The UK \TeX\ FAQ
Your 260 Questions Answered
version 3.1a, date 2002/09/05
Maintained by Robin Fairbairns
September 5, 2002

NOTE
This document is an updated and extended version of the FAQ article that
was published as the December 1994 and 1995, and March 1999 editions of the
UK TUG magazine Baskerville (which weren’t formatted like this).
The article is also available via the World Wide Web.

Contents

A Introduction 6
B The Background 7
  1 What is \TeX? ........................................... 7
  2 How should I pronounce “\TeX”? ....................... 7
  3 What is \METAFONT? ..................................... 7
  4 What is MetaPost? ..................................... 8
  5 How can I be sure it’s really \TeX? ................. 8
  6 Are \TeX{} and friends Y2K compliant? ............... 8
  7 What is \LaTeX{}? ...................................... 9
  8 What is \LaTeX{} 2\epsilon? ............................ 9
  9 How should I pronounce “\LaTeX{}(2\epsilon)””? 9
 10 Should I use Plain \TeX{} or \LaTeX{}? ................. 9
 11 How does \LaTeX{} relate to Plain \TeX{}? .......... 10
 12 What is Con\TeX{}t? .................................. 10
 13 What are the AMS packages (\LaTeX, etc.)? ....... 10
 14 What is Eplain? ...................................... 10
 15 What is Lollipop? .................................... 11
 16 What is Texinfo? ..................................... 11
 17 If \TeX{} is so good, how come it’s free? ............ 11
 18 What is the future of \TeX{}? ........................ 11
 19 Reading (\LaTeX) files ................................ 12
 20 Why is \TeX{} not a WYSIWYG system? ............... 12
 21 \TeX{} User Groups .................................. 13
C Documentation and Help 13
 22 Books on \TeX{} and its relations ................... 13
 23 Books on Type ....................................... 15
 24 Where to find FAQs .................................. 15
 25 Where to get help .................................... 16
 26 How to ask a question ................................ 16
 27 (\LaTeX) Tutorials, etc ............................... 17
 28 Learning to write (\LaTeX) classes and packages .... 18
 29 \METAFONT{} and MetaPost Tutorials ................. 18
 30 \LaTeX{} Documentation .............................. 18
 31 Where can I find the symbol for... ................ 19
 32 The \PCTeX{} manual ................................... 19
L.2 Adobe Type 1 ("PostScript") fonts
81 Using PostScript fonts with \TeX
82 Previewing files using Type 1 fonts
83 \TeX font metric files for PostScript fonts
84 Deploying Type 1 fonts
85 Choice of scalable outline fonts
86 Weird characters in \dvips output

L.3 Particular font families
87 Using the "Concrete" fonts

M Graphics
88 How to import graphics into (\LaTeX) documents
89 Graphics in \dvips
90 Graphics in PDF\LaTeX
91 Making MetaPost output display in \ghostscript
92 Drawing with \TeX
93 Drawing Feynman diagrams in \LaTeX
94 Creating a bibliography style
95 Capitalisation in \biblatex
96 ‘String too long’ in \biblatex
97 \biblatex doesn’t understand lists of names
98 URLs in \biblatex bibliographies
99 Using \biblatex with Plain \TeX
100 Separate bibliographies per chapter?
101 Multiple bibliographies?
102 Putting bibliography entries in text
103 Sorting and compressing citations
104 Multiple citations

N Bibliographies and citations
N.1 Creating bibliographies
94 Creating a bibliography style
95 Capitalisation in \biblatex
96 ‘String too long’ in \biblatex
97 \biblatex doesn’t understand lists of names
98 URLs in \biblatex bibliographies
99 Using \biblatex with Plain \TeX

N.2 Creating citations
100 Separate bibliographies per chapter?
101 Multiple bibliographies?
102 Putting bibliography entries in text
103 Sorting and compressing citations
104 Multiple citations

N.3 Manipulating whole bibliographies
105 Listing all your \biblatex entries
106 Making HTML of your Bibliography

O Installing (\LaTeX) files
107 Installing a new package
108 Where to put new files
109 Installing \miktex "known packages"
110 “Temporary” installation of (\LaTeX) files

P Adjusting the typesetting
P.1 Alternative document classes
111 Replacing the standard classes
112 Formatting a thesis in \LaTeX
113 Setting papers for journals
114 A ‘report’ from lots of ‘article’s
115 Curriculum Vitae (Resumé)
116 Letters and the like
117 Other “document font” sizes?

P.2 Document structure
118 The style of document titles
119 The style of section headings
120 Indent after section headings
121 How to create a \subsection
122 The style of captions
123 Alternative head- and footlines in \LaTeX
124 Changing the margins in \LaTeX
125 Wide figures in two-column documents
126 1-column abstract in 2-column document
127 Really blank pages between chapters
128 Balancing columns at the end of a document
Q.6 Hyphenation

176 My words aren’t being hyphenated ......................... 87
177 Weird hyphenation of words ............................... 88
178 (Merely) peculiar hyphenation ............................. 88
179 Accented words aren’t hyphenated ....................... 89
180 Using a new language with Babel ......................... 89
181 Stopping all hyphenation .................................. 90

Q.7 Odds and ends

182 Typesetting all those $\LaTeX$-related logos ............ 91
183 Referring to things by their name ....................... 91
184 How to do bold-\texttt{} or bold-\textsc{} ................. 91

R Symbols, etc.

185 Symbols for the number sets .............................. 92
186 Better script fonts for maths ............................... 92
187 Setting bold Greek letters in $\LaTeX$ ...................... 93
188 The Principal Value Integral symbol ...................... 94
189 How to use the underscore character .................... 94
190 HOW to type an ‘@’ sign? ................................. 94
191 Typesetting the Euro sign ................................ 94

S Macro programming

S.1 “Generic” macros

192 Finding the width of a letter, word, or phrase ......... 95
193 Patching existing commands ............................... 96
194 Comparing the “job name” ............................... 96
195 Is the argument a number? ................................ 97
196 Defining macros within macros ......................... 98
197 Spaces in macros ............................................. 98
198 How to break the 9-argument limit ...................... 99
199 Defining characters as macros ......................... 100
200 Active characters in command arguments ............... 101
201 Defining a macro from an argument ................... 102
202 Transcribing $\LaTeX$ command definitions ............ 102

S.2 $\LaTeX$ macros

203 How to change $\LaTeX$’s “fixed names” .................. 103
204 Changing the words \texttt{} uses ....................... 104
205 Running equation, figure and table numbering ........ 104
206 \texttt{} and \texttt{} in macro names ................... 105
207 What’s the reason for ‘protection’? ..................... 105
208 \texttt{} does not work with \texttt{} ................. 106
209 Optional arguments like \texttt{} ......................... 106
210 Making labels from a counter ......................... 106
211 Finding if you’re on an odd or an even page .......... 106
212 How to change the format of labels .................... 107
213 A command with two optional arguments .............. 107
214 Adjusting the presentation of section numbers ........ 107

T Things are Going Wrong...

T.1 Getting things to fit

215 Enlarging $\LaTeX$ ................................. 108
216 Why can’t I load $\LaTeX$? .............................. 108

T.2 Making things stay where you want them

217 Moving tables and figures in $\LaTeX$ ..................... 109
218 Underlined text won’t break ........................... 110
219 Controlling widows and orphans ....................... 111

T.3 Things have “gone away”

220 Old $\LaTeX$ font references such as \texttt{} ......... 111
221 Missing symbol commands ............................... 111
222 Where are the \texttt{max} and \texttt{min} fonts? .......... 111
223 Where are the \texttt{am} fonts? .......................... 112

U Why does it do that?

5
A Introduction

This FAQ was originated by the Committee of the UK T\TeX\ Users’ Group (UK TUG) as a development of a regular posting to the Usenet newsgroup comp.text.tex that was maintained for some time by Bobby Bodenheimer. The first UK version was much re-arranged and corrected from the original, and little of Bodenheimer’s work now remains.

An HTML translation of the FAQ is available on the World-Wide Web, via URL http://www.tex.ac.uk/faq; an alternative HTML version is also to be found on the T\TeX\ Live CD-ROM (see question 52).

Most members of the committee of UK TUG, over the years since 1994, have contributed to this FAQ to some extent. The following people, who have never been members of the committee, have also contributed help or advice: Donald Arseneau, Barbara Beeton, Karl Berry, Giuseppe Bilotta, Damian Cugley, Michael Dewey, Michael Downes, Thomas Esser, Anthony Goreham, Norman Gray, Eitan Gurari, John Hobby, Berthold Horn, Ian Hutchinson, Werner Icking, David Kastrup, Regnor Jernsletten,
Finding the Files

Unless otherwise specified, all files mentioned in this FAQ are available from a CTAN archive, or from one of their mirrors. Question 45 gives details of the CTAN archives, and how to retrieve files from them. If you don’t have access to the Internet, question 52 tells you of sources of CD-ROMs that offer snapshots of the archives.

The reader should also note that the first directory name of the path name of every file on CTAN has been elided from what follows, for the simple reason that it’s always the same (tex-archive/).

To avoid confusion, we’ve also elided the full stop¹ from the end of any sentence whose last item is a path name (such sentences are rare, and only occur at the end of paragraphs). Though the path names are set in a different font from running text, it’s not easy to distinguish the font of a single dot!

B  The Background

1 What is \TeX?

\TeX{} is a typesetting system written by Donald E. Knuth, who says in the Preface to his book on \TeX{} (see question 22) that it is “intended for the creation of beautiful books — and especially for books that contain a lot of mathematics”.

Knuth is Emeritus Professor of the Art of Computer Programming at Stanford University in California, USA. Knuth developed the first version of \TeX{} in 1978 to deal with revisions to his series “the Art of Computer Programming”. The idea proved popular and Knuth produced a second version (in 1982) which is the basis of what we use today.

Knuth developed a system of ‘literate programming’ (see question 64) to write \TeX{}, and he provides the literate (WEB) source of \TeX{} free of charge, together with tools for processing the web source into something that can be compiled and something that can be printed; there’s never any mystery about what \TeX{} does. Furthermore, the WEB system provides mechanisms to port \TeX{} to new operating systems and computers; and in order that one may have some confidence in the ports, Knuth supplied a test (see question 5) by means of which one may judge the fidelity of a \TeX{} system. \TeX{} and its documents are therefore highly portable.

\TeX{} is a macro processor, and offers its users a powerful programming capability. For this reason, \TeX{} on its own is a pretty difficult beast to deal with, so Knuth provided a package of macros for use with \TeX{} called Plain \TeX{}: Plain \TeX{} is effectively the minimum set of macros one can usefully employ with \TeX{}, together with some demonstration versions of higher-level commands (the latter are better regarded as models than used as-is). When people say they’re “programming in \TeX{}”, they usually mean they’re programming in Plain \TeX{}.

2 How should I pronounce “\TeX{}”?

The ‘X’ stands for the lower case Greek letter Chi (χ — the upper-case Greek letter doesn’t look in the least like a letter ‘X’), and is pronounced by English-speakers either a bit like the ‘ch’ in the Scots word ‘loch’ ([x] in the IPA) or like ‘k’. It definitely is not pronounced ‘ks’.

3 What is METAFONT?

METAFONT was written by Knuth as a companion to \TeX{}; whereas \TeX{} defines the layout of glyphs on a page, METAFONT defines the shapes of the glyphs and the relations between them. METAFONT details the sizes of glyphs, for \TeX{}’s benefit, and details the rasters used to represent the glyphs, for the benefit of programs that will produce printed output as post processes after a run of \TeX{}.

¹‘Full stop’ (British English)==‘period’ (American English)
METAFONT’s language for defining fonts permits the expression of several classes of things: first (of course), the simple geometry of the glyphs; second, the properties of the print engine for which the output is intended; and third, ‘meta’-information which can distinguish different design sizes of the same font, or the difference between two fonts that belong to the same (or related) families.

Knuth (and others) have designed a fair range of fonts using METAFONT, but font design using METAFONT is much more of a minority skill than is \TeX macro-writing. The complete \TeX-user nevertheless needs to be aware of METAFONT, and to be able to run METAFONT to generate personal copies of new fonts.

4 What is MetaPost?
The MetaPost system (by John Hobby) implements a picture-drawing language very much like that of METAFONT except that it outputs Encapsulated PostScript files instead of run-length-encoded bitmaps. MetaPost is a powerful language for producing figures for documents to be printed on PostScript printers, either directly or embedded in (\TeX) documents. It includes facilities for directly integrating \TeX text and mathematics with the graphics. (Knuth tells us that he uses nothing but MetaPost for diagrams in text that he is writing.)

The PostScript output is of such a simple form that MetaPost output files can be directly included in PDF \LaTeX documents (see question 253).

Much of MetaPost’s source code was copied from METAFONT’s sources with Knuth’s permission.

5 How can I be sure it’s really \TeX?
\TeX (and METAFONT and MetaPost) are written in a ‘literate’ programming language called Web (see question 64) which is designed to be portable across a wide range of computer systems. How, then, is a new version of \TeX checked?

Of course, any sensible software implementor will have his own suite of tests to check that his software runs: those who port \TeX and its friends to other platforms do indeed perform such tests.

Knuth, however, provides a ‘conformance test’ for both \TeX (trip) and METAFONT (trap). He characterises these as ‘torture tests’: they are designed not to check the obvious things that ordinary typeset documents, or font designs, will exercise, but rather to explore small alleyways off the main path through the code of \TeX. They are, to the casual reader, pretty incomprehensible!

Once an implementation of \TeX has passed its trip, or an implementation of METAFONT has passed its trap, test it may reasonably be distributed as a working version.

6 Are \TeX and friends Y2K compliant?
Crashing: None of \TeX, METAFONT or MetaPost can themselves crash due to any change whatever in the date of any sort.

Timestamps: In the original sources, a 2-digit year was stored as the creation time for format files and that value is printed in logfiles. These items should not be of general concern, since the only use of the date format file is to produce the log output, and the log file is designed for human readers only.

Knuth announced in 1998 that implementators could alter this code without fear of being accused of non-compliance. Nearly all implementations that are being actively maintained had been modified to generate 4-digit years in the format file and the log, by the end of 1998. The original sources themselves have now been modified so that 4-digit year numbers are stored.

The \texttt{\year} primitive: Certification of a \TeX implementation (see question 5) does not require that \texttt{\year} return a meaningful value (which means that \TeX can, in principle, be implemented on platforms that don’t make the value of the clock available to user programs). The \TeXbook defines \texttt{\year} as “the current year of our Lord”, which is the only correct meaning for \texttt{\year} for those implementations which can supply a meaningful value, which is to say nearly all of them.

\footnote{PDF \LaTeX cannot normally handle PostScript inclusions}
In short, \TeX implementations should provide a value in \texttt{\year} giving the 4-digit year Anno Domini, or the value 1776 if the platform does not support a date function.

Note that if the system itself fails to deliver a correct date to \TeX, then \texttt{\year} will of course return an incorrect value. \TeX cannot be considered Y2K compliant, in this sense, on a system that is not itself Y2K compliant.

\textbf{Macros:} \TeX macros can in principle perform calculations on the basis of the value of \texttt{\year}. The \LaTeX suite (see question 7) performs such calculations in a small number of places; the calculations performed in the current (supported) version of \TeX are known to be Y2K compliant.

Other macros and macro packages should be individually checked.

\textbf{External software:} Software such as DVI translators needs to be individually checked.

7 \textbf{What is \LaTeX?}

\LaTeX is a \TeX macro package, originally written by Leslie Lamport, that provides a document processing system. \LaTeX allows markup to describe the structure of a document, so that the user need not think about presentation. By using document classes and add-on packages, the same document can be produced in a variety of different layouts.

Lamport says that \LaTeX ”represents a balance between functionality and ease of use”. This shows itself as a continual conflict that leads to the need for such things as FAQs: \LaTeX can meet most user requirements, but finding out how is often tricky.

8 \textbf{What is \LaTeX\2\epsilon?}

Lamport’s last version of \LaTeX (\LaTeX\2.09, last updated in 1992) was superseded in 1994 by a new version (\LaTeX\2\epsilon) written by the \LaTeX team (see question 250). \LaTeX\2\epsilon is now the only readily-available version of \LaTeX, and draws together several threads of \LaTeX development from the later days of \LaTeX\2.09.

\LaTeX\2\epsilon has several enhancements over \LaTeX\2.09, but they were all rather minor, with a view to continuity and stability rather than the "big push" that some had expected from the team. \LaTeX\2\epsilon continues to this day to offer a compatibility mode in which most files prepared for use with \LaTeX\2.09 will run (albeit with somewhat reduced performance). Differences between \LaTeX\2\epsilon and \LaTeX\2.09 are outlined in a series of ’guide’ files that are available in every \LaTeX distribution; the most important of these files is available on the Web as \url{http://www.tex.ac.uk/tex-archive/macros/latex/doc/html/usrguide/} and outlines the differences as seen by the ordinary writer of documents and of simple macros.

9 \textbf{How should I pronounce “\LaTeX\2\epsilon”?}

Lamport never recommended how one should pronounce \LaTeX, but a lot of people pronounce it ‘Lay \TeX’ or perhaps ‘Lah \TeX’ (with \TeX pronounced as the program itself; see question 2). It is definitely not to be pronounced in the same way as the rubber-tree gum.

The ‘epsilon’ in ‘\LaTeX\2\epsilon’ is supposed to be suggestive of a small improvement over the old \LaTeX\2.09. Nevertheless, most people pronounce the name as ‘\LaTeX-two-ee’.

10 \textbf{Should I use Plain \TeX or \LaTeX?}

There’s no straightforward answer to this question. Many people swear by Plain \TeX, and produce highly respectable documents using it (Knuth is an example of this, of course). But equally, many people are happy to let someone else take the design decisions for them, accepting a small loss of flexibility in exchange for a saving of brain power.

The arguments around this topic can provoke huge amounts of noise and heat, without offering much by way of light; your best bet is to find out what those around you are using, and to go with the crowd. Later on, you can always switch your allegiance; don’t bother about it.

If you are preparing a manuscript for a publisher or journal, ask them what markup they want before you develop your own; many big publishers have developed their own (\LaTeX) styles for journals and books, and insist that authors stick closely to their markup.
11 How does LATEX relate to Plain TeX?

LATEX is a program written in the programming language \TeX. (In the same sense, any LATEX document is also a program, which is designed to run ‘alongside’, or ‘inside’ LATEX, whichever metaphor you prefer.)

Plain TeX is also a program written in the programming language TeX.

Both exist because writing your documents in ‘raw’ TeX would involve much reinventing of wheels for every document. They both serve as convenient aids to make document writing more pleasant: LATEX is a far more extensive aid.

LATEX is close to being a superset of Plain TeX. Many documents designed for Plain TeX will work with LATEX with no more than minor modifications (though some will require substantial work).

Interpretation of any (L)ATEX program involves some data-driven elements, and LATEX has a wider range of such elements than does Plain TeX. As a result, the mapping from LATEX to Plain TeX is far less clear than that in the other direction.

12 What is ConTeXt?

ConTeXt is a macro package developed by Hans Hagen, originally to serve the needs of the Dutch firm, Pragma. It was designed with the same general-purpose aims as LATEX, but (being younger) reflects much more recent thinking about the structure of markup, etc. In particular, ConTeXt can customise its markup to an author’s language (customising modules for Dutch, English and German are provided, at present).

ConTeXt is well integrated, in all of its structure, with the needs of hypertext markup, and in particular with the facilities offered by PDF TeX (see question 253). The default installation employs a version of TeX built with both the PDF TeX and \v TeX (see question 252) extensions, and makes good use of both.

ConTeXt doesn’t yet have quite such a large developer community as does LATEX, but those developers who are active seem to have prodigious energy.

ConTeXt distribution: macros/context

13 What are the AMS packages (AMS-\TeX, etc.)?

AMS-\TeX is a \TeX macro package, originally written by Michael Spivak for the American Mathematical Society (AMS) during 1983–1985 and is described in the book “The Joy of \TeX” (see question 22). It is based on Plain TeX, and provides many features for producing more professional-looking maths formulas with less burden on authors. It pays attention to the finer details of sizing and positioning that mathematical publishers care about. The aspects covered include multi-line displayed equations, equation numbering, ellipsis dots, matrices, double accents, multi-line subscripts, syntax checking (faster processing on initial error-checking \TeX runs), and other things.

As LATEX increased in popularity, authors asked to submit papers to the AMS in LATEX, and so the AMS developed AMS-LATEX, which is a collection of LATEX packages and classes that offer authors most of the functionality of AMS-\TeX.

AMS-\TeX distribution: macros/amstex

AMS-LATEX distribution: macros/latex/required/amslatex

14 What is Eplain?

The Eplain macro package expands on and extends the definitions in Plain TeX. Eplain is not intended to provide “generic typesetting capabilities”, as do LATEX or Texinfo (see question 16). Instead, it provides definitions that are intended to be useful regardless of the high-level commands that you use when you actually prepare your manuscript.

For example, Eplain does not have a command \section, which would format section headings in an “appropriate” way, as LATEX’s \section. The philosophy of Eplain is that some people will always need or want to go beyond the macro designer’s idea of “appropriate”. Such canned macros are fine — as long as you are willing to accept the resulting output. If you don’t like the results, or if you are trying to match a different format, you are out of luck.

On the other hand, almost everyone would like capabilities such as cross-referencing by labels, so that you don’t have to put actual page numbers in the manuscript. Karl Berry, the author of Eplain, says he is not aware of any generally available macro packages that do not force their typographic style on an author, and yet provide such capabilities.
15 What is Lollipop?
Lollipop is a macro package written by Victor Eijkhout; it was used in the production of his book “\TeX{} by Topic” (see question 27). The manual says of it:

Lollipop is ‘\TeX{} made easy’. Lollipop is a macro package that functions as a toolbox for writing \TeX{} macros. It was my intention to make macro writing so easy that implementing a fully new layout in \TeX{} would become a matter of less than an hour for an average document, and that it would be a task that could be accomplished by someone with only a very basic training in \TeX{} programming.

Lollipop is an attempt to make structured text formatting available for environments where previously only WYSIWYG packages could be used because adapting the layout is so much more easy with them than with traditional \TeX{} macro packages.

The manual goes on to talk of ambitions to “capture some of the \LaTeX{} market share”; it’s a very witty package, but little sign of it taking over from \LaTeX{} is detectable... An article about Lollipop appeared in TUGboat 13(3).

Lollipop distribution: macros/lollipop

16 What is Texinfo?
Texinfo is a documentation system that uses one source file to produce both on-line information and printed output. So instead of writing two different documents, one for the on-line help and the other for a typeset manual, you need write only one document source file. When the work is revised, you need only revise one document. You can read the on-line information, known as an “Info file”, with an Info documentation-reading program. By convention, Texinfo source file names end with a \texttt{.texi} or \texttt{.texinfo} extension. You can write and format Texinfo files into Info files within GNU emacs, and read them using the \texttt{emacs} Info reader. You can also format Texinfo files into Info files using \texttt{makeinfo} and read them using \texttt{info}, so you’re not dependent on \texttt{emacs}. The distribution includes a \texttt{Perl} script, \texttt{texi2html}, that will convert Texinfo sources into HTML.

Texinfo distribution: macros/texinfo/texinfo

17 If \TeX{} is so good, how come it’s free?
It’s free because Knuth chose to make it so. He is nevertheless apparently happy that others should earn money by selling \TeX{}-based services and products. While several valuable \TeX{}-related tools and packages are offered subject to restrictions imposed by the GNU General Public Licence (‘Copyleft’), \TeX{} itself is not subject to Copyleft.

There are commercial versions of \TeX{} available; for some users, it’s reassuring to have paid support. What is more, some of the commercial implementations have features that are not available in free versions. (The reverse is also true: some free implementations have features not available commercially.)

This FAQ concentrates on ‘free’ distributions of \TeX{}, but we do at least list the major vendors (see question 55).

18 What is the future of \TeX{}?
Knuth has declared that he will do no further development of \TeX{}; he will continue to fix any bugs that are reported to him (though bugs are rare). This decision was made soon after \TeX{} version 3.0 was released; at each bug-fix release the version number acquires one more digit, so that it tends to the limit \(\pi\) (at the time of writing, Knuth’s latest release is version 3.14159). Knuth wants \TeX{} to be frozen at version \(\pi\) when he dies; thereafter, no further changes may be made to Knuth’s source. (A similar rule is applied to \METAFONT{}; its version number tends to the limit \(e\), and currently stands at 2.718.)

There are projects (some of them long-term projects: see, for example, question 250) to build substantial new macro packages based on \TeX{}. For the even longer term, there are various projects to build a successor to \TeX{}; see questions 251 and 252.
19 Reading \TeX files

So you’ve been sent a \TeX file: what are you going to do with it? Well, the good news is that \TeX systems are available, free, for most sorts of computer; the bad news is that you need a pretty complete \TeX system even to read a single file, and complete \TeX systems are pretty large.

\TeX is a typesetting system that arose from a publishing project (see question 1), and its basic source is available free from its author. However, at its root, it is just a typesetting engine: even to view or to print the typeset output, you will need ancillary programs. In short, you need a \TeX distribution — a collection of \TeX-related programs tailored to your operating system: for details of the sorts of things that are available, see question 53 or 55 (for commercial distributions).

