refer to this header, you would type: `\on{page}[hasselt-by-night]`. See section 18.9 for more information on cross referencing.

A table of contents in which all headers will appear can be produced with the command

\completecontent

Note that a table of contents can only be known to TEX *after* a complete and successful run. In the *next* run the table of contents can be typeset.

Now that you know these basics you can understand all of the following example:

```
\starttext
\completecontent
\chapter{Introduction}
...
\chapter{Chapter One}
\section[firstsection]{The first section}
...
\section{The second section}
\subsection{the first subsection}
...
\subsection{the second subsection}
...
\section{The third section}
...
\chapter{Another Chapter}
...
\chapter[lastchapter]{The Last Chapter}
...
\stoptext
```

Just like in Plain TEX, there is no need to keep everything in one file. Using the command

`\input` *filename*

you can split your document into separate files. You could, e.g., keep parts or chapters in separate files. Note that if you only specify a file name (i.e., without extension), the extension `.tex` will be assumed. In case you want to specify a full path, you should use forward slashes /, not backslashes \. Here is an example:

```
\input /tex/mystyles/thesis/thesis.ctx
```

18.2.2 Set-up commands

Set-up commands are mostly placed in the set-up area of your document, the area before `\starttext`. These commands have a global effect, but you also use them mid-document. The set-up commands all have the same structure. Here is an example:

`\setuphead[`*identifier*`][`*options*`]`

The options are described below:

Option	Value
`identifier`	*name*
`0, 1, 2, ...`	*number* `each`
`style`	`normal bold slanted boldslanted type cap small ...` *command*
`width`	*dimension*
`height`	*dimension*
`align`	`left right middle` `width`
`tolerance`	`verygrigged rigged` `tolerant` `verytolerant`
`distance`	*dimension*
`before`	*command*
`after`	*command*
`inner`	*command*
`command`	*command*
`line`	`on` `off`

A set-up command consists of a more or less logical name and a number of bracket pairs.

`\setupacommand[`*name*`][options][assignments]`

The default options and assignments are underlined. Furthermore you will notice that some values are typeset in italics: *section*, *name*, *dimension*, *number*, *command* and *text*. This indicates that you can set the value yourself.

section a header name such as `chapter`, or `section` or `paragraph`.

name an identifier (logical name).

dimension a size with a unit in `cm`, `mm`, `in`, `pt`, `sp`, `em` or `ex`.

number an integer.

command a command.

text text.

In this manual we will mention several set-up commands and we will demonstrate their usage, but a complete description of their options is not given here. You can find these in H. Hagen's *ConTEXt Quick Reference Guide* (cdrom): the Adobe Acrobat PDF file `set-i-en.pdf` is an interactive document that lists all options of commands. A German version (`set-i-de.pdf`) and a Dutch version (`set-i-nl.pdf`) are also available.

Of course heading commands can be configured to your own preferences and you can even define your own headings. This is done with the command `\setuphead` which we have already discussed, and the command

`\definehead[`*identifier*`] [`*level*`]`

Here is an example of their usage:

```
\definehead[myheader]
  [section]
\setuphead
  [myheader]
  [numberstyle=bold,
   textstyle=cap,
   before=\hairline\blank,
   after=\nowhitespace\hairline]
\myheader[myhead]{Hasselt makes headlines}
```

1 HASSELT MAKES HEADLINES

There is one other command you should know about if you want to change the style of headings:

`\setupheads[`*options*`]`

You can use this command to set up the numbering of the numbered headings. If you type:

```
\setupheads
  [alternative=inmargin,
   separator=--]
```

all numbers will appear in the margin. Section 1.1 would look like 1–1. Commands like `\setupheads` should appear in the set-up area of your document.

18.3 Defining the style of a document

CONTEXT encourages you to mark up your document in a very logical way. You can specify the layout of pages, paragraphs, page header, page footers, itemizations, figures, etc. at a global level. This method ensures consistency throughout your document. As a bonus, there will be very little need for tweaking the layout in the middle of the document.

18.3.1 Page layout

The page layout can be defined with:

```
\setuplayout[options]
```

This command should appear in the set-up area of your document. Below some of the options and their meaning are listed:

Option	Meaning
`width`	*dimension*
`height`	*dimension*
`leftmargin`	*dimension*
`rightmargin`	*dimension*
`header`	*dimension*
`footer`	*dimension*
`style`	`bold slanted cap small ...`
`grid`	`yes no`

You have to familiarize yourself with the parameters that describe the page layout. A page is divided in a number of areas such as the text, margin, header and footer. The size of each area can be specified. The different areas in the page layout are shown in figure 18.1.

If you want CONTEXT to give you visual markers that show your page layout, you can use the command

```
\showframe
```

and process one page or the whole file. The areas are shown in a number of frames. The command

```
\showlayout
```

shows the values of the parameters.

The values of the layout parameters are available as commands (see table below). This enables you to work more accurately when defining measures of columns, figures and tables. A few of these parameters are:

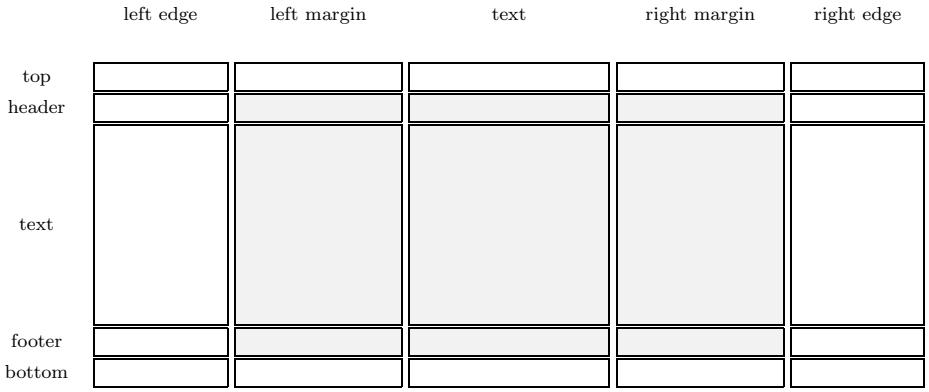


Figure 18.1: The page areas with \setupbackgrounds

Command	Meaning
\makeupwidth	width of the typing area
\makeupheight	height of the typing area
\textwidth	width of the text area
\textheight	height of the text area

If you want to define the width of a column or the height of a figure, you can do that relative to \makeupwidth or \makeupheight. Changes in page width or page height will alter columns and figures proportionally. The other distances and measures are shown below:

```
\topdistance      \leftmarginwidth   \makeupheight
\topheight        \rightmarginwidth  \makeupwidth
\headerdistance   \edgedistance      \textheight
\headerheight     \edgewidth         \textwidth
\topspace         \leftedgewidth     \footerdistance
\headerlevel      \rightedgewidth    \footerheight
\backspace        \paperheight       \bottomdistance
\margindistance   \paperwidth        \bottomheight
\marginwidth
```

In case you want to make only slight changes to the global page layout, you should use the command

\adaptlayout*[pages]*[*options*]

For reasons of consistency in layout, this command should be used with great caution. Here is an example:

```
\adaptlayout[21,38][height=+.5cm]
```

In this case pages 21 and 38 would have a height of their default height + 0.5 cm. For local changes in the page layout you can use:

```
\startlocal ... \stoplocal
```

For example:

```
\startlocal
\setuplayout[height=+.5cm]
Hasselt has a completely different layout than most other cities
because of its fortifications and moats.
\stoplocal
```

18.3.2 Page breaking and page numbering

A new page can be forced or blocked by:

```
\page[options]
```

Below some of the options and their meaning are listed:

Option	Meaning
yes	enforce a page break
makeup	enforce a page break without filling
no	do not break page here
preference	preferably a new page here
bigpreference	strong preference for a new page here
left	next page is a left hand side (even) page
right	next page is a right hand side (odd) page
disable	following commands have no effect
reset	following commands do have effect
empty	insert an empty page
last	add pages until an even number is reached
quadruple	add pages until a multiple of 4 is reached

Page numbering is automatic, but you can force a specific page number with the \page command:

```
\page[25]
```

Sometimes it is better to state a relative page number like [+2] or [-2].

The position of the page numbers on a page depend on your preferences. It also depends on whether the document is one sided or double sided. Page numbering can be set up with:

```
\setuppagenumbering[options]
```

A few of the possible options are:

Option	Value
alternative	singlesided doublesided
location	header footer middle
style	normal bold slanted ...
left	*text*
right	*text*
text	*text*
conversion	numbers characters romannumerals

18.3.3 Page headers and page footers

You can specify page headers and footers with the commands:

```
\setupfootertexts[option] [left text] [right text]
```

and

```
\setupheadertexts[option] [left text] [right text]
```

The first parameter is used to specify the location of the page footer or page header (e.g. `text`, `edge` or `margin`). The content of the page footer or page header can be specified in the second and third parameters.

In a double sided document you can specify a fourth parameter. In that case the third and fourth parameter are used for footer or header on the even pages and odd pages. In many cases you can omit parameters, as in the next example:

```
\setupheadertexts[Manual][chapter]
```

In this case the text *Manual* will appear on the left of the page header. The parameter `chapter` is interpreted as a keyword: it will yield the title of the current chapter on the right of the page header. This header will change dynamically at every new chapter. You can set up the page header and page footer with:[1]

```
\setupheader[options]
```

and

```
\setupfooter[options]
```

The most important option here is `style`, which can be given the value `normal`, `bold`, `slanted`, etc.

If you want to leave out the page header and footer on one page (e.g. the title page) you can type:

```
\noheaderandfooterlines
```

1 Do not confuse `\setuphead` and `\setupheader`.

18.3.4 Columns

Text can be formatted in columns. If you precede a piece of text by the command `\startcolumns` and append the command `\stopcolumns` to it, that piece of text will be formatted in columns.

`\startcolumns`*[options]* ... `\stopcolumns`

Here is an example:

```
\switchtobodyfont[7pt]
\startcolumns[n=2,tolerance=verytolerant]
Hasselt is an old Hanzeatic City, situated
12~km north of Zwolle at the river
Zwartewater.

The city has a long history since obtaining
the city charter around 1252. Parts and
parcel of this history can be traced back
to a large number of monuments to be admired
in the city center. There you will find the
St. Stephanus church, a late gothic church
dating back to 1479 with a magnificent organ.

The former Municipal Building is situated on
The Market Place. Constituted between 1500
and 1550 it houses a large collection of
weapons, amongst which a large collection of
black powder guns (haakbussen) in the world
should be mentioned.
\stopcolumns
```

Hasselt is an old Hanzeatic City, situated 12 km north of Zwolle at the river Zwartewater.

The city has a long history since obtaining the city charter around 1252. Parts and parcel of this history can be traced back to a large number of monuments to be admired in the city center. There you will find the St. Stephanus church, a late gothic church dating back to 1479 with a magnificent organ.

The former Municipal Building is situated on The Market Place. Constituted between 1500 and 1550 it houses a large collection of weapons, amongst which a large collection of black powder guns (haakbussen) in the world should be mentioned.

If necessary you can force switching to a new column with `\column`. You can set up columns with:

`\setupcolumns`[*options*]

The most important option you can specify is the number of columns. Specifying, e.g., `n=4` will set up a 4 column layout.

18.3.5 Typesetting paragraphs

In TEX and CONTEXT the most important unit of text is a paragraph. You can start a new paragraph by:

- an empty line
- the TEX command `\par`

We advise you to use empty lines as paragraph separators. This will lead to a clearly structured and well organized file and will prevent mistakes.

Only in situations where a command has to be 'closed' explicitly you should use `\par`.

During one of the wars Hasselt lay under siege. After some time the city was famine stricken, everything edible was eaten. Except for one cow. The cow was kept alive and treated very well.

Once a day the citizens of Hasselt took the cow for a walk on the ramparts. The besiegers saw the well fed cow and became very discouraged. They broke up their camps and Hasselt was saved.

```
During one of the wars Hasselt lay under
siege. After some time the city was famine
stricken, everything edible was eaten.
Except for one cow. The cow was kept alive
and treated very well. \par
Once a day the citizens of Hasselt took
the cow for a walk on the ramparts. The
besiegers saw the well fed cow and became
very discouraged. They broke up their camps
and Hasselt was saved.
```

This could also be typed with an empty line instead of `\par`:

During one of the wars Hasselt lay under siege. After some time the city was famine stricken, everything edible was eaten. Except for one cow. The cow was kept alive and treated very well.

Once a day the citizens of Hasselt took the cow for a walk on the ramparts. The besiegers saw the well fed cow and became very discouraged. They broke up their camps and Hasselt was saved.

```
During one of the wars Hasselt lay under
siege. After some time the city was famine
stricken, everything edible was eaten.
Except for one cow. The cow was kept alive
and treated very well.

Once a day the citizens of Hasselt took
the cow for a walk on the ramparts. The
besiegers saw the well fed cow and became
very discouraged. They broke up their camps
and Hasselt was saved.
```

If you want the paragraph to start with an indentation you can type:

```
\indenting[options]
```

in the set-up area of your document. You can specify what kind of indentation you want. By default indenting is set to `never`.

If you choose to use indentations you may sometimes have to switch it off explicitly. This can be done with the command

```
\noindenting
```

You can set up the value of the indentation with:

```
\setupindenting[...]
```

CONTEXT automatically suppresses indentation at certain points, e.g. after a chapter heading.

Sometimes you want to apply a special format to paragraphs. You can define a special format with the command

\defineparagraphs[*identifier*] [*options*]

After defining the format, you can specify the details of this format:

\setupparagraphs[*identifier*] *[columns]* [*options*]

A few of the possible options are listed below:

Option	Value
style	normal bold slanted...
width	*dimension*
height	*dimension*
before	*command*
after	*command*
tolerance	verystrict strict tolerant verytolerant
align	left right middle width

After defining a paragraph with \defineparagraphs you can format the paragraph with \setupparagraphs. Next you can start your paragraph with \start... and end it with \stop... A new paragraph starts with the name of your paragraph, in this case \mypar:

```
\defineparagraphs[mypar]
  [n=3,before={\blank},after={\blank}]
\setupparagraphs[mypar]
  [1][width=.08\textwidth,style=bold]
\setupparagraphs[mypar]
  [2][width=.42\textwidth]
\startmypar
1252   \mypar
Hasselt obtains its city charter from bishop
Hendrik van Vianden.
\mypar
Hendrik van Vianden was pressed by other
towns not to agree with the charter. It took
Hasselt a long period of time to convince
the Bishop. After supporting the Bishop in
a small war against the Drents, the charter
was released.
\stopmypar
```

1252 Hasselt obtains its city charter from bishop Hendrik van Vianden.

Hendrik van Vianden was pressed by other towns not to agree with the charter. It took Hasselt a long period of time to convince the Bishop. After supporting the Bishop in a small war against the Drents, the charter was released.

Here is another example of paragraph formatting:

Lime-kilns Hasselt has its own limekilns. These were build in 1504 and produced quick lime up to 1956. Nowadays they are a touristic attraction.

$$CaCO_3 \longrightarrow CaO + CO_2$$

```
\defineparagraphs[chempar]
   [n=3,before=,after=,distance=1em]
\setupparagraphs [chempar]
   [1] [width=.09\textwidth]
\setupparagraphs [chempar]
   [2] [width=.38\textwidth]
\startchempar
\it Limekilns
\chempar
Hasselt has its own limekilns. These were
build in 1504 and produced quick lime up
to 1956. Nowadays they are a touristic
attraction.
\chempar
\chemical{CaCO_3,~,GIVES,~,CaO,~,+,~,CO_2}
\stopchempar
```

By the way, you could also type it in a more cryptic way:

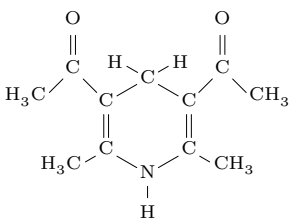
Lime-kilns Hasselt has its own limekilns. These were build in 1504 and produced quick lime up to 1956. Nowadays they are a touristic attraction.

$$CaCO_3 \longrightarrow CaO + CO_2$$

```
\chempar \it Limekilns
\\ Hasselt has its own limekilns. These
were build in 1504 and produced quick
lime up to 1956. Nowadays they are a
touristic attraction.
\\ \chemical{CaCO_3,~,GIVES,~,%
             CaO,~,+,~,CO_2} \\
```

The \\ are used as column separators and they are essential.

The chemical module is explained in H. Hagen and T. Otten's PPCHTEX manual (cdrom) because not everybody is interested in chemical stuff. However, chemical structures always look impressive.



Compound A

```
\startchemical[scale=small,width=fit,%
  top=3000,bottom=3000]
  \chemical[SIX,SB2356,DB14,Z2346,SR3,RZ3,%
    -SR6,+SR6,-RZ6,+RZ6]
           [C,N,C,C,H,H,H]
  \chemical[PB:Z1,ONE,Z0,DIR8,Z0,SB24,DB7,%
    Z27,PE][C,C,CH_3,O]
  \chemical[PB:Z5,ONE,Z0,DIR6,Z0,SB24,DB7,%
    Z47,PE][C,C,H_3C,O]
  \chemical[SR24,RZ24][CH_3,H_3C]
  \bottext{Compound A}
\stopchemical
```

CONTEXT relies on the PICTEX macros of M. Wichura to draw this kind of structures. Although the chemical module introduces only a few commands, it takes some practice to get the right results.

The vertical spacing between paragraphs can be specified by:

```
\setupwhitespace[option]
```

where *option* can be: `none`, `small`, `medium`, `big`, or a dimension, e.g. `3pt`.

When inter-paragraph spacing is specified there are two commands available that are seldom needed:

```
\nowhitespace
```

and

```
\whitespace
```

If a paragraph contains horizontal lines, the line spacing needs extra care. Consider the following example:

In the Hoogstraat in Hasselt there is a stone tablet with a representation of the

cow

that commemorates the siege and the wisdom of the citizens of Hasselt.

```
In the Hoogstraat in Hasselt there is a
stone tablet with a representation of the

\framed{cow}

that commemorates the siege and the wisdom
of the citizens of Hasselt.
```

Here we see an alignment problem that must be corrected. You could carry out a correction with:

```
\startlinecorrection ... \stoplinecorrection
```

If you enter the same text as follows, you will get better output:

In the Hoogstraat in Hasselt there is a stone tablet with a representation of the

cow

that commemorates the siege and the wisdom of the citizens of Hasselt.

```
In the Hoogstraat in Hasselt there is a
stone tablet with a representation of the

\startlinecorrection
\framed{cow}
\stoplinecorrection

that commemorates the siege and the wisdom
of the citizens of Hasselt.
```

Another command that deals with vertical spacing is:

```
\blank[options]
```

You can specify the amount of blank space using keywords such as `small`, `middle` and `big`, which are related to the current font size.

```
In official writings Hasselt always has
the affix Ov. This is an abbreviation for
the province of {\em Overijssel}.
\blank[2*big]
The funny thing is that there is no other
Hasselt in the Netherlands. So it is
redundant.
\blank
The affix is a leftover from the times
that the Netherlands and Belgium were one
country under the reign of King Philip II
of Spain.
\blank[big]
Hasselt in Belgium lies in the province
of Limburg. One wonders if the Belgian
people write Hasselt (Li) on their
letters.
```

In official writings Hasselt always has the affix Ov. This is an abbreviation for the province of *Overijssel*.

The funny thing is that there is no other Hasselt in the Netherlands. So it is redundant.

The affix is a leftover from the times that the Netherlands and Belgium were one country under the reign of King Philip II of Spain.

Hasselt in Belgium lies in the province of Limburg. One wonders if the Belgian people write Hasselt (Li) on their letters.

If you don't specify a parameter, the command `\blank` will yield the default space. Default spacing can be set up with:

```
\setupblank[options]
```

where *option* can be: `none`, `small`, `medium`, `big`, or a dimension, e.g. `3pt`.

If you want to suppress vertical spacing you can use:

```
\startpacked[options] ... \stoppacked
```

```
\defineparagraphs[city][n=2,before=,after=]
\city Hasselt (Ov) \\ Overijssel \\
\city Hasselt (Li) \\ Limburg \\
\startpacked
\city Hasselt (Ov) \\ The Netherlands \\
\city Hasselt (Li) \\ Belgium \\
\stoppacked
```

Hasselt (Ov) Overijssel
Hasselt (Li) Limburg
Hasselt (Ov) The Netherlands
Hasselt (Li) Belgium

It is not hard to imagine why there is also:

```
\startunpacked ... \stopunpacked
```

You can force vertical space with

```
\godown[distance]
```

where *distance* can be any distance (e.g. in inches) or a blank space keyword.

18.3.6 Margin text

It is very easy to put text in the margin. You just use the command

`\inmargin{`*something*`}`

The example below demonstrates how you can specify a small note in the margin of the page:

This is a very marginal note.

```
\inmarge{This is a very marginal note.}
```

18.3.7 Outlining text

You can outline a text with the command `\framed`. The command looks like this:

`\framed[`*options*`]{`*something*`}`

Here is an example:

How do you like this box?

```
\framed[height=10mm,width=fit]
  {How do you like this box?}
```

If you want to outline small pieces of text within a paragraph, it is better to use the command

`\inframed[`*options*`]{`*something*`}`

because it automatically takes care of interline spacing.

Complete paragraphs can be outlined with the following commands:

`\startframedtext[`*options*`]` ... `\stopframedtext`

We will demonstrate this here:

It was essential for Hasselt to have a bridge across the river Zwarte Water. The bishop of Utrecht gave Hasselt his consent in 1486.

Other cities in the neighbourhood of Hasselt were afraid of the toll money to be paid when crossing this bridge so they prevented the construction.

```
\startframedtext[width=.7\makeupwidth]
It was essential for Hasselt to
have a bridge across the river
Zwarte Water. The bishop of Utrecht
gave Hasselt his consent in 1486.
\blank
Other cities in the neighbourhood
of Hasselt were afraid of the toll
money to be paid when crossing
this bridge so they prevented the
construction.
\stopframedtext
```

The \blank command was used to insert a blank line.

The command to set up frames globally is

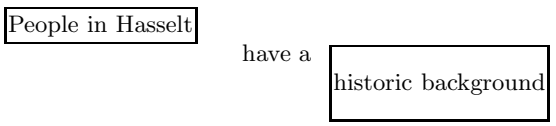
\setupframed[*options*]

Some of the options are described below:

Option	Value
height	fit broad *dimension*
width	fit broad *dimension*
align	yes no
corner	round right

18.3.8 Alignment

Single lines of text can be aligned with the commands \leftaligned, \midaligned and \rightaligned, as demonstrated in the example below:




```
\leftaligned
  {\inframed[width=fit]
    {People in Hasselt}}
\midaligned
  {\inframed[height=1.5cm,frame=off]
    {have a}}
\rightaligned
  {\inframed[height=10mm]
    {historic background}}
```

Alignment of paragraphs is done with the commands

\startalignment*[options]* ... \stopalignment

You can optionally specify the tolerance (stretchability) and the direction (vertical or horizontal).

\setuptolerance[*options*]

By default the tolerance is verystrict. In columns you could specify verytolerant like this:

```
\setuptolerance[horizontal,verytolerant]
```

Horizontal and vertical alignment can be set up with the command

\setupalign[*options*]

Some of the options are: width, left, right, middle, broad and line.

18.4 Fonts and font switches

By default CONTEXT will use the Computer Modern font family. You can also use the Lucida Bright font family. A number of PostScript font families, such as Times, are also available.

18.4.1 Font style and size

You select the font family, style and size for your document with the command `\setupbodyfont`. In the set-up area of your document you could write, e.g.

```
\setupbodyfont[sansserif,9pt]
```

For changes mid-document or on paragraph level you should use the command

`\switchtobodyfont[`*options*`]`

Here is an example:

```
\startnarrower
\switchtobodyfont[8pt,sansserif]
\startlines
Foekepotterij, foekepotterij,
Geef mij een centje dan ga'k voorbij.
Geef mij een alfje dan blijf ik staan,
'k Zak nog liever naar m'n arrenmoeder gaan.
Hier woont zo'n rieke man, die zo vulle %
g\'even kan.
G\'ef wat, old wat, g\'ef die arme wat,
'k Eb zo lange met de foekepot elopen.
'k Eb gien geld om brood te kopen.
Foekepotterij, foekepotterij,
Geef mij een centje dan ga'k voorbij.
\stoplines
\stopnarrower
```

Foekepotterij, foekepotterij,
Geef mij een centje dan ga'k voorbij.
Geef mij een alfje dan blijf ik staan,
'k Zak nog liever naar m'n arrenmoeder gaan.
Hier woont zo'n rieke man, die zo vulle gèven kan.
Gèf wat, old wat, gèf die arme wat,
'k Eb zo lange met de foekepot elopen.
'k Eb gien geld om brood te kopen.
Foekepotterij, foekepotterij,
Geef mij een centje dan ga'k voorbij.

You should notice that `\startnarrower ... \stopnarrower` is also used as a begin and end of the font switch. The function of `\startlines` and `\stoplines` in this example should be obvious: line breaks in the input file are obeyed.

If you want an overview of the available font family, you can type:

```
\showbodyfont[cmr]
```

`cmr` is an abbreviation of Computer Modern Roman. The result will be:

[cmr]													
	\tf	\sc	\sl	\it	\bf	\bs	\bi	\tfx	\tfxx	\tfa	\tfb	\tfc	\tfd
\rm	Ag	AG	*Ag*	*Ag*	**Ag**	***Ag***	***Ag***	Ag	Ag	Ag	Ag	Ag	Ag
\ss	Ag	Ag	*Ag*	*Ag*	**Ag**	**Ag**	**Ag**	Ag	Ag	Ag	Ag	Ag	Ag
\tt	Ag	Ag	*Ag*	*Ag*	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag

18.4.2 Style and size switches in commands

In a number of commands you can optionally specify the desired typestyle. For example:

```
\setuphead[chapter][style=\tfd]
```

In this case the font size for chapter headings is indicated with the command \tfd. But instead of a command you could use the predefined options that are related to the actual typeface: normal, bold, slanted, boldslanted, type, mediaeval, small, smallbold, smallslanted, smallboldslanted, smalltype, capital, cap.

18.4.3 Local font style and size

In running text you can change the font style locally to roman, sans serif and teletype with the commands \rm, \ss and \tt. You can switch to and italic or bold with the commands \sl and \bf.

Font sizes are available from 4pt to 12pt. You can switch to a specific size with the command \switchtobodyfont. The actual style is indicated with \tf. If you want to switch to a somewhat greater size, you can type \tfa, \tfb, \tfc or \tfd. An addition of a, b, c or d to \sl, \it or \bf is also allowed. In the next example we will demonstrate the effect of a few of these commands.

Mintage

In the period from 1404 till 1585 Hasselt had its own *right of coinage*. This right was challenged by other cities, but the bishops of Utrecht did not honour these *protests*.

```
{\tfc Mintage} \par
In the period from {\tt 1404} till
{\tt 1585} Hasselt had its own {\sl
right of coinage}. This right was
challenged by other cities, but the
{\switchtobodyfont[7pt] bishops of
Utrecht} did not honour these
{\slb protests}.
```

Notice how curly braces were used to keep font switching local.

18.4.4 Redefining the body font

For special purposes you can define your own font size using the command

```
\definebodyfont[size] [style] [options]
```

In the example below we will define the body font to be 'Roman', 9 points. Additionally we redefine the `\tfe` command as a Computer Modern Roman font at 24 points:

Hasselt

```
\definebodyfont[9pt][rm][tfe=cmr12 at 24pt]
{\tfe Hasselt}
```

18.4.5 Small caps

Abbreviations such as PDF (Portable Document Format) are printed in pseudo small caps. A small capital is somewhat smaller than the capital of the actual bodyfont. Pseudo small caps are produced with:

```
\cap{text}
```

Below you can see how it looks. The command `\sc` shows the real small caps. The reason for using *pseudo* small caps instead is not just a matter of taste. The pseudo small caps are more 'intelligent': they adapt to the surrounding text.

PDF club, PDF CLUB and PDF CLUB

```
PDF club, \cap{PDF club} and {\sc PDF club}
```

18.4.6 Emphasized

To emphasize words consistently throughout your document you should use:

```
\em
```

Emphasized words appear in a slanted or italic style.

If you walk through Hasselt you should ***watch out*** for *Amsterdammers*. An *Amsterdammer* is ***not*** a person from Amsterdam but a little stone pilar used to separate sidewalk and road. A pedestrian should be protected by these *Amsterdammers* against cars but more often people get hurt from tripping over them.

```
If you walk through Hasselt you should
{\bf \em watch out} for {\em Amsterdammers}.
An {\em Amsterdammer} is {\bf \em not} a
person from Amsterdam but a little stone
pilar used to separate sidewalk and road.
A pedestrian should be protected by these
{\em Amsterdammers} against cars but more
often people get hurt from tripping over
them.
```

Emphasizing something within an already emphasized environment will result in 'normal' print. This method ensured that you still some kind of contrast.

18.4.7 Verbatim text

If you want to display typed text and you want the output to reflect the line breaks as they appear in the input, you should use the commands

```
\starttyping ... \stoptyping
```

In running text you can use:

```
\type{anything}
```

The curly braces enclose the text that you want to type verbatim. You have to be careful with `\type` because it will temporarily disable the line breaking mechanism.

The command **\type** starts with a backslash ****.

```
The command \type{\type} starts with
a backslash \type{\}.
```

You can set up the layout of verbatim text with these commands:

```
\setuptype[options]
```

and

```
\setuptyping[options]
```

These commands allow you to specify how verbatim text should be typeset. There are a few neat tricks built in that we will show in the next example:

```
This is a special case.

      How's that?

      An \extraordinary{surprise}
        \for \all!
```

```
\setuptype[option=slanted]
\type{This is a <<special>> case.}

\setuptyping[margin=1cm]
\starttyping
How's that?
\stoptyping

\setuptyping[option=color]
\starttyping
An \extraordinary{surprise}
  \for \all!
\stoptyping
```

In the example above the strings `\extraordinary`, `\for` and `\all` are typeset in green and the braces are typeset in red. Note that all text, including these strings, is typeset verbatim.

18.5 Figures

Photographs and pictures can be inserted in your document as follows:

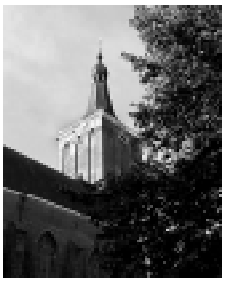


Figure 1 Stephanus Church.

```
\placefigure
  [][fig:church]
  {Stephanus Church.}
  {\externalfigure[hass24g]}
```

The command `\placefigure` handles numbering and vertical spacing before and after the figure. Furthermore this command initializes a 'float' mechanism. This means that CONTEXT checks whether there is enough space for the figure. If not, the figure will be placed at another location while open space is filled out with text: the figure starts floating in your document until the optimal location is found. You can influence this mechanism using with the first parameter of the `\placefigure` command. The command `\placefigure` is actually an instance of the command

`\place`*block*`[`*options*`] [`*ref*`,...] {`*caption*`}{`*content*`}`

Possible options are listed below:

Location	Meaning
`here`	put figure at this location if possible
`force`	ignore float mechanism and place figure
`page`	put figure at top of the next page
`top`	put the caption above figure
`bottom`	put caption under figure
`left`	place figure at the left margin
`right`	place figure at the right margin

The second parameter is used as a label that can be referred to elsewhere in the document. You can refer to this particular figure as follows:

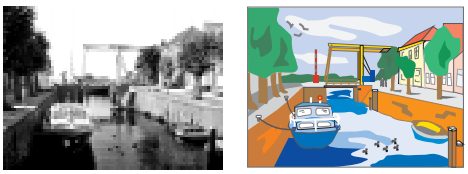
```
\in{figure}[fig:church]
```

The first parameter in curly braces is used to specify the caption. You can type any text you want. If you want no caption and no number, you can use the keyword `none`. The second parameter in curly braces is used for defining the figure itself. Here is another example:

Figure 1 A framed Hasselt.

```
\placefigure
  {A framed Hasselt.}
  {\framed{\tfd Hasselt}}
```

However, probably most of your figure were produced using graphics programs such as Corel Draw or Adobe Illustrator. Such figures are available as separate files in a specific graphics format. CONTEXT, in conjunction with the program TEXUTIL, supports the graphics formats EPS, TIF, JPG, MPS, PDF and PNG. Beware that graphics inclusion depends heavily on the capabilities of the DVI driver. Users can normally depend on CONTEXT to find the best suited file type. Before processing your document, CONTEXT needs some information on the figures, such as file format and dimensions. This is done by `\useexternalfigure`. The next example shows a photo combined with a graphic into one figure:



a bitmap picture a vector graphic

Figure 1 The Hasselt Canals.

```
\useexternalfigure
  [Graphic][gracht]
  [width=.4\textwidth]
\useexternalfigure
  [Photo][hass03g]
  [width=.4\textwidth]
\placefigure
  [here,force]
  [fig:canal]
  {The Hasselt Canals.}
  \startcombination[2*1]
     {\Photo}{a bitmap picture}
     {\Graphic}{a vector graphic}
  \stopcombination
```

`\useexternalfigure[`*identifier*`][`*file*`]` *`[options]`*

This command has three parameters. The first one contains a logical name for the figure. This is optional and used when one figure appears more than once in your document.

The parameter contains the file name (without extension). The third parameter leaves room to set up parameters. In well structured documents you would type `\useexternalfigure` in the set-up area of your document.

`\startcombination[`*combination*`]` ... `\stopcombination`

These commands are used for combining two pictures in one figure. You can specify the number of pictures as a parameter. If you want to display one picture below the other you should type `[1*2]`. You can imagine what happens if you combine 6 pictures as `[3*2]` ([horizontal*vertical]).

The examples shown above are enough for creating illustrated documents. Sometimes, however, you want a more integrated layout of pictures and text. For that purpose you can use:

\start*block*text *[location] [ref]* {*caption*}{*content*} ... \stop*block*text

Here is an example:

Hasselt has always had a varying number of citizens due to economic events. For example the Dedemsvaart was dug around 1810. This canal runs through Hasselt and therefore trade flourished. This led to a population growth of almost 40% within 10 years. Nowadays the Dedemsvaart has no commercial value anymore and the canals have become a touristic attraction.

```
\startfiguretext
  [left]
  [fig:citizens]
  {none}
  {\externalfigure[hass07g]
   [width=.5\makeupwidth,frame=on]}
  Hasselt has always had a varying number
  of citizens due to economic events. For
  example the Dedemsvaart was dug around
  1810. This canal runs through Hasselt and
  therefore trade flourished. This led to a
  population growth of almost 40\% within
  10~years. Nowadays the Dedemsvaart has no
  commercial value anymore and the canals
  have become a touristic attraction.
\stopfiguretext
```

In the last example we used the command

\externalfigure[*file name*] [*options*]

The file name should be specified *without* extension, since CONTEXT will automatically choose the best available type. The second parameter can be used to specify the size of the figure. You can set up the layout of figures with:

\setupfloats[*options*]

The most important options for \setupfloats are:

Option	Value
location	left middle right
width	fit *dimension*

You can set up the numbering and the labels with:

\setupcaptions[*options*]

The most important options for \setupcations are:

Option	Value
location	top bottom none
width	fit max *dimension*
style	bold slanted cap ...
number	yes no
align	left middle right no

These commands should be used in the set-up area of your document. They have a global effect on all floating blocks. Here is an example:

A picture on the right:

Figure 1 ***Just a picture.***

```
\setupfloats
  [location=right]
\setupcaptions
  [location=top,
   height=.4\makeupheight,
   style=boldslanted]
A picture
on the right:
\placefigure
  {Just a picture.}
  {\externalfigure
      [hass18g]
      [frame=on,
       width=50mm]}
```

18.6 Tables

In general, a table consists of columns which may be independently left adjusted, centered, right adjusted, or aligned on decimal points. Headings may be placed over single columns or groups of columns. Table entries may contain equations or several rows of text. Horizontal and vertical lines may be drawn wholly or partially across the table.

This is what M. Wichura wrote in the preface of the manual of TABLE (TABLE manual, 1988). M. Wichura is also the author of the TABLE macros that CONTEXT relies on for processing tables. A few CONTEXT macros were added to take care of consistent line spacing and to make the interface a little less cryptic.

For placing a table the command \placetable is used, which is a predefined version of \placeblock (see section 18.5). For defining the table you use:

```
\starttable[template] ... \stoptable
```

In the next example we will demonstrate these commands.

Year	Number of ships
1645	450
1671	480
1676	500
1695	930

Table 1 Ships that moored at Hasselt.

```
\placetable[here][tab:ships]
   {Ships that moored at Hasselt.}
\starttable[|c|c|]
\HL
\NC \bf Year \NC
\bf Number of ships \NC\SR
\HL
\NC 1645 \NC 450 \NC\FR
\NC 1671 \NC 480 \NC\MR
\NC 1676 \NC 500 \NC\MR
\NC 1695 \NC 930 \NC\LR
\HL
\stoptable
```

The first command `\placetable` is very similar to `\placefigure`. It takes care of numbering and of spacing before and after the table. Furthermore, the float mechanism can be initialized, to make the table 'float' to its optimal location.

In the table *template* you can use several keys to specify the formatting of columns:

Key	Meaning
`|`	column separator
`c`	center
`l`	flush left
`r`	flush right
`s<n>`	set intercolumn space at value $n = 0, 1, 2$
`w<>`	set minimum column width at specified value

In addition to the format *keys* there are format *commands*. The table below shows a few of the essential commands. These are original TABLE commands.

Command	Meaning
`\JustLeft`	flush left and omit column format
`\JustRight`	flush right and omit column format
`\JustCenter`	center and omit column format
`\SetTableToWidth{}`	specify exact table width
`\use{n}`	use the space of the next *n* columns

In the examples we have shown we used a number of CONTEXT formatting commands. The table below gives an overview of these commands.

Command	Meaning	
\NR	next row	make row with no vertical space adjustment
\FR	first row	make row, adjust upper spacing
\LR	last row	make row, adjust lower spacing
\MR	mid row	make row, adjust upper and lower spacing
\SR	separate row	make row, adjust upper and lower spacing
\VL	vertical line	draw a vertical line, go to next column
\NC	next column	go to next column
\HL	horizontal line	draw a horizontal
\DL	division line	draw a division line over the next column
\DL[n]	division line	draw a division line over *n* columns
\DC	division column	draw a space over the next column
\DR	division row	make row, adjust upper and lower spacing
\LOW{text}	—	lower *text*
\TWO, \THREE, ...	—	use the space of the next *two*, *three*, ... columns

Below are a few more examples.

Year	Citizens
1675	428
1795	1124
1880	2405
1995	7408

standard

Year	Citizens
1675	428
1795	1124
1880	2405
1995	7408

only \NR

Table 1 Effect of formatting commands.

```
\placetable
  [here,force][tab:effects of commands]
  {Effect of formatting commands.}
\startcombination[1*2]
{\starttable[|c|c|]
\HL \VL \bf Year \VL \bf Citizens \VL\SR \HL
\VL 1675 \VL ~428 \VL\FR
\VL 1795 \VL 1124 \VL\MR
\VL 1880 \VL 2405 \VL\MR
\VL 1995 \VL 7408 \VL\LR \HL
\stoptable}{standard}
{\starttable[|c|c|]
\HL \VL \bf Year \VL \bf Citizens \VL\NR \HL
\VL 1675 \VL ~428 \VL\NR
\VL 1795 \VL 1124 \VL\NR
\VL 1880 \VL 2405 \VL\NR
\VL 1995 \VL 7408 \VL\NR \HL
\stoptable}{only \type{\NR}}
\stopcombination
```

In the example above the first table \SR, \FR, \MR and \LR are used. These commands take care of line spacing within a table. As you can see in the second table, the command \NR only starts a new row.

In the examples below column spacing is demonstrated. The first example show the 'standard' spacing:

Year	Citizens
1675	428
1795	1124
1880	2405
1995	7408

```
\starttable[|c|c|]
\HL \VL \bf Year \VL \bf Citizens \VL\SR \HL
\VL 1675 \VL ~428 \VL\FR
\VL 1795 \VL 1124 \VL\MR
\VL 1880 \VL 2405 \VL\MR
\VL 1995 \VL 7408 \VL\LR \HL
\stoptable
```

The next example shows the effect of the `s0` key:

Year	Citizens
1675	428
1795	1124
1880	2405
1995	7408

```
\starttable[s0 | c | c |]
\HL \VL \bf Year \VL \bf Citizens \VL\SR \HL
\VL 1675 \VL ~428 \VL\FR
\VL 1795 \VL 1124 \VL\MR
\VL 1880 \VL 2405 \VL\MR
\VL 1995 \VL 7408 \VL\LR \HL
\stoptable
```

The next example shows the effect of `s0` in column 1:

Year	Citizens
1675	428
1795	1124
1880	2405
1995	7408

```
\starttable[| s0 c | c |]
\HL \VL \bf Year \VL \bf Citizens \VL\SR \HL
\VL 1675 \VL ~428 \VL\FR
\VL 1795 \VL 1124 \VL\MR
\VL 1880 \VL 2405 \VL\MR
\VL 1995 \VL 7408 \VL\LR \HL
\stoptable
```

The next example shows the effect of `s1`:

Year	Citizens
1675	428
1795	1124
1880	2405
1995	7408

```
\starttable[s1 | c | c |]
\HL \VL \bf Year \VL \bf Citizens \VL\SR \HL
\VL 1675 \VL ~428 \VL\FR
\VL 1795 \VL 1124 \VL\MR
\VL 1880 \VL 2405 \VL\MR
\VL 1995 \VL 7408 \VL\LR \HL
\stoptable
```

Columns are often separated with a vertical line | and rows by a horizontal line.