But beware — \TeX makes no attempt to look like the sort of WYSIWYG system you’re probably used to (see question 20): while many modern versions of \TeX have a compile–view cycle that rivals the best commercial word processors in its responsiveness, what you type is usually mark-up, which typically defines a logical (rather than a visual) view of what you want typeset.

However, in this context markup proves to be a blessing in disguise: a good proportion of most \TeX documents is immediately readable in an ordinary text editor. So, while you need to install a considerable system to attain the full benefits of the \TeX document that you were sent, the chances are you can understand quite a bit of it with nothing more than the ordinary tools you already have on your computer.

20 Why is \TeX not a WYSIWYG system?

WYSIWYG is a marketing term ("What you see is what you get") for a particular style of text processor. WYSIWYG systems are characterised by two principal claims: that you type what you want to print, and that what you see on the screen as you type is a close approximation to how your text will finally be printed.

The simple answer to the question is, of course, that \TeX was conceived long before the marketing term, at a time when the marketing imperative wasn’t perceived as significant. However, that was a long time ago: why has nothing been done with the “wonder text processor” to make it fit with modern perceptions?

There are two answers to this. First, the simple “things have been done” (but they’ve not taken over the \TeX world); and second, “there are philosophical reasons why the way \TeX has developed is ill-suited to the WYSIWYG style”. Indeed, there is a fundamental problem with applying WYSIWYG techniques to \TeX: the complexity of \TeX makes it hard to get the equivalent of \TeX’s output without actually running \TeX.

A celebrated early system offering “WYSIWYG using \TeX” came from the Vor\TeX project: a pair of (early) Sun workstations worked in tandem, one handling the user interface while the other beavered away in the background typesetting the result. Vor\TeX was quite impressive for its time, but the two workstations combined had hugely less power than the average sub-thousand dollar Personal Computer nowadays, and its code has not proved portable (it never even made the last ‘great’ \TeX version change, at the turn of the 1990s, to \TeX version 3). Modern systems that are similar in their approach are Lightning Textures (an extension of Blue Sky’s original \TeX system for the Macintosh), and Scientific Word (which can also cooperate with a computer algebra system); both these systems are commercially available (see question 55).

The issue has of recent years started to attract attention from \TeX developers, and several interesting projects addressing the “\TeX document preparation environment” (see question 256) are in progress.

Nevertheless, The \TeX world has taken a long time to latch onto the idea of WYSI-WYG. Apart from simple arrogance (“we’re better, and have no need to consider the petty doings of the commercial word processor market”), there is a real conceptual difference between the word processor model of the world and the model \LaTeX and Con\TeXt employ — the idea of “markup”. “Pure” markup expresses a logical model of a document, where every object within the document is labelled according to what it is rather than how it should appear: appearance is deduced from the properties of the type of object. Properly applied, markup can provide valuable assistance when it comes to re-use of documents.

Established WYSIWYG systems find the expression of this sort of structured markup difficult; however, markup is starting to appear in the lists of the commercial world’s requirements, for two reasons. First, an element of markup helps impose style on a
document, and commercial users are increasingly obsessed with uniformity of style; and second, the increasingly pervasive use of XML-derived document archival formats demands it. The same challenges must be addressed by \TeX-based document preparation support schemes, so we are observing a degree of confluence of the needs of the two communities: interesting times may be ahead of us.

21 \TeX User Groups

There has been a \TeX User Group since very near the time \TeX first appeared. That first group, TUG, is still active and its journal \textit{TUGboat} continues in publication (4 issues a year) with articles about \TeX, \textsc{Metafont} and related technologies, and about document design, processing and production. TUG holds a yearly conference, whose proceedings are published in \textit{TUGboat}.

TUG’s web site is a valuable resource for all sorts of \TeX-related matters, such as details of \TeX software, and lists of \TeX vendors and \TeX consultants. Back articles from \textit{TUGboat} are slowly (subject to copyright issues, etc.) making their way to the site, too.

Some time ago, TUG established a “technical council”, whose task was to oversee the development of \TeXXnical projects. Most such projects nowadays go on their way without any support from TUG, but TUG’s web site lists its Technical Working Groups (TWGs).

TUG has a reasonable claim to be considered a world-wide organisation, but there are many national and regional user groups, too; TUG’s web site maintains a list of “Local User Groups” (LUGs).

Contact TUG itself via:

\begin{verbatim}
TeX Users Group
1466 NW Front Avenue, Suite 3141
Portland, OR 97209
USA
Tel: +1 503-223-9994
Fax: +1 503-223-3960
Email: tug@mail.tug.org
Web: \url{http://www.tug.org/}
\end{verbatim}

C Documentation and Help

22 Books on \TeX and its relations

While Knuth’s book is the definitive reference for \TeX, there are other books covering \TeX:

\begin{description}
\item[A Beginner’s Book of \TeX] by Raymond Seroul and Silvio Levy, (Springer Verlag, 1992, ISBN 0-387-97562-4)
\item[Introduction to \TeX] by Norbert Schwarz (Addison-Wesley, 1989, ISBN 0-201-51141-X — now out of print)
\item[A Plain \TeX Primer] by Malcolm Clark (Oxford University Press, 1993, ISBNs 0-198-53724-7 (hardback) and 0-198-53784-0 (paperback))
\item[\TeX by Topic] by Victor Eijkhout (Addison-Wesley, 1992, ISBN 0-201-56882-9 — now out of print, but see question 27)
\item[\TeX for the Beginner] by Wynter Snow (Addison-Wesley, 1992, ISBN 0-201-54799-6)
\end{description}
\TeX: Starting from Square One\footnote{That’s ‘Starting from Square One’} by Michael Doob (Springer Verlag, 1993, ISBN 3-540-56441-1 — now out of print)


A collection of Knuth’s publications about typography has recently been published:


and in late 2000, a “Millennium Boxed Set” of all 5 volumes of Knuth’s “Computers and Typesetting” series (about \TeX and \METAFONT) was published by Addison Wesley:


For \LaTeX, see:


*Math into \LaTeX: An Introduction to \LaTeX and \AMS-\LaTeX* by George Grätzer (third edition Birkhäuser and Springer Verlag, 2000, ISBN 0-8176-4431-9, ISBN 3-7643-4131-9)

A list of errata for the first printing is available from: \url{http://www.springer-ny.com/catalog/np/jan99np/0-387-98708-8.html}


A sample of George Grätzer’s “Math into \LaTeX”, in Adobe Acrobat format, and example files for the \LaTeX Graphics and Web Companions, and for Grätzer’s “First Steps in \LaTeX”, are all available on CTAN.

The list for \METAFONT is rather short:


Alan Hoenig’s ‘\TeX Unbound’ includes some discussion and examples of using \METAFONT.

This list only covers books in English: notices of new books, or warnings that books are now out of print are always welcome. However, we do not carry reviews of current published material.


Examples for First Steps in \TeX: info/FirstSteps
Examples for \TeX\ Graphics Companion: info/lgc
Examples for \TeX\ Web Companion: info/lwc
Sample of Math into \TeX: info/mil/mil.pdf

23 Books on Type

The following is a partial listing of books on typography in general. Of these, Bringhurst seems to be the one most often recommended.


There are many catalogues of type specimens but the following books provide a more interesting overall view of types in general and some of their history.


The above lists are limited to books published in English. Typographic styles are somewhat language-dependent, and similarly the ‘interesting’ fonts depend on the particular writing system involved.

24 Where to find FAQs

Bobby Bodenheimer’s article, from which this FAQ was developed, used to be posted (nominally monthly) to newsgroup comp.text.tex and cross-posted to newsgroups news.answers and comp.answers. The (long obsolete) last posted copy of that article is kept on CTAN for auld lang syne; it is no longer kept in the news.answers archives.

A version of the present FAQ may be browsed via the World-Wide Web, and its sources are available from CTAN.

Another excellent information source, available in English, is the \TeX\ navigator.

Both the Francophone \TeX\ usergroup Gutenberg and the Czech/Slovak usergroup CS-TUG have published translations of this FAQ, with extensions appropriate to their languages.

Other non-English FAQs are available:
25 Where to get help

First … read any FAQ you can find. (Which is what you’re doing now, isn’t it?)

An ambitious FAQ-like project to collect all \TeX-related information into one place is under way at http://www.ctv.es/USERS/irmina/TeEncontreX.html; as with any FAQ, this project needs the support and help of all users — as yet, it carries an incomplete set of answers to potential questions. The sources of the package (including a complete set of HTML files) are available: info/spanish/TeEncontreX

The tutorials and other on-line documentation (see question 27) can get you started but for more in-depth understanding you should get and read at least one of the many good books on the subject (see question 22). The definitive source for \LaTeX is the source code itself, but that is not something to be approached lightly (if at all).

If you are seeking a particular package or program, look on your own system first: you might already have it — the better \TeX distributions contain a wide range of supporting material.

If you have access to the internet, and in particular newsgroups, then \LaTeX discussions, including METAFONT and MetaPost, are on comp.text.tex. It is best to spend a little time familiarising yourself with the current threads before asking a question. The group is normally very responsive but asking a question that has just been answered is likely to dampen people’s enthusiasm to help you.

http://groups.google.com/ archives Usenet news discussions, and comp.text.tex may be found there. Google’s archive now goes impressively far back in time (before comp.text.tex even existed), and it is a powerful resource for tracking down that recommendation that no-one can now remember. Google also now allows you to post your own questions or followups.

The few people who can’t use the World Wide Web, can’t access Usenet news, but can use electronic mail can seek help through mailing lists.

The \TeXhax digest is nowadays operated as a moderated “MailMan” mailing list: its members now have the option of receiving messages in ‘real time’, and answers are more quickly forthcoming than ever they were in the past. Subscribe via http://lists.nottingham.ac.uk/mailman/listinfo/texhax; past digests are available on CTAN.

Many mailing lists exist that cover some small part of the \TeX arena. A good source of pointers is http://www.tug.org/

Announcements of \TeX-related installations on the CTAN archives are sent to the mailing list ctan-ann. Subscribe to the list by sending a message ‘subscribe ctan-ann <your email address>’ to majordomo@dante.de

Issues related to METAFONT (and, increasingly, MetaPost) are discussed on the metafont mailing list; subscribe by sending a message ‘subscribe metafont <your name>’ to listserv@ens.fr

A few other \TeX-related lists may be accessed via listserv@rz.uni-heidelberg.de. Send a message containing the line ‘help’ to this address, or browse http://listserv.uni-heidelberg.de/cgi-bin/wa

\TeXhax digests: Browse digests/texhax

26 How to ask a question

You want help from the community at large; you’ve decided where you’re going to ask your question (see question 25), but how do you phrase it?
Excellent “general” advice (how to ask questions of anyone) is contained in Eric Raymond’s article on the topic. Eric’s an extremely self-confident person, and this comes through in his advice; but his guidelines are very good, even for us in the un-self-confident majority. It’s important to remember that you don’t have a right to advice from the world, but that if you express yourself well, you will usually find someone who will be pleased to help.

So how do you express yourself in the \TeX{} world? There aren’t any comprehensive rules, but a few guidelines may help in the application of your own common sense.

1. Make sure you’re asking the right people. Don’t ask in a \TeX{} forum about printer device drivers for the \tool{fooobar} operating system. Yes, \TeX{} users need printers, but no, \TeX{} users will typically \emph{not} be \tool{fooobar} systems managers. Similarly, avoid posing a question in a language that the majority of the group don’t use: post in Ruritanian to \texttt{de.comp.text.tex} and you may have a long wait before a German- and Ruritanian-speaking \TeX{} expert notices your question.

2. If your question is (or may be) \TeX{}-system-specific, report what system you’re using, or intend to use: “I can’t install \TeX{}” is as good as useless, whereas “I’m trying to install the \tool{mumbleTeX} distribution on the \tool{grobble} operating system” gives all the context a potential respondent might need. Another common situation where this information is important is when you’re having trouble installing something new in your system.

3. If you need to know how to do something, make clear what your environment is. “I want to do \texttt{x} in Plain \TeX{}”, or “I want to do \texttt{y} in \LaTeX{} running the \texttt{boggle} class”.

4. If something’s going wrong, pretend you’re submitting a \LaTeX{} bug report (see question 259), and try to generate a minimum failing example. If your example needs your local \texttt{xyzthesis} class, or some other resource not generally available, be sure to include a pointer to how the resource can be obtained.

5. Be as succinct as possible. Your helpers probably don’t need to know \emph{why} you’re doing something, just \emph{what} you’re doing and where the problem is.

\LaTeX{} Tutorials, etc.

Some very fine tutorials have been written, over the years. Michael Doob’s splendid ‘Gentle Introduction’ to Plain \TeX{} has been stable for a very long time. Another contender in the same game is from one D. R. Wilkins, available on the web at \url{http://www.ntg.nl/doc/wilkins/pllong.pdf}

More dynamic is Tobias Oetiker’s ‘(Not so) Short Introduction to \LaTeX{} 2ε’, which is regularly updated, as people suggest better ways of explaining things, etc. The introduction has been translated into several languages other than its original English.

Harvey Greenberg’s ‘Simplified Introduction to \LaTeX{}’ was written for a lecture course, and is available on CTAN (in PostScript only, unfortunately).

Peter Flynn’s ‘Beginner’s \LaTeX{}’ isn’t yet (April 2002) complete, but is another pleasing read. It too started as course material.

The AMS publishes a “Short Math Guide for \LaTeX{}”, which is available (in several formats) via \url{http://www.ams.org/tex/short-math-guide.html}

Reference material is somewhat more difficult to come by. A sterling example is set by David Bausum’s list of \TeX{} primitives.

Some university departments make their local documentation available on the web. Most straightforwardly, there’s the simple translation of existing documentation into HTML, for example the INFO documentation of the \LaTeX{} installation, of which a sample is the \LaTeX{} documentation available at \url{http://www.tac.dk/cgi-bin/info2www?latex}

More ambitiously, some departments have enthusiastic documenters who make public record of their \LaTeX{} support. For example, Tim Love (of Cambridge University Engineering Department) maintains his department’s excellent pages \url{http://www-h.eng.cam.ac.uk/help/tpl/textprocessing/}

Another summary by command (somewhat akin to David Basum’s work for Plain \TeX{}) is to be found at \url{http://www.giss.nasa.gov/latex/ltex-2.html}

People have long argued for \LaTeX{} books to be made available on the web, and Victor Eijkhout’s excellent “\TeX{} by Topic” (previously published by Addison-Wesley,
but long out of print) was offered in this way at Christmas 2001. The book is currently available at http://www.eijkhout.net/tbt/; it’s not a beginner’s tutorial but it’s a fine reference (contributions are invited, and the book is well worth the suggested contribution).

_Gentle Introduction:_ info/gentle/gentle.pdf

_Not so Short Introduction:_ info/lshort/english/lshort.pdf

_Simplified \LaTeX:_ info/simplified-latex/latex.ps

## 28 Learning to write \LaTeX classes and packages

There’s nothing particularly magic about the commands you use when writing a package, so you can simply bundle up a set of \LaTeX \texttt{(re)newcommand} and \texttt{(re)newenvironment} commands, put them in a file \texttt{package.sty} and you have a package.

However, any but the most trivial package will require rather more sophistication. Some details of \LaTeX commands for the job are to be found in ‘\LaTeX for class and package writers’ (http://www.tex.ac.uk/tex-archive/macros/latex/doc/html/clsguide: the \LaTeX source of this document appears in the \LaTeX distribution). Beyond this, a good knowledge of \TeX itself is valuable. With such knowledge it is possible to use the documented source of \LaTeX as reference material (dedicated authors will acquaint themselves with the source as a matter of course). A complete set of the documented source of \LaTeX may be prepared by processing the file \texttt{source2e.tex} in the \LaTeX distribution.

Writing good classes is not easy; it’s a good idea to read some established ones (\texttt{classes.dtx}, for example, produces the standard classes other than \texttt{letter}, and may itself be formatted with \LaTeX). Classes that are not part of the distribution are commonly based on ones that are, and start by loading the standard class with \texttt{\LoadClass} — an example of this technique may be seen in \texttt{ltxguide.cls}

\texttt{classes.dtx}: macros/latex/base/classes.dtx

\texttt{ltxguide.cls}: macros/latex/base/ltxguide.cls

\texttt{source2e.tex}: macros/latex/base/source2e.tex

## 29 METAFONT and MetaPost Tutorials

Unfortunately there appear to be no tutorials on how to use METAFONT, except for the information provided by Knuth. There are, though, a couple of articles on how to run both METAFONT and MetaPost.

Geoffrey Tobin has provided \texttt{METAFONT for Beginners} (see question 78). This describes how the METAFONT system works and how to avoid some of the potential pitfalls.

Peter Wilson’s \textit{Some Experiences in Running METAFONT and MetaPost} offers the benefit of Peter’s experience (he has designed a number of ‘historical’ fonts using METAFONT). On the METAFONT side the article is more geared towards testing and installing new METAFONT fonts than on the system, while on the other side it describes how to use MetaPost illustrations in \LaTeX and PDF\LaTeX documents, with an emphasis on how to use appropriate fonts for any text or mathematics.

Hans Hagen (of Con\TeXt fame) offers a MetaPost tutorial called MetaFun (which admittedly concentrates on the use of MetaPost with in Con\TeXt). It may be found on his company’s MetaPost page.

\textit{Beginners’ guide}: info/metafont-for-beginners.tex

Peter Wilson’s “experiences”: info/metafp.ps (PostScript) or info/metafp.pdf (PDF format)

## 30 BIB\TeX Documentation

\BIB\TeX, a program originally designed to produce bibliographies in conjunction with \LaTeX, is explained in Section 4.3 and Appendix B of Leslie Lamport’s \LaTeX manual (see question 22). The document “\BIB\TeXing”, contained in the file \texttt{btxdoc.tex}, gives a more complete description. \textit{The \BIB\TeX Companion} (see question 22) also has information on \BIB\TeX and writing \BIB\TeX style files.

The document “Designing \BIB\TeX Styles”, contained in the file \texttt{btxhak.tex}, explains the postfix stack-based language used to write \BIB\TeX styles (.bst files). The file \texttt{btxbst.doc} is the template for the four standard styles (plain, abbrv, alpha,
unsrt). It also contains their documentation. The complete BiBTeX documentation set (including the files above) is available on CTAN.

There is a Unix BiBTeX man page in the web2c package (see question 53). Any copy you may find of a man page written in 1985 (before “BiBTeXing” and “Designing BiBTeX Styles” appeared) is obsolete, and should be thrown away.4

BiBTeX documentation: biblio/bibtex/distrib/doc

31 Where can I find the symbol for...

There is a wide range of symbols available for use with TeX, most of which are not shown (or even mentioned) in (La)TeX books. The Comprehensive BiBTeX Symbol List (by Scott Pakin et al.) illustrates over 2000 symbols, and details the commands and packages needed to produce them.

Other questions in this FAQ offer specific help on kinds of symbols:

• Script fonts for mathematics (question 186)
• Fonts for the number sets (question 185)
• Typesetting the principal value integral (question 188)

Symbol List: Browse info/symbols/comprehensive; there are processed versions in both PostScript and PDF forms for both A4 and letter paper.

32 The PCTeX manual

PCTeX is a set of macros by Michael Wichura for drawing diagrams and pictures. The macros are freely available; however, the PCTeX manual itself is not free. Unfortunately, TUG is no longer able to supply copies of the manual (as it once did), and it is now available only through Personal TeX Inc, the vendors of PCTeX (http://www.pctex.com/). The manual is not available electronically.

pctex: graphics/pictex

D Bits and pieces of TeX

33 What is a DVI file?

A DVI file (that is, a file with the type or extension .dvi) is TeX’s main output file, using TeX in its broadest sense to include PSTeX, etc. ‘DVI’ is supposed to be an acronym for DeVice-Independent, meaning that the file can be printed on almost any kind of typographic output device. The DVI file is designed to be read by a driver (see question 34) to produce further output designed specifically for a particular printer (e.g., a LaserJet) or to be used as input to a previewer for display on a computer screen. DVI files use TeX’s internal coding; a TeX input file should produce the same DVI file regardless of which implementation of TeX is used to produce it.

A DVI file contains all the information that is needed for printing or previewing except for the actual bitmaps or outlines of fonts, and possibly material to be introduced by means of \special commands (see question 38).

The canonical reference for the structure of a DVI file is the source of dvitype.

dvitype: systems/knuth/texware/dvitype.web

34 What is a driver?

A driver is a program that takes as input a DVI file (see question 33) and (usually) produces a file that can be sent to a typographic output device, called a printer for short.

A driver will usually be specific to a particular printer, although any PostScript printer ought to be able to print the output from a PostScript driver.

As well as the DVI file, the driver needs font information. Font information may be held as bitmaps or as outlines, or simply as a set of pointers into the fonts that the printer itself ‘has’. Each driver will expect the font information in a particular form. For more information on the forms of fonts, see questions 35, 36, 37 and 81.

4Or submitted for inclusion in a museum dedicated to the history of computing: the stricture was conceivably reasonable in the first version of this FAQ…
35 What are PK files?
PK files (packed raster) contain font bitmaps. The output from METAFONT (see question 78) includes a generic font (GF) file and the utility gftopk produces the PK file from that. There are a lot of PK files, as one is needed for each font, that is each magnification (size) of each design (point) size for each weight for each family. Further, since the PK files for one printer do not necessarily work well for another, the whole set needs to be duplicated for each printer type at a site. As a result, they are often held in an elaborate directory structure, or in ‘font library files’, to regularise access.

36 What are TFM files?
TFM stands for TEX font metrics, and TFM files hold information about the sizes of the characters of the font in question, and about ligatures and kerns within that font. One TFM file is needed for each font used by TEX, that is for each design (point) size for each weight for each family; one TFM file serves for all magnifications, so that there are (typically) fewer TFM files than there are PK files. The important point is that TFM files are used by TEX (LaTEX, etc.), but are not, generally, needed by the printer driver.

37 Virtual fonts
Virtual fonts for TEX were first implemented by David Fuchs in the early days of TEX, but for most people they started when Knuth redefined the format, and wrote some support software, in 1989 (he published an article in TUGboat at the time, and a copy is available on CTAN). Virtual fonts provide a way of telling TEX about something more complicated than just a one-to-one character mapping. The entities you define in a virtual font look like characters to TEX (they appear with their sizes in a font metric file), but the DVI processor may expand them to something quite different. You can use this facility just to remap characters, to make a composite font with glyphs drawn from several sources, or to build up an effect in arbitrarily complicated ways — a virtual font may contain anything which is legal in a DVI file. In practice, the most common use of virtual fonts is to remap PostScript fonts (see question 83) or to build ‘fake’ maths fonts.

It is important to realise that TEX itself does not see virtual fonts; for every virtual font read by the DVI driver there is a corresponding TFM file read by TEX. Virtual fonts are normally created in a single ASCII vpl (Virtual Property List) file, which includes both sets of information. The vptovf program is then used to the create the binary TFM and VF files. The commonest way (nowadays) of generating vpl files is to use the fontinst package, which is described in detail in question 83. qdtexvpl is another utility for creating ad-hoc virtual fonts.

38 \special commands
TEX provides the means to express things that device drivers can do, but about which TEX itself knows nothing. For example, TEX itself knows nothing about how to include PostScript figures into documents, or how to set the colour of printed text; but some device drivers do.

Such things are introduced to your document by means of \special commands; all that TEX does with these commands is to expand their arguments and then pass the command to the DVI file. In most cases, there are macro packages provided (often with the driver) that provide a comprehensible interface to the \special; for example, there’s little point including a figure if you leave no gap for it in your text, and changing colour proves to be a particularly fraught operation that requires real wizardry. \LaTEX 2e has standard graphics and colour packages that make figure inclusion, rotation and scaling, and colour typesetting via \specials all easy.

The allowable arguments of \special depend on the device driver you’re using. Apart from the examples above, there are \special commands in the \emTEX drivers (e.g., dvihplj, dviscr, etc.) that will draw lines at arbitrary orientations, and commands in dvitoln03 that permit the page to be set in landscape orientation.
39 Documented \LaTeX{} sources (.dtx files)
\LaTeX{} 2\epsilon, and many support macro packages, are now written in a literate programming style (see question 64), with source and documentation in the same file. This format, known as ‘doc’, was originated by Frank Mittelbach. The documented sources conventionally have the suffix .dtx, and should normally be stripped of documentation before use with \LaTeX{}. Alternatively you can run \LaTeX{} on a .dtx file to produce a nicely formatted version of the documented code. An installation script (with suffix .ins) is usually provided, which needs the standard \LaTeX{} 2\epsilon docstrip package (among other things, the installation process strips all the comments that make up the documentation for speed when loading the file into a running \LaTeX{} system). Several packages can be included in one .dtx file, with conditional sections, and there facilities for indices of macros etc. Anyone can write .dtx files; the format is explained in The \LaTeX{} Companion (see question 22), and a tutorial is available from CTAN (together with skeleton .dtx and .ins files). There are no programs yet to assist in composition of .dtx files. .dtx files are not used by \LaTeX{} after they have been processed to produce .sty or .cls (or whatever) files. They need not be kept with the working system; however, for many packages the .dtx file is the primary source of documentation, so you may want to keep .dtx files elsewhere.