Steenwijk	Zwartsluis	Hasselt
Zwartsluis	Hasselt	Steenwijk
Hasselt	Steenwijk	Zwartsluis

```
\starttable[|c|c|c|]
\NC Steenwijk  \NC Zwartsluis \NC Hasselt
\NC\SR \DC \DL \DC \DR
\NC Zwartsluis \VL Hasselt    \VL Steenwijk
\NC\SR \DC \DL \DC \DR
\NC Hasselt    \NC Steenwijk  \NC Zwartsluis
\NC\SR
\stoptable
```

A more sensible example is given in the table below.

City council elections in 1994				
Party	Districts			Total
	1	2	3	
PvdA	351	433	459	1243
CDA	346	350	285	981
VVD	140	113	132	385
HKV/RPF/SGP	348	261	158	767
GPV	117	192	291	600

```
\starttable[|l|c|c|c|c|]
\HL \VL \FIVE \JustCenter{City council
  elections in 1994} \VL\SR \HL
\VL \LOW{Party} \VL \THREE{Districts} \VL
  \LOW{Total} \VL\SR
\DC \DL[3] \DC \DR
\VL \VL 1 \VL 2 \VL 3 \VL \VL\SR
\HL \VL PvdA \VL 351 \VL 433 \VL 459
  \VL 1243 \VL\FR
\VL CDA \VL 346 \VL 350 \VL 285 \VL ~981
  \VL\MR
\VL VVD \VL 140 \VL 113 \VL 132 \VL ~385
  \VL\MR
\VL HKV/RPF/SGP \VL 348 \VL 261 \VL 158
  \VL ~767 \VL\MR
\VL GPV \VL 117 \VL 192 \VL 291 \VL ~600
  \VL\LR \HL
\stoptable
```

In the last column a ~ is used to simulate a four digit number. The ~ has the width of a digit.

In case a table becomes too big, you may want to adjust, for instance, the font size or the vertical and/or horizontal spacing. This is done with the command

```
\setuptables[options]
```

which we will use in the next example to squeeze a table that otherwise would not fit:

```
{\setuptables[bodyfont=5pt,distance=small]
\starttable[|c|c|c|c|c|]
\HL \VL \use5 \JustCenter{Decline of wealth
in Dutch guilders (Dfl)} \VL\SR
\HL \VL Year
\VL 1.000--2.000
\VL 2.000--5.000
\VL 5.000--10.000
\VL over 10.000 \VL\SR \HL
\VL 1675 \VL 22 \VL 12 \VL 4 \VL 5
  \VL\FR
\VL 1724 \VL ~4 \VL ~4 \VL 4 \VL 3
  \VL\MR
\VL 1750 \VL 12 \VL ~5 \VL 2 \VL --
  \VL\MR
\VL 1808 \VL ~9 \VL ~2 \VL -- \VL --
  \VL\LR \HL
\stoptable}
```

Decline of wealth in Dutch guilders (Dfl)				
Year	1.000–2.000	2.000–5.000	5.000–10.000	over 10.000
1675	22	12	4	5
1724	4	4	4	3
1750	12	5	2	–
1808	9	2	–	–

You can also set up the layout of tables with the commands `\setupfloats` and `\setupcaptions`, as explained in section 18.5. Here is an example:

```
\setupfloats[location=left]
\setupcaption[table][style=cap]
\placetable{Opening hours library.}
\starttable[|l|c|c|]
\HL \VL \bf Day \VL \use2
    \bf Opening hours \VL\SR\HL
\VL Monday \VL 14.00 -- 17.30
\VL 18.30 -- 20.30 \VL\FR
\VL Tuesday \VL \VL \VL\MR
\VL Wednesday \VL 10.00 -- 12.00
\VL 14.00 -- 17.30 \VL\MR
\VL Thursday \VL 14.00 -- 17.30
\VL 18.30 -- 20.30 \VL\MR
\VL Friday \VL 14.00 -- 17.30
\VL \VL\MR
\VL Saturday \VL 10.00 -- 12.30
\VL \VL\LR\HL
\stoptable
```

Day	**Opening hours**	
Monday	14.00 – 17.30	18.30 – 20.30
Tuesday		
Wednesday	10.00 – 12.00	14.00 – 17.30
Thursday	14.00 – 17.30	18.30 – 20.30
Friday	14.00 – 17.30	
Saturday	10.00 – 12.30	

Table 1 OPENING HOURS LIBRARY.

18.7 Postponing

Figures or tables (or other pieces of text) may take up a lot of space. Yet you don't want to break them over two pages. If there is insufficient room on the current page, you would rather postpone a figure or table until the next page break. The commands to 'postpone' are

```
\startpostponing ... \stoppostponing
```

Here is an example of specifying a postponed figure:

```
\startpostponing
\placefigure
  {A postponed figure.}
  {\externalfigure
      [hass16g][frame=on,width=\textwidth]}
\stoppostponing
```

The figure will be placed at the top of the next page. It will cause minimal disruption of running text. Of course this method can be applied to any other piece of your document.

18.8 Item lists

If you need to typeset a list of items, you can use the command `\startitemize`. This command allows you to make almost any kind of list, using bullets, numbers or other symbols to identify items. In section 18.8.1 we will describe in detail how this command can be used.

Definition lists typically have a somewhat different layout. In section 18.8.2 we will describe command to typeset definition lists.

Numbered definition lists are also supported. This is in fact a variation on definition lists. In section 18.8.3 we will explain this feature.

18.8.1 Itemizing an enumerating

Itemizations and enumerations are made with:

```
\startitemize[keywords] [options] ... \stopitemize
```

Between these commands you can specify items using the command

```
\item
```

Here is an example:

```
\startitemize[R,packed,broad]
\item Hasselt was founded in the 14th
century.
\item Hasselt is known as a so-called
Hanze town.
\item Hasselt's name stems from a tree.
\stopitemize
```

I. Hasselt was founded in the 14th century.
II. Hasselt is known as a so-called Hanze town.
III. Hasselt's name stems from a tree.

In the example above, the `R` specifies Roman numbering and the keyword `packed` instructs CONTEXT to keep line spacing tight. The parameter `broad` takes care of vertical spacing after the item separator. The table below shows several other keywords you can use to configure a specific item list:

Keyword	Item separator
1	–
2	•
3	⋆
⋮	⋮
n	1 2 3 4 . . .
a	a b c d . . .
A	A B C D . . .
r	i ii iii iv . . .
R	I II III IV . . .

You can also define your own item separator by using the command

`\definesymbol[`*name*`][`*typeset symbol*`]`

For example:

```
\definesymbol[5][\small\inframed{+}]
\startitemize[5,packed]
\item Hasselt was built on a riverdune.
\item Hasselt lies at the crossing of
two rivers.
\stopitemize
```

[+] Hasselt was built on a riverdune.
[+] Hasselt lies at the crossing of two rivers.

The set-up parameters are described below:

Option	Meaning
`standard`	standard set-up (global set-up)
`packed`	no vertical spacing between items
`serried`	no horizontal spacing between separator and text
`joinedup`	no vertical spacing before and after itemize
`broad`	horizontal spacing between separator and text
`inmargin`	place separator in margin
`atmargin`	place separator on margin
`stopper`	place full stop after separator
`columns`	put items in columns
`intro`	prevent page breaking after introduction line
`continue`	continue numbering or lettering

The option `columns` is used in conjunction with a number, as demonstrated in the next example:

```
\startitemize[n,columns,two]
\item Achter 't Werk
\item Meestersteeg
\item Heerengracht
\item Julianakade
\item Baangracht
\item Eiland
\item Kastanjelaan
\item Justitiebastion
\stopitemize
```

1. Achter 't Werk
2. Meestersteeg
3. Heerengracht
4. Julianakade
5. Baangracht
6. Eiland
7. Kastanjelaan
8. Justitiebastion

Sometimes you may want to continue an enumeration after a short intermezzo. The example below shows how to continue the itemization in a three column format. The parameter `broad` enlarges the horizontal space between item separator and item text.

```
\startitemize[n,columns,two,broad]
\item Keppelstraat
\item Justitiebastion
\item Brouwersgracht
\item Gasthuisstraat
\stopitemize
\startitemize[continue,columns,three]
\item Eikenlaan
\item Kaai
\item Kalverstraat
\item Hofstraat
\item Markt
\item Hoogstraat
\stopitemize
```

1. Keppelstraat
2. Justitiebastion
3. Brouwersgracht
4. Gasthuisstraat
5. Eikenlaan
6. Kaai
7. Kalverstraat
8. Hofstraat
9. Markt
10. Hoogstraat

To ensure consistency throughout your document, you can make global settings with

\setupitemize[*level*] [*style*] [*options*]

where *level* represents the nesting level of an itemization. The first is 1. *style* can be standard, n*broad, n*serried, packed or joinedup. Some of the options are:

Option	Value
align	left right normal
style	bold slanted cap ...
width	*dimension*
distance	*dimension*

In the example below we set the width of all secondary itemizations to 2 cm:

```
\setupitemize[2]
  [width=2cm,style=boldslanted]
In The Netherlands the cities can determine
the height of a number of taxes. So the
costs of living can differ from town to town.
There are differences of upto 50\% in taxes
like:
\startitemize[n]
\item real estate tax \par
The real estate tax is divided in two
components:
\startitemize[a,packed]
\item the ownership tax
\item the tenant tax
\stopitemize
If the real estate has no tenant the owner
pays both components.
\item dog license fee \par
The owner of one or more dogs pays a fee.
When a dog has died or been sold the owner
has to inform cityhall.
\stopitemize
```

In The Netherlands the cities can determine the height of a number of taxes. So the costs of living can differ from town to town. There are differences of upto 50% in taxes like:

1. real estate tax

   The real estate tax is divided in two components:

   a. the ownership tax
   b. the tenant tax

   If the real estate has no tenant the owner pays both components.

2. dog license fee

   The owner of one or more dogs pays a fee. When a dog has died or been sold the owner has to inform cityhall.

18.8.2 Definition lists

If you want to display notions, concepts, ideas, etc. in a consistent manner you can use the command

\definedescription[*identifier*] [*options*]

Here is an example:

Hasselter juffer *A sort of cookie of puff pastry and covered with sugar. It tastes very sweet.*

```
\definedescription
  [concept] [location=serried,
             style=slanted,width=broad]
  \concept{Hasselter juffer} A sort of cookie
  of puff pastry and covered with sugar.
  It tastes very sweet. \par
```

You can also choose other layouts:

Hasselter bitter

A very strong alcoholic drink (up to 40%) mixed with herbs to give it a special taste. It is sold in a stone flask and it should be served *ijskoud* (as cold as ice).

```
\definedescription
  [concept]
  [location=top,headstyle=bold,width=broad]
\concept{Hasselter bitter} A very strong
alcoholic drink (up to 40\%) mixed with
herbs to give it a special taste. It is
sold in a stone flask and it should be
served {\em ijskoud} (as cold as ice).\par
```

If you have more paragraphs in a definition you can use a `\start...` and `\stop...` pair:

Euifeest A harvest home to celebrate the end of a period of hard work.

This event takes place at the end of August and lasts one week. The city is completely illuminated and the streets are decorated. This feast week ends with a *Braderie*.

```
\definedescription
  [concept]
  [location=right,headstyle=bold,width=broad]
\startconcept{Euifeest} A harvest home to
celebrate the end of a period of hard work.

This event takes place at the end of August
and lasts one week. The city is completely
illuminated and the streets are decorated.
This feast week ends with a {\em Braderie}.
\stopconcept
```

The layout is normally specified in the second parameter, but you can also use

`\setupdescription[identifier][options]`

to specify global settings.

18.8.3 Numbered definitions

As a variation on definition lists, you can (automatically) number text elements. This can be handy for defining, e.g. remarks or questions.

`\defineenumeration[name][options]`

Here is an example:

```
\defineenumeration
  [remark]
  [location=top,
   text=Remark,
   inbetween=\blank,
   after=\blank]
```

Now the new commands \remark, \subremark, \resetremark and \nextremark are available and you can type remarks like this:

Remark 1

In the early medieval times Hasselt was a place of pilgrimage. The *Heilige Stede* (Holy Place) was torn down during the Reformation. In 1930, after 300 years the *Heilige Stede* was reopened.

Remark 1.1

Nowadays the *Heilige Stede* is closed again but once a year an open air service is held on the same spot.

```
\remark In the early medieval times Hasselt
was a place of pilgrimage. The {\em Heilige
Stede} (Holy Place) was torn down during
the Reformation. In 1930, after 300 years
the {\em Heilige Stede} was reopened.

\subremark Nowadays the {\em Heilige Stede}
is closed again but once a year an open air
service is held on the same spot.
```

You can reset numbering with \resetremark or \resetsubremark or increment a number with \nextremark or \nextsubremark. CONTEXT will reset counters automatically at every chapter, section, etc.

You can set up the layout of \defineenumeration with

\setupenumerations*[name]*[*options*]

You can also change the layout of \remark and \subremark in the example above, e.g. like this:

```
\setupenumeration[remark][headstyle=cap]
\setupenumeration[subremark][headstyle=italic]
```

18.9 Cross referencing

For referring from one location in a document to another you can use the command

\in{*element*}[*reference*]

The *element* should contain a keyword such as chapter, paragraph, figure or table. The *reference* should be a logical label. An example will demonstrate the usage of the \in command:

```
\chapter[hotel]{Hotels in Hasselt}
```

Now you can refer to this chapter like this:

```
\in{chapter}[hotel]
```

which will yield something like ‘chapter 9’. Here is another example:

There are a number of things you can do in Hasselt:

1. swimming
2. sailing
3. hiking
4. biking

Activities like activity 3 described on page 1 are very tiresome.

```
There are a number of things you
can do in Hasselt:
\startitemize[n,packed]
\item swimming
\item sailing
\item[hiking] hiking
\item biking
\stopitemize
Activities like \in{activity}[hiking]
described on \at{page}[hiking] are
very tiresome.
```

You can also refer to pages. This is done with the command

`\at{`*type*`}[`*ref*`]`

For example with:

```
\at{page}[hiking]
```

This command can be used in combination with:

`\pagereference[`*ref*`]`

and

`\textreference[`*ref*`]{`*text*`}`

If you want to refer to the chapter ‘Hotels in Hasselt’ you could type:

```
Look in \in{chapter}[hotel] on \at{page}[hotel] for a
complete listing of the accommodations in
\pagereference[accommodations]{Hasselt}.
```

A chapter number and a page number will be generated when processing the document. Elsewhere in the document you can refer to accommodations with `\at{page}[accommodations]`.

`\about[`*ref*`]`

will generate the *title* of a chapter, section, etc.

18.10 Indexes

It is possible to generate one or more indexes. You can use the command

\index*[sortkey]*{*realkey*}

to specify an entry in the index:

```
\index{town hall}
```

The words 'town hall' will appear as one index entry. An index is sorted in alphabetical order by an auxiliary program. Sometimes the index entry cannot be alphabetized. For instance, mathematical symbols have to be stated in a redundant way in order to produce a correct alphabetical list:

```
\index[alpha]{$\alpha$}
```

Sometimes you may want to specify sub, or even subsub entries. These can be defined as follows:

```
\index{town hall+location}
\index{town hall+architecture}
```

You can generate your index list with the command

\placeindex

or

\completeindex

But of course you can also define your own index. In CONTEXT we call them 'registers'. You can define registers using the command

\defineregister[*identifier*] [*plurale*]

For instance, if you want to make a new register based on the streets in Hasselt you could enter:

```
\defineregister[street][streets]
```

Now a new register command \street is available, and you can add entries like this: \street{Ridderstraat}. To produce a list of entries you could now use:

```
\placestreet
\completestreet
```

You can change the display of the registers with the command

\setupregister[*identifier*] [*options*]

Some of the options are described below:

Option	Value
balance	yes no
align	yes no
style	bold slanted small ...

18.11 Footnotes

If you want to annotate your text you can use the command

\footnote[*ref*]{*text*}

The *ref* parameter is optional and contains a logical name. The parameter in curly braces contains the text you want to display at the foot of the page. The footnote number can be referenced by its logical name:

\note[*ref*]

If you type this text:

```
The Hanzepact was a late medieval commercial alliance of towns in the
regions of the North and the Baltic Sea. The association was formed
for the furtherance and protection of the commerce of its
members.\footnote[war]{This was the source of jealousy and fear among
other towns that caused a number of wars.} In the Hanze period there
was a lively trade in all sorts of articles such as wood, wool,
metal, cloth, salt, wine and beer.\note[war] The prosperous trade
caused an enormous growth of welfare in the Hanzeatic
towns.\footnote{Hasselt is one of these towns.}
```

It would be typeset like this:

The Hanze pact was a late medieval commercial alliance of towns in the regions of the North and the Baltic Sea. The association was formed for the furtherance and protection of the commerce of its members.[2] In the Hanze period there was a lively trade in all sorts of articles such as wood, wool, metal, cloth, salt, wine and beer.[2] The prosperous trade caused an enormous growth of welfare in the Hanzeatic towns.[3]

2 This was the source of jealousy and fear among other towns that caused a number of wars.
3 Hasselt is one of these towns.

Footnotes are numbered automatically. The command

```
\setupfootnotes[options]
```

enables you to change the layout of footnotes. Some of the options are described below:

Option	Value
conversion	numbers characters romannumerals
line	on off
style	bold slanted cap small ...

18.12 Writing text

You already know that the backslash \ and the curly braces {} have a special meaning. But there are more characters that have a special meaning. This means that if you simply type them in your document, the TEX compiler will complain, or at least you will get unexpected results.

In section 18.12.1 we will show which characters are 'special', and how to type them anyway in your document. In section 18.12.2 we will show how to write accented characters. Some 'foreign' characters that may not be available on your keyboard are discussed in section 18.12.3.

18.12.1 Reserved characters

Next to \ there are other characters that need special attention when you want them to appear in verbatim mode or in text mode. Other reserved characters have a meaning in typesetting mathematical expressions and some can be used in math mode only. Table 18.1 gives an overview of these characters and what you have to type to produce them.

18.12.2 Accented characters

Accented characters have to be composed in CONTEXT. The table below shows you how to do this. The character u is just an example here.

Table 18.1: Reserved characters

In text mode:

Reserved character	Verbatim		Text	
	Type	To produce	Type	To produce
#	\type{#}	#	\#	#
$	\type{$}	$	\$	$
&	\type{&}	&	\&	&
%	\type{%}	%	\%	%
In math mode:				
+	\type{+}	+	$+$	+
-	\type{-}	-	$-$	−
=	\type{=}	=	$=$	=
<	\type{<}	<	$<$	<
>	\type{>}	>	$>$	>

You type	You get	You type	You get
\‘{u}	ù	\u{u}	ŭ
\’{u}	ú	\v{u}	ǔ
\^{u}	û	\H{u}	ű
\"{u}	ü	\t{uu}	u͡u
\~{u}	ũ	\c{u}	ų
\={u}	ū	\d{u}	ụ
\.{u}	u̇	\b{u}	u̲

You don’t want ï or ǰ so for an accented i and j you compose the characters as follows: \"\i gives ï; \’{\j} gives ǰ.

A more readable and intuitive way of typing accented characters can bet set up with the command

\useencoding[*encoding*]

With this command you can specify which input encoding you are using. If you are using a Windows editor program (Notepad, MED, PFE, WINEDT, etc.) you will probably be using the ISO-Latin 1 encoding. In that case you should specify the encoding `win` as demonstrated in the next example:

```
\useencoding[win]
e-acute = \’e, i-umlaut = \"\i, n-tilde =
\~n, o-grave = \‘o, c-cedille = \c{c},
u-circonflex = \^u, A-umlaut = \"A
```

e-acute = é, i-umlaut = ï, n-tilde = ñ, o-grave = ò, c-cedille = ç, u-circonflex = û, A-umlaut = Ä

In case you still use a DOS style editor (edit.com, PE, QEDIT, etc.), you should use the encoding `ibm`.

Instead of using the `\useencoding` command, you could use one of TEX's input translation tables, which are explained in section 13.3.2).

You should either use TEX's translation feature *or* the `\useencoding` command in CONTEXT files, but never both.

18.12.3 Foreign symbols

The composition of characters that appear in foreign languages is shown in the table below.

You type	You get	You type	You get
`\oe`	œ	`\O`	Ø
`\OE`	Œ	`\l`	ł
`\ae`	æ	`\L`	Ł
`\AE`	Æ	`\SS`	ß
`\aa`	å	`?‘`	¿
`\AA`	Å	`!‘`	¡
`\o`	ö		

18.12.4 Superscript and subscript in text

Superscripts and subscripts in running text are a bit unusual, but nevertheless, it is possible:

It is very easy to put $^{\text{superscript}}$ and $_{\text{subscript}}$ in your text. What would you call this version $^{\text{superscript}}_{\text{subscript}}$? It looks dab though!

```
It is very easy to put \high{superscript}
and \low{subscript} in your text. What
would you call this version
\lohi{subscript}{superscript}?
It looks dab though!
```

18.12.5 Date

You can use the command

```
\currentdate
```

to insert the current date into your document.

18.12.6 Units

To force yourself to use dimensions and units consistently throughout the document you can make your own list with units. These should specified in the set-up area.

In ConTeXt there is an external module available that contains almost all SI-units. You can load this module by using the command `\usemodule[unit]` (see section 18.18). Then you can write units like this:

$\mathrm{m/m^2}$
$\mathrm{m^3/s}$
$\mathrm{mm^2/inch}$
$\mathrm{cl/s}$
$\mathrm{ms^{-1}}$
$\mathrm{N/inch^2}$
$\mathrm{N \cdot m/s^2}$

```
\startlines
\Meter \Per \Square \Meter
\Kubic \Meter \Per \Sec
\Square \Milli \Meter \Per \Inch
\Centi \Liter \Per \Sec
\Meter \Inverse \Sec
\Newton \Per \Square \Inch
\Newton \Times \Meter \Per \Square \Sec
\stoplines
```

It looks like a lot of typing but it does guarantee consistency of units. The command `\unit` also prevents line breaks between value and unit. You can define your own units with:

```
\unit[Ounce]{oz}{British ounce}
```

Later on in the document you can type '`15.6 \Ounce`' which will yield '15.6 oz'. The legend on page 450 would be more consistent if you type it like this:

s = place m
v = velocity m/ s
t = time s
a = acceleration $\mathrm{m/s^2}$

```
\startlegend
\leg s \\ place \\ \Meter \\
\leg v \\ velocity \\ \Meter \Per \Sec \\
\leg t \\ time \\ \Sec \\
\leg a \\ acceleration \\ \Meter \Per
  \Square \Sec \\
\stoplegend
```

The command `\unit` is an application of synonyms. See section 18.17.2 for more information on synonyms.

In order to write the percent and promille sign in a consistent way there are two specific commands: `\percent` and `\permille`.

18.12.7 Storing text for later use

You can store information temporarily for future use in your document with:

`\startbuffer[`*identifier*`] ... \stopbuffer`

For example:

```
\startbuffer[visit]
If you want to see what Hasselt has in store you come and visit it
some time.
\stopbuffer
...
\getbuffer[visit]
```

With `\getbuffer[visit]` you recall the stored text. The file name is optional. With `\typebuffer[visit]` you could recall the typed version of the content of the buffer.

18.12.8 Hiding text

Text can be hidden with:

```
\starthiding ... \stophiding
```

The text inbetween will not be processed.

18.12.9 Drawing lines

There are many commands to draw lines. For a single line you type:

```
\hairline
```

or

```
\thinrule
```

For more lines you type:

```
\thinrules[options]
```

Text in combination with lines is also possible:

Hasselt – Amsterdam

If you draw a straight line from Hasselt to Amsterdam you would have to cover a distance of almost 145 km.

If you draw two straight lines from Hasselt to Amsterdam you would have to cover a distance of almost 290 km. Amsterdam

Hasselt

```
\starttextrule{Hasselt -- Amsterdam}
If you draw a straight line from Hasselt
to Amsterdam you would have to cover a
distance of almost 145 \Kilo \Meter.
\stoptextrule
If you draw two straight lines from Hasselt
to Amsterdam you would have to cover a
distance of almost 290 \Kilo \Meter.
Amsterdam\thinrules[n=3]Hasselt
```

You always have to be careful in drawing lines. Empty lines around `\thinrules` must not be forgotten and the vertical spacing is always a point of concern.

You can set up line spacing with:

`\setupthinrules[`*options*`]`

Some of the options are:

Option	Value
`distance`	`small` `medium` `big`
`n`	*number*
`height`	*dimension*
`depth`	*dimension*

There are a few complementary commands that might be very useful:

`\setupfillinrules[`*options*`]`

and

`\setupfillinlines[`*options*`]`

Some of the options are:

Option	Value
`width`	*dimension*
`distance`	*dimension*

These commands are introduced in the examples below:

name _____

address _____

_____

_____

Can you please state the number of cars used in your family. _____

Strike any word ~~in~~ ~~this~~ ~~text~~. . . . . .

```
\setupfillinrules[width=15mm]
\setupfillinlines[width=35mm]
\fillinrules[n=1]{\bf name}
\fillinrules[n=3]{\bf address}
\fillinline{Can you please state the
\underbar{number} of cars used in your
family.} \par
Strike any word \overstrikes{in this
text}\periods[7]
```

These commands are used in questionnaires. Beware that text that is struck out or underlined will not be hyphenated.

18.12.10 Positioning

Sometimes you may need to position text on a page or within a text element. You can position text with:

`\position(`*x*`,`*y*`){`*something*`}`

The parentheses enclose the *x* and *y* coordinates, the curly braces enclose the text you want to position. You can set up the *x* and *y* axes with:

```
\setuppositioning[options]
```

Some of the options are:

Option	Value
unit	mm cm in pt sp em ex
factor	*number*
scale	*number*

You can use units and scaling factors. This rather complex example will illustrate \position.

```
\def\dicefive%
{\framed
  [width=42pt,height=42pt,offset=0pt]
{\setuppositioning
  [unit=pt,factor=12,xoffset=-11pt,
   yoffset=-8pt]%
\startpositioning
\position(1,1){$\bullet$}%
\position(1,3){$\bullet$}%
\position(2,2){$\bullet$}%
\position(3,1){$\bullet$}%
\position(3,3){$\bullet$}%
\stoppositioning}}
\placefigure{This is five.}{\dicefive}
```

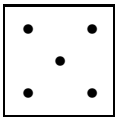


Figure 1 This is five.

18.12.11 Rotated text, figures and tables

You can rotate text (e.g. a table) or any object with:

```
\rotate[options]{something}
```

The first parameter specifies the rotation in degrees. This should be 90, 180 or 270. The curly braces contain the text or any object (perhaps a complete picture) you want to rotate.

Hasselt got its municipal rights in 1252. From that time on it had the right to use its own seal on official documents. This seal showed Holy Stephanus known as one of the first Christian martyrs and was the **patron** of Hasselt.

```
Hasselt got its municipal rights in
1252. From that time on it had the
\rotate[rotation=90]{right} to use
its own seal on official documents. This
seal showed Holy Stephanus known as one
of the first Christian martyrs and was
the \rotate[rotation=270]{\bf patron} of
Hasselt.
```

18.12.12 Carriage return

A new line can be forced with:

```
\crlf
```

The abbreviation stands for 'CarriageReturn LineFeed'. In case a number of lines should be followed by carriage return and line feed it is more convenient to use:

```
\startlines ... \stoplines
```

On a wooden panel in the town hall of Hasselt one can read:

Heimelijcken haet
eigen baet
jongen raet
Door diese drie wilt verstaen
is het Roomsche Rijck vergaen.

This little rhyme contains a warning for the magistrates of Hasselt: don't allow personal benefits or feelings influence your wisdom in decision making.

```
On a wooden panel in the town hall
of Hasselt one can read:
\startlines
Heimelijcken haet
eigen baet
jongen raet
Door diese drie wilt verstaen
is het Roomsche Rijck vergaen.
\stoplines
This little rhyme contains a warning
for the magistrates of Hasselt: don't
allow personal benefits or feelings
influence your wisdom in decision making.
```

In a few commands new lines are generated by `\\`. For example if you type `\inmargin{in the\\margin}` then the text will be divided over two lines.

18.12.13 Hyphenation

When writing multilingual texts you have to be aware of the fact that hyphenation may differ from one country to another. To activate a language you type:

```
\language[identifier]
```

The identifier can be `nl`, `fr`, `en`, `de`, `sp`, etc. To change from one language to another you can use the shorthand versions: `\nl`, `\en`, `\de`, `\fr`, `\sp`, etc.

If you want to know more about Hasselt you could probably best read *Uit de geschiedenis van Hasselt* by F. Peereboom.

```
If you want to know more about
Hasselt you could probably best
read {\nl \em Uit de geschiedenis
van Hasselt} by F.~Peereboom.
```

If a word is hyphenated incorrectly, you can mark valid hyphenation points yourself. This is done in the set-up area:

```
\hyphenation{hi-sto-ry hanze-pact}
```

Note that you only have to supply one version of a word. The hyphenation routine is not case sensitive.

18.12.14 Comments

All text between `\starttext` and `\stoptext` will be processed by CONTEXT.

However, sometimes you may want to exclude fragments. If you precede a piece of text with the percent sign % it will not be processed.

```
% In very big documents you can use the command input for
% different files.
%
% For example:
%
% \input hass01.tex % chapter 1 on Hasselt
% \input hass02.tex % chapter 2 on Hasselt
% \input hass03.tex % chapter 3 on Hasselt
```

When you delete the % before `\input` the three files will be processed. The comments describing the content of the files will not be processed.

18.13 Formulas

You can typeset numbered formulas with:

`\placeformula[`*ref,...*`]`

After this command a formula should follow, using the commands

`\startformula ... \stopformula`

Here are two examples:

Here's a formula:

$$y = x^2 \qquad (1)$$

And another one:

$$\int_0^1 x^2 dx \qquad (2)$$

```
Here's a formula:
\placeformula[formula:aformula]
\startformula
y=x^2
\stopformula
And another one:
\placeformula
\startformula
\int_0^1 x^2 dx
\stopformula
```

Between `\startformula` and `\stopformula` you are in 'math mode' so you can define any formula you want by using TEX commands. TEX is exceptionally good at typesetting mathematics, but you will need to learn some more commands. If you want to typeset math with TEX, we advise you to consult other documentation. See for instance:

- section 16.16 of this book
- *The TEXbook* by D. Knuth

- *The Beginners Book of TEX* by S. Levy and R. Seroul

Math mode can occur in two modes: in text mode and display mode. Mathematical expressions in text mode begin with $ and end with $.

The Hasselt community covers an area of $42{,}05\ \mathrm{km}^2$. Now if you consider a circular area of this size with the market place of Hasselt as the center point M you can calculate its diameter with $\frac{1}{4}\pi r^2$.

```
The Hasselt community covers an area of
42,05 \Square \Kilo \Meter. Now if you
consider a circular area of this size with
the market place of Hasselt as the center
point $M$ you can calculate its diameter
with ${{1}\over{4}} \pi r^2$.
```

The many {} in $\frac{1}{4}\pi r^2$ are essential for separating operations in the expression. If you omit the outer curly braces like this: `${1}\over{4} \pi r^2$`, you would get a non-desired result: $\frac{1}{4\pi r^2}$

Display mode however begins with $$ and ends with $$. You will get an expression that is displayed in the middle of a page.

Integrating is fun!

$$\int_0^1 x^2 dx$$

```
Integrating is fun!
$$
\int_0^1 x^2 dx
$$
```

If you compare $$ with the first examples you will notice that `\startformula` and `\stopformula` is sort of equivalent to $$.

The command `\placeformula` handles spacing around the formulas and the numbering of the formula. The parameter is used for cross referencing and for switching numbering off.

A cubic relation:

$$y = x^3 \qquad (1)$$

```
A cubic relation:
\placeformula[cubic]
\startformula
y=x^3
\stopformula
```

The label `cubic` is used for referring to this formula. Such a reference is made with `\in{formula}[cubic]`. If no numbering is required, you type:

```
\placeformula[-]
```

Numbering of formulas is set up with `\setupnumbering`. One way to set up numbering is `\setupnumbering[way=bychapter]`. This means that the chapter number precedes the formula number and numbering is reset at each new chapter.

Formulas can be set up with the command

`\setupformulas[`*options*`]`

The most important option is `location`, which can have a value `left` or `right`, the latter being the default.

18.14 Legends

The commands

```
\startlegend ... \stoplegend
```

are used to explain the meaning of symbols in formulas. The following example demonstrates this.

$$s = vt + \frac{1}{2}at^2 \qquad (1)$$

s = displacement m
v = velocity m/s
t = time s
a = acceleration m/s^2

```
\placeformula$$ s =
   v t + {{1}\over{2}} a t^2 $$
\startlegend
\leg s \\ displacement \\ m \\
\leg v \\ velocity \\ m/s \\
\leg t \\ time \\ s \\
\leg a \\ acceleration \\ m/s^2 \\
\stoplegend
```

The command `\leg` starts a new row; `\\` is a column separator. The last `\\` in each row is essential. The spaces are optional and have no effect on the outcome. However, these commands are easily mistyped and misread while checking. So take some time to make readable input.

The first column is used for symbols and is typeset in mathematical mode. The second column is used for explanations of the symbols and the third one for units and dimensions. Because of the necessity of sub- and superscripts the third column is in mathematical mode, but the text is displayed in the current text font.

Complementary to legends is a command that displays facts in a consistent way when formulas are used for calculations. Such a command looks like this:

velocity $v = 10$ m/s
acceleration $a = -3$ m/s^2
time $t = 3$ s

```
\startfact
\fact velocity \\ v \\ 10~m/s \\
\fact acceleration \\ a \\ - 3~m/s^2 \\
\fact time \\ t \\ 3~s \\
\stopfact
```

The two last columns are in mathematical mode, the last column displays text in the actual font. The `~` is necessary because spaces are ignored in mathematical mode.

18.15 Using color

Text can be set in color. You can set up colors with the command

`\setupcolors[`*options*`]`

The use of colors has to be activated as follows:

```
\setupcolors[state=start]
```

Now the following predefined colors are available:

```
red, green, blue, cyan, magenta, yellow, white, black, darkred,
middlered, lightred, darkgreen, middlegreen, lightgreen, darkblue,
middleblue, lightblue, darkcyan, middlecyan, darkmagenta,
middlemagenta darkyellow, middleyellow, darkgray, middlegray,
lightgray.
```

These colors can be used as parameter of the command

`\color[`*color*`]{`*text*`}`

Here is an example:

Hasselt is a very colorful town.

```
\startcolor[red]
{\tfb Hasselt is a very
\color[green]{colorful} town.}
\stopcolor
```

On a black and white printer colors will typically be rendered as shades of gray. On a color printer or in an electronic document you will see real colors. You can define your own colors with:

`\definecolor[`*identifier*`] [`*options*`]`

For example:

```
\definecolor[pink] [r=1.0,g=0.8,b=0.8]
\definecolor[darkgreen] [r=0,g=.5,b=0]
```

Now the colors ‘pink’ and ‘darkgreen’ are available.

18.15.1 Background of text

To emphasize a paragraph you can use backgrounds. A background is used with the commands:

`\startbackground ... \stopbackground`

An example will illustrate this:

Hasselt has produced a few well known persons. Only recently it turned out that Kilian van Rensselaer has played a prominent role in the foundation of the State of New York.

```
\setupbackground[background=color,
   color=pink,corner=round]
\startbackground
Hasselt has produced a few well known
persons. Only recently it turned out that
Kilian van Rensselaer has played a prominent
role in the foundation of the State of New
York.
\stopbackground
```

Backgrounds can span multiple pages. With the command

`\setupbackground[`*options*`]`

you can change the layout of the backgrounds. Here are a few of the options:

Option	Value
`height`	`fit broad` *dimension*
`width`	`fit broad` *dimension*
`background`	`color screen none`
`backgroundcolor`	*color*

18.15.2 Page backgrounds

The background of each page area can also be set. This command looks like this:

`\setupbackgrounds[`*vertical*`] [`*horizontal*`] [`*options*`]`

The first parameter can be `top`, `header`, `footer`, `bottom`, `page`, `leftpage` or `rightpage`. The second parameter can be `leftedge`, `leftmarging`, `text`, `rightmargin` or `rightedge`.

If you want to have backgrounds in the gray areas of the page layout of figure 18.1 (page 407), you should write:

```
\setupbackgrounds
  [header,text,footer]
  [leftmargin,text,rightmargin]
  [background=screen]
```

Suppose that you had defined a color named 'DeepPurple' that you want to use as background. You could write:

```
\setupbackgrounds
  [header,text,footer]
  [leftmargin,text,rightmargin]
  [background=color,
   color=DeepPurple]
```

18.16 Interactive mode in electronic documents

Nowadays documents can be made electronically available for consulting on a computer and displaying on a computer screen.

Interaction means that you can click on active areas and jump to the indicated areas. Such active areas are often called 'hyperlinks'. For example, if you consult a register, you can click on an (active) entry and you will jump to the corresponding page. Interaction relates to:

- chapter numbers in table of contents
- page numbers in registers
- page numbers, chapter numbers and figure numbers in internal references to pages, chapters, figures, etc.
- titles, page numbers, chapter numbers in external references to other interactive documents
- menus as navigation tools

Interactive documents in CONTEXT are actually PDF documents. PDF (Adobe's Portable Document Format) is an international standard for documents that can be printed on displayed on screen. If displayed on screen, hyperlinks (if any) will be active.

We assume that you will use PDFTEX for producing a PDF document directly. Alternatively you could produce DVI, convert it to PostScript (using DVIPS) and use Adobe Acrobat Distiller to convert the PostScript output to PDF. For viewing (interactive) PDF documents on screen you can use Adobe Acrobat Reader, Adobe Acrobat Exchange, or GSview.

CONTEXT is a very powerful system for producing electronic or interactive PDF documents. However, only a few standard features are described in this chapter. As the authors of CONTEXT are planning to make all CONTEXT related manuals electronically available (including sources), studying these is one of the options to become more acquainted with the possibilities of CONTEXT.

18.16.1 Interactive mode

The interactive mode is set up with:

```
\setupinteraction[options]
```

Here are a few of the options:

Option	Value
state	start stop
menu	on off
color	*name*
contrastcolor	*name*
style	normal bold slanted cap ...
author	*text*
title	*text*
subtitle	*text*
date	*text*

Here is an example:

```
\setupinteraction
  [state=start,
   color=green,
   style=bold]
```

The hyperlinks are now generated automatically and 'active' words are displayed in bold green.

The file size of the interactive document is considerably bigger than the file size of its paper pendant because hyperlinks consume space. You will also notice that processing time becomes longer. Therefore it is advisable to switch off interactive mode while your document is under construction.

18.16.2 Interaction within a document

Earlier you have seen how to make references with \in and \at. You may have wondered why you had to type \in{chapter}[chap:introduction]. There are two reasons for this. In the first place this prevents line breaks within the reference. In the second place the word 'chapter' and its number are typeset differently in interactive mode. This results in a larger 'clickable' area.

In interactive mode there is one other command that has little meaning in the paper variant.

\goto{*text*}[*ref*]

Here is an example:

```
In \goto{Hasselt}[fig:cityplan] all streets
are build in a circular way.
```

In the interactive document the word 'Hasselt' will be a hyperlink. It will enable you to jump to a map of Hasselt.

18.16.3 Interaction between documents

It is possible to link one document to another. First you have specify which other file you want to link. This can be done with the command

```
\useexternaldocument[identifier] [file] [title]
```

The first parameter must contain a logical name for the document. The second parameter contains the file name of the other document. The third parameter specifies the title of that document.

For referring to other documents you can use:

```
\from[ref]
```

Below is an example:

```
\useexternaldocument[hia][hasbook][Hasselt in August]
Most touristic attractions are described in \from[hia].
A description of the Eui-feest is found in \from[hia::euifeest].
A description of the \goto{Eui-feest}[hia::euifeest] is
found in \from[hia]. The eui-feest is described on
\at{page}[hia::euifeest] in \from[hia]. See for more information
\in{chapter}[hia::euifeest] in \from[hia].
```

The command `\useexternaldocument` should be used in the set-up area. The double `::` indicates a reference to an external document.