DTX tutorial: info/dtxtut

40 What are encodings?
Let’s start by defining two concepts, the character and the glyph. The character is the abstract idea of the ‘atom’ of a language or other dialogue: so it might be a letter in an alphabetic language, a syllable in a syllabic language, or an ideogram in an ideographic language. The glyph is the mark created on screen or paper which represents a character. Of course, if reading is to be possible, there must be some agreed relationship between the glyph and the character, so while the precise shape of the glyph can be affected by many other factors, such as the capabilities of the writing medium and the designer’s style, the essence of the underlying character must be retained.

Whenever a computer has to represent characters, someone has to define the relationship between a set of numbers and the characters they represent. This is the essence of an encoding: it is a mapping between a set of numbers and a set of things to be represented. \TeX{} of course deals in encoded characters all the time: the characters presented to it in its input are encoded, and it emits encoded characters in its DVI (or PDF) output. These encodings have rather different properties.

The \TeX{} input stream was pretty unruly back in the days when Knuth first implemented the language. Knuth himself prepared documents on terminals that produced all sorts of odd characters, and as a result \TeX{} contains some provision for translating the input encoding to something regular. Nowadays, the operating system translates keystrokes into a code appropriate for the user’s language: the encoding used is often a national or international standard, though many operating systems use “code pages” defined by MicroSoft. These standards and code pages often contain characters that can’t appear in the \TeX{} system’s input stream. Somehow, these characters have to be dealt with — so an input character like “ê” needs to be interpreted by \TeX{} in a way that that at least mimics the way it interprets “\’{e}”.

The \TeX{} output stream is in a somewhat different situation: characters in it are to be used to select glyphs from the fonts to be used. Thus the encoding of the output stream is notionally a font encoding (though the font in question may be a virtual one — see question 37). In principle, a fair bit of what appears in the output stream could be direct transcription of what arrived in the input, but the output stream also contains the product of commands in the input, and translations of the input such as ligatures like fi⇒“fi”.

Font encodings became a hot topic when the Cork encoding (see question 42) appeared, because of the possibility of suppressing \texttt{\textbackslash accent} commands in the output stream (and hence improving the quality of the hyphenation of text in inflected languages, which is interrupted by the \texttt{\textbackslash accent} commands — see question 41). To take advantage of the diacriticised characters represented in the fonts, it is necessary to arrange that whenever the command sequence “\’{e}” has been input (explicitly, or implicitly via the sort of mapping of input mentioned above), the character that codes the position of the “ê” glyph is used.
Thus we could have the odd arrangement that the diacriticised character in the \TeX input stream is translated into \TeX commands that would generate something looking like the input character; this sequence of \TeX commands is then translated back again into a single diacriticised glyph as the output is created. This is in fact precisely what the \LaTeX packages `inputenc` and `fontenc` do, if operated in tandem on (most) characters in the ISO Latin-1 input encoding and the T1 font encoding. At first sight, it seems eccentric to have the first package do a thing, and the second precisely undo it, but it doesn’t always happen that way: most font encodings can’t match the corresponding input encoding nearly so well, and the two packages provide the sort of symmetry the \LaTeX system needs.

41 How does hyphenation work in \TeX?

Everyone knows what hyphenation is: we see it in most books we read, and (if we’re alert) often spot ridiculous mis-hyphenation from time to time (at one time, British newspapers were a fertile source).

Hyphenation styles are culturally-determined, and the same language may be hyphenated differently in different countries — for example, British and American styles of hyphenation of English are very different. As a result, a typesetting system that is not restricted to a single language at a single locale needs to be able to change its hyphenation rules from time to time.

\TeX uses a pretty good system for hyphenation (originally designed by Frank Liang), and while it’s capable of missing “sensible” hyphenation points, it seldom selects grossly wrong ones. The algorithm matches candidates for hyphenation against a set of “hyphenation patterns”. The candidates for hyphenation must be sequences of letters (or other single characters that \TeX may be persuaded to think of as letters) — things such as \TeX’s \texttt{\textbackslash accent} primitive interrupt hyphenation.

Sets of hyphenation patterns are usually derived from analysis of a list of valid hyphenations (the process of derivation, using a tool called patgen, is not ordinarily a participatory sport).

The patterns for the languages a \TeX system is going to deal with may only be loaded when the system is installed. To change the set of languages, a partial reinstalation (see question 180) is necessary.

\TeX provides two “user-level” commands for control of hyphenation: \texttt{\textbackslash language} (which selects a hyphenation style), and \texttt{\textbackslash hyphenation} (which gives explicit instructions to the hyphenation engine, overriding the effect of the patterns).

The ordinary \LaTeX user need not worry about \texttt{\textbackslash language}, since it is very thoroughly managed by the babel package; use of \texttt{\textbackslash hyphenation} is discussed in question 176.

42 What are the EC fonts?

A font consists of a number of glyphs. In order that the glyphs may be printed, they are encoded (see question 40), and the encoding is used as an index into tables within the font. For various reasons, Knuth chose deeply eccentric encodings for his Computer Modern family of fonts; in particular, he chose different encodings for different fonts, so that the application using the fonts has to remember which font of the family it’s using before selecting a particular glyph.

When \TeX version 3 arrived, most of the excuses for the eccentricity of Knuth’s encodings went away, and at TUG’s Cork meeting, an encoding for a set of 256 glyphs, for use in \TeX text, was defined. The intention was that these glyphs should cover ‘most’ European languages that use Latin alphabets, in the sense of including all accented letters needed. (Knuth’s CMR fonts missed things necessary for Icelandic and Polish, for example, but the Cork fonts have them. Even Cork’s coverage isn’t complete: it misses letters from Romanian, Eastern and Northern Sami, and Welsh, at least. The Cork encoding does contain “NG“ glyphs that allows it to support Southern Sami.) \LaTeX refers to the Cork encoding as T1, and provides the means to use fonts thus encoded to avoid problems with the interaction of accents and hyphenation (see question 179).

The only METAFONT-fonts that conform to the Cork encoding are the EC fonts. They look CM-like, though their metrics differ from CM-font metrics in several areas. The fonts are now regarded as ‘stable’ (in the same sense that the CM fonts are stable: their metrics are unlikely ever to change). Their serious disadvantages for the casual user are their size (each EC font is roughly twice the size of the corresponding CM font), and there are far more of them than there are CM fonts. The simple number of fonts
has acted as a disincentive to the production of Adobe Type 1 versions of the fonts., but several commercial suppliers offer EC or EC-equivalent fonts in type 1 or TrueType form — see question 55, and free are available (see question 255). What’s more, until corresponding fonts for mathematics are produced, the CM fonts must be retained because some mathematical symbols are drawn from text fonts in the CM encodings.

The EC fonts are distributed with a set of “Text Companion” (TC) fonts that provide glyphs for symbols commonly used in text. The TC fonts are encoded according to the \TeX TS1 encoding, and are not viewed as ‘stable’ in the same way as are the EC fonts are.

The Cork encoding is also implemented by virtual fonts provided in the PSNFSS system (see question 81), for PostScript fonts, and also by the txfonts and pxfonts font packages (see question 85).

EC and TC fonts: fonts/ec

43 What is the TDS?

TDS stands for the \TeX Directory Structure, which is a standard way of organising all the \TeX-related files on a computer system.

Most modern distributions conform to the TDS, which provides for both a ‘standard’ hierarchy and a ‘local’ hierarchy. The TDS reserves the name \texmf as the name of the root directory (folder) of the hierarchies. Files supplied as part of the distribution are put into the standard hierarchy. The location of the standard hierarchy is system dependent, but on a Unix system it might be at /usr/local/texmf, or /usr/local/share/texmf, or /opt/texmf, or a similar location, but in each case the \TeX files will be under the /texmf subdirectory.

There can be multiple ‘local’ hierarchies in which additional files can be put. In the extreme an installation can have a local hierarchy and each user can also have an individual local hierarchy. The location of any local hierarchy is not only system dependent but also user dependent. Again, though, all files should be put under a local /texmf directory.

TDS specification: tds/draft-standard/tds-0.9996

44 What is “Encapsulated PostScript”?

PostScript has over the years become a lingua franca of powerful printers; since PostScript is also a powerful graphical programming language, it is commonly used as an output medium for drawing (and other) packages.

However, since PostScript is such a powerful language, some rules need to be imposed, so that the output drawing may be included in a document as a figure without “leaking” (and thereby destroying the surrounding document, or failing to draw at all).

Appendix H of the PostScript Language Reference Manual (second and subsequent editions), specifies a set of rules for PostScript to be used as figures in this way. The important features are:

- certain “structured comments” are required: important ones are the identification of the file type, and information about the “bounding box” of the figure (i.e., the minimum rectangle enclosing it);
- some commands are forbidden — for example, a showpage command will cause the image to disappear, in most \TeX-output environments; and
- “preview information” is permitted, for the benefit of things such as word processors that don’t have the ability to draw PostScript in their own right — this preview information may be in any one of a number of system-specific formats, and any viewing program may choose to ignore it.

A PostScript figure that conforms to these rules is said to be in “Encapsulated PostScript” format. Most (\LaTeX) packages for including PostScript are structured to use Encapsulated PostScript; which of course leads to much hilarity as exasperated (\LaTeX) users struggle to cope with the output of drawing software whose authors don’t know the rules.
E  Acquiring the Software

45 Repositories of \TeX{} material

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, almost everything mentioned in this FAQ is archived at the CTAN sites (see the lists of software at the end of the questions).

The CTAN sites are currently dante.ctan.org (Mainz, Germany), cam.ctan.org (Cambridge, UK) and tug.ctan.org (Colchester, Vermont, USA). The organisation of \TeX{} files on all CTAN sites is identical and starts at `tex-archive/`. Each CTAN node may also be accessed via the Web at URLs http://www.dante.de/tex-archive, http://www.tex.ac.uk/tex-archive and http://www.ctan.org/tex-archive respectively; not all CTAN mirrors are Web-accessible. As a matter of course, to reduce network load, please use the CTAN site or mirror closest to you. A complete and current list of CTAN sites and known mirrors may be obtained by using the `finger` utility on 'user' ctan@cam.ctan.org, ctan@dante.ctan.org or ctan@tug.ctan.org; it is also available as file CTAN.sites on the archives themselves.

For details of how to find files at CTAN sites, see questions 49 (searching by ftp) and 50 (Web searching).

The email servers ftpmail@dante.ctan.org and ftpmail@tug.ctan.org provide an ftp-like interface through mail. Send a message containing just the line 'help' to your nearest server, for details of use.

The \TeX{} user who has no access to any sort of network may buy a copy of the archive on CD-ROM (see question 52).

46 What’s the CTAN nonfree tree?

The CTAN archives are currently restructuring their holdings so that files that are ‘not free’ are held in a separate tree. The definition of what is ‘free’ (for this purpose) is influenced by, but not exactly the same as the Debian Free Software Guidelines (DFSG).

Material is placed on the nonfree tree if it is not freely-usable (e.g., if the material is shareware, commercial, or if its usage is not permitted in certain domains at all, or without payment). Users of the archive should check that they are entitled to use material they have retrieved from the nonfree tree.

The Catalogue (one of the prime sources for finding \TeX{}-related material via web search — see question 50) lists the licence details in each entry in its lists. The catalogue also provides For details of the licence categories, see its list of licences.

47 Contributing a file to the archives

If you are able to use anonymous ftp, get yourself a copy of the file README.uploads from the root directory of any CTAN archive (see question 45). The file tells you where to upload, what to upload, and how to notify the CTAN management about what you want doing with your upload.

You may also upload via the Web: each of the principle CTAN sites offers an upload page — choose from http://www.ctan.org/upload.html, http://www.dante.de/upload.html or http://www.tex.ac.uk/upload.html; the pages lead you through the process, showing you the information you need to supply.

If you can use neither of these methods, ask advice of the CTAN management: if the worst comes to the worst, it may be possible to mail a contribution.

You will make everyone’s life easier if you choose a descriptive and unique name for your submission. Descriptiveness is in the eye of the beholder, but do try and be reasonable; and it’s probably a good idea to check that your chosen name is not already in use by browsing the archive (see question 49), or the Catalogue (see question 50).

If it’s appropriate (if your package is large, or regularly updated), the CTAN management can arrange to mirror your contribution direct into the archive. At present this may only be done if your contribution is available via ftp, and of course the directory structure on your archive must ‘fit’.

24
Finding (b)\TeX macro packages

Before you ask for a \TeX macro or \LaTeX class or package file to do something, try searching Graham Williams’ catalogue. You can also search the catalogue over the web (see question 50).

If you have learnt of a file, by some other means, that seems interesting, search a CTAN archive for it (see question 49). For packages listed in The \LaTeX Companion (see question 22) the file may be consulted as an alternative to searching the archive’s index. It lists the current location in the archive of such files.

Graham Williams’ catalogue: help/Catalogue/catalogue.html

\LaTeX Companion: info/companion.ctan

Finding files in the CTAN archives

To find software at a CTAN site, you can use anonymous \texttt{ftp} to the host with the command ‘quote site index \lt<\text{term}\gt\rt’, or the searching script at \url{http://www.dante.de/cgi-bin/ctan-index}

To get the best use out of the \texttt{ftp} facility you should remember that \lt<\text{term}\gt\rt is a \textit{Regular Expression} and not a fixed string, and also that many files are distributed in source form with an extension different to the final file. (For example \LaTeX\ packages are often distributed sources with extension \texttt{dtx} rather than as package files with extension \texttt{sty}.)

One should make the regular expresion general enough to find the file you are looking for, but not too general, as the \textit{ftp} interface will only return the first 20 lines that match your request.

The following examples illustrate these points. To search for the \LaTeX package ‘caption’, you might use the command:

\begin{verbatim}
quote site index caption.sty
\end{verbatim}

but it will fail to find the desired package (which is distributed as \texttt{caption.dtx}) and does return unwanted ‘hits’ (such as \texttt{hangcaption.sty}). Also, although this example does not show it the ‘.’ in ‘caption.sty’ is used as the regular expression that matches \textit{any} character. So

\begin{verbatim}
quote site index doc.sty
\end{verbatim}

matches such unwanted files as \texttt{language/swedish/slatex/doc2sty/makefile}

Of course if you know the package is stored as \texttt{.dtx} you can search for that name, but in general you may not know the extension used on the archive. The solution is to add ‘/’ to the front of the package name and ‘\.’ to the end. This will then search for a file name that consists solely of the package name between the directory separator and the extension. The two commands:

\begin{verbatim}
quote site index /caption\.
quote site index /doc\.
\end{verbatim}

do narrow the search down sufficiently. (In the case of doc, a few extra files are found, but the list returned is sufficiently small to be easily inspected.)

If the search string is too wide and too many files would match, the list will be truncated to the first 20 items found. Using some knowledge of the CTAN directory tree you can usually narrow the search sufficiently. As an example suppose you wanted to find a copy of the \texttt{dvips} driver for MS-DOS. You might use the command:

\begin{verbatim}
quote site index dvips
\end{verbatim}

but the result would be a truncated list, not including the file you want. (If this list were not truncated 412 items would be returned!) However we can restrict the search to MS-DOS related drivers as follows.

\begin{verbatim}
quote site index mdos.*dvips
\end{verbatim}

Which just returns relevant lines such as \texttt{systems/msdos/dviware/dvips/dvips5528.zip}

A basic introduction to searching with regular expressions is:

- Most characters match themselves, so "a" matches "a" etc.;
- "." matches any character;
- "[abcD-F]" matches any single character from the set {"a", "b", "c", "D", "E", "F"};
- "*" placed after an expression matches zero or more occurrences so "a*" matches "a" and "aaaa", and "[a-zA-Z]*" matches a ‘word’;
• "\" quotes a special character such as "." so "." just matches ".".
• "-" matches the beginning of a line;
• "." matches the end of a line.

For technical reasons in the quote site index command, you need to ‘double’ any \ hence the string /caption\., in the above example. The quote site command ignores the case of letters. Searching for caption or CAPTION would produce the same result.

50 Finding files by Web search
Two of the CTAN web servers offer a search facility: http://www.tex.ac.uk/search and http://www.ctan.org/search; you can look for a file whose name you already know (in pretty much the same way as the ftp-based quote site index command — see question 49), or you can do a keyword-based search of the catalogue.

The search script produces URLs for files that match your search criteria. The URLs point to the CTAN site or mirror of your choice; when you first use the script, it asks you to choose a site, and stores its details in a cookie on your machine. Choose a site that is close to you, to reduce network load.

51 Finding new fonts
A comprehensive list of METAFONT fonts used to be posted to comp.fonts and to comp.text.tex, roughly every six weeks, by Lee Quin.

Nowadays, authors of new material in METAFONT are few and far between (and mostly designing highly specialised things with limited appeal to ordinary users). Most new fonts that appear are prepared in some scalable outline form or other (see question 85), and they are almost all distributed under commercial terms.

METAFONT font list: info/metafont-list

52 \TeX\ CD-ROMs
If you don’t have access to the Internet, there are obvious attractions to \TeX\ collections on a CD-ROM. Even those with net access will find large quantities of \TeX\-related files to hand a great convenience.

Ready-to-run \TeX\ systems on CD-ROM are available:

• A consortium of User Groups (notably TUG, UK TUG and GUTenberg) distributes the \TeX\ Live CD-ROM, now in its sixth edition. All members of several User Groups receive copies free of charge. Some user groups will also sell additional copies: contact your local user group or TUG (see question 21). Details of \TeX\ Live are available from its own web page on the TUG site.
• The Dutch \TeX\ Users Group (NTG) publish the whole 4All\TeX\ workbench on a 2-CD-ROM set packed with all the Windows \TeX\ software, macros and fonts you can want. It is available from NTG direct, from TUG for $40 and from UK TUG for £30 (a manual is included). It is a useful resource for anyone to browse, not just for Windows users.

An alternative to the ready-to-run system is the CTAN archive snapshot; in general one would expect that such systems would be harder to use, but that the volume of resources offered would balance this extra inconvenience. There were once commercial offerings in this field, but nowadays the snapshot supplied to user group members annually is about the only source of such things.

F \TeX\ Systems

53 \TeX\ for different machines
We list here the free or shareware packages; see see question 55 for details of commercial packages.

Unix Instructions for retrieving the web2c Unix \TeX\ distribution via anonymous ftp are to be found in unix.tex.ftp, though nowadays the sensible installer will take (and possibly customise) one of the packaged distributions such as \TeX\ (which often has a more recent version of web2c embedded than has been released "in the wild"), or the \TeX\ Live CD-ROM (see question 52).
For `\texttt{\TeX}` you need at most one each of the `.tar.gz` files for `\texttt{\TeX}-src`, `\texttt{\TeX}-texmf` and `\texttt{\TeX}-texmfsrc`.

Sets of binaries for many common Unix systems are to be found as part of the `\texttt{\TeX}` distribution, or on the `\texttt{\TeX}` Live CD-ROM. There are rather more to be found on CTAN; you’ll find compressed `.tar` archive for each supported architecture in the directory. In default of a precompiled version, `\texttt{\TeX}` will compile on most Unix systems, though it was originally developed for use under Linux (see below).

MacOS X users should refer to the information below, under item “Mac”.

---

Linux

There are at least two respectable implementations of `\texttt{\TeX}` to run on Linux, \texttt{\LaTeX} and `\texttt{\TeX}` (see above). Beware the Slackware '96 CD-ROM distribution of \texttt{\LaTeX}: it includes a version of the CM fonts that has deeply offended Don Knuth (since it contravenes his distribution conditions). The Slackware updates now offer `\texttt{\TeX}`, as do most Linux distributions. The most recent offering is a free version of the commercial \texttt{\LaTeX} (see question 55), which specialises in direct production of PDF from (\LaTeX) input.

---

PC; MS-DOS or OS/2

`\texttt{\LaTeX}`, by Eberhard Mattes, includes \texttt{\LaTeX}, \texttt{\BIBL\LaTeX}, previewers, and drivers, and is available as a series of zip archives. Documentation is available in both German and English. Appropriate memory managers for using \texttt{\LaTeX} with 386 (and better) processors and under Windows, are included in the distribution. \texttt{\LaTeX} will operate under Windows, but Windows users are better advised to use a distribution tailored for the Windows environment. A version of \texttt{\LaTeX}, packaged to use a TDS directory structure (see question 43), is separately available as an `\texttt{\LaTeX}` ‘contribution’.

---

PC: Win32

`\texttt{\LaTeX}`, by Fabrice Popineau, is a version of `\texttt{\LaTeX}` for Windows systems. As such, it is particularly attractive to those who need to switch back and forth between Windows and Unix environments, and to administrators who need to maintain both (\texttt{\LaTeX} can use the same `\texttt{\texmf}` tree as a `\texttt{\TeX}` installation). \texttt{\LaTeX}’s previewer (`\texttt{Windvi}`) is based on `\texttt{xdvi}`, and takes advantage of extra facilities in the Win32 environment. `\texttt{Windvi}` is capable of printing directly, and a version of `\texttt{dvips}` is also available.

\texttt{\LaTeX}, by Christian Schenk, is also a comprehensive distribution, developed separately from the `\texttt{\LaTeX}` work. It has its own previewer, YAP, which is itself capable of printing, though the distribution also includes a port of `\texttt{dvips}`. The current version is available for file-by-file download (the HTML files in the directory offer hints on what you need to get going). A prepackaged version of the whole directory is also available.

A further (free) option arises from the “\texttt{CygWin}” bundle, which presents a Unix-like environment over the Win32 interface; an X-windows server is available. If you run CygWin on your Windows machine, you have the option of using `\texttt{\LaTeX}`.
too (you will need the X-server, to run xdvi). Of course, \TeX\ components will look like Unix applications (but that’s presumably what you wanted), but it’s also reputedly somewhat slower than native Win32 implementations such as MikTEX or fp\TeX. \TeX\ is available as part of the CygWin distribution (in the same way that a version is available with most Linux distributions, nowadays), and you may also build your own copy from the current sources.

BaKoMa \TeX, by Basil Malyshev, is a comprehensive (shareware) distribution, which focuses on support of Acrobat. The distribution comes with a bunch of Type 1 fonts packaged to work with BaKoMa \TeX, which further the focus.

\texttt{bakoma: nonfree/systems/win32/bakoma}

\texttt{fp\TeX: systems/win32/fptex}

\texttt{mikTeX: Acquire systems/win32/miktex/setup/setup.exe and read systems/win32/miktex/setup/install.html}

\texttt{tetex: systems/unix/teTeX/1.0/distrib/sources}

\textbf{Windows NT, other platforms} Ports of \MikTeX\ for NT on Power PC and AXP are available. Neither version has been updated for version 1.2 (or later) of \MikTeX\ — they may not be satisfactory.

\texttt{mikTeX for AXP: systems/win32/miktext-AXP}

\texttt{mikTeX for Power PC: systems/win32/miktexppc}

\textbf{Mac} Oz\TeX, by Andrew Trevorrow, is a shareware version of \TeX\ for the Macintosh. A DVI previewer and PostScript driver are also included. UK TUG preps the shareware fee, so that its members may acquire the software without further payment. Questions about Oz\TeX\ may be directed to \texttt{oztex@midway.uchicago.edu}

Another partly shareware program is CMac\TeX, put together by Tom Kiffe. This is much closer to the Unix \TeX\ setup (it uses dvips, for instance). CMac\TeX\ includes a port of the latest version of Omega (see question 251). Both Oz\TeX\ and CMac\TeX\ are additionally available on MacOS X, but OS X users also have the option of a build of \TeX\ by Gerben Wierda. This is naturally usable from the command line, just like any other Unix-based system, but it can also be used Mac-style as the engine behind Richard Koch’s (free) \TeX\Shop, which is an integrated \TeX\ editor and previewer.

A useful resource for Mac users has a news and ‘help’ section, as well as details of systems and tools. A useful resource for Mac users has a news and ‘help’ section, as well as details of systems and tools.

\texttt{cmactex: systems/mac/cmactex}

\texttt{oztex: nonfree/systems/mac/oztex}

\texttt{MacOS X \TeX\Shop: ftp://ftp.nluug.nl/pub/comp/macosx/tex-gs/}

\texttt{\TeX\Shop: http://darkwing.uoregon.edu/~koch/texshop/texshop.html}

\textbf{OpenVMS} \TeX\ for OpenVMS is available.

\texttt{OpenVMS: systems/OpenVMS/TEX97_CTAN.ZIP}

\textbf{Atari} \TeX\ is available for the Atari ST.

If anonymous \texttt{ftp} is not available to you, send a message containing the line ‘help’ to \texttt{atari@atari.archive.umich.edu}

\texttt{Atari TeX: systems/atari}

\textbf{Amiga} Full implementations of \TeX\ 3.1 (Pas\TeX) and METAFONT 2.7 are available.