After processing your document (at least twice to get the references right) and `hasbook.tex`, you will have two PDF documents. The references above have the following meaning:

`\from[hia]`
: This will produce the active title ‘Hasselt in August’. It links to the first page of `hasbook.pdf`.

`\from[hia::euifeest]`
: This will produce an active title that links to the page where the chapter ‘Eui-feest’ begins.

`\goto{Eui-feest}[hia::euifeest]`
: This will produce the active word ‘Eui-feest’ and is linked to the page where chapter ‘Eui-feest’ begins.

`\at{page}[hia::euifeest]`
: This will produce an active word ‘page and page number’ that links to that page.

`\in{chapter}[hia::euifeest]`
: This will produce on active word ‘chapter and chapter number’ that links to that chapter.

As you can see the `::` separates the (logical) file name and the destination.

18.16.4 Menus

You can define navigation tools with:

```
\defineinteractionmenu[identifier] [location] [options]
```

The first parameter specifies a logical name that can be used to recall the menu. The second parameter specifies the location on the screen. A typical menu definition will look like this:

```
\setupcolors
  [state=start]
\setupinteraction
  [state=start,menu=on]
\defineinteractionmenu
  [mymenu]
  [right]
  [state=start,        align=middle,
   background=screen,  frame=on,
   width=\marginwidth, style=smallbold,
   color=]
\setupinteractionmenu
  [mymenu]
  [{Content[content]},
   {Index[index]},
   {\vfill},
   {Stop[ExitViewer]}]
```

This will produce a menu on the right hand side of every screen. The menu contains buttons that read *Content*, *Index* and *Stop*. Their functions are, respectively: jump to the table of contents, jump to the index and leave the viewer. The labels to obvious destinations such as table of contents and index are predefined. Other predefined destinations are `FirstPage`, `LastPage`, `NextPage` and `PreviousPage`.

The action `ExitViewer` is necessary to make the electronic document self-contained. Other predefined actions you can use are `PrintDocument`, `SearchDocument` and `PreviousJump`. The meaning of these actions should be obvious.

18.17 Defining your own elements

You can define a new command with the command

```
\definecommand[number of parameters] \command{content}
```

Without any further explanation this command is used in the following example. You may have a well illustrated document and you are tired of typing:

```
\placefigure
  [here,force]
  [fig:logical name]
  {Caption}
  {\externalfigure[filename]
      [width=5cm,frame=on]}
```

You could define your own command with a few variables such as:

- logical name
- caption
- file name

Your command definition could look like this:

```
\define[3]\myputfigure
  {\placefigure
     [here,force]
     [fig:#1]
     {#2}
     {\externalfigure[#3]
         [width=5cm,frame=on]}}
```

The parameter `[3]` indicates that the command expects three parameters `#1`, `#2` and `#3`. In the command call `\myputfigure` you have to supply these parameters using curly braces. Now you can use this definition like this:

```
\myputfigure{lion}{The Dutch lion is a sentry.}{hass13g}
```

Very sophisticated commands can be programmed but this is left to your own imagination and ingenuity.

In addition to defining commands you can also define `\start ... \stop` command pairs.

`\definestartstop`[*identifier*] [*options*]

For example:

```
\startnarrower
\definestartstop
  [stars]
  [commands={\inleft{\hbox to 5mm{%
     \leaders\hbox{$\star$}\hfill}}},
   before=\blank,after=\blank]
\startstars
{\em Hasselter Juffers} are sweet cookies
but the name is no coincidence. On July 21
in 1233 the {\em Zwartewaterklooster}
(Blackwater Monastery) was founded. The
monastery was meant for unmarried girls and
women belonging to the nobility of Hasselt.
These girls and women were called {\em
juffers}.
\stopstars
\stopnarrower
```

⋆⋆⋆

Hasselter Juffers are sweet cookies but the name is no coincidence. On July 21 in 1233 the *Zwartewaterklooster* (Blackwater Monastery) was founded. The monastery was meant for unmarried girls and women belonging to the nobility of Hasselt. These girls and women were called *juffers*.

18.17.1 Table of contents (lists)

A table of contents usually contains chapter numbers, chapter titles and page numbers. It can be extended with sections, subsections, etc. A table of contents is generated automatically with:

```
\placecontent
```

It depends on the location of this command in your document what kind of table of contents is produced. At the top of the document it will generate a list of chapters, sections, etc. But at the top of a chapter:

```
\chapter{Hasselt in Summer}
\placecontent
\section{Hasselt in July}
\section{Hasselt in August}
```

it will only produce a list of (sub) section titles with the corresponding section numbers and page numbers.

The command `\placecontent` is available after definition with:

```
\definecombinedlist[identifier] [section] [options]
```

This command and `\definelist` allows you to define your own lists.

```
\definelist[chapter]
\setuplist
  [chapter]
```

```
    [before=\blank,
     after=\blank,
     style=bold]
\definelist[paragraph]
\setuplist
    [section]
    [alternative=d]
```

Now there are two lists of chapters and sections. These will be combined in a table of contents with the command \definecombinedlist.

```
\definecombinedlist
    [content]
    [chapter,section]
    [alternative=b]
```

Now the commands \placecontent and

\completecontent

are available. The layout of lists can easily be varied using the keyword alternative. Several predefined layouts are available:

Alternative	Displays
a	number – title – page number
b	number – title – spaces – page number
c	number – title – dots – page number
d	number – title – page number (continuing)
e, f, g	reserved for interactive purposes

Lists are set up with:

\setuplist[*identifier*] [*options*]

and

\setupcombinedlist[*identifier*] [*options*]

If you want to change the layout of the generated table of contents you will have to remember that it is a list.

```
\setupcombinedlist
    [content]
    [alternative=c,
     aligntitle=no,
     width=2.5cm]
```

This will result in a somewhat different layout than the default one. Lists are placed with:

\placelist[*heading level*] *[options]*

So, if you want a table of contents you type:

```
\placecontent[level=section]
```

or

```
\completecontent[level=section]
```

Now only sections will be listed. You may need this option if your document that has very deeply nested headings and you don't want all levels to appear in the table of contents.

A long list or a long table of contents will fill more than one page. To force page breaks at specific points you can type:

```
\completecontent[2.2,8.5,12.3.3]
```

A page break will occur after section 2.2, section 8.5 and subsection 12.3.3.

18.17.2 Synonyms

In many texts people want to use specific words consistently throughout the document. To enforce consistency the command below is available.

\definesynonyms[*identifier*] [*plurale*] [*command*]

The first parameter contains the name of the synonym in singular, the second in plural. The third parameter contains a command. For example, the command `\abbreviation` can be defined like this:

```
\definesynonyms[abbreviation][abbreviations][\infull]
\setupsynonyms[style=cap]
```

Now the command `\abbreviation` is available and it can be used to specify your abbreviations:

```
\abbreviation{PDF}{Portable Document Format}
\abbreviation{AMS}{American Mathematical Society}
\abbreviation{NS}{Nederlandse Spoorwegen}
```

They can be used as follows:

```
\abbreviation{VVV}{Vereniging voor
   Vreemdelingen Verkeer}

The Dutch \VVV\ (\infull{VVV}) can
provide you with touristic
information on Hasselt.
```

The Dutch VVV (Vereniging voor Vreemdelingen Verkeer) can provide you with touristic information on Hasselt.

The list of synonyms or abbreviations should be defined in the set-up area of your document. You could also store this kind of information in an external file, and load this file (e.g., `abbreviations.tex`) like this:

```
\input abbreviations
```

If you want to list all used abbreviations in your document you should type:

```
\placelistofabbreviations
```

or

```
\completelistofabbreviations
```

A complete and sorted list of abbreviations and their meanings is produced. The layout of synonyms can be changed with:

```
\setupsynonyms[options]
```

You can use this command to make synonyms appear in, e.g., small caps, a bold font or a slanted font.

18.17.3 Floating blocks

A block in ConTeXt is a text element, for example a table or a figure that you can process in a special way. You have already seen the use of `\placefigure` and `\placetable`. These are both examples of floating blocks. You can define such blocks yourself with:

```
\definefloat[identifier] [plurale]
```

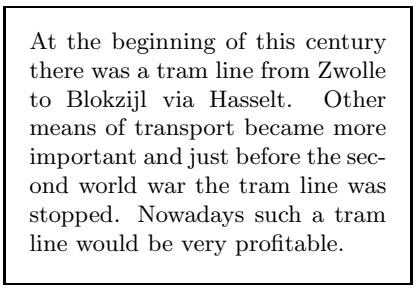
The parameters are used to specify the name in singular and plural form. For example:

```
\definefloat[intermezzo][intermezzi]
```

Now the following commands are available:

```
\placeintermezzo[][]{}{}
\startintermezzotext ... \stopintermezzotext
\placelistofintermezzi
\completelistofintermezzi
```

Here is an example:



Intermezzo 1 The tram line.

```
\setupfloats[location=middle]
\setupcaption[location=bottom,
  headstyle=bold]
\placeintermezzo{The tram line.}
\startframedtext
At the beginning of this century there
was a tram line from Zwolle to Blokzijl
via Hasselt. Other means of transport
became more important and just before
the second world war the tram line was
stopped. Nowadays such a tram line would
be very profitable.
\stopframedtext
```

18.17.4 Text blocks

Another type of block is a text block. A text block contains one or more paragraphs that you want to use more than once. You can define a text block with:

`\defineblock[`*identifier*`]`

Here is an example:

```
\defineblock[dutch]
```

After defining the text block the following commands are available:

```
\begindutch ... \enddutch
```

Text blocks are manipulated with:

`\hideblocks[`*identifier*`]` *`[identifier]`*

and

`\useblocks[`*identifier*`]` *`[identifier]`*

An example shows some of the possibilities of text blocks.

```
\defineblock[dutch,english]
\hideblocks[dutch,english]

\beginenglish[dedemsvaart-e]
After 1810 the Dedemsvaart brought some
prosperity to Hasselt. All ships went
through the canals of Hasselt and the
shops on both sides of the canals
prospered.
\endenglish

\begindutch[dedemsvaart-d]
Sinds 1810 veroorzaakte de Dedemsvaart
enige welvaart in Hasselt. Alle schepen
voeren door de grachten en de winkels
aan weerszijden van de gracht floreerden.
\enddutch

\useblocks[english][dedemsvaart-e]
```

After 1810 the Dedemsvaart brought some prosperity to Hasselt. All ships went through the canals of Hasselt and the shops on both sides of the canals prospered.

If you define these blocks consequently you easily make a bilingual manual. For that purpose it is also possible to store text blocks in external files like this:

```
\setupblocks
  [dutchman]
  [file=manual-dutch]
```

The Dutch text blocks are stored in `manual-dutch.tex`. The text fragments can be called by the logical name ‘dutchman’.

18.18 Using modules

For reasons of efficiency it was decided to implement some functionality of ConTeXt by means of external modules. Loading is done in the set up area of your input file and done by means of:

```
\usemodule[name]
```

Currently the following modules are available:

chemic for typesetting chemical structures

units for using SI units

pictex for drawing pictures (is used in conjunction with module chemic)

18.19 User specifications

When CONTEXT is run a number of predefined parameters are loaded. These parameters are set up in the file `cont-sys.tex`. Users can define their own preferences (house style) in this file. In this file you could write the following command:

```
\usespecials[reset,ps,tr,pdf]
```

or more 'hard wired':

```
\setupoutput[dvips,acrobat]
```

Now CONTEXT will know that DVIPS is your DVI driver for producing PostScript output. CONTEXT needs to know this in order to generate correct instructions in the DVI file. Likewise you could write:

```
\setupoutput[pdftex]
```

Now CONTEXT will assume that you use PDFTEX to generate PDF output. It will automatically set-up the basics for this kind of output.

18.20 Processing steps

During processing CONTEXT writes information in the file `myfile.tui`. This information is used in the next pass. Part of this information is processed by the program TEXUTIL.

Information on registers and lists are written in the file `myfile.tuo`. The information in this file is filtered and used (when necessary) by CONTEXT.

TEXUTIL is available as a computer platform independent Perl script called `texutil.pl`. So you must have the Perl program installed to run it (cdrom). You can start the script as follows:

≡ perl textutil.pl *options*

If no options are given, TEXUTIL will show a list of options. In order to process `myfile.tui` file you should enter:

▷ `perl texutil.pl --references myfile`

It is a good idea to run TEXUTIL automatically after each CONTEXT run. A simple batch file containing just a few lines can do the trick:

```
@echo off
context.exe %1
perl.exe texutil.pl --references %1
```

Another useful application of TEXUTIL is to gather information about graphic files. The information will be written to a file called `texutil.tuf` and consists (amongst others) of the graphic type (EPS, TIF, PNG, PDF, etc.), the width and the height of the figure.

```
▷  perl texutil.pl --figures *.eps
```

An even more sophisticated way to run CONTEXT jobs is implemented in H. Hagen's `texexec.pl` script. This script will automatically run a CONTEXT job as many times as necessary to resolve all references, optimize typesetting, etc.

18.21 Auxiliary files

CONTEXT will produce some auxiliary files during processing. The following file types may appear on your working directory. The column 'remove?' indicates whether these files can be removed without affecting the production process. But remember that even if you accidentally remove files that you shouldn't, it only takes one or more CONTEXT runs to regenerate everything.

File	Content	Written by	Read by	Remove?
`.tui`	input information	CONTEXT	TEXUTIL	yes
`.tuo`	output information	TEXUTIL	CONTEXT	no
`.tub`	block information	CONTEXT	CONTEXT	no
`.tmp`	buffer information	CONTEXT	CONTEXT	yes
`texutil.tuf`	figure information	TEXUTIL	CONTEXT	no

Appendix A

File types

File types are given in `roman typewriter` font if human-readable or in *`slanted typewriter`* font if machine-readable ('binary'). By files written by 'humans' we mean file not generated from other sources, but written by a user or programmer.

Table A.1: File types for TEX and friends

	Written by	Read by	Explanation
`aliases`	humans	Web2c programs	file name aliases
`.afm`	font design program	PS2PK, AFM2TFM	Adobe font metrics
`.base`	iniMF	METAFONT	MF format file
`.bib`	humans	BIBTEX	bibliography
`.bbl`	BIBTEX	TEX	bibliographic references
`.blg`	BIBTEX	humans	log file
`.bst`	humans	BIBTEX	BIBTEX style
`.ch`	humans	Tangle	change file
`.cnf`	humans	Web2c programs	configuration file
`.cfg`	humans	DVIPS, WINDVI LATEX	configuration file
`.dvi`	TEX	DVI driver	TEX output
`.enc`	humans	DVI driver, PDFTEX	font encoding definition
`.eps`	DVIPS, many graphic programs	TEX, many graphic programs, DVI driver	Encapsulated PostScript

... continued on next page

Table A.1: (continued)

	Written by	Read by	Explanation
.fmt	iniTEX	TEX	format file
*.*gf*	METAFONT	GFtoPK	graphic font
.hyp	humans, Patgen	iniTEX	hyphenation patterns
.idx	TEX	MakeIndex	raw index
.ind	MakeIndex	TEX	sorted index
.ilg	MakeIndex	humans	log file
.ini	Windows programs	Windows programs	initialization/ configuration
.ist	humans	MakeIndex	index style
.log	many programs	humans	log file
ls-R	MKTEXLSR, ls	Web2c programs	file index
.map	humans	Web2c programs	font names
.mem	iniMpost	METAPOST	format file
.mf	humans	METAFONT	font description
.mft	humans	METAFONT	METAFONT style file
.mp	humans	METAPOST	picture description
.p	Tangle	Pascal compiler	Pascal program
.pdf	PDFTEX, DVI2PDF, Ghostscript, Acrobat Distiller	Ghostscript, Acrobat Reader	Adobe Portable Document Format
.pfa	font design program	DVI driver, Ghostscript	PostScript font (ASCII)
.pfb	font design program	DVI driver, Ghostscript, GSFtoPK	PostScript font (binary)
.pk	GFtoPK, PS2PK, GSFtoPK	DVI driver	bitmap font
.pl	TFtoPL, humans	PLtoTF	font property list
.pool	Tangle	TEX, METAFONT, METAPOST	messages of TEX, METAFONT or METAPOST
.pro	humans	DVIPS	PostScript profile header
.ps	DVIPS, humans	Ghostscript, Acrobat Distiller,	PostScript output

... continued on next page

Table A.1: (continued)

	Written by	Read by	Explanation
		PS printer	
`.tex`	humans, Weave	TEX	document or macro definitions
.tfm	PLtoTF, METAFONT, AFM2TFM	TEX, DVI driver, TFtoPL	TEX font metrics
.vf	VPtoVF, AFM2TFM	DVI driver, VFtoVP	virtual font
`.vpl`	humans, VFtoVP	VPtoVF	virtual font property list
`.web`	humans	Weave, Tangle	TEX sources

Table A.2: File types not specific to TEX and friends

	Written by	Read by	Explanation
`.bat`	humans	OS*	batch file
`.btm`	humans	4DOS	advanced batch file
.exe	compiler	OS*	executable program
.com	compiler	OS*	executable program
.dll	compiler	executables	dynamic link library
`.pl`	humans	Perl	Perl script
`.c`	humans	C compiler	C programs
`.h`	humans	C compiler	C header files
.tar	Unix TAR	Unix TAR	Unix 'Tape' Archive
.gz	Gzip	Gzip	'gzip' compression
.tgz	TAR/Gzip	TAR/Gzip	'tar'ed and 'gzip'ed
.Z	Unix Compress	Unix Compress	compression
.zip	Zip	Unzip	Zip compression

* = operating system

Table A.3: METAFONT modes defined in modes.mf

Mode name	Description
agfafzz	AGFA 400PS (406 dpi)
agfatfzz	AGFA P3400PS (400 dpi)
amiga	Commodore Amiga (100 dpi)
aps	Autologic APS-Micro5 (723 dpi)
apssixhi	Autologic APS-Micro6 (1016 dpi)

... continued on next page

Table A.3: (continued)

Mode name	Description
atariezf	Atari ST SLM 804 printer (300 dpi)
atarinf	Atari previewer (95 dpi)
atarins	Atari previewer (96 dpi)
atariotf	Atari ST SM 124 screen (101 dpi)
bitgraph	BBN Bitgraph (118 dpi)
bjtenex	Canon BubbleJet 10ex (360 dpi)
bjtzzex	Canon BubbleJet 200ex (360 dpi)
bjtzzs	Canon BubbleJet 200 (720 x 360 dpi)
bjtzzl	BubbleJet 200 landscape (360 x 720 dpi)
boise	HP 2680A (180 dpi)
canonbjc	Canon BJC-600 (360 dpi)
canonex	LaserWriter Pro 630 (600 dpi)
canonlbp	Symbolics LGP-10 (240 dpi)
cg	Compugraphic 8600 (1301 x 1569 dpi)
cgl	Compugraphic 8600 landscape (1569 x 1302 dpi)
cgnszz	Compugraphic 9600 (1200 dpi)
crs	Alphatype CRS (5333 dpi)
cx	Canon CX, SX, LBP-LX (300 dpi)
datadisc	DataDisc (70 dpi)
newdd	DataDisc (70 x 93 dpi)
declarge	DEC 19-inch, 1280 x 1024 (100 dpi)
decsmall	DEC 17-inch, 1024 x 768 (82 dpi)
deskjet	HP DeskJet 500 (300 dpi)
docutech	Xerox 8790 or 4045 (600 dpi)
dover	Xerox Dover (384 dpi)
eighthre	EightThree (83 dpi)
epscszz	Epson Stylus Color 600 (720 dpi)
epsdrft	Epson (120 x 72 dpi)
epsdrftl	Epson (72 x 120 dpi)
epsfast	Epson (60 x 72 dpi)
epsfastl	Epson (72 x 60 dpi)
epson	9-pin Epson MX/FX (240 x 216 dpi)
epsonl	9-pin Epson MX/FX landscape (216 x 240 dpi)
epsonact	Epson Action Laser 1500 (300 dpi)
epsonlo	Epson (120 x 216 dpi)
epsonlol	Epson landscape (216 x 120 dpi)
epsonsq	Epson SQ 870 (360 dpi)
epstypro	Epson Stylus Pro (360 dpi)
epstyplo	Epson Stylus Pro (180 dpi)
epstypmd	Epson Stylus Pro (720 x 360 dpi)

... continued on next page

Table A.3: (continued)

Mode name	Description
esphi	Epson Stylus Pro (720 dpi)
epstylus	Epson Stylus (360 dpi)
fourfour	FourFour (44 dpi)
gtfax	G3fax (204 x 196 dpi)
gtfaxl	G3fax landscape (196 x 204 dpi)
gtfaxlo	G3fax (204 x 98 dpi)
gtfaxlol	G3fax landscape (98 x 204 dpi)
highfax	G3fax (200 dpi)
hprugged	HP RuggedWriter 480 (180 dpi)
ibm_a	IBM 38xx (240 dpi)
ibmd	IBM 38xx (240 dpi)
ibmega	IBM EGA monitor (96 x 81 dpi)
ibmegal	IBM EGA monitor landscape (81 x 96 dpi)
ibmfzon	IBM 4019 (300 dpi)
ibmfztn	IBM 4029-30-39, 4250 (600 dpi)
ibmpp	IBM ProPrinter (240 x 216 dpi)
ibmppl	IBM ProPrinter (216 x 240 dpi)
ibmsoff	IBM 6154 display (118 dpi)
sherpa	IBM 6670 (Sherpa) (240 dpi)
ibmteot	IBM 3812 (240 dpi)
ibmtetz	IBM 3820 (240 dpi)
ibmtont	IBM 3193 screen (100 dpi)
ibmtosn	IBM 3179 screen (87 x 65 dpi)
ibmtosnl	IBM 3179 screen landscape (65 x 87 dpi)
ibmvga	IBM VGA monitor (110 dpi)
ibx	Chelgraph IBX (9600 dpi)
itoh	CItoh 8510A (160 x 144 dpi)
itohl	CItoh 8510A landscape (144 x 160 dpi)
itohtoz	CItoh 310 (240 x 144 dpi)
itohtozl	CItoh 310 landscape (144 x 240 dpi)
iw	Apple ImageWriter (144 dpi)
jetiiisi	HP Laser Jet IIISi (300 dpi)
lasf	DEC LA75 (144 dpi)
lexmarkr	Lexmark Optra R 4049 (1200 dpi)
lexmarks	Lexmark Optra S 1250/1650/2450 (1200 dpi)
lexmarku	Lexmark Optra R+ 4049 (600 dpi)
linolo	Linotype Linotronic [13]00 (635 dpi)
linolttz	Linotronic L-300 with RIP-50 (3386 dpi)
linoone	Linotronic [13]00 (1270 dpi)
linotzzh	Linotype Linotronic 300 (2540 dpi)

... continued on next page

Table A.3: (continued)

Mode name	Description
ljfive	HP LaserJet 5 (600 dpi)
ljfivemp	HP LaserJet 5MP (600 dpi)
ljfour	HP LaserJet 4 (600 dpi)
ljfzzz	LaserJet 4000N, ProRes mode (1200 dpi)
ljfzzzfr	HP LaserJet 4000 FastRes (600 dpi)
ljlo	HP LaserJet (150 dpi)
lmaster	LaserMaster (1000 dpi)
lnotr	DEC LN03R Scriptprinter (300 dpi)
lnzo	DEC LN01 (300 dpi)
lpstz	DEC lps20 (300 dpi)
lqlores	Epson LQ-500 (180 dpi)
lqmed	Epson LQ-500 (360 x 180 dpi)
lqmedl	Epson LQ-500 landscape (180 x 360 dpi)
lview	Sigma L-View monitor (118 x 109 dpi)
lwpro	Apple LaserWriterPro 810 (800 dpi)
macmag	Mac screens at magstep 1 (86 dpi)
mactrue	Mac screen (72 dpi)
ncd	NCD 19-inch (95 dpi)
nec	NEC (180 dpi)
nechi	NEC-P6 (360 dpi)
neclm	NEC PC-PR406LM (320 dpi)
nectzo	NEC PC-PR201 series (160 dpi)
nexthi	NeXT Newgen (400 dpi)
nextscrn	NeXT monitor (100 dpi)
nineone	NineOne (91 x 91) (91 dpi)
nullmode	TFM files only (101 dpi)
onetz	OneTwoZero (120/120) (120 dpi)
ocessfz	OCE 6750-PS (508 dpi)
okidata	Okidata (240 x 288 dpi)
okidatal	Okidata landscape (288 x 240 dpi)
okifte	Okidata 410e in 600 dpi mode (600 dpi)
pcscreen	also, e.g., high-resolution Suns (118 dpi)
pcprevw	PC screen preview (118 dpi)
phaser	Tektronix Phaser PXi (300 dpi)
phaserfs	Tektronix Phaser 560 (1200 dpi)
phasertf	Tektronix Phaser 350 (600 x 300 dpi)
pixpt	one pixel per point (72.27 dpi)
prntware	Printware 720IQ (1200 dpi)
qms	QMS (Xerox engine) (300 dpi)
qmsostf	QMS 1725 (600 dpi)

. . . continued on next page

Table A.3: (continued)

Mode name	Description
qmsoszz	QMS 1700 (600 dpi)
qmstftf	QMS 2425 (1200 dpi)
ricoh	e.g., TI Omnilaser (300 dpi)
ricoha	e.g., IBM 4216 (300 dpi)
ricohlp	e.g., DEC LN03 (300 dpi)
ricohsp	Ricoh sp10ps/lp7200-ux (600 dpi)
sparcptr	Sun SPARCprinter (400 dpi)
starnlt	Star NL-10 (240 x 216 dpi)
starnltl	Star NL-10 landscape (216 x 240 dpi)
stylewri	Apple StyleWriter (360 dpi)
sun	Sun and BBN Bitgraph (85 dpi)
supre	Ultre*setter (2400 dpi)
toshiba	Toshiba 13XX, EpsonLQ (180 dpi)
ultre	Ultre*setter (1200 dpi)
vs	VAXstation monitor (78 dpi)
vtftzz	Varityper 4200 B-P (1800 dpi)
vtftzzhi	Varityper 4300P (2400 dpi)
vtftzzlo	Varityper 4300P (1200 dpi)
vtfzszw	Varitype 5060W, APS 6 (600 dpi)
vtszz	Varityper Laser 600 (600 dpi)
xrxesnz	Xerox 8790 or 4045 (300 dpi)
xrxfzfz	Xerox 4050/4075/4090/4700 (300 dpi)
xrxnszz	Xerox 9700 (300 dpi)
xrxtszz	Xerox 3700 (300 dpi)

Table A.4: Types of graphics files

Extension	Meaning
`.avs`	AVS X image file
`.bmp`	Microsoft Windows bitmap image
`.bmp24`	Microsoft Windows 24-bit bitmap
`.cmyk`	Raw cyan, magenta, yellow, and black bytes
`.dcx`	ZSoft IBM PC multi-page Paintbrush
`.dib`	Microsoft Windows bitmap image
`.eps`	Adobe Encapsulated PostScript
`.eps2`	Adobe Level II Encapsulated PS
`.epsf`	Adobe Encapsulated PostScript
`.epsi`	Adobe EPS Interchange format
`.fax`	Fax Group 3

... continued on next page

Table A.4: (continued)

Extension	Meaning
.fig	TransFig image format
.fits	Flexible Image Transport System
.fpx	FlashPix Format
.gif	CompuServe graphics interchange format
.gif87	Graphics interchange format (version 87a)
.gradation	Gradual passing from one shade to another
.granite	Granite texture
.gray	Raw gray bytes
.hdf	Hierarchical Data Format
.jbig	Joint Bi-level Image experts Group format
.jpeg, .jpg	Joint Photographic Experts Group
.map	Colormap intensities and indices
.matte	Raw matte bytes
.miff	Magick image file format
.mono	Bi-level bitmap in LSB
.mpeg, .mpg	Motion Picture Experts Group
.mtv	MTV Raytracing image
.netscape	Netscape 216 color cube
.pbm	Portable bitmap format
.pcd	Photo CD
.pcx	ZSoft IBM PC Paintbrush
.pdf	Portable Document Format
.pgm	Portable graymap format
.pict	Apple Macintosh QuickDraw/PICT
.plasma	Plasma fractal image
.png	Portable Network Graphics
.pnm	Portable anymap
.ppm	Portable pixmap format
.ps	Adobe PostScript
.ps2	Adobe Level II PostScript
.rad	Radiance image
.rgb	Raw red, green, and blue bytes
.rgba	Raw red, green, blue, and matte bytes
.rla	Alias/Wavefront image
.rle	Utah Run length encoded image
.sgi	Irix RGB image file
.sun	SUN Rasterfile
.text, .txt	Raw text file
.tga	Truevision Targa image
.tiff, .tif	Tagged Image File Format

... continued on next page

Table A.4: (continued)

Extension	Meaning
.tiff24	24-bit Tagged Image File Format
.uyvy	16bit/pixel interleaved YUV
.tile	Tile image with a texture
.uil	X-Motif UIL table
.vid	Visual Image Directory
.viff	Khoros Visualization image file
.xbm	X Windows system bitmap
.xpm	X Windows system pixmap file
.xwd	X Windows system window dump file
.yuv	CCIR 601 4:1:1 file

Appendix B

Flowcharts

In the flowcharts below we have tried to give you insight into the relations between programs and files that exist in a typical TEX system. Files are indicated by rectangular boxes in which the file name extension is given. A description of the function of each file type/extension is given in appendix A.

Files typeset in *upright* typewriter font are human-readable (ASCII); files typeset in *slanted* typewriter font are machine-readable ('binary').

Programs (often also called 'executables' or 'binaries', which can be quite confusing) are indicated by oval boxes.

The arrows indicate what output a program produces. Dashed sections are explained in detail in other charts.

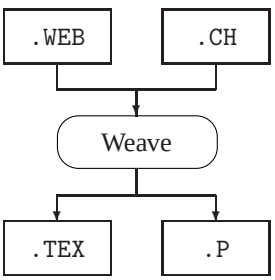


Figure B.1: Weave: from Web sources to program and documentation

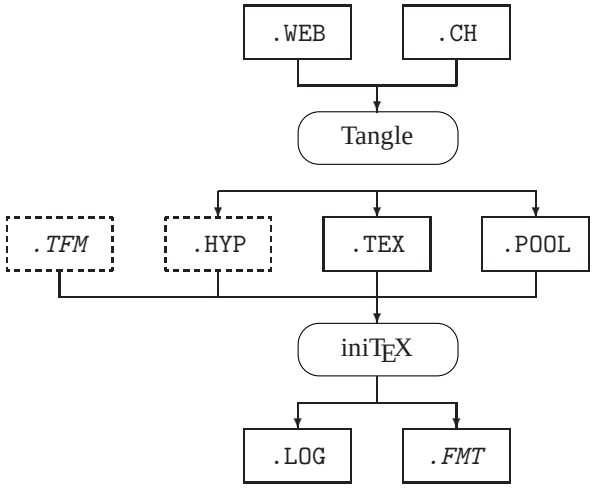


Figure B.2: Tangle: from Web sources to iniTEX

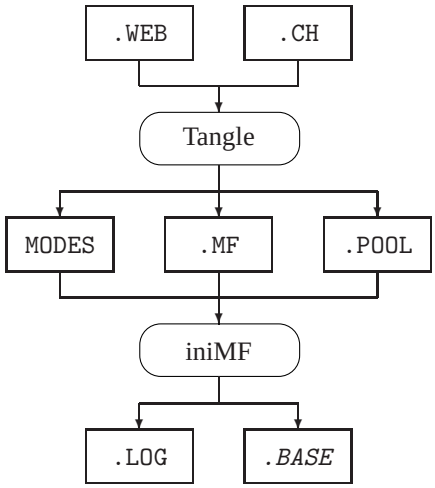


Figure B.3: Tangle: from Web sources to iniMF

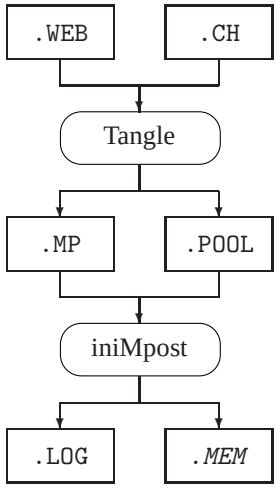


Figure B.4: Tangle: from Web sources to program iniMpost

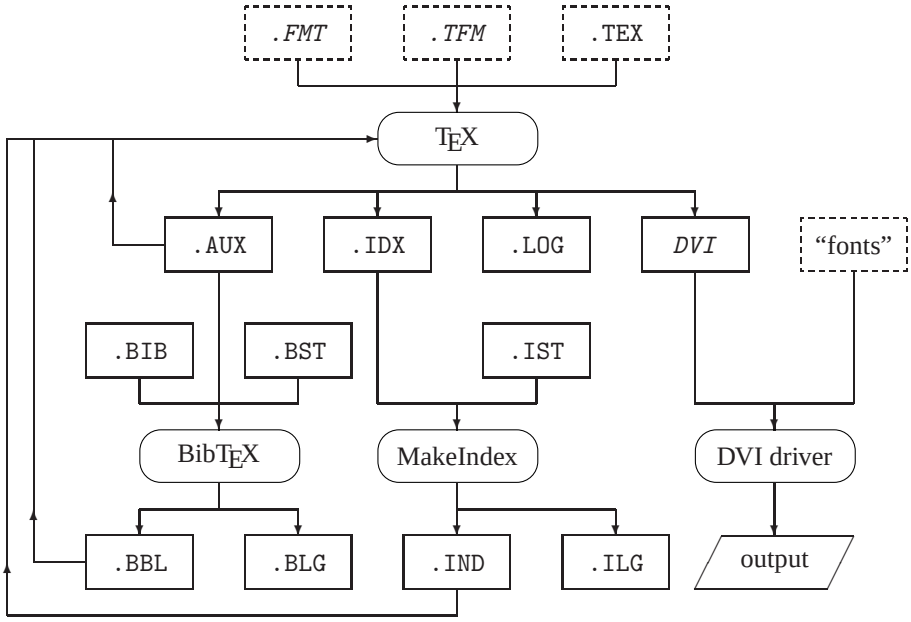


Figure B.5: Files used in a typical TEX run
(adapted from Schrod, 1991 (cdrom)) 'fonts' can be `.pk`, `.pfb`, `.tfm` and/or `.vf` files.

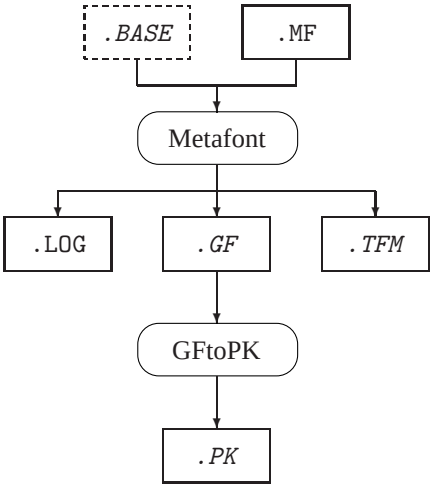


Figure B.6: From METAFONT to bitmapped fonts

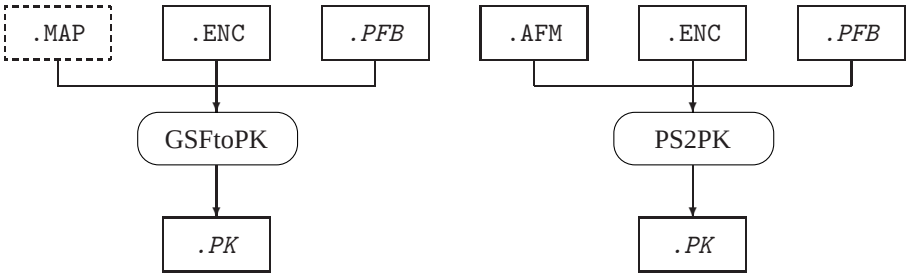


Figure B.7: From PostScript Type 1 fonts to bitmapped fonts

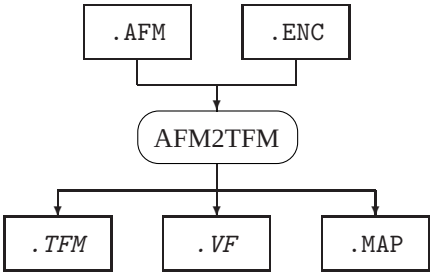


Figure B.8: From PostScript font metrics to TEX font metrics

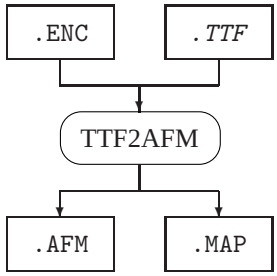


Figure B.9: From TrueType font to TEX PostScript font metrics

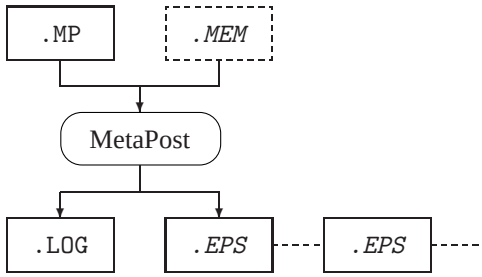


Figure B.10: From METAPOST to EPS pictures

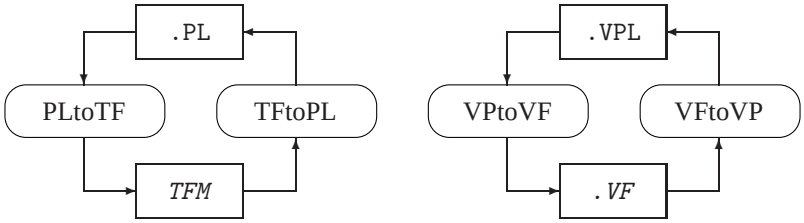


Figure B.11: (Virtual) font property lists to (virtual) font metrics

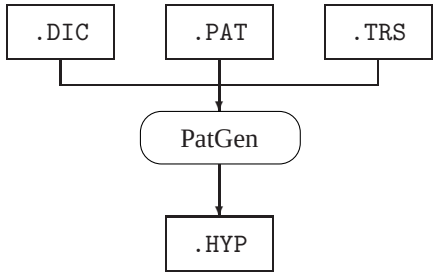


Figure B.12: Hyphenation pattern generation

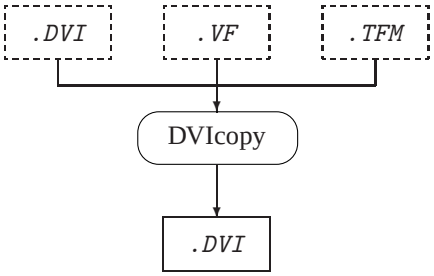


Figure B.13: Devirtualizing a DVI file

Appendix C

Overview of software

In this appendix we will list software that is used by 4TEX directly of indirectly, or which you may find on the 4allTEX CDROM and use for your own purposes. In addition to the names of the authors we will indicate what kind of license applies to the software. In this context software means executables, batch files, Perl scripts, dynamic link libraries, help files, macros, fonts, etc.

Naturally TEX dialects such as Plain TEX, LATEX and CONTEXT should also be regarded as software. Many authors wrote packages that run on top of these dialects. However, the list of these packages is far too long to print here. On the 4allTEX CDROM you can find lists of free and non-free packages in the directory `\texmf\lists`.

In case a sofware product is not 'free' you may decide not to use it, or to get a license for it. Note that it may not always be easy to trace which software you are actually using. E.g., a LATEX package may use another non-free LATEX package, or a program may run silently in the background.

We recommend that you check the license agreement of *any* software product you use. Licenses and license fees of different software products differ enormously, depending e.g. on the situation in which you use it. Some sofware may run perfectly without ever warning you that you are in violation of its license agreement. However, this is *not* a valid excuse for not paying your license fee.

The 4allTEX CDROM comes with pre-paid licenses for the following programs:

- The editor program MED: you can use the version supplied on the CDROM without any limitations. You may *not* hand the program to others, and you are not entitled to support or to updates.
- The scripting tool 4DOS: you can use the version supplied on the CDROM for running any 4DOS batch file from within 4TEX.

In all cases you are free to obtain a full license from the author(s) if you find the current license agreement too limitated, or if you simply want to support good products.