\texttt{PasTeX: systems/amiga}

\textbf{TOPS-20} \TeX\ was originally written on a DEC-10 under WAITS, and so was easily ported to TOPS-20. A distribution that runs on TOPS-20 is available via anonymous \texttt{ftp} from \texttt{ftp.math.utah.edu} in \texttt{pub/tex/pub/web}

54 \TeX\-friendly editors and shells

There are good \TeX\-writing environments and editors for most operating systems; some are described below, but this is only a personal selection:

\textbf{Unix} Try GNU \texttt{emacs} or \texttt{xemacs}, and the AUC-\TeX\ mode (AUC-\TeX\ is available from
CTAN, but *emacs* itself isn’t). AUC-\TeX{} provides menu items and control sequences for common constructs, checks syntax, lays out markup nicely, lets you call \TeX{} and drivers from within the editor, and everything else like this that you can think of. Complex, but very powerful.

*Nedit* is another free, programmable, editor available for Unix systems. An AUC-\TeX{}-alike package for *Nedit* is available from CTAN.

**MS-DOS** There are several choices:
- \TeX{}shell is a simple, easily-customisable environment, which can be used with the editor of your choice.
- Eddi4\TeX{} (also shareware) is a specially-written \TeX{} editor which features intelligent colouring, bracket matching, syntax checking, online help and the ability to call \TeX{} programs from within the editor. It is highly customisable, and features a powerful macro language.

You can also use GNU *emacs* and AUC-\TeX{} under MS-DOS.

**Windows ’9x, NT, etc.** *Winedt*, a shareware package, is highly spoken of. It provides a shell for the use of \TeX{} and related programs, as well as a powerful and well-configured editor.

The 4Alt\TeX{} CD-ROM (see question 52) contains another powerful Windows-based shell.

**OS/2** Eddi4\TeX{} works under OS/2; an alternative is *epmtex*, which offers an OS/2-specific shell.

**Macintosh** The commercial *Textures* provides an excellent integrated Macintosh environment with its own editor. More powerful still (as an editor) is the shareware *Alpha* which is extensible enough to let you perform almost any \TeX{}-related job. It works well with Ot\TeX{}.

Atari, Amiga and NeXT users also have nice environments. \Bib\TeX{} users looking for *make*-like facilities should try *latexmk*.

There is another set of shell programs to help you manipulate \Bib\TeX{} databases.

```
alpha: systems/mac/support/alpaha
auctex: support/auctex
epmtex: systems/os2/epmtex
latexmk: support/latexmk
nedit latex support: support/NEdit-LaTeX-Extensions
TeXshell: systems/msdos/texshell
TeXtelExtel: systems/msdos/emtex-contrib/TeXtelExtel
winedt: systems/win32/winedt/winedt32.exe
```

**55 Commercial \TeX{} implementations**

There are many commercial implementations of \TeX{}. The first appeared not long after \TeX{} itself appeared.

What follows is probably an incomplete list. Naturally, no warranty or fitness for purpose is implied by the inclusion of any vendor in this list. The source of the information is given to provide some clues to its currency.

In general, a commercial implementation will come ‘complete’, that is, with suitable previewers and printer drivers. They normally also have extensive documentation (*i.e.*, not just the \TeX{}book!) and some sort of support service. In some cases this is a toll free number (probably applicable only within the USA and or Canada), but others also have email, and normal telephone and fax support.

**PC; **\TeX{} Runs on all versions of Windows.

Richard J. Kinch
TrueTeX Software
6994 Pebble Beach Court
Lake Worth FL 33467
USA
Tel: +1 561-966-8400
Email: kinch@truetex.com
Web: http://www.truetex.com/
PC; **Y\&Y \TeX**  “Bitmap free \TeX for Windows.”

Y\&Y, Inc.
106 Indian Hill, MA 01741
USA
Tel: 800-742-4059 (within North America)
Tel: +1 978-371-3286
Fax: +1 978-371-2004
Email: sales-help@YandY.com and
tech-help@YandY.com

Source: Mail from Y\&Y, July 2001

**pc\TeX** Long-established: pc\TeX32 is a Windows implementation.

Personal \TeX Inc
12 Madrona Street
Mill Valley, CA 94941
USA
Tel: 800-808-7906 (within the USA)
Fax: +1 415-388-8865
Email: texsales@pctex.com and
texsupp@pctex.com
Web: [http://www.pctex.com/](http://www.pctex.com/)

Source: Mail from Personal \TeX Inc, September 1997

**PC; \VT\TeX** DVI, PDF and HTML backends, Visual Tools and Type 1 fonts

MicroPress Inc
68-30 Harrow Street
Forest Hills, NY 11375
USA
Tel: +1 718-575-1816
Fax: +1 718-575-8038
Email: support@micropress-inc.com

Source: Mail from MicroPress, Inc., July 1999

**PC; Scientific Word** Scientific Word and Scientific Workplace offer a mechanism for near-WYSIWYG input of \LaTeX documents; they ship with True\TeX from Kinch (see above). Queries within the UK and Ireland should be addressed to Scientific Word Ltd., others should be addressed directly to the publisher, MacKichan Software Inc.

Dr Christopher Mabb
Scientific Word Ltd.
49 Queen Street
Peterhead
Aberdeenshire, AB42 1TU
UK
Tel: 0845 766 0340 (within the UK)
Tel: +44 1779 490500
Fax: 01779 490600 (within the UK)
Email: christopher@sciword.demon.co.uk
Web: [http://www.sciword.demon.co.uk](http://www.sciword.demon.co.uk)

MacKichan Software Inc.
600 Ericksen Ave. NE, Suite 300
Bainbridge Island WA 98110
USA
Tel: +1 206 780 2799
Fax: +1 206 780 2857
Email: info@mackichan.com
Web: [http://www.mackichan.com](http://www.mackichan.com)
**Macintoshs; Textures** “A \TeX{} system ‘for the rest of us’”; also gives away a *META-FONT* implementation and some font manipulation tools.

Blue Sky Research  
534 SW Third Avenue  
Portland, OR 97204  
USA  
Tel: 800-622-8398 (within the USA)  
Tel: +1 503-222-9571  
Fax: +1 503-222-1643  
Email: sales@bluesky.com  
Web: [http://www.bluesky.com/](http://www.bluesky.com/)

**Amiga\TeX{}** A full implementation for the Commodore Amiga, including full, on-screen and printing support for all PostScript graphics and fonts, IFF raster graphics, automatic font generation, and all of the standard macros and utilities.

Radical Eye Software  
PO Box 2081  
Stanford, CA 94309  
USA

Source: Mail from Tom Rokicki, November 1994

### G DVI Drivers and Previewers

#### 56 DVI to PostScript conversion programs

The best public domain DVI to PostScript conversion program, which runs under many operating systems, is Tom Rokicki’s *dvips*. *dvips* is written in C and ports easily. All current development is in the context of Karl Berry’s *kpathsea* library, and sources are available from the \TeX{} live repository.

An VMS versions is available as part of the CTAN distribution of \TeX{} for VMS.

A precompiled version to work with em\TeX{} is available on CTAN.

Macintosh users can use either the excellent drivers built into Oz\TeX{} or Textures, or a port of *dvips* in the CMac\TeX{} package.

*MS-DOS and OS/2*: [systems/msdos/dviware/dvips](http://www.ctan.org)  
*VMS distribution*: [systems/OpenVMS/TEX97_CTAN.ZIP](http://www.ctan.org)

#### 57 DVI drivers for HP LaserJet

The em\TeX{} distribution (see question 53) contains a driver for the LaserJet, *dvihplj*.

Version 2.10 of the Beebe drivers supports the LaserJet. These drivers will compile under Unix, VMS, and on the Atari ST and DEC-20s.

For Unix systems, Karl Berry’s *dvijl* uses the same path-searching library as *web2c*. *dvijl* is no longer distributed as a separate source, but the te\TeX{} distribution holds a copy under the name *dvijl*.

*Beebe drivers*: [dviware/beebe](http://www.ctan.org)

#### 58 Output to “other” printers

In the early years of \TeX{}, there were masses of DVI drivers for any (then) imaginable kind of printer, but the steam seems rather to have gone out of the market for production of such drivers for printer-specific formats. There are several reasons for this, but the primary one is that few formats offer the flexibility available through PostScript, and *ghostscript* is so good, and has such a wide range of printer drivers (perhaps this is where the DVI output driver writers have all gone?).

The general advice, then, is to generate PostScript (see question 56), and to process that with *ghostscript* set to generate the format for the printer you actually have. If you are using a Unix system of some sort, it’s generally quite easy to insert *ghostscript* into the print spooling process.

*ghostscript*: Browse [nonfree/support/ghostscript](http://www.ctan.org)
DVI previewers

EmTeX for PCs running MS-DOS or OS/2, MikTeX and fpTeX for PCs running Windows and OZTeX for the Macintosh, all come with previewers that can be used on those platforms. EmTeX’s previewer can also be run under Windows 3.1.

Commercial PC TeX packages (see question 55) have good previewers for PCs running Windows, or for Macintosches.

For Unix systems, there is one ‘canonical’ viewer, xdvi. Xdvik is a version of xdvi using the web2c libraries. Unix TeX distributions (such as teTeX or NTeX) include a version of xdvik using the same version of web2c as the rest of the distribution.

Alternatives to previewing include

- conversion to ‘similar’ ASCII text (see question 67) and using a conventional text viewer to look at that,
- generating a PostScript version of your document and viewing it with a Ghostscript-based previewer (see question 82), and
- generating PDF output, and viewing that with Acrobat Reader or one of the substitutes for that.

H Support Packages for TeX

Fig, a TeX-friendly drawing package

(X)Fig is a menu driven tool that allows you to draw objects on the screen of an X workstation; transfig is a set of tools which translate the code fig produces to other graphics languages including PostScript and the LATEX picture environment.

Fig and transfig are maintained by Brian Smith. Another tool for fig conversion is fig2mf which generates METAFONT code from fig input.

There’s no explicit port of xfig to windows (although it is believed to work under cygwin with their X-windows system). However, the program jfig is thought by many to be an acceptable substitute, written in Java.

fig2mf: graphics/fig2mf
xfig: graphics/xfig
transfig: graphics/transfig

TeXCAD, a drawing package for LATEX

TeXCAD is a program for the PC which enables the user to draw diagrams on screen using a mouse or arrow keys, with an on-screen menu of available picture-elements. Its output is code for the LATEX picture environment. Optionally, it can be set to include lines at all angles using the emTeX driver-family \specials (see question 38). TeXCAD is part of the emTeX distribution.

A Unix port of the program (xtexcad) has been made.

emtex: systems/msdos/emtex
xtexcad: nonfree/graphics/xtexcad/xtexcad-2.4.1.tar.gz

Spelling checkers for work with TeX

For Unix, ispell is probably the program of choice; it is well integrated with the emacs, and deals with some TeX syntax.

For Windows, there is a good spell checker incorporated into the WinEDT and 4AllTeX shell/editors (see question 54). The 4AllTeX checker is also available as a separate package, 4spell.

For the Macintosh, Excalibur is the program of choice. It will run in native mode on both sorts of Macintosh. The distribution comes with dictionaries for several languages.

The VMS Pascal program spell makes special cases of some important features of LATEX syntax.

For MS-DOS, there are several programs. Amspell can be called from within an editor, and jspell is an extended version of ispell.
4spell: support/4spell
amspell: support/amspell
excalibur: systems/mac/support/excalibur/Excalibur-3.0.2.hqx
ispell: Browse support/ispell, but beware of any version with a number 4.x — such versions represent a divergent version of the 3.x series.
jspell: support/jspell
VMS spell: support/vmspell
winedt: systems/win32/winedt/winedt32.exe

63 How many words have you written?

One often has to submit a document (e.g., a paper or a dissertation) under some sort of constraint about its size. Sensible people set a constraint in terms of numbers of pages, but there are some that persist in limiting the numbers of words you type.

A simple solution to the requirement can be achieved following a simple observation: the powers that be are unlikely to count all the words of a document submitted to them. Therefore, a statistical method can be employed: find how many words there are on a full page; find how many full pages there are in the document (allowing for displays of various sorts, this number will probably not be an integer); multiply the two. However, if the document to be submitted is to determine the success of the rest of one’s life, it takes a brave person to thumb their nose at authority quite so comprehensively...

The simplest method is to strip out the (LATEX) markup, and to count what’s left. On a Unix-like system, this may be done using detex and the built-in wc:

detex <filename> | wc -w

Winedt (see question 54) in the Windows environment provides this functionality direct.

Simply stripping (LATEX) markup isn’t entirely reliable, however: that markup itself may contribute typeset words, and this could be a problem. The wordcount package contains a Bourne shell (i.e., typically Unix) script for running a LATEX file with a special piece of supporting LATEX code, and then counting word indications in the log file. This is probably as accurate automatic counting as you can get.

detex: support/detex
wordcount: macros/latex/contrib/supported/wordcount

I Literate programming

64 What is Literate Programming?

Literate programming is the combination of documentation and source together in a fashion suited for reading by human beings. In general, literate programs combine source and documentation in a single file. Literate programming tools then parse the file to produce either readable documentation or compilable source. The WEB style of literate programming was created by D. E. Knuth during the development of LATEX.

The “documented LATEX” style of programming (see question 39) is regarded by some as a form of literate programming, though it only contains a subset of the constructs Knuth used.

Discussion of literate programming is conducted in the newsgroup comp.programming.
literate, whose FAQ is stored on CTAN. Another good source of information is http://www.literateprogramming.com/

Literate Programming FAQ: help/LitProg-FAQ

65 WEB systems for various languages

LATEX is written in the programming language WEB; WEB is a tool to implement the concept of “literate programming”. Knuth’s original implementation will be in any respectable distribution of LATEX, but the sources of the two tools (tangle and weave), together with a manual outlining the programming techniques, may be had from CTAN.

CWEB, by Silvio Levy, is a WEB for C programs.
Spidery WEB, by Norman Ramsey, supports many languages including Ada, awk, and C and, while not in the public domain, is usable without charge.

*FWEB*, by John Krommes, is a version for Fortran, Ratfor, and C.

*SchemeWEB*, by John Ramsdell, is a Unix filter that translates SchemeWEB into \LaTeX source or Scheme source.

*APLWEB* is a version of WEB for APL.

*FunnelWeb* is a version of WEB that is language independent.

Other language independent versions of WEB are *nuweb* (which is written in ANSI C) and *noweb*.

*Tweb* is a WEB for Plain \LaTeX macro files, using *noweb*.

```
aplweb: web/apl/aplweb
cweb: web/c_cpp/cweb
funnelweb: web/funnelweb
fweb: web/fweb
noweb: web/noweb
nuweb: web/nuweb
schemeweb: web/schemeweb
spiderweb: web/spiderweb
tangle: systems/knuth/web
tweb: web/tweb
weave: systems/knuth/web
```

### J Format conversions

66 Conversion between \LaTeX and others

*troff* *troff-to-latex*, written by Kamal Al-Yahya at Stanford University (California, USA), assists in the translation of a *troff* document into \LaTeX format. It recognises most -ms and -man macros, plus most *eqn* and some *tbl* preprocessor commands. Anything fancier needs to be done by hand. Two style files are provided. There is also a man page (which converts very well to \LaTeX...). The program is copyrighted but free. *tr2latex* is an enhanced version of this *troff-to-latex*.

**WordPerfect** *wp2latex* has recently been much improved, and is now available either for MS-DOS or for Unix systems, thanks to its current maintainer Jaroslav Fojtík.

**PC-Write** *pcwrite.arc* is a print driver for PC-Write that “prints” a PC-Write V2.71 document to a \LaTeX-compatible disk file. It was written by Peter Flynn at University College, Cork, Republic of Ireland.

**runoff** Peter Vanroose’s *rnototex* conversion program is written in VMS Pascal. The sources are distributed with a VAX executable.

**refer/tib** There are a few programs for converting bibliographic data between Bib\LaTeX and *refer/tib* formats. The collection includes a shell script converter from Bib\LaTeX to *refer* format as well. The collection is not maintained.

**RTF** *Rtf2tex*, by Robert Lupton, is for converting Microsoft’s Rich Text Format to \LaTeX. There is also a converter to \LaTeX by Erwin Wechtl, called *rtf2latex*. The latest converter, by Ujwal Sathyam and Scott Prahl, is *rtf2latex2e*; this system seems rather good already, and is still being improved.

Translation to RTF may be done (for a somewhat constrained set of \LaTeX documents) by *\LaTeX2RTF*, which can produce ordinary RTF, Windows Help RTF (as well as HTML, see question 69). *\LaTeX2RTF* is supported on various Unix platforms and under Windows 3.1

**Microsoft Word** A rudimentary program for converting MS-Word to \LaTeX is *wd2latex*, for MS-DOS; a better idea, however, is to convert the document to RTF format and use one of the RTF converters mentioned above.

**Excel** *Excel2Latex* converts an Excel file into a \LaTeX tabular environment; it comes as a .xls file which defines some *Excel* macros to produce output in a new format.
A separate FAQ by Wilfried Hennings deals specifically with conversions between TeX-based formats and word processor formats may be referred to for more detailed information.

A group at Ohio State University (USA) is working on a common document format based on SGML, with the ambition that any format could be translated to or from this one. FrameMaker provides “import filters” to aid translation from alien formats (presumably including TeX) to FrameMaker’s own.

\texttt{excel2latex: support/excel2latex/xl2latex.zip}
\texttt{pcwritex.arc: support/pcwritex}
\texttt{refer and bib tools: biblio/bibtex/utils/refer-tools}
\texttt{rnototex: support/rnototex}
\texttt{rtf2latex: support/rtf2latex}
\texttt{rtf2latex2e: support/rtf2latex2e}
\texttt{rtf2tex: support/rtf2tex}
\texttt{tex2rtf: support/tex2rtf}
\texttt{tr2latex: support/tr2latex}
\texttt{troff-to-latex: support/troff-to-latex}
\texttt{wd2latex: dviware/wd2latex}
\texttt{wp2latex: support/wp2latex}

Word processor FAQ: \url{http://www.tug.org/utilities/texconv/index.html}
Word processor FAQ (source): \url{help/wp-conv/wp-conv.zip}

67 Conversion from \(\LaTeX\) to plain ASCII

The aim here is to emulate the Unix \texttt{nroff}, which formats text as best it can for the screen, from the same input as the Unix typesetting program \texttt{troff}.

Converting DVI to plain text is the basis of many of these techniques; sometimes the simple conversion provides a good enough response. Options are:

- \texttt{dvi2tty} (one of the earliest)
- \texttt{crudetype}
- \texttt{catdvi}, which is also capable of generating Latin-1 or UTF-8 encoded output.
  
  \texttt{Catdvi} was conceived as a replacement for \texttt{dvi2tty}, but can’t (quite) be recommended as a complete replacement yet.

Ralph Droms provides a \texttt{txt} bundle of things in support of ASCII generation, but it doesn’t do a good job with tables and mathematics. An alternative is the \texttt{screen} package.

Another possibility is to use the \LaTeX-to-ASCII conversion program, \texttt{l2a}, although this is really more of a de-\LaTeX{}Xing program.

The canonical de-\LaTeX{}Xing program is \texttt{detex}, which removes all comments and control sequences from its input before writing it to its output. Its original purpose was to prepare input for a dumb spelling checker, and it’s only usable for preparing useful ASCII versions of a document in highly restricted circumstances.

\texttt{Tex2mail} is slightly more than a de-TeXer — it’s a Perl script that converts \TeX{} files into plain text files, expanding various mathematical symbols (sums, products, integrals, sub/superscripts, fractions, square roots, …) into “ASCII art” that spreads over multiple lines if necessary. The result is more readable to human beings than the flat-style \TeX{} code.

Another significant possibility is to use one of the HTML-generation solutions (see question 69), and then to use a browser such as \texttt{lynx} to dump the resulting HTML as plain text.

\texttt{catdvi: dviware/catdvi}
\texttt{crudetype: dviware/crudetype}
\texttt{detex: support/detex}
\texttt{dvi2tty: nonfree/dviware/dvi2tty}
\texttt{l2a: support/l2a}
Conversion from SGML or HTML to \TeX

SGML is a very important system for document storage and interchange, but it has no formatting features; its companion ISO standard DSSSL (see http://www.jclark.com/dsssl/) is designed for writing transformations and formatting, but this has not yet been widely implemented. Some SGML authoring systems (e.g., SoftQuad Author/Editor) have formatting abilities, and there are high-end specialist SGML typesetting systems (e.g., Miles33’s Genera). However, the majority of SGML users probably transform the source to an existing typesetting system when they want to print. \TeX is a good candidate for this. There are three approaches to writing a translator:

1. Write a free-standing translator in the traditional way, with tools like yacc and lex; this is hard, in practice, because of the complexity of SGML.
2. Use a specialist language designed for SGML transformations; the best known are probably Omnimark and Balise. They are expensive, but powerful, incorporating SGML query and transformation abilities as well as simple translation.
3. Build a translator on top of an existing SGML parser. By far the best-known (and free!) parser is James Clark’s nsgmls, and this produces a much simpler output format, called ESIS, which can be parsed quite straightforwardly (one also has the benefit of an SGML parse against the DTD). Two good public domain packages use this method:
   - David Megginson’s sgmlspm, written in Perl 5.
   - Joachim Schrod and Christine Detig’s stil, written in Common Lisp.

Both of these allow the user to write ‘handlers’ for every SGML element, with plenty of access to attributes, entities, and information about the context within the document tree.

If these packages don’t meet your needs for an average SGML typesetting job, you need the big commercial stuff.

Since HTML is simply an example of SGML, we do not need a specific system for HTML. However, Nathan Torkington developed html2latex from the HTML parser in NCSA’s Xmosaic package. The program takes an HTML file and generates a \TeX file from it. The conversion code is subject to NCSA restrictions, but the whole source is available as support/html2latex.

Michel Goossens and Janne Saarela published a very useful summary of SGML, and of public domain tools for writing and manipulating it, in TUGboat 16(2).

\LaTeX conversion to HTML

\TeX and \LaTeX are well suited to producing electronically publishable documents. However, it is important to realize the difference between page layout and functional mark-up. \TeX is capable of extremely detailed page layout; HTML is not, because HTML is a functional mark-up language not a page layout language. HTML’s exact rendering is not specified by the document that is published but is, to some degree, left to the discretion of the browser. If you require your readers to see an exact replication of what your document looks like to you, then you cannot use HTML and you must use some other publishing format such as PDF. That is true for any HTML authoring tool.

\TeX’s excellent mathematical capabilities remain a challenge in the business of conversion to HTML. There are only two generally reliable techniques for generating mathematics on the web: creating bitmaps of bits of typesetting that can’t be translated, and using symbols and table constructs. Neither technique is entirely satisfactory. Bitmaps lead to a profusion of tiny files, are slow to load, and are inaccessible to those with visual disabilities. The symbol fonts offer poor coverage of mathematics, and their use requires configuration of the browser. The future of mathematical browsing may be brighter — see question 254.

For today, possible packages are:

\LaTeX2HTML, a perl script package that supports \TeX only, and generates mathematics (and other “difficult” things) using bitmaps. The original version was written by Nikos Drakos for Unix systems, but the package is now also available.
for Windows systems. Michel Goossens and Janne Saarela published a detailed discussion of \LaTeX2HTML, and how to tailor it, in \textit{TUGboat 16}(2).

\textbf{\textit{TtH}} a compiled program that supports either \LaTeX{} or plain \TeX{}, and uses the font/table technique for representing mathematics. It is written by Ian Hutchinson, using \textit{flex}. The distribution consists of a single C source (or a compiled executable), which is easy to install and very fast-running.

\textbf{\textit{Tex4ht}} a compiled program that supports either \LaTeX{} or plain \TeX{}, by processing a DVI file; it uses bitmaps for mathematics, but can also use other technologies where appropriate. Written by Eitan Gurari, it parses the DVI file generated when you run (\LaTeX{}) over your file with tex4ht's macros included. As a result, it’s pretty robust against the macros you include in your document, and it’s also pretty fast.

\textbf{\textit{TeXspider}} a commercial program from MicroPress (see question 55), which is described on \url{http://www.micropress-inc.com/webb/wbstart.htm}; it uses bitmaps for equations.

\textbf{\textit{Hevea}} a compiled program that supports \LaTeX{} only, and uses the font/table technique for equations (indeed its entire approach is very similar to \textit{TtH}). It is written in Objective CAML by Luc Maranget. \textit{Hevea} isn’t archived on CTAN; details (including download points) are available via \url{http://pauillac.inria.fr/~maranget/hevea/}

The World Wide Web Consortium maintains a list of “filters” to HTML, with sections on (\LaTeX) and \textit{BIBTEX} — see \url{http://www.w3.org/Tools/Word_proc_filters.html}

\textit{latex2html}: Browse \url{support/latex2html}
\textit{tex4ht}: \url{support/Tex4ht/tex4ht.zip}
\textit{tth}: \url{nonfree/support/tth/dist/tth_C.tgz}

70 Using \TeX{} to read SGML or XML directly

This can nowadays be done, with a certain amount of clever macro programming. David Carlisle’s \textit{xmltex} is the prime example; it offers a practical solution to typesetting XML files.

One use of a \TeX{} that can typeset XML files is as a backend processor for XSL formatting objects, serialized as XML. Sebastian Rahtz’s Passive\TeX{} uses \textit{xmltex} to achieve this end.

\textit{xmltex}: \url{macros/xmltex/base}
\textit{passivetex}: \url{macros/xmltex/contrib/passivetex}

71 Retrieving (\LaTeX) from DVI, etc.

The job just can’t be done automatically: DVI, PostScript and PDF are “final” formats, supposedly not susceptible to further editing — information about where things came from has been discarded. So if you’ve lost your (\LaTeX) source (or never had the source of a document you need to work on) you’ve a serious job on your hands. In many circumstances, the best strategy is to retypew the whole document, but this strategy is to be tempered by consideration of the size of the document and the potential typists’ skills.

If automatic assistance is necessary, it’s unlikely that any more than text retrieval is going to be possible; the (\LaTeX) markup that creates the typographic effects of the document needs to be recreated by editing.

If the file you have is in DVI format, many of the techniques for converting (\LaTeX) to ASCII (see question 67) are applicable. Consider \textit{dvi2tty}, \textit{crudetype} and \textit{catdvi}. Remember that there are likely to be problems finding included material (such as included PostScript figures, that don’t appear in the DVI file itself), and mathematics is unlikely to convert easily.

To retrieve text from PostScript files, the \textit{ps2ascii} tool (part of the \textit{ghostscript} distribution) is available. One could try applying this tool to PostScript derived from an PDF file using \textit{pdf2ps} (also from the \textit{ghostscript} distribution), or \textit{Acrobat Reader} itself; an alternative is \textit{pdftotext}, which is distributed with \textit{xpdf}.

Another avenue available to those with a PDF file they want to process is offered by Adobe \textit{Acrobat} (version 5 or later): you can tag the PDF file into an structured
document, output thence to well-formed XHTML, and import the results into Microsoft Word (2000 or later). From there, one may convert to (L)\TeX using one of the techniques discussed in question 66.

The result will typically (at best) be poorly marked-up. Problems may also arise from the oddity of typical \TeX font encodings (notably those of the maths fonts), which Acrobat doesn’t know how to map to its standard Unicode representation.

catdvi: dviware/catdvi
crudetype: dviware/crudetype
dvi2tty: nonfree/dviware/dvi2tty
ghostscript: nonfree/support/ghostscript
xpdf: Browse support/xpdf

72 Translating $\LaTeX$ to Plain $\TeX$

Unfortunately, no “general”, simple, automatic process is likely to succeed at this task. See “How does $\LaTeX$ relate to Plain $\TeX$” (see question 11) for further details.

Translating a document designed to work with $\LaTeX$ into one designed to work with Plain $\TeX$ is likely to amount to carefully including (or otherwise re-implementing) all those parts of $\LaTeX$, beyond the provisions of Plain $\TeX$, which the document uses.

K Hypertext and PDF

73 Making hypertext documents from $\TeX$

If you want on-line hypertext with a (L)\TeX source, probably on the World Wide Web, there are four technologies to consider:

1. Direct (L)\TeX conversion to HTML (see question 69);
2. Use TeXinfo (see question 16), and use the info viewer, or convert the texinfo to HTML;
3. Use Adobe Acrobat, which will preserve your typesetting perfectly (see question 74);
4. The hyper$\TeX$ conventions (standardised \special commands); there are supporting macro packages for Plain $\TeX$ and $\LaTeX$).

The Hyper$\TeX$ project extended the functionality of all the $\LaTeX$ cross-referencing commands (including the table of contents) to produce \special commands which are parsed by DVI processors conforming to the Hyper$\TeX$ guidelines; it provides general hypertext links, including those to external documents.

The Hyper$\TeX$ specification says that conformant viewers/converters must recognize the following set of \special commands:

href: html:<a href = "href_string">
name: html:<a name = "name_string">
end: html:</a>
image: html:<img src = "href_string">
base_name: html:<base href = "href_string">

The href, name and end commands are used to do the basic hypertext operations of establishing links between sections of documents.

Further details are available on http://arXiv.org/hypertex/; there are two commonly-used implementations of the specification, a modified xdvi and (recent releases of) dvips. Output from the latter may be used in recent releases of ghostscript or Acrobat Distiller.

74 Making Acrobat documents from $\LaTeX$

There are three general routes to Acrobat output: Adobe’s original ‘distillation’ route (via PostScript output), conversion of a DVI file, and the use of a direct PDF generator such PDF$\TeX$ (see question 253) or MicroPress’s VT$\TeX$ (see question 55).

For simple documents (with no hyper-references), you can either

• process the document in the normal way, produce PostScript output and distill it;
• (on a Windows or Macintosh machine with the appropriate Adobe tools installed)
pass the output through the PDFwriter in place of a printer driver (this route is a
dead end: the PDFwriter cannot create hyperlinks);
• process the document in the normal way and generate PDF direct from the DVI
with dvipdfm; or
• process the document direct to PDF with PDF\TeX{} or VT\TeX{}: PDF\TeX{} has
the advantage of availability for a wide range of platforms, VT\TeX{} (available
commercially for Windows, or free of charge for Linux or OS/2) has wider graphics
capability, dealing with encapsulated PostScript and some in-line PostScript.

To translate all the \TeX{} cross-referencing into Acrobat links, you need a \TeX{}
package to suitably redefine the internal commands. There are two of these for \TeX{},
both capable of conforming to the Hyper\TeX{} specification (see question 73): Sebastian
Rahtz’s hyperref, and Michael Mehlich’s hyper. Hyperref uses a configuration file to
determine how it will generate hypertext: it can operate using PDF\TeX{} primitives, the
hyper\TeX{} \verb+$\backslash$specials$, or DVI driver-specific \verb+$\backslash$special commands. Both dvips and
Y\&Y’s DVIPS\textsc{One} translate the DVI with these \verb+$\backslash$special commands into PostScript
acceptable to Distiller, and dvipdfm has \verb+$\backslash$special commands of its own.

There is no free implementation of all of Adobe Distiller’s functionality, but recent
versions of ghostscript provide pretty reliable distillation (but beware of the problems
discussed in question 75). Also, Distiller itself is now remarkably cheap (for academics
at least).

For viewing (and printing) the resulting files, Adobe’s Acrobat Reader is available
for a fair range of platforms; for those for which Adobe’s reader is unavailable,
remotely current versions of ghostscript can display and print PDF files.
dvipdfm: dvilware/dvipdfm
ghostscript: Browse nonfree/support/ghostscript
hyper.sty: macros/latex/contrib/supported/hyper
hyperref.sty: macros/latex/contrib/supported/hyperref
vtex: systems/vtex/linux (for Linux) or systems/vtex/os2 (for OS/2),
together with systems/vtex/common

75 Quality of PDF from PostScript

Any output of dvips may (in principle) be converted to PDF, using either a sufficiently
recent version of ghostscript or Adobe’s (commercial) Distiller, but the results can be
pretty poor if certain simple precautions are omitted.

First, it’s important to use type 1 fonts in preparing the output; METAFONT bitmap
fonts get converted to type 3 fonts when converting to PDF, and Adobe’s Acrobat
Reader (which most of the world uses to view PDF files) makes a very poor fist of
displaying type 3 fonts. If you’ve written a document that uses nothing but fonts such
as Adobe Times, which only exist in type 1 format, no further action is actually re-
quired; however, most ordinary (\TeX{} documents contain at least odd characters that
originate in METAFONT, and it’s advisable to tell dvips to use type 1 versions regard-
less. One does this by using the dvips switches –Pcmz and –Pamz.

Second, if you’re using ghostscript (or the ps2pdf script that’s distributed with it)
make sure you have an appropriate version. Again, if your document contains nothing
but the “basic PostScript” set of fonts (Times, etc), the restrictions are less burdensome
and you can get away with using ghostscript version 5.50; however, that version makes
type 3 fonts of any type 1 fonts embedded in the PostScript, which is pretty unsatisfac-
tory. To be safe, ensure that you’re using ghostscript version 6.00 at least (at the time
of writing, version 7.04 is current).

Third, some versions of Acrobat Reader are confused by characters that are in
positions where Adobe fonts don’t hold characters, and most METAFONT-supplied
fonts have such characters (even after they’ve been converted to type 1). Dvips provides
a means for remapping these characters to places where they’ll be harmless; to invoke
this facility, execute dvips with the switch –G1

Recent distributions of dvips come with a “pdf” pseudo-printer description file;
this bundles selection of the type 1 fonts and character remapping with setting a very
high nominal printer resolution, thus sidestepping dvips’s tendency to ‘optimise’ output
destined for low-resolution printers. Use this by including the \-Ppdf switch in your \dvips command line.

76 Finding ‘8-bit’ Type 1 fonts

Elsewhere, these FAQs recommend that you use an ‘8-bit’ font to permit accentuation of inflected languages (see question 179), and also recommend the use of Type 1 fonts to ensure that you get good quality PDF (see question 75). Unfortunately, the combination proves not to be entirely straightforward: it’s not always easy to find a satisfactory set of Type 1 fonts.

The recommendations prove to be contradictory: there are obstacles in the way of achieving both at the same time. You can use one of the myriad text fonts available in Type 1 format (with appropriate PSNFSS metrics for T1 encoding, or metrics for some other 8-bit encoding such as LY1); you can use a commercial or shareware CM-like Type 1 fonts; or you can use virtual font manipulations — but all these options have their drawbacks.

If you use someone else’s text font (even something as simple as Adobe’s Times family) you have to find a matching family of mathematical fonts, which is a non-trivial undertaking — see question 85.

Commercial CM-like fonts cost money: remarkably little money for commercial products with such a large intellectual property input, but more than this author could ordinarily expend… Y&Y offer their “European Modern” set: an extension of the CM fonts that may used either with T1 or LY1 encoding; these are fonts from the same stable that gave us the free AMS/Blue Sky Research/Y&Y fonts, sensitively extended (though they don’t cover the more eccentric areas of the T1 encoding, and don’t come in the same welter of design sizes that the EC fonts offer). Micropress offer the complete EC set in Type 1 format, as part of their range of outline versions of fonts that were originally distributed in METAFONT format. See question 55.

The shareware BaKoMa \TeX distribution (see question 53) offers a set of Type 1 EC fonts, as an extra shareware option. (As far as the present author can tell, these fonts are only available to users of BaKoMa \TeX: they are stored in an archive format that seems not to be publicly available.)

Virtual fonts (see question 37) help us deal with the problem, since they allow us to map “bits of DVI file” to single characters in the virtual font; so we can create an “é” character by recreating the DVI commands that would result from the code “\‘e”. However, since this involves two characters being selected from a font, the arrangement is sufficient to fool Acrobat Reader: you can’t use the program’s facilities for searching for text that contains inflected characters, and if you cut text from a window that contains such a character, you’ll find something unexpected (typically the accent and the ‘base’ characters separated by a space) when you paste the result. However, if you can live with this difficulty, virtual fonts are a useful, straightforward, and cheap solution to the problem.

There are two virtual-font offerings of CM-based 8-bit fonts — the ae (“almost EC”) and zefonts sets; the zefonts set has wider coverage (though the ae set may be extended to offer guillemets by use of the aeguill package).

\ae fonts: \texttt{fonts/ae}
\aeguill.sty: \texttt{macros/latex/contrib/supported/aeguill}
\zefonts: \texttt{fonts/zefonts}

77 Replacing Type 3 fonts in PostScript

One often comes across a PostScript file generated by \dvips which contains embedded PK fonts; if you try to generate PDF from such a file, the quality will be poor.

Of course, the proper solution is to regenerate the PostScript file, but if neither the sources nor the DVI file are available, one must needs resort to some sort of patching to replace the bitmap fonts in the file by outline fonts.

The program pkfix (by Heiko Oberdiek) will do this patching, for files created by “not too old versions” of \dvips: it finds the fonts to be replaced by examining the PostScript comments \dvips has put in the file. For each font, pkfix puts appropriate \TeX commands in a file, which it then processes and runs through \dvips (with switch \-Ppdf) to acquire an appropriate copy of the font; these copies are then patched back into the original file.
Another program, *dvistrip*, is available from Y&Y’s web site for Windows users who also have Adobe Acrobat Distiller available. *Dvistrip* simply removes the fonts: the idea is that you then reinstate them in the course of a run through *distiller* (which only works if *distiller* ‘knows’ about the fonts: it can be instructed via its Settings→Font Locations tool).


*pkfix*: support/pkfix

# Fonts

## METAFONT fonts

### Getting METAFONT to do what you want

METAFONT allows you to create your own fonts, and most \TeX{} users will never need to use it. METAFONT, unlike \TeX{}, requires some customisation: each output device for which you will be generating fonts needs a mode associated with it. Modes are defined using the `mode_def` convention described on page 94 of *The METAFONT Book* (see question 22). You will need a file, which conventionally called `local.mf`, containing all the `mode_def`s you will be using. If `local.mf` doesn’t already exist, Karl Berry’s collection of modes, available as `fonts/modes/modes.mf`, is a good starting point (it can be used as a ‘local.mf’ without modification in a ‘big enough’ implementation of METAFONT). Lists of settings for various output devices are also published periodically in *TUGboat* (see question 21). Now create a plain base file using `inimf`, `plain.mf`, and `local.mf`:

```
\% inimf
This is METAFONT...
**plain** you type ‘plain’
(output)
*input local you type this
(output)
*dump you type this
(output)
```

Beginning to dump on file plain...

This will create a base file named `plain.base` (or something similar; for example, it will be `PLAIN.BAS` on MS-DOS systems) which should be moved to the directory containing the base files on your system (note that some systems have two or more such directories, one for each ‘size’ of METAFONT used).

Now you need to make sure METAFONT loads this new base when it starts up. If METAFONT loads the plain base by default on your system, then you’re ready to go. Under Unix (using the default `web2c` distribution) this does indeed happen, but we could for instance define a command `mf` which executes `virmf &plain` loading the plain base file.

The usual way to create a font with plain METAFONT is to start it with the line

```
\mode=<mode name>; mag=<magnification>; input <font file name>
```

in response to the ‘**’ prompt or on the METAFONT command line. (If `<mode name>` is unknown or omitted, the mode defaults to ‘proof’ and METAFONT will produce an output file called `<font file name>.2602gf`.) The `<magnification>` is a floating point number or ‘magstep’ (magsteps are defined in *The METAFONT Book* and *The \TeX{}book*). If `<magnification>` is omitted, then the default is 1 (magstep 0). For example, to generate cmr10 at 12pt for an epson printer you would type

```
mf \mode=epson; mag=magstep 1; input cmr10
```

Note that under Unix the `\` and `;` characters must usually be quoted or escaped, so this would typically look something like

```
mf '\mode=epson; mag=magstep 1; input cmr10'
```

---

5The command name is symbolically linked to `virmf`, and `virmf` loads command_name.base
If you don’t have inimf or need a special mode that isn’t in the base, you can put its commands in a file (e.g., ln03.mf) and invoke it on the fly with the \smode command. For example, to create cmr10.300gf for an LN03 printer, using the file

```
% This is ln03.mf as of 1990/02/27
% mode_def courtesy of John Sauter
proofing:=0;
fontmaking:=1;
tracingtitles:=0;
pixels_per_inch:=300;
blacker:=0.65;
fillin:=-0.1;
o_correction=.5;
```

(note the absence of the mode_def and enddef commands), you would type

```
mf \smode="ln03"; input cmr10
```

This technique isn’t one you should regularly use, but it may prove useful if you acquire a new printer and want to experiment with parameters, or for some other reason are regularly editing the parameters you’re using. Once you’ve settled on an appropriate set of parameters, you should use them to rebuild the base file that you use.

A summary of the above written by Geoffrey Tobin, and tips about common pitfalls in using METAFONT, is available as info/metafont-for-beginners.tex

### 79 Which font files should be kept

METAFONT produces from its run three files, a metrics (TFM) file, a generic font (GF) file, and a log file; all of these files have the same base name as does the input (e.g., if the input file was cmr10.mf, the outputs will be cmr10.tfm, cmr10.nnngf, and cmr10.log).

For \TeX to use the font, you need a TFM file, so you need to keep that. However, you are likely to generate the same font at more than one magnification, and each time you do so you’ll (incidentally) generate another TFM file; these files are all the same, so you only need to keep one of them.

To preview or to produce printed output, the DVI processor will need a font raster file; this is what the GF file provides. However, while there used (once upon a time) to be DVI processors that could use GF files, modern processors use packed raster (PK) files. Therefore, you need to generate a PK file from the GF file; the program gftopk does this for you, and once you’ve done that you may throw the GF file away.

The log file should never need to be used, unless there was some sort of problem in the METAFONT run, and need not be ordinarily kept.

### 80 Acquiring bitmap fonts

When CTAN was established, most people would start using \TeX with a 300 dots-per-inch (dpi) laser printer, and sets of Computer Modern bitmap fonts for this resolution are available on CTAN: fonts/cm/pk/pk300.zip (for write-black printer engines) and fonts/cm/pk/pk300w.zip (for write-white engines).

At that time, there were regular requests that CTAN should hold a wider range of resolutions, but they were resisted for two reasons:

1. When a bitmap font is created with METAFONT, it needs to know the characteristics of the device; who knows what 600 or 1270 dpi device you have? (Of course, this objection applies equally well to 300 dpi printers.)
2. Bitmap fonts get big at high resolutions. Who knows what fonts at what sizes people are going to need?

Fortunately, (D)\TeX distribution technology has put a stop to these arguments: most (if not all) current distributions generate bitmap fonts as needed, and cache them for later re-use. The impatient user, who is determined that all bitmap fonts should be created once and for all, may be supported by scripts such as allcm (distributed with \TeX, at least; otherwise such a person should consult question 78).

If your output is to a PostScript-capable device, it may be worth switching to Type 1 versions of the CM fonts. Two free versions are currently available; the older (bakoma)
is somewhat less well produced than the bluesky fonts, which were originally professionally produced and sold, but were then donated to the public domain by their originators Y&Y and Bluesky Research, in association with the AMS. Unfortunately, the coverage of the sets is slightly different, but the present author hasn’t found the need to use bakoma since bluesky became available. In recent years, several other ‘METAFONT’ fonts have become available in Type 1 format; it’s common never to find the need of generating bitmap fonts for any purpose other than previewing (see question 82).

The commercial font suppliers continue just to keep ahead of the free software movement, and provide Type 1 versions of the EC fonts, CM-style Cyrillic fonts, as well as a range of mathematical fonts to replace those in the CM family (see question 85).

bakoma: fonts/cm/ps-type1/bakoma
bluesky: Browse fonts/cm/ps-type1/bluesky

L.2 Adobe Type 1 (“PostScript”) fonts
81 Using PostScript fonts with \TeX
In order to use PostScript fonts, \TeX needs metric (called TFM) files. Several sets of metrics are available from the archives; for mechanisms for generating new ones, see question 83. You also need the fonts themselves; PostScript printers come with a set of fonts built in, but to extend your repertoire you almost invariably need to buy from one of the many commercial font vendors (see, for example, question 85).

If you use \TeX2ε, the best way to get PostScript fonts into your document is to use the PSNFSS package maintained by Walter Schmidt; it’s supported by the \TeX3 project team, so bug reports can and should be submitted. PSNFSS gives you a set of packages for changing the default roman, sans-serif and typewriter fonts; e.g., times, avanat.sty will set up Times Roman, Helvetica and Courier in place of Computer Modern, while avant.sty just changes the sans-serif family to AvantGarde. To go with these packages, you will need the font metric files (watch out for encoding problems! — see question 83) and font description (.fd) files for each font family you want to use. Many sets of metrics, etc., can be obtained from the psfonts area of CTAN, arranged by vendor (e.g., Adobe, Monotype, etc.). For convenience, metrics for the common ‘35’ PostScript fonts found in most printers are provided with PSNFSS, packaged as the “Laserwriter set”.

For older versions of \TeX there are various schemes, of which the simplest to use is probably the PS\TeX macros distributed with dvips.

For Plain \TeX, you load whatever fonts you like; if the encoding of the fonts is not the same as Computer Modern it will be up to you to redefine various macros and accents, or you can use the font re-encoding mechanisms available in many drivers and in ps2pk and afm2tfm.

Victor Eijkhout’s Lollipop package (see question 15) supports declaration of font families and styles in a similar way to \TeX’s NFSS, and so is easy to use with PostScript fonts.

Some common problems encountered are discussed elsewhere (see question 84).

\textit{Laserwriter set of 35 fonts}: macros/latex/required/psnfss/lw35nfss.zip
lollipop: macros/lollipop
psfonts: Browse fonts/psfonts
psnfss: macros/latex/required/psnfss

82 Previewing files using Type 1 fonts
Until recently, free \TeX previewers have only been capable of displaying bitmap PK fonts. (Y&Y’s commercial previewer DVl\input window (see question 55) has long used Adobe Type Manager to display Type 1 fonts directly, and the most recent releases of xdvi sport a Type 1 font renderer.) Other previewers of the current generation offer automatic generation of the requisite PK files (using gsftopk, or similar, behind the scenes). If your previewer isn’t capable of this, you have three options:

1. Convert the DVI file to PostScript and use a PostScript previewer. Some systems offer this capability as standard, but most people will need to use a separate
2. Under Windows on a PC, or on a Macintosh, let Adobe Type Manager display the fonts. Textures (Macintosh) works like this, and under Windows you can use Y&Y’s dviwindo for bitmap-free previewing. (See question 55 for details of these suppliers.)

3. If you have the PostScript fonts in Type 1 format, use ps2pk or gsftopk (designed for use with the ghostscript fonts) to make PK bitmap fonts which your previewer will understand. This can produce excellent results, also suitable for printing with non-PostScript devices. Check the legalities of this if you have purchased the fonts. The very commonest PostScript fonts such as Times and Courier come in Type 1 format on disk with Adobe Type Manager (often bundled with Windows, and part of OS/2).

83 \TeX{} font metric files for PostScript fonts

Reputable font vendors such as Adobe supply metric files for each font, in AFM (Adobe Font Metric) form; these can be converted to TFM (\TeX{} Font Metric) form. The CTAN archives have prebuilt metrics which will be more than enough for many people; but you may need to do the conversion yourself if you have special needs or acquire a new font. One important question is the encoding of (Latin character) fonts; while we all more or less agree about the position of about 96 characters in fonts (the basic ASCII set), the rest of the (typically) 256 vary. The most obvious problems are with floating accents and special characters such as the ‘pounds sterling’ sign. There are three ways of dealing with this: either you change the \TeX{} macros which reference the characters (not much fun, and error-prone); or you change the encoding of the font (easier than you might think); or you use virtual fonts (see question 37) to pretend to \TeX{} that the encoding is the same as it is used to. \LaTeX{} 2\epsilon has facilities for dealing with fonts in different encodings; read the \LaTeX{} Companion (see question 22) for more details. In practice, if you do much non-English (but Latin script) typesetting, you are strongly recommended to use the fontenc package with option ‘T1’ to select ‘Cork’ (see question 42) encoding. A useful alternative is Y&Y’s “private” LY1 encoding, which is designed to sit well with “Adobe standard” encoded fonts. Basic support of LY1 is available on CTAN: note that the “relation with Adobe’s encoding” means that there are no virtual fonts in the LY1 world.

Alan Jeffrey’s fontinst package is an AFM to TFM converter written in \TeX{}; it is used to generate the files used by \LaTeX{} 2\epsilon’s PSNFSS package to support use of PostScript fonts. It is a sophisticated package, not for the faint-hearted, but is powerful enough to cope with most needs. Much of its power relies on the use of virtual fonts (see question 37).

For slightly simpler problems, Rokicki’s afm2tfm, distributed with dvips, is fast and efficient; note that the metrics and styles that come with dvips are not currently \LaTeX{} 2\epsilon compatible.

For the Macintosh, there is a program called EdMetrics which does the job (and more). EdMetrics comes with the (commercial) Textures distribution, but is itself free software, and is available on CTAN.

Windows users can buy Y&Y’s (see question 55) Font Manipulation Tools package which includes a powerful afmtofm program among many other goodies.
84 Deploying Type 1 fonts

For the \LaTeX{} user trying to use the PSNFSS (see question 81) package, three questions may arise.

First, you have to declare to the DVI driver that you are using PostScript fonts; in the case of dvips, this means adding lines to the \texttt{psfonts.map} file, so that \texttt{dvips} will know where the proper fonts are, and won’t try to find PK files. If the font isn’t built into the printer, you have to acquire it (which may mean that you need to purchase the font files).

Second, your previewer must know what to do with the fonts: see question 82.

Third, the stretch and shrink between words is a function of the font metric; it is not specified in AFM files, so different converters choose different values. The PostScript metrics that come with PSNFSS used to produce quite tight setting, but they were revised in mid 1995 to produce a compromise between American and European practice. Really sophisticated users may not find even the new the values to their taste, and want to override them. Even the casual user may find more hyphenation or overfull boxes than CMR produces; but CMR is extremely generous.

85 Choice of scalable outline fonts

If you are interested in text alone, you can use any of over 20,000 fonts(!) in Adobe Type 1 format (called ‘PostScript fonts’ in the \LaTeX{} world and ‘ATM fonts’ in the DTP world), or any of several hundred fonts in TrueType format. That is, provided of course, that your previewer and printer driver support scalable outline fonts.