The kind of license that applies to each software product is categorized as follows:

0	Absolutely free
1	GNU General Public License or very similar. See the file `\license.gnu` on the 4allTEX CDROM
2	Aladdin Free Publice License. See the file `\bin\win32\LICENCE` on the 4allTEX CDROM
3	Free for personal use, not free for professional use
4	Free for non-commercial or educational use
5	Shareware. See individual license agreement
6	Shareware for which the 4allTEX CDROM contains a licensed version
?	Unknown

Table C.1: Software overview

Software	**Author(s)**	**License**
4dosrt	R. Conn, JP Software Inc.	6
4ntrt	R. Conn, JP Software Inc.	6
4project	W. Dol, E. Frambach	4
4spell	W. Dol, E. Frambach	4
4tex	W. Dol, E. Frambach	4
abc2mtex	C. Walshaw	0
access	K. Berry, O. Weber	1
acrord32	Adobe Inc.	0
afm2tfm	T. Rokicki	1
animate	E. du Pont de Nemours	0
asc2tex	T. Götz	0
asc2wp	G. Johannsen	0
bdftops	Aladdin Enterprises	2
bg5conv	W. Lemberg	1
bibedit	J. Björnerstedt	0
bibtex	O. Patashnik	1
catdoc	V. Wagner	1
cef5conv	W. Lemberg	1
cefconv	W. Lemberg	1
cefsconv	W. Lemberg	1
cep	B. Jackowski, P. Pianowski, P. Strzelczyk	0
chi2ltx	M. Gomulinski	0
chi2tex2	I. Zakharevich	0
chktex	J. Berger	1
contextnl.el	B. de Boer	?

... continued on next page

Table C.1: (continued)

Program(s)	Author(s)	License
convert	E. du Pont de Nemours	0
convdw	CrossCourt Systems	5
cop	B. Jackowski, P. Pianowski, P. Strzelczyk	0
cstocs2	J. Tkadlec	?
detex	D. Trinkle	0
disdvi	?	?
dmp	J. Hobby	1
dt2dv	G. Tobin	1
dv2dt	G. Tobin	1
dvi2dvi	P. Sawatzki	?
dvi2tty	M. Mol	?
dvibook	A. Duggan	1
dviconcat	C. Torek	?
dvicopy	P. Breitenlohner	1
dvidvi	Radical Eye, M. Kohm	?
dvihp	G. Neumann	1
dvilj	G. Neumann	1
dvilj2p	G. Neumann	1
dvilj4	G. Neumann	1
dvilj4l	G. Neumann	1
dvilj6	G. Neumann	1
dvipdfm	M. Wicks	1
dvips	T. Rokicki	1
dviselect	C. Torek	1
dvitomp	J. Hobby	1
dvitype	D. Knuth	1
e32	SemWare Corp.	5
eps2pdf	S. Rahtz	0
epsffit	A. Duggan	1
epstool	R. Lang	2
etex	NTS-team, P. Breitenlohner	1
extractres	A. Duggan	1
filt	L.G. Institut Fourier	?
fixdlsrps	A. Duggan	1
fixfmps	A. Duggan	1
fixmacps	A. Duggan	1
fixpsditps	A. Duggan	1
fixpspps	A. Duggan	1
fixscribeps	A. Duggan	1

... continued on next page

Table C.1: (continued)

Program(s)	Author(s)	License
fixtpps	A. Duggan	1
fixwfwps	A. Duggan	1
fixwpps	A. Duggan	1
fixwwps	A. Duggan	1
fmtutil	K. Berry, O. Weber	1
gawk	Free Software Foundation	1
gftodvi	D. Knuth	1
gftopk	T. Rokicki	1
gftype	D. Fuchs	1
go32	D. Delorie	1
grep	Free Software Foundation	1
gsftopk	P. Vojta	1
gsview32	R. Lang	2
gswin32	Aladdin Enterprises	2
gswin32c	Aladdin Enterprises	2
gvwgs32	R. Lang	2
gvwgs32	R. Lang	2
hbf2gf	W. Lemberg	1
html2latex	N. Norkington	?
i_view32	I. Skiljan	0
identify	E. du Pont de Nemours	0
includeres	A. Duggan	1
inimf	D. Knuth	1
inimpost	J. Hobby	1
initex	D. Knuth	1
jpeg2ps	T. Merz	?
kill	L. Kahn	?
kpsestat	K. Berry, O. Weber	1
kpsewhich	K. Berry, O. Weber	1
lacheck	K. Krab Thorup	?
latex2e.hlp	T. Martinsen	0
latex2eps	I. Podlubny	0
latex2rtf	D. Taupin	1
latexcad	J. Leis	4
latexmac	J. Aguirregabiria	3
libpng	G. Randers-Pehrson, E. Dilger, G. Schalnat	1
lnexe	K. Berry, O. Weber	1
ltxwizrd	P. Wiladt	3
mag	P. Tutelaers	0

... continued on next page

Table C.1: (continued)

Program(s)	Author(s)	License
makeindex	J. Hobby	1
makempx	J. Hobby	1
mdraw	Mayura Software	5
med	M. Pfersdorff	6
mf	D. Knuth	1
mft	D. Knuth	1
midi2tex	H. Kuykens	?
mktex	K. Berry, O. Weber	1
mktexdir	K. Berry, O. Weber	1
mktexlsr	K. Berry, O. Weber	1
mktexmf	K. Berry, O. Weber	1
mktexnam	K. Berry, O. Weber	1
mktexpk	K. Berry, O. Weber	1
mktextfm	K. Berry, O. Weber	1
mktexupd	K. Berry, O. Weber	1
mogrify	E. du Pont de Nemours	0
mpost	J. Hobby	1
mpto	J. Hobby	1
musixflx	D. Taupin, ?	1
newer	J. Hobby	1
notepro	E. Fookes	5
notetab	E. Fookes	0
omega	J. Plaice, Y. Haralambous	1
patgen	F. Liang, P. Breitenlohner	1
pcwtex	J. Breen	0
pdf2ps	Aladdin Enterprises	2
pdfetex	Hàn Thế Thành, NTS-team, P. Breitenlohner	1
pdfimages	D. Noonburg	?
pdfinfo	D. Noonburg	?
pdftex	Hàn Thế Thành	1
pdftops	D. Noonburg	?
pdftotext	D. Noonburg	?
perl	L. Wall	1
pfb2pfa	P. Tutelaers	1
pfe32	A. Phillips	5
pftogsf	Aladdin Enterprises	2
pk2bm	P. Tutelaers	1
pktogf	T. Rokicki	1
pktype	T. Rokicki	1

… continued on next page

Table C.1: (continued)

Program(s)	Author(s)	License
pltotf	D. Knuth	1
pooltype	D. Knuth	1
poster	J. van Eijndhoven	
prfile32	P. Lerup	0
ps	L. Kahn	?
ps2ai	Aladdin Enterprises	2
ps2ascii	Aladdin Enterprises	2
ps2epsi	Aladdin Enterprises	2
ps2gif	Aladdin Enterprises	2
ps2pdf	Aladdin Enterprises	2
ps2pk	P. Tutelaers	1
ps2ps	Aladdin Enterprises	2
ps_conv	B. Jackowski, P. Pianowski	0
psbook	A. Duggan	1
psnup	A. Duggan	1
psp	Jasc Software, Inc.	5
psresize	A. Duggan	1
psselect	A. Duggan	1
pstops	A. Duggan	1
quickfinger32	D. Weekly	0
rtflatex	D. Taupin	1
sjis5conv	W. Lemberg	1
t1ascii	I. Hetherington, E. Kohler	1
t1asm	I. Hetherington, E. Kohler	1
t1binary	I. Hetherington, E. Kohler	1
t1disasm	I. Hetherington, E. Kohler	1
t1unmac	I. Hetherington, E. Kohler	1
tangle	D. Knuth	1
tex	D. Knuth	1
tex2rtf	J. Smart	0
tex4ht	E. Gurari	1
texexec	H. Hagen	1
texi2html	L. Cons	0
texutil	H. Hagen	1
tftopl	D. Knuth	1
tgrind	?	?
thumbpdf	H. Oberdiek	1
tie	K. Guntermann	?
tr2latex	K. Al-Yahya, C. Engel	?
tran	H. Hagen	1

... continued on next page

Table C.1: (continued)

Program(s)	**Author(s)**	**License**
ttb	U. Fastenrath	0
ttf2afm	?	?
ttf2pk	F. Loyer, W. Lemberg	1
ttf2tfm	F. Loyer, W. Lemberg	1
tth	I. Hutchinson	3
uedit32	I. Mead	5
uncep	B. Jackowski, P. Pianowski, P. Strzelczyk	0
uncop	B. Jackowski, P. Pianowski, P. Strzelczyk	0
ungsvw32	Aladdin Enterprises	2
unpost	I. Hetherington	?
untex	M. Staats	0
unzip	J. Gailly	0
vftovp	D. Knuth	1
virtex	D. Knuth	1
vlna	P. Olsak	?
vptovf	D. Knuth	1
wbibdb	E. Doron	0
weave	D. Knuth	1
wget	H. Niksic	1
wgnuplot	C. Kelley, T. Williams	0
windvi	F. Popineau	1
winedt	A. Simonic	5
wp2latex	R. Houtepen, J. Fojtik	?
zip	M. Adler, R. Wales, J. Gailly, O. van der Linden, K. Rommel	0
zlib	J. Gailly, M. Adler	0
∗.pfb	URW Software	1

Appendix D

Electronic documents on the CDROM

Here is a bibliography of electronic documents you can find on the 4allTEX CDROM in `\ElectronicDocs`. In that directory you will find a subdirectory for each author. All documents are available in PDF; some are available also in other formats, such as TEX, plain text or DVI.

The documents are in English unless indicated otherwise by a language marker in the left margin.

(ES) Bautista, T. 1996. *Una Descripción de LaTeX 2ε (translation of Knappen et al.: 'LaTeX-Kurzbeschreibung' into Spanish)*. 82 pages.

(FR) Bayart, B. 1995. *Joli manuel pour LaTeX 2ε — Guide local de l'ESIEE*. 143 pages.

Berry, K. 1997a. *KPATHSEA manual*. 43 pages.

Berry, K. 1997b. *Web2c manual*. 58 pages.

Berry, K. and Smith, S. 1998. *Expanded Plain TeX*. 62 pages.

Biemesderfer, C. 1990. *LaTeX Command Summary*. 14 pages.

Carlisle, D.P. 1998. *Packages in the 'graphics' bundle*. 16 pages.

Chen, P. and Harrison, M. 1990. *Index Preparation and Processing*. 21 pages.

(DE) Cremer, F. 1993. *Das kleine TeXBuch*. 150 pages.

Doob, M. 1984. *A Gentle Introduction to TeX: A Manual for Self-study*. 95 pages.

(MN) Дорж, Д. 1998. *LaTeX 2ε – Товч заагар (translation and adaptation of Partl's 'LaTeX-Kurzbeschreibung' into Mongolian)*. 19 pages.

Doron, E. 1997. *BibDB, an Interactive* BIBTEX *Bibliography Manager*. 62 pages.

(NL) Eijkhout, V. 1989. *Wat is TEX?* 4 pages.

Eijkhout, V. 1996. *The TEX ruler* 3 pages.

(GR) Filippou, D. 1997. *Μία εὔκολη εἰσαγωγὴ στὸ TEX (translation of Doob's 'A gentle introduction to TEX' into Greek, including an extra chapter on typesetting Greek text)*. 127 pages.

Frambach, E. 1999. *Is TEX Y2K-compliant?* 6 pages.

Gibbons, J. 1993. *What exactly are virtual fonts?* 4 pages.

(NL) Hagen, H. 1998a. *ConTEXt voor Beginners*. 132 pages.

Hagen, H. 1998b. *Getting Started with ConTEXt*. 130 pages.

Hàn Thế, T., Rahtz, S. and Hagen, H. 1999. *The PDFTEX user manual*, 24 pages.

(FI) Hellgren, T. 1999. *Pitkänpuoleinen johdanto LATEX 2ε:n Käyttöön (translation of Oetiker's 'The Not So Short Introduction to LATEX 2ε' into Finnish)*. 101 pp.

(FR) Herrb, M. 1998. *Une courte (?) introduction à LATEX 2ε — ou LATEX 2ε en 74 minutes (translation of Oetiker's 'The Not So Short Introduction to LATEX 2ε' into French)*. 88 pages.

Hobby, J. 1995a. *Drawing Graphs with MetaPost*. 21 pages.

Hobby, J. 1995b. *The MetaPost System*. 20 pages.

Hobby, J. 1995c. *A User's Manual for MetaPost*. 91 pages.

Hoekwater, T. 1998. *Comparing ConTEXt and LATEX*. 6 pages.

Hoenig, A. 1991. *An Introduction to TEX for New Users*. 4 pages.

Knuth, D. 1990. *Virtual fonts: More fun for Grand Wizards*. 12 pages.

Kotz, D. 1991. *LATEX and the GNUPLOT Plotting Program*. 9 pages.

Kowk, C. 1988. *Extensions to epic and LATEX Picture Environment*. 12 pages.

Laan, C. v.d. 1994. *What is TEX and Metafont all about?* 23 pages.

Laan, C. v.d. 1995. *Publishing with TEX*. 131 pages.

Lamport, L. 1987a. *MakeIndex: An Index Processor for LATEX*. 8 pages.

Lamport, L. 1987b. *LATEX 2ε — The macro package for TEX*. 70 pages.

LATEX3 Team 1997a. *LATEX 2ε for Authors*. 30 pages.

LATEX3 Team 1997b. *LATEX 2ε for Class and Package Writers*. 32 pages.

Maltby, G. 1992. *An introduction to TEX and friends*. 63 pages.

(PL) Marszałkowska, J. 1997. *System MetaPost (translation of Hobby's 'The MetaPost System' into Polish)*. 19 pages.

(HU) Németh, L. 1998. *Nem olyan rövid bevezeto a LaTeX2e használatába (translation of Oetiker's 'The Not So Short Introduction to LaTeX 2ε' into Hungarian)*. 90 pp.

Oetiker, T., Partl, H., Hyna, I. and Schlegl, E. 1999. *The Not So Short Introduction to LaTeX 2ε — or LaTeX 2ε in 87 minutes*. 101 pages.

(NL) Oostrum, P. van. 1997. *Handleiding LaTeX*. 89 pages.

Patashnik, O. 1988a. *BibTeXing*. 16 pages.

Patashnik, O. 1988b. *Designing BibTeX Styles*. 10 pages.

Podar, S. 1986. *Enhancements to the Picture Environment of LaTeX*. 33 pages.

Popineau, F. 1997. *Web2c under WIN32*. 6 pages.

Reckdahl, K. 1997. *Using Imported Graphics in LaTeX 2ε*. 86 pages.

(DE) Reichert, A. 1997. *Gleitobjekte — die richtige Schmierung*. 18 pages.

Rokicki, T. 1997. *DVIPS: A TeX Driver (5.66a)*. 65 pages.

Rose, K. 1999. *XY-pic User's Guide*. 16 pages.

Rose, K. and Moore, R. 1999. *XY-pic Reference Manual*. 81 pages.

(DE) Rupprecht, R. 1989. *Schneewittchen oder Übungen zum LaTeX-Kurs*. 8 pages.

(DE) Schmidt, W., Knappen, J., Partl, H., and Hyna, I. 1999. *LaTeX 2ε-Kurzbeschreibung*. 49 pages.

Schrod, J. 1991. *The Components of TeX*. 9 pages.

Silverman, J. 1992. *Plain TeX Reference Card*. 2 pages.

(LT) Statulevičius, V. 1997. *LaTeX 2ε: Matematiniai Simboliai ir Sriftai*. 7 pages.

Taylor, C. 1996. *What has WYSIWYG done to us?* 17 pages.

(FR) Taylor, C. 1997. *Mais qu'est ce qu'ont bien pu nous apporter les systèmes WYSIWYG? (CFTTR's translation of* What has WYSIWYG done to us *into French)*. 29 pages.

Tobin, G. 1994. *Metafont for beginners*. 21 pages.

(RU) Тоботрас, Б. *1998: Не очень краткое введение в LaTeX 2ε. (translation of Knappen's 'LaTeX-Kurzbeschreibung' into Russian)*. 84 pages.

TUG Working Group on the TeX Directory Structure. 1997. *A Directory Structure for TeX Files*. 20 pages.

UK TeX Users Group Committee. 1998. *The New TeX FAQ — Your 119 Questions Answered*. 34 pages.

Valiente Feruglio, G. 1996. *Do journals honor LaTeX submissions?* 9 pages.

(PL) Wawrykiewicz, S. 1994. *Łagodne wprowadzenie do TeX-a (translation of Doob's 'A gentle introduction to TeX' into Polish)*. 74 pages.

Wilkins, D. 1994a. *Getting Started with Plain TeX*. 40 pages.

Wilkins, D. 1994b. *Summary of Commonly-Used Features of Plain TeX*. 18 pages.

Williams, T. and Kelley, C. 1993. *GNUplot — An Interactive Plotting Program*. 64 pages.

Woo, K. 1991. *GNUPLOT Quick Reference*. 6 pages.

Appendix E

Glossary

4DOS A command line interpreter that can be used as an alternative to the common Windows 'DOS box' or 'Command Prompt'. 4DOS supports a much more powerful scripting language and it is much more user-friendly.

ASCII American Standard Code for Information Interchange. A standard for representing characters, numbers, punctuation, carriage returns, line feeds, etc. An ASCII file contains no formatting information encoded through the use of special characters, but sequences of ASCII characters may be interpreted as formatting or structuring tags.

ANSI American National Standards Institute. Mostly used to specify a codepage defined by this institute.

anti-aliasing A technique to smooth edges of low resolution bitmapped images. It is often used to make text on computer screens more readable.

BibTeX A program that can generate bibliographic references and bibliographies automatically.

bitmapped image An image that is defined in terms a matrix of individually colored pixels.

bitmapped font A font that consists of glyphs as bitmapped images.

codepage A definition of the layout of a character set, in terms of positions of each character in a font.

ConTeXt A TeX dialect such as Plain TeX and LaTeX.

console application A program that has no graphical user interface, but instead writes screen output (if any) to the console. Such programs are usually run from the command line in a DOS box.

CMYK Cyan, Magenta, Yellow, Black. A method for specifying an exact color value.

CTAN Comprehensive TEX Archive Network. A network of FTP servers that store TEX resources.

DDE Dynamic Data Exchange. A method for (Windows) programs to exchange data, commands, etc.

DVI DeVice Independent. Used to describe an important feature of TEX output: regardless of the device you use to render it on, the output will look the same. Line breaks, page breaks, in fact positions of any item on a page are fixed.

DTP Desk Top Publishing. The common name for programs that are used to design magazines, advertisements, etc. Usually these programs support a highly interactive 'What You See Is What You Get' graphical user interface.

***ε*-TEX** An extended version of the standard TEX program.

FAT File Allocation Table. A method used by the operating system to store information on the contents of a disk.

environment 1. A small piece of computer memory in which some variables can be stored; 2. A LATEX construction that implements specific local formatting rules.

font metrics A list of values that any typesetting system needs when using a specific font. This list contains sizes of each character, interline spacing, kerning information, etc.

font glyphs The actual character shapes of a font.

FTP File Transfer Protocol. A standardized method for transferring files over the Internet.

HTML HyperText Markup Language, the language in which World Wide Web pages are written.

HTML HELP An online help format based on HTML.

ISO International Standards Organization.

ISO-9660 A standard for CDROM production. CDROMs that comply with this standard can be read by nearly every operating system.

Java A fairly new programming language that is designed to run on any operating system.

Joliet An extension of ISO-9660 that allows for long file names and a few more features. Mostly used on Windows systems.

kernel The part of a system that performs the most basic functions.

kerning A method for optimizing typesetting by specifying small adjustments to intercharacter spacing.

LaTeX A TeX dialect such as Plain TeX or ConTeXt.

ligature A replacement glyph for two or more characters that looks better than the combination.

macro A small subprogram made up of primitive functions and/or other macros.

MetaFont A program for designing fonts.

MetaPost A program for designing graphs, based on the MetaFont language. Output is generated in EPS format.

NTFS Windows NT File System.

NTS New Typesetting System, a possible successor to TeX that is still under construction.

Omega An extended version of TeX that uses Unicode natively.

PCL Printer Control Language. A language designed by Hewlett Packard that can be used on many common printer types.

Omega An extension to the original TeX program. Omega supports Unicode natively.

PDF Portable Document Format. A fully self-contained and efficient file format, designed by Adobe, that can easily be rendered on computer screens and printers.

PDFTeX An extension of the standard TeX program that is capable of generating PDF instead of DVI.

Perl A powerful scripting language that can be used on almost every operating system.

pica A measure that is often used in the typesetting world. One pica is 12 points, which is roughly 4 mm or 0.16 inch.