\TeX{} itself only cares about metrics, not the actual character programs. You just need to create a \TeX{} metric file TFM using some tool such as \texttt{afm2tfm} (possibly in combination with vptovf), \texttt{afmtof}m (from Y\&Y, see question 55) or \texttt{fontinst}. For the previewer or printer driver you need the actual outline font files themselves (pfa for Display PostScript, pfb for ATM on IBM PC, Mac outline font files on Macintosh).

If you also need mathematics, then you are severely limited by the demands that \TeX{} makes of maths fonts (for details, see the paper by B.K.P.\ Horn in \textit{TUGboat} \textbf{14}(3)). For maths, then, there are relatively few choices (though the list is at last growing). There are several font families available that are based on Knuth’s original designs, and some that complement other commercial text font designs; one set (MicroPress’s ‘informal math’) stands alone.

\textit{Computer Modern} (75 fonts — optical scaling) Donald E. Knuth

The CM fonts were originally designed in \texttt{METAFONT}, but are also now available in scalable outline form. There are commercial as well as public domain versions, and there are both Adobe Type 1 and TrueType versions. A set of outline versions of the fonts was developed as a commercial venture by Y\&Y and Blue Sky Research; they have since assigned the copyright to the AMS, and the fonts are now freely available from CTAN. Their quality is such that they have become the \textit{de facto} standard for Type 1 versions of the fonts.

\textit{AMS fonts} (52 fonts, optical scaling) The AMS

This set of fonts offers adjuncts to the CM set, including two sets of symbol fonts (\texttt{msam} and \texttt{msbm}) and Euler text fonts. These are not a self-standing family, but merit discussion here (not least because several other families mimic the symbol fonts). Freely-available Type 1 versions of the fonts are available on CTAN. The \textit{eulervm} package permits use of the Euler maths alphabet in conjunction with text fonts that do not provide maths alphabets of their own (for instance, Adobe Palatino or Minion).

\textit{Computer Modern Bright} (62 fonts — optical scaling) Walter Schmidt

CM Bright is a family of sans serif fonts, based on Knuth’s CM fonts. It comprises the fonts necessary for mathematical typesetting, including AMS symbols, as well as text and text symbol fonts of various shapes. The collection comes with its own set of files for use with \LaTeX{}. The CM Bright fonts are supplied in Type 1 format by MicroPress, Inc.
Concrete Math (25 fonts — optical scaling) Ulrik Vieth

The Concrete Math font set was derived from the Concrete Roman typefaces designed by Knuth. The set provides a collection of math italics, math symbol, and math extension fonts, and fonts of AMS symbols that fit with the Concrete set, so that Concrete may be used as a complete replacement for Computer Modern. Since Concrete is considerably darker than CM, the family may particularly attractive for use in low-resolution printing or in applications such as posters or transparencies. Concrete Math fonts, as well as Concrete Roman fonts, are supplied in Type 1 format by MicroPress, Inc.

For further information (including samples) see http://www.micropress-inc.com/fonts/ccmath/ccmain.htm

BA Math (13 fonts) MicroPress Inc.

BA Math is a family of serif fonts, inspired by the elegant and graphically perfect font design of John Baskerville. BA Math comprises the fonts necessary for mathematical typesetting (maths italic, maths symbols and extensions) in normal and bold weights. The family also includes all OT1 and T1 encoded text fonts of various shapes, as well as fonts with most useful glyphs of the TS1 encoding. Macros for using the fonts with Plain \TeX, \LaTeXX 2.09 and current \LaTeXX are provided.

For further details (including samples) see http://www.micropress-inc.com/fonts/bamath/bamain.htm

HV Math (14 fonts) MicroPress Inc.

HV Math is a family of sans serif fonts, inspired by the Helvetica (TM) typeface. HV Math comprises the fonts necessary for mathematical typesetting (maths italic, maths symbols and extensions) in normal and bold weights. The family also includes all OT1 and T1 encoded text fonts of various shapes, as well as fonts with most useful glyphs of the TS1 encoding. Macros for using the fonts with Plain \TeX, \LaTeXX 2.09 and current \LaTeXX are provided. Bitmapped copies of the fonts are available free, on CTAN.

For further details (including samples) see http://www.micropress-inc.com/fonts/hvmath/hvmain.htm

Informal Math (7 outline fonts) MicroPress Inc.

Informal Math is a family of fanciful fonts loosely based on the Adobe’s Tekton (TM) family, fonts which imitate handwritten text. Informal Math comprises the fonts necessary for mathematical typesetting (maths italic, maths symbols and extensions) in normal weight, as well as OT1 encoded text fonts in upright and oblique shapes. Macros for using the fonts with Plain \TeX, \LaTeXX 2.09 and current \LaTeXX are provided.

For further details (including samples) see http://www.micropress-inc.com/fonts/ifmath/ifmain.htm

Lucida Bright with Lucida New Math (25 fonts) Chuck Bigelow and Kris Holmes

Lucida is a family of related fonts including seriffed, sans serif, sans serif fixed width, calligraphic, blackletter, fax, Kris Holmes’ connected handwriting font, etc; they’re not as ‘spindly’ as Computer Modern, with a large x-height, and include a larger set of maths symbols, operators, relations and delimiters than CM (over 800 instead of 384: among others, it also includes the AMS \textbf{msam} and \textbf{msbm} symbol sets). ‘Lucida Bright Expert’ (14 fonts) adds seriffed fixed width, another handwriting font, smallcaps, bold maths, upright ‘maths italic’, etc., to the set. The distribution includes support for use with Plain \TeX and \LaTeXX 2.09. Support under \LaTeXX 2\epsilon is provided in PSNFSS (see question 81) thanks to Sebastian Rahtz and David Carlisle.

For a sample, see http://www.YandY.com/download/chironlb.pdf

MathTime 1.1 (3 fonts) Publish or Perish (Michael Spivak)

The set contains maths italic, symbol, and extension fonts, designed to work well with Times-Roman. These are typically used with Times, Helvetica and Courier (which are resident on many printers, and which are supplied with some PC
versions). In addition you may want to complement this basic set with Adobe’s Times Smallcap, and perhaps the set of Adobe ‘Math Pi’ fonts, which include blackboard bold, blackletter, and script faces.

For a sample, see http://www.YandY.com/download/chironmt.pdf

**MathTime Plus** (12 fonts) Publish or Perish (Michael Spivak)

Adds bold and heavy versions of the basic math fonts, as well as upright math “italic”. There are also Greek letters for use in typesetting terms commonly used in physics, as well as regular and bold script faces. Both MathTime distributions include support for use with Plain \TeX \ and \LaTeX\ 2.09 (including code to link in Adobe Math Pi 2 and Math Pi 6). Support under \LaTeX\ 2ε is provided in PSNFSS (see question 81) thanks to Frank Mittelbach and David Carlisle.

For a sample, see http://www.YandY.com/download/mathplus.pdf

**TM Math** (14 fonts) MicroPress Inc.

TM Math is a family of serif fonts, inspired by the Times (TM) typeface. TM Math comprises the fonts necessary for mathematical typesetting (maths italic, maths symbols and extensions) in normal and bold weights. The family also includes all OT1 and T1 encoded text fonts of various shapes, as well as fonts with most useful glyphs of the TS1 encoding. Macros for using the fonts with Plain \TeX, \LaTeX\ 2.09 and current \LaTeX\ are provided. Bitmapped copies of the fonts are available free, on CTAN.

For further details (and samples) see http://www.micropress-inc.com/fonts/tmmath/tmmain.htm

**Belleek** (3 fonts) Richard Kinch

Belleek is the upshot of Kinch’s thoughts on how META\font might be used in the future: they were published simultaneously as META\font source, as Type 1 fonts, and as TrueType fonts. The fonts act as “drop-in” replacements for the basic MathTime set (as an example of “what might be done”).

The paper outlining Kinch’s thoughts, proceeding from considerations of the ‘intellectual’ superiority of META\font to evaluations of why its adoption is so limited and what might be done about the problem, is to be found at http://truetex.com/belleek.pdf (the paper is a good read, but exhibits the problems discussed in question 75 — don’t try to read it on-screen in Acrobat reader).

**PA Math** PA Math is a family of serif fonts loosely based on the Palatino (TM) typeface. PAMath comprises the fonts necessary for mathematical typesetting (maths italics, maths, calligraphic and oldstyle symbols, and extensions) in normal and bold weights. The family also includes all OT1, T1 encoded text fonts of various shapes, as well as fonts with the most useful glyphs of the TS1 encoding. Macros for using the fonts with Plain \TeX, \LaTeX\ 2.09 and current \LaTeX\ are provided.

For further details (and samples) see http://www.micropress-inc.com/fonts/pamath/pamain.htm

**mathpazo version 1.003** (5 fonts) by Diego Puga

The Pazo Math fonts are a family of type 1 fonts suitable for typesetting maths in combination with the Palatino family of text fonts. Four of the five fonts of the distribution are maths alphabets, in upright and italic shapes, medium and bold weights; the fifth font contains a small selection of “blackboard bold” characters (chosen for their mathematical significance). Support under \LaTeX\ 2ε is available in PSNFSS (see question 81); the fonts are licensed under the GPL, with legalese permitting the use of the fonts in published documents.

**pxfonts set version 1.0** (26 fonts) by Young Ryu

The px\font set consists of

- virtual text fonts using Adobe Palatino (or the URW replacement used by ghostscript) with modified plus, equal and slash symbols;
- maths alphabets using times;
- maths fonts of all symbols in the computer modern maths fonts (cmy, cmmi, cmex and the Greek letters of cmr)
- maths fonts of all symbols corresponding to the AMS fonts (msam and msbm);
• additional maths fonts of various symbols.

The text fonts are available in OT1, T1 and LY1 encodings, and TS encoded symbols are also available. The sans serif and monospaced fonts supplied with the txfonts set (see below) may be used with pxfonts; the txfonts set should be installed whenever pxfonts are. \LaTeX, dvips and PDF\LaTeX support files are included. The documentation is readily available.

The fonts are licensed under the GPL; use in published documents is permitted.

txfonts set version 3.1 (42 fonts) by Young Ryu

The txfonts set consists of

• virtual text fonts using Adobe Times (or the URW replacement used by ghostscript) with modified plus, equal and slash symbols;
• matching sets of sans serif and monospace ('typewriter') fonts (the sans serif set is based on Adobe Helvetica);
• maths alphabets using times;
• maths fonts of all symbols in the computer modern maths fonts (cmay, cmmi, cmex and the Greek letters of cmr)
• maths fonts of all symbols corresponding to the AMS fonts (msam and msbm);
• additional maths fonts of various symbols.

The text fonts are available in OT1, T1 and LY1 encodings, and TS encoded symbols are also available. \LaTeX, dvips and PDF\LaTeX support files are included. The documentation is readily available.

The fonts are licensed under the GPL; use in published documents is permitted.

Adobe Lucida, LucidaSans and LucidaMath (12 fonts)

Lucida and LucidaMath are generally considered to be a bit heavy. The three maths fonts contain only the glyphs in the CM maths italic, symbol, and extension fonts. Support for using LucidaMath with \TeX is not very good; you will need to do some work reencoding fonts etc. (In some sense this set is the ancestor of the LucidaBright plus LucidaNewMath font set.)

Proprietary fonts Various sources.

Since having a high quality font set in scalable outline form that works with \TeX can give a publisher a real competitive advantage, there are some publishers that have paid (a lot) to have such font sets made for them. Unfortunately, these sets are not available on the open market, despite the likelihood that they’re more complete than those that are.

Mathptm Alan Jeffrey, Walter Schmidt and others.

This set contains maths italic, symbol, extension, and roman virtual fonts, built from Adobe Times, Symbol, Zapf Chancery, and the Computer Modern fonts. The resulting mixture is not really entirely acceptable, but can pass in some circumstances. The real advantage is that the mathptm fonts are (effectively) free, and the resulting PostScript files can be freely exchanged. Support under \TeX is available in PSNFSS (see question 81).

The very limited selection of commercial maths font sets is a direct result of the fact that a maths font has to be explicitly designed for use with \TeX and as a result it is likely to lose some of its appeal in other markets. Furthermore, the \TeX market for commercial fonts is minute (in comparison, for example, to Microsoft TrueType font pack #1, which sold something like 10 million copies in a few weeks after release of Windows 3.1!).

Text fonts in Type 1 format are available from many vendors including Adobe, Monotype, Bitstream. Avoid cheap rip-offs: not only are you rewarding unethical behaviour, destroying the cottage industry of innovative type design, but you are also very likely to get junk. The fonts may not render well (or at all under ATM), may not have the ‘standard’ complement of 228 glyphs, or may not include metric files (needed to make TFM files).

TrueType remains the “native” format for Windows. Some \TeX implementations such as True\TeX (see question 55) use TrueType versions of Computer Modern and
Times Maths fonts to render \TeX{} documents in Windows without the need for additional system software like ATM.

When choosing fonts, your own system environment may not be the only one of interest. If you will be sending your finished documents to others for further use, you should consider whether a given font format will introduce compatibility problems. Publishers may require TrueType exclusively because their systems are Windows-based, or Type 1 exclusively, because their systems are based on the early popularity of that format in the publishing industry. Many service bureaus don’t care as long as you present them with a finished print file for their output device.

CM family collection: Browse \texttt{fonts/cm/ps-type1/bluesky}
AMS font collection: Browse \texttt{fonts/amsfonts/ps-type1}
Belleek fonts: \texttt{fonts/belleek/belleek.zip}
eulervm.sty and supporting metrics: \texttt{fonts/eulervm}
hvmath (free bitmapped version): \texttt{fonts/micropress/hvmath}
pfonts: \texttt{fonts/pfonts}
tmmath (free bitmapped version): \texttt{fonts/micropress/tmmath}
tfonts: \texttt{fonts/tfonts}

86 Weird characters in \texttt{dvips} output

You’ve innocently generated output, using \texttt{dvips}, and there are weird transpositions in it: for example, the fi ligature has appeared as a £ symbol. This is an unwanted side-effect of the precautions about generating PostScript for PDF outlined in question 75. The \texttt{–G1} switch discussed in that question is appropriate for Knuth’s text fonts, but doesn’t work with text fonts that don’t follow Knuth’s patterns (such as fonts supplied by Adobe).

If the problem arises, suppress the \texttt{–G1} switch: if you were using it explicitly, \texttt{don’t}; if you were using \texttt{–Ppdf}, add \texttt{–G0} to suppress the implicit switch in the pseudo-printer file.

The problem has been corrected in \texttt{dvips} of May 2002 (the version distributed with the \TeX{}-Live 7 CD-ROM and in other \TeX{} distributions from that date onwards).

L.3 Particular font families

87 Using the “Concrete” fonts

The Concrete Roman fonts were designed by Don Knuth for a book called “Concrete Mathematics”, which he wrote with Graham and Patashnik (the Patashnik, of Bit\TeX{} fame). Knuth only designed text fonts, since the book used the Euler fonts for mathematics. The book was typeset using Plain \TeX{}, of course, with additional macros that may be viewed in a file \texttt{gkpmac.tex}, which is available on CTAN. A few years later, Ulrik Vieth designed the Concrete Math fonts. The packages \texttt{beton}, \texttt{concmath}, and \texttt{ccfonts} are \LaTeX{} packages that change the default text fonts from Computer Modern to Concrete and slightly increase the \texttt{\baselineskip}. The packages \texttt{concmath}, and \texttt{ccfonts} also change the default math fonts from Computer Modern to Concrete and use the Concrete versions of the AMS fonts (this last behaviour is optional in the case of the \texttt{concmath} package).

There are no bold Concrete fonts, but it is generally accepted that the Computer Modern Sans Serif demibold condensed fonts are an adequate substitute. If you are using \texttt{concmath} or \texttt{ccfonts} and you want to follow this suggestion, then use the package with \texttt{boldsans} class option (in spite of the fact that the \texttt{concmath} documentation calls it \texttt{sansbold} class option). If you are using \texttt{beton}, add \texttt{\renewcommand{\bfdefault}{sbc}} to the preamble of your document.

Type 1 versions of the fonts are available. For the OT1 encoding, they are available from MicroPress (see question 85). The CM-Super fonts (see question 255) contain Type 1 versions of the Concrete fonts in the T1 encoding.

\texttt{beton.sty}: \texttt{macros/latex/contrib/support/beton}
\texttt{ccfonts.sty}: \texttt{macros/latex/contrib/support/ccfonts}
\texttt{CM-Super fonts}: \texttt{fonts/ps-type1/cm-super}
\texttt{concmath.sty}: \texttt{macros/latex/contrib/other/concmath}
M  Graphics

88 How to import graphics into \LaTeX \document

Knuth, when designing the current version of \TeX \backslash \textup{back in the early 1980}s, could discern no "standard" way of expressing graphics in documents. He reasoned that this state could not persist for ever, but that it would be foolish for him to define \TeX \ primitives that allowed the import of graphical image definitions. He therefore deferred the specification of the use of graphics to the writers of DVI \textsc{drivers}; \TeX \ \document\textsc{s would control the drivers by means of 'special commands (see question \textup{38}).

There is therefore a straightforward way for anyone to import graphics into their document: read the specification of the \texttt{special} commands your driver uses, and 'just' code them. This is the hair-shirt approach: it definitely works, but it’s not for everyone.

Over the years, therefore, "graphics inclusion" packages have sprung up; most were designed for inclusion of Encapsulated PostScript graphics (see question \textup{44}) — which has become the \textit{lingua franca} of graphics inclusion over the last decade or so.

Notable examples are the \texttt{epsf} package (distributed with \texttt{dvips}) and the \texttt{psfig} package. (Both of these packages were designed to work well with both Plain \TeX \ and \LaTeX \ 2.09; they are both still available.) All such packages were tied to a particular DVI \textsc{driver} (\texttt{dvips}, in the above two cases), but their code could be configured for others.

The obvious next step was to make the code configurable dynamically. The \LaTeX \ standard \texttt{graphics} package and its derivatives made this step: it is strongly preferred for all current work. It can also be used (with the help of the \texttt{miniltx \LaTeX \ emulator") in documents written in Plain \TeX.

The \texttt{graphics} package takes a variety of "driver options" — package options that select code to generate the commands appropriate to the DVI \textsc{driver} in use. In most cases, your \LaTeX \ distribution will provide a \texttt{graphics.cfg} file that will select the correct driver for what you’re doing (for example, a distribution that provides both \TeX \ and \PDF \ will usually provide a configuration file that determines whether \PDF \ is running, and selects the definitions for it if so).

The \texttt{graphics} package provides a toolkit of commands (insert graphics, scale a box, rotate a box), which may be composed to provide most facilities you need; the basic command, \texttt{\includegraphics}, takes one optional argument, which specifies the bounding box of the graphics to be included.

The \texttt{graphicx} package uses the facilities of of \texttt{graphics} behind a rather more sophisticated command syntax to provide a very powerful version of the \texttt{\includegraphics} command. \texttt{graphicx}'s version can combine scaling and rotation, viewporting and clipping, and many other things. While this is all a convenience (at some cost of syntax), it is also capable of producing noticeably more efficient PostScript, and some of its combinations are simply not possible with the \texttt{graphics} package version.

The \texttt{epsfig} package provides the same facilities as \texttt{graphicx}, but via a \texttt{\psfig} command, capable of emulating the behaviour (if not the bugs) the old \texttt{psfig} package. \texttt{Epsfig} also supplies homely support for users of the \texttt{epsf} package. There is no rational reason to stick with the old packages, which have never been entirely satisfactory in the \LaTeX \ context.

A wide variety of detailed techniques and tricks have been developed over the years, and Keith Reckdahl’s \texttt{epslatex} outlines them in compendious detail: this highly recommendable document is available from CTAN.

epsf.tex: \texttt{macros/plain/contrib/epsf.tex}
epsfig.sty: Part of the \texttt{macros/latex/required/graphics} bundle
epslatex.pdf: info/epslatex.pdf; the document is also available in PostScript format as info/epslatex.ps

\texttt{graphics.sty}: Part of the \texttt{macros/latex/required/graphics}

\texttt{graphicx.sty}: Part of the \texttt{macros/latex/required/graphics} bundle
89 Graphics in dvips

Dvips, as originally conceived, can only import a single graphics format: encapsulated PostScript (.eps files, see question 44). Dvips also deals with the slightly eccentric .eps that is created by MetaPost (see question 4).

Apart from the fact that a depressing proportion of drawing applications produce corrupt EPS when asked for such output, this is pretty satisfactory for vector graphics work.

To include bitmap graphics, you need some means of converting them to PostScript; in fact many standard image manipulators (such as ImageMagick’s convert) make a good job of creating EPS files. (Though Unix users should beware of xv’s claims: it has a tendency to downsample your bitmap to your screen resolution.)

Special purpose applications jpeg2ps (which converts JPEG files using PostScript level 2 functionality) and bmeps (which converts both JPEG and PNG files) are also considered “good bets”. Bmeps comes with patches to produce your own version of dvips that can cope with JPEG and PNG direct, using bmeps’s conversion library.

90 Graphics in PDF\LaTeX

PDF\LaTeX itself has a rather wide range of formats that it can “natively” incorporate into its output PDF stream: JPEG (.jpg files) for photographs and similar images, TIFF (.tif files) and PNG files for artificial bitmap images, and PDF for vector drawings.

In addition, the standard PDF\LaTeX graphics package setup causes Hans Hagen’s supp-pdf macros to be loaded: these macros are capable of translating the output of MetaPost to PDF “on the fly”; thus MetaPost output (.mps files) may also be included in PDF\LaTeX documents.

The commonest problem users encounter, when switching from \TeX, is that there is no straightforward way to include EPS files: since PDF\LaTeX is its own “driver”, and since it contains no means of converting PostScript to PDF, there’s no direct way the job can be done.

The simple solution is to convert the EPS to an appropriate PDF file. The epstopdf program will do this: it’s available either as a Windows executable or as a Perl script to run on Unix and other similar systems. A \LaTeX package, epstopdf, can be used to generate the requisite PDF files “on the fly”; this is convenient, but requires that you suppress one of \TeX’s security checks: don’t use it in files from sources you don’t entirely trust.

An alternative (and, I find, deeply satisfying) solution is to use purifyeps, a perl script which uses the good offices of pstoedit and of MetaPost to convert your Encapsulated PostScript to “Encapsulated PostScript that comes out of MetaPost”, and can therefore be included directly.

91 Making MetaPost output display in ghostscript

MetaPost ordinarily expects its output to be included in some context where the ‘standard’ METAFONT fonts (that you’ve specified) are already defined — for example, as a figure in \LaTeX document. If you’re debugging your MetaPost code, you may want to view it in ghostscript (or some other PostScript previewer). However, the PostScript ‘engine’ in ghostscript doesn’t ordinarily have the fonts loaded, and you’ll experience an error such as

```
Error: /undefined in cmmi10
```
There is provision in MetaPost for avoiding this problem: issue the command
\prologues := 2; at the start of the .mp file.

Unfortunately, the PostScript that MetaPost inserts in its output, following this command, is incompatible with ordinary use of the PostScript inclusions into \(\LaTeX\) documents, so it’s best to make the \prologues command optional. Furthermore, MetaPost takes a very simple-minded approach to font encoding: since \(\LaTeX\) font encodings regularly confuse sophisticated minds, this can prove troublesome. If you’re suffering such problems (the symptom is that characters disappear, or are wrongly presented) the only solution is to view the ‘original’ metapost output after processing through \(\LaTeX\) and dvips.

Conditional compilation may be done either by inputting MyFigure.mp indirectly from a simple wrapper MyFigureDisplay.mp:
\begin{verbatim}
prologues := 2;
input MyFigure
\end{verbatim}

or by issuing a shell command such as
\begin{verbatim}
mp \{\prologues:=2; input MyFigure\}
\end{verbatim}
(which will work without the quote marks if you’re not using a Unix shell).

A suitable \(\LaTeX\) route would involve processing MyFigure.tex, which contains:
\begin{verbatim}
\documentclass{article}
\usepackage{graphicx}
\begin{document}
\thispagestyle{empty}
\includegraphics{MyFigure.1}
\end{document}
\end{verbatim}
Processing the resulting DVI file with the dvips command
\begin{verbatim}
dvips -E -o MyFigure.eps MyFigure
\end{verbatim}
would then give a satisfactory Encapsulated PostScript file.

92 Drawing with \(\LaTeX\)

There are many packages to do pictures in \(\LaTeX\) itself (rather than importing graphics created externally), ranging from simple use of \(\LaTeX\) picture environment, through enhancements like epic, to sophisticated (but slow) drawing with \(\text{PCTeX}\). Depending on your type of drawing, and setup, four systems should be at the top of your list to look at:

1. \text{pstricks}; this gives you access to all the power of PostScript from \(\LaTeX\) itself, by sophisticated use of \texttt{\special}. You need a decent DVI to PostScript driver (like dvips), but the results are worth it. The well-documented package gives you not only low-level drawing commands (and full colour) like lines, circles, shapes at arbitrary coordinates, but also high-level macros for framing text, drawing trees and matrices, 3D effects, and more.

2. MetaPost; you liked \METAFONT, but never got to grips with font files? Try MetaPost (see question 4) — all the power of \METAFONT, but it generates PostScript figures; MetaPost is nowadays part of most serious \(\LaTeX\) distributions. Knuth uses it for all his work...

3. \text{Mfpic}; you liked \METAFONT, but can’t understand the language? The package makes up \METAFONT or MetaPost code for you within using familiar-looking \(\LaTeX\) macros. Not \textit{quite} the full power of \METAFONT, but a friendlier interface.

4. You liked \text{PCTeX} but don’t have enough memory or time? Look at Eitan Gurari’s \textit{dratex}, which is as powerful as most other \(\LaTeX\) drawing packages, but is an entirely new implementation, which is not as hard on memory, is much more readable (and is fully documented).

\begin{verbatim}
\textit{dratex}: graphics/dratex
\textit{mfpic}: graphics/mfpic
\textit{pstricks}: graphics/pstricks
\end{verbatim}

93 Drawing Feynman diagrams in \(\LaTeX\)

Michael Levine’s \textit{feynman} bundle for drawing the diagrams in \(\LaTeX\) 2.09 is still available.
Thorsten Ohl’s `feynmf` is designed for use with current \LaTeX, and works in combination with \LaTeX\font (or, in its `feynmp` incarnation, with MetaPost). The `feynmf` or `feynmp` package reads a description of the diagram written in \LaTeX, and writes out code. \LaTeX\font (or MetaPost) can then produce a font (or PostScript file) for use in a subsequent \LaTeX run. For new users, who have access to MetaPost, the PostScript version is probably the better route, for document portability and other reasons.

Jos Vermaeren’s `axodraw` is mentioned as an alternative in the documentation of `feynmf`, but it is written entirely in terms of \dvips special commands, and is thus rather imperfectly portable.

An alternative approach is implemented by Norman Gray’s `feyn` package. Rather than creating complete diagrams as postscript images, `feyn` provides a font (in a variety of sizes) containing fragments, which you can compose to produce complete diagrams. It offers fairly simple diagrams which look good in equations, rather than complicated ones more suitable for display in figures.

`axodraw`: `graphics/axodraw/export`

`feyn` font bundle: `fonts/feyn`

`feynman` bundle: `macros/latex209/contrib/feynman`

`feynmf/feynmp` bundle: `macros/latex/contrib/supported/feynmf`

\section{Bibliographies and citations}

\subsection{Creating bibliographies}

\subsection*{94 Creating a bibliography style}

It is possible to write your own: the standard bibliography styles are distributed in a commented form, and there is a description of the language (see question 30). However, it must be admitted that the language in which \BIB\TeX styles are written is pretty obscure, and one would not recommend anyone who’s not a confident programmer to write their own, though minor changes to an existing style may be within the grasp of many.

If your style isn’t too ‘far out’, you can probably generate it by using the facilities of the `custom-bib` bundle. This contains a file `makebst.tex`, which runs you through a text menu to produce a file of instructions, with which you can generate your own `.bst` file. This technique doesn’t deal with entirely new styles of document (the present author needed “standards committee papers” and “ISO standards” for his dissertation; another commonly-required type is the Web page — see question 98).

\BIB\TeX documentation: `biblio/bibtex/distrib/doc`

`makebst.tex`: Distributed with `macros/latex/contrib/supported/custom-bib`

\subsection*{95 Capitalisation in \BIB\TeX}

The standard \BIB\TeX bibliography styles impose fixed ideas about the capitalisation of titles of things in the bibliography. While this is not unreasonable by \BIB\TeX’s lights (the rules come from the \Chicago Manual of Style) it can be troublesome, since \BIB\TeX fails to recognise special uses (such as acronyms).

The solution is to enclose the letter or letters, whose capitalisation \BIB\TeX should not touch, in braces, as:

\begin{verbatim}
title = {The {THE} operating system},
\end{verbatim}

Sometimes you find \BIB\TeX changing the case of a single letter inappropriately. No matter; the technique can be applied to single letters, as in:

\begin{verbatim}
title = {Te{X}niques and tips},
\end{verbatim}

There’s more on the subject in the \BIB\TeX documentation (see question 30).

\subsection*{96 ‘String too long’ in \BIB\TeX}

The \BIB\TeX diagnostic “Warning–you’ve exceeded 1000, the global-string-size, for entry foo” usually arises from a very large abstract or annotation included in the database. The diagnostic usually arises because of an infelicity in the coding of `abstract.bst`, or styles derived from it. (One doesn’t ordinarily output annotations in other styles.)
The solution is to make a copy of the style file (or get a clean copy from CTAN — \bibliography/contrib/abstract.bst), and rename it (e.g., on a long file-name system to abstract-long.bst). Now edit it: find function outputnonnull and

- change its first line (line 60 in the version on CTAN) from "{ 's :=" to "{ swap$", and
- delete its last line, which just says "s" (line 84 in the version on CTAN).

Finally, change your \bibliographystyle command to refer to the name of the new file.

This technique applies equally to any bibliography style: the same change can be made to any similar outputnonnull function.

If you’re reluctant to make this sort of change, the only way forward is to take the entry out of the database, so that you don’t encounter \BibTeX’s limit, but you may need to retain the entry because it will be included in the typeset document. In such cases, put the body of the entry in a separate file:

```latex
@article{long.boring,
  author = "Fred Verbose",
  ...
  abstract = "{\input{abstracts/long.tex}}"
}
```

In this way, you arrange that all \BibTeX has to deal with is the file name, though it will tell \TeX (when appropriate) to include all the long text.

97 \BibTeX doesn’t understand lists of names

\BibTeX has a strict syntax for lists of authors’ (or editors’) names in the \BibTeX data file; if you write the list of names in a “natural”-seeming way, the chances are you will confuse \BibTeX, and the output produced will be quite different from what you had hoped.

Names should be expressed in one of the forms

First Last
Last, First
Last, Suffix, First

and lists of names should be separated with “and”. For example:

```latex
AUTHOR = {Fred Q. Bloggs, John P. Doe \& Robin Fairbairns}
```

falls foul of two of the above rules: a syntactically significant comma appears in an incorrect place, and \& is being used as a name separator. The output of the above might be something like:

John P. Doe \& Robin Fairbairns Fred Q. Bloggs

because “John P. Doe & Robin Fairbairns” has become the ‘first name’, while “Fred Q. Bloggs” has become the ‘last name’ of a single person. The example should have been written:

```latex
AUTHOR = {Fred Q. Bloggs and John P. Doe and Robin Fairbairns}
```

Some bibliography styles implement clever acrobatics with very long author lists. You can force truncation by using the pseudo-name “others”, which will usually translate to something like “et al” in the typeset output. So, if Mr. Bloggs wanted to distract attention from his co-authors, he would write:

```latex
AUTHOR = {Fred Q. Bloggs and others}
```

98 URLs in \BibTeX bibliographies

There is no citation type for URLs, *per se*, in the standard \BibTeX styles, though Oren Patashnik (the author of \BibTeX) is considering developing one such for use with the long-awaited \BibTeX version 1.0.

The actual information that need be available in a citation of an URL is discussed at some length in the publicly available on-line extracts of ISO 690–2; the techniques below do not satisfy all the requirements of ISO 690–2, but they offer a solution that is at least available to users of today’s tools.
Until the new version arrives, the simplest technique is to use the \texttt{howpublished} field of the standard styles' \texttt{@misc} function. Of course, the strictures about typesetting URLs (see question 143) still apply, so the entry will look like:

\begin{verbatim}
@misc{...,  
  ...,  
  howpublished = "\url{http://...}"
}
\end{verbatim}

Another possibility is that some conventionally-published paper, technical report (or even book) is also available on the Web. In such cases, a useful technique is something like:

\begin{verbatim}
@techreport{...,  
  ...,  
  note = "Also available as \url{http://...}"}
\end{verbatim}

There is good reason to use the \texttt{url} or \texttt{hyperref} packages in this context, since (by default) the \texttt{\url} command ignores spaces in its argument. \LaTeX{} has a habit of splitting lines it considers excessively long, and if there are no space characters for it to use as ‘natural’ breakpoints, \LaTeX{} will insert a comment (‘%’) character ... which is an acceptable character in a URL, so that \texttt{\url} will typeset it. If you’re using \texttt{url}, the way around the problem is to insert odd spaces inside the URL itself in the .\texttt{bib} file, to enable \LaTeX{} to make reasonable decisions about breaking the line. Note that the version of \texttt{\url} that comes with recent versions of the \texttt{hyperref} package doesn’t suffer from the ‘%’-end of line problem: \texttt{hyperref} spots the problem, and suppresses the unwanted characters.

A possible alternative approach is to use the \texttt{harvard} package (if its citation styles are otherwise satisfactory for you). \textsl{Harvard} bibliography styles all include a “\texttt{url}” field in their specification; however, the typesetting offered is somewhat feeble (though it does recognise and use \texttt{LaTeX2HTML} macros if they are available, to create hyperlinks).

\begin{verbatim}
harvard: macros/latex/contrib/supported/harvard
hyperref: macros/latex/contrib/supported/hyperref
url: macros/latex/contrib/other/misc/url.sty
\end{verbatim}

99 Using \LaTeX{} with Plain \TeX{}

The file \texttt{btxmac.tex} contains macros and documentation for using \LaTeX{} with Plain \TeX{}, either directly or with Eplain (see question 14). See question 30 for more information about \LaTeX{} itself.

\begin{verbatim}
btxmac.tex: macros/eplain/btxmac.tex
\end{verbatim}

N.2 Creating citations

100 Separate bibliographies per chapter?

A separate bibliography for each ‘chapter’ of a document can be provided with the package \texttt{chapterbib} (which comes with a bunch of other good bibliographic things). The package allows you a different bibliography for each \texttt{\input} file (i.e., despite the package’s name, the availability of bibliographies is related to the component source files of the document rather than to the chapters that logically structure the document).

The package \texttt{bibunits} ties bibliographies to logical units within the document: the package will deal with chapters and sections (as defined by \LaTeX{} itself) and also defines a \texttt{bunite} environment so that users can select their own structuring.

\begin{verbatim}
chapterbib.sty: macros/latex/contrib/supported/cite
bibunits.sty: macros/latex/contrib/supported/bibunits
\end{verbatim}

101 Multiple bibliographies?

If you’re thinking of multiple bibliographies tied to some part of your document (such as the chapters within the document), please see question 100.

For more than one bibliography, there are three options.

The \texttt{multibib} package offers a very simple interface: it redefines the bibliography commands so that each time you use any one of them, you tell it which bibliography
you want the citations to go to or to come from. The \bibliography command itself also takes a further extra argument that says what title to use for the resulting section or chapter (i.e., it patches \refname and \bibname — see question 203 — in a \texttt{babel}-safe way).

The \texttt{multibib} package allows you to define a series of “additional topics”, each of which comes with its own series of bibliography commands (e.g., a topic “sec” for secondary literature would have commands \texttt{citesec, nocitesec, \bibliographystylesec} and \texttt{\bibliographysec}. You can pull citations from any bibliography (.bib file) into any one of the multiple bibliographies (indeed, they may all come from the same .bib file).

The \texttt{bibtopic} package allows you separately to cite several different bibliographies. At the appropriate place in your document, you put a sequence of \texttt{btSect} environments (each of which specifies a bibliography database to scan) to typeset the separate bibliographies. Thus, one might have

\begin{verbatim}
\begin{btSect}{books}
  \section{References from books}
  \btPrintCited
\end{btSect}
\begin{btSect}{articles}
  \section{References from articles}
  \btPrintCited
\end{btSect}
\end{verbatim}

There is also a command \texttt{btPrintNotCited}, which gives the rest of the content of the database (if nothing has been cited from the database, this is equivalent to \LaTeX standard \texttt{\nocite{*}}).

\texttt{bibtopic.sty}: \texttt{macros/latex/contrib/supported/bibtopic}
\texttt{multibib.sty}: \texttt{macros/latex/contrib/supported/multibib}

102 Putting bibliography entries in text

This is a common requirement for journals and other publications in the humanities. Sometimes the requirement is for the entry to appear in the running text of the document, while other styles require that the entry appear in a footnote.

Options for entries in running text are

- The package \texttt{bibentry}, which puts slight restrictions on the format of entry that your .bst file generates, but is otherwise undemanding of the bibliography style.
- The package \texttt{inlinebib}, which requires that you use its \texttt{inlinebib.bst}
- The package \texttt{jurabib}, which was originally targetted at German law documents, and has comprehensive facilities for the manipulation of citations. The package comes with four bibliography styles that you may use: \texttt{jurabib.bst, jhuman.bst} and two Chicago-like ones.

Options for entries in footnotes are

- The package \texttt{footbib}, and
- The package \texttt{jurabib}, again.

\texttt{bibentry.sty}: Distributed with \texttt{macros/latex/contrib/supported/natbib}
\texttt{footbib.sty}: \texttt{macros/latex/contrib/supported/footbib}
\texttt{inlinebib.sty}: \texttt{biblio/bibtex/contrib/inlinebib}
\texttt{jurabib.sty}: \texttt{macros/latex/contrib/supported/jurabib}

103 Sorting and compressing citations

If you give \LaTeX \texttt{\cite{fred, joe, harry, min}}, its default commands could give something like “[2,6,4,3]”; this looks awful. One can of course get the things in order by rearranging the keys in the \texttt{\cite} command, but who wants to do that sort of thing for no more improvement than “[2,3,4,6]”?

The \texttt{cite} package sorts the numbers and detects consecutive sequences, so creating “[2–4,6]”. The \texttt{natbib} package, with the numbers and \texttt{sort&compress} options, will do the same when working with its own numeric bibliography styles (\texttt{plainnat.bst} and \texttt{unsrtnat.bst}).
If you might need to make hyperreferences to your citations, cite isn’t adequate. If you add the hypernat package:
\usepackage[...]{hyperref}
\usepackage[numbers,sort&compress]{natbib}
\usepackage{hypernat}

... \\
\bibliographystyle{plainnat}
the natbib and hyperref packages will interwork.
cite.sty: macros/latex/contrib/supported/cite
hypernat.sty: macros/latex/contrib/supported/misc/hypernat.sty
hyperref.sty: macros/latex/contrib/supported/hyperref
plainnat.bst: macros/latex/contrib/supported/hyperref
unsrtnat.bst: macros/latex/contrib/supported/hyperref

104 Multiple citations
A convention sometimes used in physics journals is to “collapse” a group of related citations into a single entry in the bibliography. \LaTeX, by default, can’t cope with this arrangement, but the mcite package deals with the problem.

The package overloads the \cite command, so that citations of the form \cite{paper1,paper2} appear in the document as a single citation, and appear arranged appropriately in the bibliography itself. You need to alter the bibliography style (.bst) file you use; the package documentation tells you how to do that.
mcite.sty: macros/latex/contrib/supported/mcite

N.3 Manipulating whole bibliographies
105 Listing all your \LaTeX entries
\LaTeX and \BibTeX co-operate to offer special treatment of this requirement. The command \nocite{*} is specially treated, and causes \BibTeX to generate bibliography entries for every entry in each .bib file listed in your \bibliography statement, so that after a \LaTeX–\BibTeX–\LaTeX sequence, you have a document with the whole thing listed.

Note that \LaTeX doesn’t produce “Citation ... undefined” or “There were undefined references” warnings in respect of \nocite{*}. This isn’t a problem if you’re running \LaTeX “by hand” (you know exactly how many times you have to run things), but the lack might confuse automatic processors that scan the log file to determine whether another run is necessary.

106 Making HTML of your Bibliography
A neat solution is offered by the noTeX bibliography style. This style produces a .bb1 file which is in fact a series of HTML ‘P’ elements of class noTeX, and which may therefore be included in an HTML file. Provision is made for customising your bibliography so that its content when processed by noTeX is different from that presented when it is processed in the ordinary way by \LaTeX.

A more conventional translator is the awk script bbl2html, which translates the .bb1 file you’ve generated: a sample of the script’s output may be viewed on the web, at http://rikblok.cjb.net/lib/refs.html
bbl2html.awk: biblio/bibtex/utils/bbl2html.awk
noTeX.bst: biblio/bibtex/utils/noTeX.bst

O Installing (\LaTeX) files
107 Installing a new package
The first step in installing a new package for your \LaTeX system is usually to find where it is (see question 48) and then to get it, usually from CTAN (see question 49). Note that Mik\LaTeX 2.1 offers a simpler procedure (see question 109) than that described here, for packages it knows about.

Ordinarily, you should download the whole distribution directory; the only occasion when this is not necessary is when you are getting something from one of the
A small package \langle smallpack \rangle might be just a single .sty file (typically smallpack.sty) with the usage instructions either included as comments in the file or in a separate user manual or README file. More often a package pack will come as a pair of files, pack.ins and pack.dtx, written to be used with the \LaTeX doc system. The package code must be extracted from these files. If there is a README file as part of the package distribution, read it!

In the doc system, the user manual and documented package code is in the .dtx file, and the .ins file contains \LaTeX instructions on what code should be extracted from the .dtx file. To unpack a doc package \langle pack \rangle, do the following:

- Run latex on pack.ins. This will generate one or more files (normally a pack.sty file but there may be others depending on the particular package).
- Run latex on pack.dtx as a start to getting the user manual and possibly a commented version of the package code.
- Run latex again on pack.dtx, which should resolve any references and generate a Table of Contents if it was called for.
- \LaTeX may have said \textit{No file pack.ind}; this is the source for the command index; if you want the index, process the raw material with:
  - \texttt{makeindex -s gind.ist pack}
  - and run \LaTeX again.
- Print and read pack.dvi

Sometimes a user manual is supplied separately from the .dtx file. Process this after doing the above, just in case the user manual uses the package it is describing.

Almost the final stage of the installation is to put the package file(s) \textit{where \LaTeX can find them}. Where the magic place is, and how you put the files there depends on your particular \LaTeX system and how it is set up (see question 43 for general principles, question 108 for specific advice).

The final stage is to tell \LaTeX that there is a new file, or files, that it should be able to go and find. Most free \LaTeX systems maintain a database of the names and locations of \TeX-related files to enable faster searching. In these systems the database must be updated, using the script or program provided with the distribution for this purpose.

\texttt{texhash}

\texttt{web2c} On a current \texttt{web2c} distribution,\texttt{texhash} ought to work; if it doesn’t, run: \texttt{mktexlsr}

\texttt{MikTeX} On a \texttt{MikTeX} distribution earlier than v2.0, do:
- Start→Programs→MikTeX→Maintenance→Refresh filename database
- or get a DOS window and run:
  - initexmf --update-fndb

On a \texttt{MikTeX} distribution v2.0 or later, do:
- Start→Programs→MikTeX 2→MikTeX Options, and press the Update filename database button.

Remember that a \texttt{\usepackage\{pack\}} command must be put in the preamble of each document in which you want to use the \texttt{pack} package.

\textbf{108 Where to put new files}

Where precisely you put files that you have downloaded does depend on what \TeX distribution you have. However, assuming that you have one of the modern TDS-compliant distributions (such as \texttt{teTeX}, \texttt{fpTeX} or \texttt{mikTeX}) there are some general rules that you can follow:

(1) Always install new files in a local texmf tree. The root directory will be named something like:

\begin{verbatim}
teTeX: /usr/share/texmf-local/
fpTeX: c:\fptex\texmf.local\mikTeX: c:\localtexmf\
\end{verbatim}

Let’s write \texttt{TEXMF} for this root, whatever it is for your system.

(2) In your local texmf tree, imitate the directory structure in your main tree. Here’s some examples of where files of given extensions should go:
109 Installing Mik\TeX \textipa{“known packages”}

Mik\TeX\textipa{2.1} maintains a database of packages it \textipa{“knows about”}, together with (coded) installation instructions that enable it to install with minimal user intervention.

If Mik\TeX\textipa{2.1} does know about a package you need installed, it’s worth using the system.

First, open the Mik\TeX\textipa{2.1} packages window: click on \textipa{Start} → \textipa{Programs} → Mik\TeX\textipa{2.1} → Mik\TeX\textipa{2.1} Options, and select the Packages tab.

On the tab, there is an Explorer-style display of packages. Right-click on the root of the tree, “Mik\TeX\textipa{2.1} Packages”, and select “Search”: enter the name of the package you’re interested in, and press the “Search” button. If Mik\TeX\textipa{2.1} knows about your package, it will open up the tree to show you a tick box for you package: check that box.

If Mik\TeX\textipa{2.1} doesn’t know about the package you’re interested in, you have to use the long-winded procedure (see question \textipa{107}) outlined elsewhere in this FAQ.

If necessary, repeat to select other packages, and then press “OK”; Mik\TeX\textipa{2.1} tells you how many packages you have selected — if you’re happy, press “OK” again. Mik\TeX\textipa{2.1} will go off, download the package (as a .cab file), if necessary, install the files of the package, and then refresh the filename database so that the files will be found.

110 “Temporary” installation of (\LaTeX\textipa{X} files

Operating systems and applications need to know where to find files: many files that they need are \textipa{“just named”} — the user doesn’t necessarily know where they are, but knows to ask for them. The commonsest case, of course, is the commands whose names you type to a shell (yes, even Windows’ “MS-DOS prompt”) are using a shell to read what you type: many of the commands simply involve loading and executing a file, and the PATH variable tells the shell where to find those files.

Modern \LaTeX\textipa{X} implementations come with a bunch of search paths built in to them. In most circumstances these paths are adequate, but one sometimes needs to extend them to pick up files in strange places: for example, we may wish to try a new bundle of packages before installing them ‘properly’ (see question \textipa{107}). To do this, we need to change the relevant path as \LaTeX\textipa{X} perceives it. However, we don’t want to throw away \LaTeX\textipa{X}’s built-in path (all of a sudden, \LaTeX\textipa{X} won’t know how to deal with all sorts of things).

To extend a \LaTeX\textipa{X} path, we define an operating system environment variable in ‘path format’, but leaving a gap which \LaTeX\textipa{X} will fill with its built-in value for the path. The commonsest case is that we want to place our extension in front of the path, so that our new things will be chosen in preference, so we leave our ‘gap to be filled’ at the end of the environment variable. The syntax is simple (though it depends which shell you’re actually using): so on a Unix-like system, using the bash shell, the job might be done like:

\begin{verbatim}
export TEXINPUTS=/tmp:
\end{verbatim}

while in a Windows system, within a MS-DOS window, it would be:

\begin{verbatim}
set TEXINPUTS=C:/temp;
\end{verbatim}

In either case, we’re asking \LaTeX\textipa{X} to load files from the root disc temporary files directory; in the Unix case, the “empty slot” is designated by putting the path separator ‘:’ on its own at the end of the line, while in the Windows case, the technique is the same, but the path separator is ‘;’.

Note that in either sort of system, the change will only affect instances of \LaTeX\textipa{X} that are started from the shell where the environment variable was set. If you run \LaTeX\textipa{X} from another window, it will use the original input path. To make a change of input path
that will “stick” for all windows, set the environment variable in your login script or profile (or whatever) in a Unix system and log out and in again, or in autoexec.bat in a Windows system, and reboot the system.

While all of the above has talked about where \TeX\ finds its macro files, it’s applicable to pretty much any sort of file any \TeX-related program reads — there are lots of these paths, and of their corresponding environment variables. In a web2c-based system, the copious annotations in the texmf.cnf system configuration file help you to learn which path names correspond to which type of file.

P Adjusting the typesetting

P.1 Alternative document classes

111 Replacing the standard classes

People are forever concocting classes that replace the standard ones: the present author produced an ukart class that used the sober package, and a few British-specific things (such as appear in the babel package’s British-english specialisation), which is still occasionally used.

Similar public efforts were available well back in the days of \LaTeX\ 2.09: a notable example, whose pleasing designs seem not to have changed much over all that time, is the ntgclass bundle. Each of the standard classes is replaced by a selection of classes, named in Dutch, sometimes with a single numeric digit attached. So we have classes artikel2, rapport1, boek3 and brief. These classes are moderately well documented in English.

The koma-script bundle (classes named scr...) are a strong current contender. They are under active current development, are comprehensive in their coverage, produce good-looking output and are well documented in both English and German.

The other class currently under development is memoir. This aims to replace book and report classes directly, and (like koma-script) is comprehensive in its coverage of small issues. Memoir’s documentation is very highly spoken of, and is sometimes recommended as an introductory tutorial on typesetting.

Koma-script bundle: macros/latex/contrib/support/koma-script
memoir.cls: macros/latex/contrib/support/memoir
NTGclass bundle: macros/latex/contrib/support/ntgclass
sober.sty: macros/latex239/contrib/misc/sober.sty

112 Formatting a thesis in \LaTeX

Thesis styles are usually very specific to your University, so it’s usually not profitable to ask around for a package outside your own University. Since many Universities (in their eccentric way) still require double-spacing, you may care to refer to question 135.

If you want to write your own, a good place to start is the University of California style, but it’s not worth going to a lot of trouble. (If officials won’t allow standard typographic conventions, you won’t be able to produce an aesthetically pleasing document anyway!)