Plain TeX A TeX dialect such as LaTeX or ConTeXt.

point A measure that is often used in the typesetting world. One point is 1/72.27 inch. Note that a 'big point' is 1/72 inch.

PostScript An extremely powerful typesetting language that is used by all professional printers.

registry A Windows method for storing information about the operating system, user settings and program settings.

regular expressions A method for using 'wildcards' and other special characters in strings to be searched for. For instance, using the expression `a.used\.$` you could find any instance of the word `amused.` or `abused.` appearing at the end of a line in a file. Some editor programs and search programs support regular expressions.

RGB Red, Green, Blue. A method for specifying an exact color value.

SGML Standard Generalized Markup Language. An ISO standard for tagging up the logical structure of ASCII based documents. A corresponding DTD (Document Type Definition) defines these tags. HTML and XML are derived from SGML.

TDS TEX Directory Structure. A standard for structuring all resources available in a typical TEX system.

TrueType A format for fonts that is commonly used by the Windows operating system and by Macintosh computers.

Type 1 A format of PostScript fonts that defines a font in a vector (outline) format. Such fonts can be scaled to any size without losing quality.

Type 3 A format of PostScript fonts that defines a font in terms of arbitrary PostScript code, including (colored) bitmaps. Such fonts are typically more difficult to render, especially on low-resolution devices.

Type 42 A format of PostScript fonts that is in fact a wrapper around a TrueType font. Such fonts can only be rendered on devices that support both a PostScript and a TrueType engine.

UNC Universal Naming Convention: A common method for accessing files on a network drive without using a 'mapped' drive letter. Names specified this way are called UNC names, and typically appear as `\\server\volume\path\filename`, where `server` is the name of the network server where the files reside, `volume` is the name of a disk volume on that server, and the `path\filename` portion is a directory name and file name.

Unicode An international standard that defines codepages for all languages in a 16-bit encoding.

URI Uniform Resource Indentifier: sequence of characters chosen from a limited subset of the repertoire of ASCII characters both for transmission in network protocols and representation in spoken and written human communication.

URL Uniform Resource Locator: a compact string representation of the location for a resource that is available via the Internet.

Usenet An Internet method for sharing email messages about specific topics. Also known as 'News'.

UTC Coordinated Universal Time: the basis of civil time-keeping.

vectorized picture A picture or graph defined in terms of vectors, as opposed to bitmapped pictures.

virtual font A method for mapping one of or more (parts of) 'real' fonts on one font that TEX 'sees'.

WEB A system for writing programs and their documentation simultaneously in one document. From this one document either the program source or the documentation

can be distilled. Note that when we talk about 'the Web', we usually mean the World Wide Web, which is something entirely different.

Web2c The name of a TEX system implementation based on a conversion of the original Pascal sources to C. Web2c also adds many extra features to the original system.

Win32 The name of an application programming interface supported by Windows 95, 98 and NT. It allows for all 32-bit features such as long file names, multi-tasking and multi-treading.

Windows directory The main directory in which the Windows operating system stores files that are part of the operating system. On computers running Windows 95/98 this directory is usually `c:\windows`; on Windows NT it is usually `c:\winnt`. The operating system also stores this path in the environment variable `windir`.

WWW World Wide Web.

WYSIWYG Acronym for 'What You See Is What You Get', pronounced 'wizzywig'. Formatted text and graphics presented on a computer screen exactly as they will appear when printed.

XML Extendable Markup Language, the successor of HTML.

Appendix F

Bibliography

Abdelhamid, R. 1995. *Das Vieweg LaTeX 2ε Buch*. Vieweg-Verlag Wiesbaden.

Abrahams, P., Berry, K. and Hargreaves, K. 1990. *TeX for the impatient*. Addison-Wesley.

Adobe Systems Incorporated 1985. *PostScript Language Tutorial and Cookbook*. Addison-Wesley.

Adobe Systems Incorporated 1990a. *Adobe Type 1 Font Format*. Addison-Wesley.

Adobe Systems Incorporated 1990b. *PostScript Language Reference Manual*. Addison-Wesley.

Adobe Systems Incorporated 1990c. *PostScript Language Reference Manual*. 2nd edition. Addison-Wesley.

Adobe Systems Incorporated 1995. *Adobe Font Metric File Format Specification*. Technical Report 5004.

American Mathematical Society 1991a. *AMS-LaTeX Version 1.1, User's Guide*. American Mathematical Society.

American Mathematical Society 1991b. *User's Guide to AMS-TeX, Version 2.1*. American Mathematical Society.

Appelt, W. 1994. *TeX für Fortgeschrittene*. Addison-Wesley.

Bautista, T. 1996. *Una Descripcion de LaTeX 2ε (translation of Knappen's 'LaTeX-Kurzbeschreibung' into Spanish)*. cdrom.

Bayart, B. 1995. *Joli manuel pour LaTeX 2ε – Guide local de l'ESIEE*. cdrom.

Bechtolsheim, S. v. 1993. *TeX in Practice: 1. Basics*. Springer-Verlag.

Berry, K. 1990. Filenames for fonts. *TUGBoat*, **11** (4), 517–520.

Berry, K. 1997a. *KPATHSEA manual*. (cdrom).

Berry, K. 1997b. *Web2c manual*. (cdrom).

Berry, K. and Smith, S. 1998. *Expanded Plain TEX*. (cdrom).

Biemesderfer, C. 1990. LATEX Command Summary. (cdrom).

Borde, A. 1992. *TEX by Example, a beginner's guide*. Academic Press Professional.

Braams, J. 1991. Babel, a multilingual style-option system for use with LATEX's standard document styles. *TUGBoat*, **12** (2), 291–301.

Bruin, R. d., Laan, C. v. d. and Luyten, J. 1988. *Publiceren met LATEX*. Rekencentrum Universiteit Groningen.

Carlisle, D. 1998. *Packages in the 'graphics' bundle*. (cdrom).

Chen, P. and Harrison, M. 1990. *Index Preparation and Processing*. (cdrom).

Cremer, F. 1993. *Das kleine TEXBuch*. (cdrom).

Detig, C. 1997. *Der LATEX Wegweiser*. International Thompson.

Dietsche, L. and Lammarsch, J. 1994. *LATEX zum Loslegen – Ein Soforthelfer für den Alltag*. Springer-Verlag.

Diller, T. 1998. *LATEX Line by Line*. 2nd edition. Wiley.

Dol, W. and Frambach, E. 1997. *4allTEX*. 4th edition. Dutch language oriented TEX Users Group NTG.

Dol, W. and Frambach, E. 1999. 4Spell, a spell-checker for Windows 95/98/NT. *MAPS*, **22**, 123–126.

Dol, W., Frambach, E. and Vlerk, M. v. d. 1993. 4TEX, a TEX Workbench for ms-dos pc's. *MAPS*, **93** (1), 53–56.

Doob, M. 1984. *A Gentle Introduction to TEX, A Manual for Self-study*. (cdrom).

Doron, E. 1997. *BibDB, an Interactive BibTEX Bibliography Manager*. (cdrom).

Eijkhout, V. 1989. *Wat is TEX?* (cdrom).

Eijkhout, V. 1991. *TEX by Topic*. Addison-Wesley.

Eijkhout, V. 1996. *The TEX ruler*. (cdrom).

Filippou, D. 1997. *Μία εὔκολη εἰσαγωγὴ στὸ TEX (translation of Doob's 'A gentle introduction to TEX' into Greek, including an extra chapter on typesetting Greek text)*. (cdrom).

Frambach, E. 1999a. 4Project, a project manager for TEX. *MAPS*, **22**, 127–129.

Frambach, E. 1999b. Is TEX Y2K-compliant? *MAPS*, **22**, 136–141. (cdrom).

Furuta, R., Scofield, J. and Shaw, A. 1982. Document Formatting Systems: Survey, Concepts, and Issues. *Computing Surveys*, **14** (3), 417–472.

Gibbons, J. 1993. What exactly are virtual fonts? *Usenet group 'comp.text.tex'*. cdrom.

Goossens, M. 1993. Postscript en LaTeX, de komplementariteit in de praktijk. *MAPS*, **93** (1), 101–112.

Goossens, M., Mittelbach, F. and Samarin, A. 1994. *The LaTeX Companion*. Addison-Wesley.

Goossens, M., Mittelbach, F. and Samarin, A. 1996. *Der LaTeX-Begleiter*. Addison-Wesley.

Goossens, M., Rahtz, R. and Mittelbach, F. 1997. *The LaTeX Graphics Companion*. Addison-Wesley.

Greenwade, G. 1993. The Comprehensive TeX Archive Network (CTAN). *TUGBoat*, **14** (3), 342–351.

Gurari, E. 1994a. *TeX and LaTeX: Drawing and Literate Programming*. McGraw-Hill.

Gurari, E. 1994b. *Writing with TeX*. McGraw-Hill.

Hagen, H. 1998a. *ConTeXt voor Beginners*. cdrom.

Hagen, H. 1998b. *Getting Started with ConTeXt*. cdrom.

Hàn Thế, T., Rahtz, S. and Hagen, H. 1999. *The PDFTeX user manual*. cdrom.

Heilmann, A. 1995. *LaTeX-Vademecum, Ein Kompaktführer für Einsteiger und Fortgeschrittene*. Springer-Verlag.

Hellgren, T. 1999. *Pitkänpuoleinen johdanto LaTeX 2ε:n Käyttöön (translation of Oetiker's 'The Not So Short Introduction to LaTeX 2ε' into Finnish)*. cdrom.

Henderson, D. 1989. Introduction to Metafont. *TUGBoat*, **10** (4), 467–479.

Herrb, M. 1998. *Une courte (?) introduction à LaTeX 2ε – ou LaTeX 2ε en 74 minutes (translation of Oetiker's 'The Not So Short Introduction to LaTeX 2ε' into French)*. cdrom.

Hewlett Packard 1990. *PCL5 Printer Language Technical Reference Manual, First edition*. Hewlett Packard.

Hobby, J. 1995a. *Drawing Graphs with MetaPost*. cdrom.

Hobby, J. 1995b. *The MetaPost System*. cdrom.

Hobby, J. 1995c. *A User's Manual for MetaPost*. cdrom.

Hoekwater, T. 1998. Comparing ConTeXt and LaTeX. *MAPS*, **20**, 280–285. cdrom.

Hoenig, A. 1991. *An Introduction to TEX for New Users.* (cdrom).

Hoenig, A. 1997. *TEX Unbound: Strategies for Fonts, Graphics and More*. Oxford University Press.

Horn, B. 1993. *Complete bitmap-free TEX packages!* Y&Y.

ISO/IEC 1993. *Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane*. Technical Report 10646-1.

Katzenbeisser, S. 1997. *Von der Idee zum Dokument, Einführung in TEX und LATEX*. Oldenbourg-Verlag.

Knuth, D. 1982a. The GF to PK Processor.

Knuth, D. 1982b. The PL to TF Processor.

Knuth, D. 1982c. The TF to PL Processor.

Knuth, D. 1982d. The VF to VP Processor.

Knuth, D. 1982e. The VP to VF Processor.

Knuth, D. 1984. A course on Metafont programming. *TUGBoat*, **5** (2), 105–118.

Knuth, D. 1986b. *The TEXbook*, volume A of *Computers and Typesetting*. Addison-Wesley.

Knuth, D. 1986a. *TEX: The Program*, volume B of *Computers and Typesetting*. Addison-Wesley.

Knuth, D. 1986e. *The Metafront book*, volume C of *Computers and Typesetting*. Addison-Wesley.

Knuth, D. 1986d. *Metafont: The Program*, volume D of *Computers and Typesetting*. Addison-Wesley.

Knuth, D. 1986c. *Computer Modern Typefaces*, volume E of *Computers and Typesetting*. Addison-Wesley.

Knuth, D. 1990a. The future of TEX and Metafont. *TUGboat*, **11** (4), 489.

Knuth, D. 1990b. Virtual fonts: more fun for grand wizards. *TUGboat*, **11** (1), 13–23.

Дорж, Д. 1999. *LATEX 2ε – Товч заагар (translation and adaptation of Partl's 'LATEX-Kurzbeschreibung' into Mongolian)*. (cdrom).

Kopka, H. 1997a. *LATEX – Band 1: Einführung*. Addison-Wesley.

Kopka, H. 1997b. *LATEX – Band 2: Ergänzungen – mit einer Einführung in Metafont*. Addison-Wesley.

Kopka, H. 1997c. *LATEX – Band 3: Erweiterungen*. Addison-Wesley.

Kopka, H. and Daly, P. 1999. *A Guide to LaTeX – Document Preparation for Beginners and Advanced Users*. 3rd edition. Addison-Wesley.

Kotz, D. 1991. *LaTeX and the GNUPLOT Plotting Program*. (cdrom).

Kwok, C. 1988. *Extensions to epic and LaTeX Picture Environment*. (cdrom).

Laan, C. v. d. 1994. *What is TeX and Metafont all about?* (cdrom).

Laan, C. v. d. 1995. *Publishing with TeX*. (cdrom).

Lammarsch, J. and Schoppmann, H. 1996. *CTAN/3, Das TeX-/LaTeX-Archiv von DANTE e.V.* Addison-Wesley.

Lamport, L. 1987a. *MakeIndex: An Index Processor for LaTeX*. (cdrom).

Lamport, L. 1987b. *LaTeX 2ε – The macro package for TeX*. (cdrom).

Lamport, L. 1994. *LaTeX, a Document Preparation System*. 2nd edition. Addison-Wesley.

Lamport, L. 1995. *Das LaTeX-Handbuch*. Addison-Wesley.

LaTeX3 Project Team 1995. *Configuration options for LaTeX 2ε*. (cdrom).

LaTeX3 Project Team 1997a. *LaTeX 2ε for authors*. (cdrom).

LaTeX3 Project Team 1997b. *LaTeX 2ε for Class and Package Writers*. (cdrom).

LaTeX3 Project Team 1998. *LaTeX 2ε font selection*. (cdrom).

Liang, F. 1983. *Word hy-phen-a-tion by com-puter*. Ph.D. thesis, Stanford University.

Maltby, G. 1992. *An introduction to TeX and friends*. (cdrom).

Marszałkowska, J. 1997. *System MetaPost (translation of Hobby's 'The MetaPost System' into Polish)*. (cdrom).

Mittelbach, F. 1990. E-TeX: Guidelines for future TeX extensions. *TUGboat*, **11** (3), 337–345.

Mittelbach, F. and Rowley, C. 1992. LaTeX 2.09 ↪ LaTeX 3. *TUGboat*, **13** (1), 96–101.

Mittelbach, F. and Schöpf, R. 1989. With LaTeX into the nineties. *TUGboat*, **10** (4), 681–690.

Mittelbach, F. and Schöpf, R. 1991. Towards LaTeX 3.0. *TUGboat*, **12** (1), 74–79.

Németh, L. 1996. *Nem olyan rövid bevezetõ a LaTeX 2ε használatába (translation of Oetiker's 'The Not So Short Introduction to LaTeX 2ε' into Hungarian)*. (cdrom).

Oetiker, T., Partl, H., Hyna, I. and Schlegl, E. 1999. *The Not So Short Introduction to LaTeX 2ε – or LaTeX 2ε in 87 minutes*. (cdrom).

Oostrum, P. v. 1997. *Handleiding LaTeX*. (cdrom).

Partl, H. 1988. *Layout-Änderungen mit LaTeX*. EDV-Zentrum der TU Wien.

Patashnik, O. 1988a. *BibTeXing*. (cdrom).

Patashnik, O. 1988b. *Designing BibTeX Styles*. (cdrom).

Podar, S. 1986. *Extensions to the Picture Environment of LaTeX*. (cdrom).

Popineau, F. 1997. *Web2c under WIN32*. (cdrom).

Reckdahl, K. 1997. *Using Imported Graphics in LaTeX 2ε*. (cdrom).

Reichert, A. 1997. Gleitobjekte – die richtige Schmierung. (cdrom).

Reid, G. 1988. *PostScript Language Program Design.* Addison-Wesley.

Reid, G. 1990. *Thinking in PostScript.* Addison-Wesley.

Rokicki, T. 1997. *DVIPS: A TeX Driver (5.66a)*. (cdrom).

Rose, K. 1999. *XY-pic User's Guide*. (cdrom).

Rose, K. and Moore, R. 1999. *XY-pic Reference Manual*. (cdrom).

Rupprecht, R. 1989. *Schneewittchen oder Übungen zum LaTeX-Kurs*. (cdrom).

Salomon, D. 1992. *Advanced TeX course: Insights and Hindsights*. Dutch language oriented TeX Users Group NTG.

Salomon, D. 1995. *The Advanced TeXbook.* Springer Verlag.

Samuel, A. 1985. *First Grade TeX: A Beginner's TeX manual*. TeX User Group.

Schmidt, W., Knappen, J., Partl, H., and Hyna, I. 1999. *LaTeX 2ε-Kurzbeschreibung*. (cdrom).

Schrod, J. 1991. *The Components of TeX*. (cdrom).

Schwartz, N. 1990. *Introduction to TeX*. Addison-Wesley.

Schwartz, N. 1991. *Einführung in TeX*. Addison-Wesley.

Schwarz, S. and Potucek, R. 1996. *Das TeXikon — Referenzhandbuch für TeX, LaTeX und LaTeX 2ε*. Addison-Wesley.

Seroul, R. and Levy, S. 1992. *A Beginner's Book of TeX*. Springer Verlag.

Silverman, J. 1992. Plain TeX Reference Card. (cdrom).

Smedinga, R. 1991. Hoe met LaTeX een boek kan worden gemaakt. *MAPS*, **91** (2), 97–101.

Sowa, F. 1994. *TeX/LaTeX und Graphik. Ein Überblick über die Verfahren*. Springer-Verlag.

Spivak, M. 1986. *The Joy of TeX*. American Mathematical Society.

Statulevičius, V. 1997. LaTeX 2ε: Matematiniai Simboliai ir Sriftai. (cdrom).

Taylor, C. 1996. What has WYSIWYG done to us? *The Seybold Report on Publishing Systems*, **26** (2), 1–12. cdrom.

Taylor, C. 1997. Mais qu'est ce qu'ont bien pu nous apporter les systèmes WYSIWYG? (CFTTR's translation of 'What has WYSIWYG done to us' into French. *Cahiers GUTenberg*, **27**, 5–33. cdrom.

Taylor, P. 1992. The future of TEX. In J. Zlatuška (ed.), *Proceedings of the 7th European TEX Conference, Prague*, pp. 235–254. Czechoslovak TEX Users Group CSTUG.

Тоботрас, Б. 1998. *Не очень краткое введение в LATEX 2ε*. cdrom.

Tobin, G. 1994. *Metafont for beginners*. cdrom.

TUG Working Group on the TEX Directory Structure 1995. A directory structure for TEX files. *TUGboat*, **16** (4), 401–413.

TUG Working Group on the TEX Directory Structure 1997. *A Directory Structure for TEX Files*. cdrom.

UK TEX Users Group Committee 1998. *The New TEX FAQ – Your 119 Questions Answered*. cdrom.

University of Chicago Press 1993. *The Chicago Manual of Style*. 14th edition. University of Chicago Press.

Urban, M. 1986. *An Introduction to LATEX*. TEX User Group.

Valiente Feruglio, G. 1996. Do journals honor LATEX submissions? *MAPS*, **17**, 23–31. cdrom.

Walsh, N. 1994. *Making TEX work*. O'Reilly.

Wawrykiewicz, S. 1994. *Łagodne wprowadzenie do TEX-a (translation of Doob's 'A gentle introduction to TEX' into Polish)*. cdrom.

Wilkins, D. 1994a. *Getting Started with Plain TEX*. cdrom.

Wilkins, D. 1994b. *Summary of Commonly-Used Features of Plain TEX*. cdrom.

Williams, T. and Kelley, C. 1993. *GNUplot – An Interactive Plotting Program*. cdrom.

Wonneberger, R. 1993. *LATEX Kompaktführer*. 3rd edition. Addison-Wesley Verlag.

Woo, K. 1991. *GNUPLOT Quick Reference*. cdrom.

Appendix G

Index

This index is in three parts; first we list all the programs and packages mentioned in the book, and all the commands and files used in each. Second, starting on page 518, we provide a summary index of authors, packages, programs, files, etc. Finally, starting on page 523 is a complete list of all commands described in the book.

Throughout the index bold face page numbers are used to indicate pages with important information about the entry, e.g., the precise definition of a command or a detailed explanation, while page numbers in normal type indicate a textual reference.

Summary Index

Authors

Files and File Extensions

All Commands