UC thesis style: macros/latex/contrib/support/uchesis

113 Setting papers for journals

Publishers of journals have a wide range of requirements for the presentation of papers, and while many publishers do accept electronic submissions in \LaTeX, they don’t often submit recommended macros to public archives.

Nevertheless, there are considerable numbers of macros of one sort or another available on CTAN; searching for your journal name in the CTAN catalogue (see question 50) may well turn up what you’re seeking.

Failing that, you may be well advised to contact the prospective publisher of your paper; many publishers have macros on their own web sites, or otherwise available only upon application.

Check that the publisher is offering you macros suitable to an environment you can use: a few still have no macros for current \LaTeX, for example, claiming that \LaTeX\ 2.09 is good enough…

Some publishers rekey anything sent them anyway, so that it doesn’t really matter what macros you use. Others merely encourage you to use as few extensions of a
standard package as possible, so that they will find it easy to transform your paper to their own internal form.

114 A ‘report’ from lots of ‘article’s

This is a requirement, for example, if one is preparing the proceedings of a conference whose papers were submitted in \LaTeX.

The nearest things to canned solutions are Peter Wilson’s combine and Federico Garcia’s subfiles classes.

Combine defines the means to ‘\import’ entire documents, and provides means of specifying significant features of the layout of the document, as well as a global table of contents, and so on. An auxiliary package, combinet, allows use of the \title, \author (etc.) of the \imported documents to appear in the global table of contents.

Subfiles is used in the component files of a multi-file project, and the corresponding subfiles is used in the master file; arrangements may be made so that the component files will be typeset using different page format, etc., parameters than those used when they are typeset as a part of the main file.

A more ‘raw’ toolkit is offered by Matt Swift’s includex and moredefs packages, both part of the frankenstein bundle) offer a possible way forward.

Includex enables you to ‘\includedoc’ complete articles (in the way that you ‘\include’ chapter files in an ordinary report). It doesn’t do the whole job for you, though. You need to analyse the package use of the individual papers, and ensure that a consistent set is loaded in the preamble of the main report.

A completely different approach is to use the pdfpages package, and to include articles submitted in PDF format into a a PDF document produced by PDF\LaTeX. The package defines an \includepdf command, which takes arguments similar to those of the \includegraphics command. With keywords in the optional argument of the command, you can specify which pages you want to be included from the file named, and various details of the layout of the included pages.

combine.cls: macros/latex/contrib/supported/combine
combinet.sty: macros/latex/contrib/supported/combine
includex.sty: Distributed in the “unsupported” part of macros/latex/contrib/supported/frankenstein
moredefs.sty: Distributed as part of macros/latex/contrib/supported/frankenstein
pdfpages.sty: macros/latex/contrib/supported/pdfpages
subfiles.cls, etc.: macros/latex/contrib/supported/subfiles

115 Curriculum Vitae (Resumé)

A framework class, vita, for Curricula Vitae is provided by Andrej Brodnik.

The class can be customised both for subject (example class option files are offered for both computer scientists and singers), and for language (both the options provided are available for both English and Slovene). Extensions may be written by creating new class option files, or by using macros defined in the class to define new entry types, etc.

Didier Verna’s class, curve, is based on a model in which the CV is made of a set of rubrics (each one dealing with a major item that you want to discuss, such as ‘education’, ‘work experience’, etc. The class’s documentation is supported by a couple of example files, and an emacs mode is provided.

The alternative to using a separate class is to impose a package on one of the standard classes. An example, Axel Reichert’s currvita package, has been recommended to the FAQ team. Its output certainly looks good.

There is also a \LaTeX\ 2.09 package resume, which comes with little but advice against trying to use it.

currvita.sty: macros/latex/contrib/supported/currvita
curve.cls: macros/latex/contrib/supported/curve
resume.sty: obsolete/macros/latex209/contrib/resume/resume.sty
vita.cls: macros/latex/contrib/other/vita

116 Letters and the like

\LaTeX{} itself provides a \textit{letter} document class, which is widely disliked; the present author long since gave up trying with it. If you nevertheless want to try it, but are irritated by its way of vertically-shifting a single-page letter, try the following hack:

\Ce{\makeatletter\let\@texttop\relax\makeatother}

in the preamble of your file.

Doing-it-yourself is a common strategy; Knuth (for use with plain \TeX{}, in the \TeX{}book), and Kopka and Daly (in their Guide to \LaTeX{}) offer worked examples.

Nevertheless, there are contributed alternatives — in fact there are an awfully large number of them: the following list, of necessity, makes but a small selection.

Axel Kielhorn’s \textit{akletter} class is the only other one, recommended for inclusion in this FAQ, whose documentation is available in English.

The \textit{dinbrief} class, while recommended, is only documented in German.

There are letter classes in each of the excellent \textit{koma-script} (\textit{scrlettr}: documentation is available in English) and \textit{ntgclass} (\textit{brief}: documentation in Dutch only) bundles. While these are probably good (since the bundles themselves inspire trust) they’ve not been specifically recommended by any users.

\texttt{akletter.cls}: \texttt{macros/latex/contrib/supported/akletter}

\texttt{brief.cls}: Part of the \texttt{macros/latex/contrib/supported/ntgclass} bundle

\texttt{dinbrief.cls}: \texttt{macros/latex/contrib/supported/dinbrief}

\texttt{newlfm.cls}: \texttt{macros/latex/contrib/supported/newlfm}

\texttt{scrlettr.cls}: Part of the \texttt{macros/latex/contrib/supported/koma-script} bundle

117 Other “document font” sizes?

The \LaTeX{} standard classes have a concept of a (base) “document font” size; this size is the basis on which other font sizes (those from \texttt{\tiny} to \texttt{\Huge}) are determined. The classes are designed on the assumption that they won’t be used with sizes other than the set that \LaTeX{} offers by default (10–12pt), but people regularly find they need other sizes. The proper response to such a requirement is to produce a new design for the document, but many people don’t fancy doing that.

Pragmatists, therefore, will tend to go for the classes in the \texttt{extsizes} bundle. This bundle offers “extended” versions of the article, report, book and letter classes, at sizes of 8, 9, 14, 17 and 20pt as well as the standard 10–12pt. Since little has been done to these classes other than to adjust font sizes and things directly related to them, they won’t be optimal — but they’re certainly practical.

\texttt{extsizes} bundle: \texttt{macros/latex/contrib/other/extsizes}

P.2 Document structure

118 The style of document titles

Limited resources in the titlepage option/environment in the standard styles, inflexibility of \texttt{\maketitle}… non-standard things in the title (such as logo images, etc.)

The \textit{titling} package provides a number of facilities that permit manipulation of the appearance of a \texttt{\maketitle} command, the \texttt{\thank} commands within it, and so on. The package also defines a \textit{titlingpage} environment, that offers something in between the standard classes’ \texttt{titlepage} option and the \texttt{titlepage} environment, and is itself somewhat configurable.

\texttt{titling.sty}: \texttt{macros/latex/contrib/supported/titling}

119 The style of section headings

Suppose that the editor of your favourite journal has specified that section headings must be centred, in small capitals, and subsection headings ragged right in italic, but that you don’t want to get involved in the sort of programming described in \textit{The \LaTeX\}
The following hack will probably satisfy your editor. Define yourself new commands
\newcommand{\ssection}[1]{\section[1]{\centering\sc #1}}
\newcommand{\ssubsection}[1]{\subsection[1]{\raggedright\it #1}}
and then use \ssection and \ssubsection in place of \section and \subsection. This isn’t perfect: section numbers remain in bold, and starred forms need a separate re-definition. Also, this will not work if you are using the prototype NFSS with \LaTeX\ 2.09, because the font-changing commands behave differently there.

The package sectsty provides an easy-to-use set of tools to do this job, while the package titlesec permits more advanced usage as well. (Titlesec comes with a second package, titeloc, which is used to adjust the format of table of contents entries.) The fncychap package provides a nice collection of customised chapter heading designs. The anonchap package provides a simple means of typesetting chapter headings “like section headings” (i.e., without the “Chapter” part of the heading); the tocbibind package provides the same commands, in pursuit of another end.

anonchap.sty: macros/latex/contrib/supported/misc/anonchap.sty
fncychap.sty: macros/latex/contrib/supported/fncychap
sectsty.sty: macros/latex/contrib/supported/sectsty
titlesec.sty: macros/latex/contrib/supported/titlesec
tocbibind.sty: macros/latex/contrib/supported/tocbibind

120 Indent after section headings
\LaTeX\ implements a style that doesn’t indent the first paragraph after a section heading. There are coherent reasons for this, but not everyone likes it. The indentfirst package suppresses the mechanism, so that the first paragraph is indented.

indentfirst.sty: Distributed as part of macros/latex/required/tools

121 How to create a \subsubsubsection
\LaTeX\’s set of “sections” stops at the level of \subsubsection. This reflects a design decision by Lamport — for, after all, who can reasonably want a section with such huge strings of numbers in front of it?

In fact, \LaTeX\ standard classes do define “sectioning” levels lower than \subsubsection, but they don’t format them like sections (they’re not numbered, and the text is run-in after the heading). These deeply inferior section commands are \paragraph and \subparagraph: you can (if you must) arrange that these two commands produce numbered headings, so that you can use them as \subsubsections and lower.

The titlesec allows you to adjust the definitions of the sectioning macros, and it may be used to transform a \paragraph’s typesetting so that it looks like that of a \section.

If you want to program the change yourself, you’ll find that the commands (\section all the way down to \subparagraph) are defined in terms of the internal \@startsection command, which takes 6 arguments. Before attempting this sort of work, you are well advised to read the \LaTeX\ sources (ltxsect.dtx in the \LaTeX\ distribution) and the source of the standard packages (classes.dtx). The \LaTeX\ companion (see question 22) discusses use of \@startsection for this sort of thing.

LaTeX\ source: macros/latex/base
titlesec.sty: macros/latex/contrib/supported/titlesec

122 The style of captions
Changes to the style of captions may be made by redefining the commands that produce the caption. So, for example, \fnum@figure (which produces the float number for figure floats) may be redefined:
\renewcommand{\fnum@figure}{\textbf{Fig. \`\#\thefigure}}
which will cause the number to be typeset in bold face. (Note that the original definition used \figurename — see question 203.) More elaborate changes can be made by patching the \caption command, but since there are packages to do the job, such changes (which can get rather tricky) aren’t recommended for ordinary users.
The float package provides some control of the appearance of captions, though it’s principally designed for the creation of non-standard floats). The caption2 and ccaption (note the double “c”) packages provide a range of different formatting options; ccaption is the more modern and comprehensive, and also provides ‘continuation’ captions and captions that can be placed outside of float environments.

caption2.sty: macros/latex/contrib/supported/caption; note that caption2’s documentation is incomplete, and the documentation of the older caption should be typeset as well as that of the newer package.

caption2.sty: macros/latex/contrib/supported/ccaption

float.sty: macros/latex/contrib/supported/float

123 Alternative head- and footlines in \LaTeX

The standard \LaTeX document classes define a small set of ‘page styles’ which (in effect) specify head- and footlines for your document. The set defined is very restricted, but \LaTeX is capable of much more; people occasionally set about employing \LaTeX facilities to do the job, but that’s quite unnecessary — Piet van Oostrum has already done the work.

The package fancyhdr provides simple mechanisms for defining pretty much every head- or footline variation you could want; the directory also contains some (rather good) documentation and one or two smaller packages. Fancyhdr also deals with the tedious behaviour of the standard styles with initial pages (see question 130), by enabling you to define different page styles for initial and for body pages.

fancyhdr.sty: macros/latex/contrib/supported/fancyhdr

124 Changing the margins in \LaTeX

Changing the size of the body of a \LaTeX document’s text is a surprisingly difficult task: the best advice to the beginner is “don’t do it”. There are interactions between fundamental \TeX constraints, constraints related to the design of \LaTeX, and good typesetting and design practice, that mean that any change must be very carefully considered, both to ensure that it “works” and to ensure that the result is pleasing to the eye.

Lamport’s warning in his section on ‘Customizing the Style’ needs to be taken seriously. One-inch margins on A4 paper are fine for 10- or 12-pitch typewriters, but not for 10pt (or even 11pt or 12pt) type because readers find such wide, dense, lines difficult to read: there should ideally be no more than 75 characters per line (though the constraints change for two-column text).

\LaTeX’s controls allow you to change the distance from the edges of a page to the left and top edges of your typeset text, and the width and height of the text. Changing the last two requires more skill than you might expect: the height should bear a certain relationship to \baselineskip, and the width should be constrained as mentioned above.

The controls are expressed as a set of page parameters; they are somewhat complex, and it is easy to get their interrelationships wrong when redefining the page layout. The layout package defines a \layout command which draws a diagram of your existing page layout, with the dimensions (but not their interrelationships) shown. This FAQ recommends that you use a package to establish consistent settings of the parameters: the interrelationships are taken care of in the established packages, without you needing to think about them.

The ‘ultimate’ tool for adjusting the dimensions and position of the printed material on the page is the geometry package: a very wide range of adjustments of the layout may be relatively straightforwardly programmed, and documentation in the .dtx file (see question 39) is good and comprehensive.

Somewhat simpler to use is the vmargin package, which has a canned set of paper sizes (a superset of that provided in \LaTeX\ 2\epsilon), provision for custom paper, margin adjustments and provision for two-sided printing.

If you’re still eager to “do it yourself”, start by familiarising yourself with \LaTeX’s page layout parameters. For example, see section C.5.3 of the \LaTeX manual (pp. 181–182), or corresponding sections in many of the other good \LaTeX manuals (see question 22). The parameters \oddsidemargin and \evensidemargin are so-called because it is conventionally taken that odd-numbered pages appear on the right-hand side of a two-page spread (‘recto’) and even-numbered pages on the left-hand side (‘verso’).
Both parameters refer to the left-hand margin; the right-hand margin is specified by implication, from the size of \textwidth. The origin in DVI coordinates is one inch from the top of the paper and one inch from the left side; positive horizontal measurements extend right across the page, and positive vertical measurements extend down the page. Thus, for margins closer to the left and top edges of the page than 1 inch, the corresponding parameters, \textit{i.e.}, \texttt{evensidemargin}, \texttt{oddsidemargin}, \texttt{topmargin}, can be set to negative values.

Another surprise is that you cannot change the width or height of the text within the document, simply by modifying the text size parameters. The simple rule is that the parameters should only be changed in the preamble of the document, \textit{i.e.}, before the \texttt{\begin{document}} statement. To adjust text width within a document we define an environment:

```latex
\newenvironment{changemargin}[2]{%\begin{list}{}{% \setlength{\topsep}{0pt} \setlength{\leftmargin}{#1} \setlength{\rightmargin}{#2} \setlength{\listparindent}{\parindent} \setlength{\itemindent}{\parindent} \setlength{\parsep}{\parskip} }% \item[]}{\end{list}}
```

This environment takes two arguments, and will indent the left and right margins, respectively, by the parameters’ values. Negative values will cause the margins to be narrowed, so \texttt{\begin{changemargin}{-1cm}{-1cm}} narrows the left and right margins by 1cm.

The \texttt{chngpage} package provides ready-built commands to do the above; it includes provision for changing the shifts applied to your text according to whether you’re on an odd or an even page of a two-sided document. The package’s documentation (in the file itself) suggests a strategy for changing text dimensions between pages — as mentioned above, changing the text dimensions within the body of a page may lead to unpredictable results.

\texttt{chngpage.sty}: macros/latex/contrib/supported/misc/chngpage.sty
\texttt{geometry.sty}: macros/latex/contrib/supported/geometry
\texttt{layout.sty}: Distributed as part of macros/latex/required/tools
\texttt{vmargin.sty}: macros/latex/contrib/supported/vmargin

### 125 Wide figures in two-column documents

Floating figures and tables ordinarily come out the same width as the page, but in two-column documents they’re restricted to the width of the column. This is sometimes not good enough; so there are alternative versions of the float environments — in two-column documents, \texttt{figure*} provides a floating page-wide figure (and \texttt{table*} a page-wide table) which will do the necessary.

The “*"ed float environments can only appear at the top of a page, or on a whole page — \texttt{h} or \texttt{b} float placement directives are simply ignored.

Unfortunately, page-wide equations can only be accommodated inside float environments. You should include them in \texttt{figure} environments, or use the \texttt{float} or \texttt{ccaption} package to define a new float type.

\texttt{ccaption.sty}: macros/latex/contrib/supported/ccaption
\texttt{float.sty}: macros/latex/contrib/supported/float

### 126 1-column abstract in 2-column document

One often requires that the abstract of a paper should appear across the entire page, even in a two-column paper. The required trick is:

```latex
\documentclass[twocolumn]{article}
...
\begin{document}
... \% \author, etc
\twocolumn[
```
Unfortunately, with the above \thanks won’t work in the \author list. If you need such specially-numbered footnotes, you can make them like this:

\maketitle
\begin{abstract}...
\end{abstract}
\end{@twocolumnfalse}

\title{Demonstration}
\author{Me, You\thanks{}%}
\twocolumn[
... as above ...
]
\renewcommand{\thefootnote}\
{\fnsymbol{footnote}}
\footnotetext[1]{Thanks for nothing}
and so on.

As an alternative, among other facilities the \abstract package (available from macros/latex/contrib/supported/abstract) provides a \saythanks command and a onecolabstract environment which remove the need to fiddle with the \thanks and footnoting. They can be used like this:

\twocolumn[
\maketitle % full width title
\begin{onecolabstract} % full width abstract
... text
\end{onecolabstract}
]
\saythanks % typeset any \thanks

127 Really blank pages between chapters

Book (by default) and report (with openright class option) ensure that each chapter starts on a right-hand (recto) page; they do this by inserting a \cleardoublepage command between chapters (rather than a mere \clearpage). The empty page thus created gets to have a normal running header, which some people don’t like.

The (excellent) fancyhdr manual covers this issue, basically advising the creation of a command \clearemptydoublepage:
\let\origdoublepage\cleardoublepage
\newcommand{\clearemptydoublepage}{% \clearpage \{\pagestyle{empty}\origdoublepage\}%
}

The “obvious” thing is then to use this command to replace \cleardoublepage in a patched version of the chapter command. (Make a package of your own containing a copy of the command out of the class.) This isn’t particularly difficult, but you can instead simply subvert \cleardoublepage (which isn’t often used elsewhere):
\let\cleardoublepage\clearemptydoublepage

Note: this command works because \clearemptydoublepage uses a copy of \cleardoublepage: see question 193 for explanation.

Note that the KOMA-Script replacement for the book class (scrbook) offers class options that control the appearance of these empty pages, and Peter Wilson’s memoir class has similar facilities.

memoir.cls: macros/latex/contrib/supported/memoir
scrbook.cls: Part of macros/latex/contrib/supported/koma-script

128 Balancing columns at the end of a document

The twocolumn option of the standard classes causes \LaTeX to set the text of a document in two columns. However, the last page of the document typically ends up with columns
of different lengths — such columns are said to be “unbalanced”. Many (most?) people
don’t like unbalanced columns.

The simplest solution to the problem is to use the multicol package in place of the
twocolumn option, as multicol balances the columns on the final page by default. However,
the use of multicol does come at a cost: its special output routine disallows the use of
in-column floats, though it does still permit full-width (e.g., figure* environment)
floats.

As a result, there is a constant push for a means of balancing columns at the end
of a two-column document. Of course, the job can be done manually: \pagebreak
inserted at the appropriate place on the last page can often produce the right effect, but
this seldom appeals, and if the last page is made up of automatically-generated text (for
example, bibliography or index) inserting the command will be difficult.

The flushend package offers a solution to this problem. It’s a somewhat dangerous
piece of macro code, which patches one of the most intricate parts of the \LaTeX
kernel without deploying any of the safeguards discussed in question 193. The package
only changes the behaviour at end document (its \flushend command is enabled by
default), and one other command permits adjustment of the final balance; other pack-
ages in the bundle provide means for insertion of full width material in two-column
documents.

The balance package also patches the output routine (somewhat more carefully
than flushend).

The user should be aware that any of these packages are liable to become confused
in the presence of floats: if problems arise, manual adjustment of the floats in the
document is likely to be necessary. It is this difficulty (what’s required in any instance
can’t really be expressed in current \LaTeX) that led the author of multicol to suppress
single-column-wide floats.

balance.sty: Distributed as part of macros/latex/contrib/other/preprint
flushend.sty: Distributed as part of macros/latex/contrib/supported/
sttools
multicol.sty: Distributed as part of macros/latex/required/tools

P.3 Page layout

129 How to get rid of page numbers

The package nopageno will suppress page numbers in a whole document.

To suppress page numbers from a single page, use \thispagestyle{empty}
somewhere within the text of the page. (Note that \maketitle and \chapter both
use \thispagestyle internally, so you need to call it after you’ve called them.)

To suppress page numbers from a sequence of pages, you may use \pagestyle
{empty} at the start of the sequence, and restore the original page style at the
end. Unfortunately, you still have to use \thispagestyle after any \maketitle
or \chapter command.

An alternative is to use the rather delightful \pagenumbering{gobble}; this has
the simple effect that any attempt to print a page number produces nothing, so there’s
no issue about preventing any part of \LaTeX from printing the number. However, the
\pagenumbering command does have the side effect that it resets the page number
(to 1), which may be undesirable.

nopageno: macros/latex/contrib/support/carlisle/nopageno.sty

130 \pagestyle{empty} on first page in \LaTeX

If you use \pagestyle{empty}, but the first page is numbered anyway, you are probably
using the \maketitle command too. The behaviour is not a bug but a feature! The
standard \LaTeX styles are written so that initial pages (pages containing a \maketitle,
\part, or \chapter) have a different page style from the rest of the document; to
achieve this, the commands internally issue \thispagestyle{plain}. This is usually
not acceptable behaviour if the surrounding page style is ‘empty’.

Possible workarounds include:

• Put \thispagestyle{empty} immediately after the \maketitle command,
  with no blank line between them.
• Use the `fancyhdr` package, which allows you to customise the style for initial pages independently of that for body pages.
• Use the `nopageno` package, which suppresses all page numbers by affecting the behaviour of page style commands.

`fancyhdr.sty`: macros/latex/contrib/support/fancyhdr
`nopageno.sty`: macros/latex/contrib/support/carlisle/nopageno.sty

131 How to create crop marks

If you’re printing something that’s eventually to be reproduced in significant quantities, and bound, it’s conventional to print on paper larger than your target product, and to place “crop marks” outside the printed area. These crop marks are available to the production house, for lining up reproduction and trimming machines.

You can save yourself the (considerable) trouble of programming these marks for yourself by using the package `crop`, which has facilities to satisfy any conceivable production house.

`crop.sty`: macros/latex/contrib/support/crop

132 ‘Watermarks’ on every page

It’s often useful to place some text (such as ‘DRAFT’) in the background of every page of a document. For L\TeX\ users, this can be achieved with the `draftcopy` package. This can deal with many types of DVI processors (in the same way that the graphics package does) and knows translations for the word ‘DRAFT’ into a wide range of languages (though you can choose your own word, too).

More elaborate watermarks may be achieved using the `eso-pic` package, which in turn uses the package `everyshi`, both part of Martin Schröder’s `ms` bundle.

`draftcopy.sty`: macros/latex/contrib/support/draftcopy
`eso-pic.sty` and `everyshi.sty`: Distributed in macros/latex/contrib/\ms\ supported/ms

133 Typesetting things in landscape orientation

It’s often necessary to typeset part of a document in landscape orientation; to achieve this, one needs not only to change the page dimensions, but also to instruct the output device to print the strange page differently.

There are two “ordinary” mechanisms for doing two slight variations of landscape typesetting:

• If you have a single floating object that is wider than it is deep, and will only fit on the page in landscape orientation, use the `rotating` package; this defines `sidewaysfigure` and `sidewaystable` environments which create floats that occupy a whole page.
• If you have a long sequence of things that need to be typeset in landscape (perhaps a code listing, a wide `tabbing` environment, or a huge table typeset using `longtable` or `supertabular`), use the `iscape` package (or `pdflscape` if you’re generating PDF output). This defines an environment `landscape`, which clears the current page and restarts typesetting in landscape orientation (and clears the page at the end of the environment before returning to portrait orientation).

No currently available package makes direct provision for typesetting in both portrait and landscape orientation on the same page (it’s not the sort of thing that \TeX\ is well set-up to do). If such behaviour was an absolute necessity, one would use the techniques described in question 163, and would rotate the landscape portion using the rotation facilities of the `graphics` package. (Returning from landscape to portrait orientation would be somewhat easier: the portrait part of the page would be a bottom float at the end of the landscape section, with its content rotated.)

To set an entire document in landscape orientation, one might use `iscape` around the whole document. A better option is the `landscape` option of the `geometry` package; if you also give it `dvips` or `pdftex` option, `geometry` also emits the rotation instructions to cause the output to be properly oriented.
A word of warning: most current \TeX\ previewers do not honour rotation requests in .dvi files (the exceptions are the (commercial) Y&Y previewer \texttt{dviwindo} (see question \ref{q:55}), and the \texttt{fp\TeX} previewer \texttt{WinDVI}). If your previewer is not capable of rotation, your best bet is to convert your output to PostScript or to PDF, and to view these ‘final’ forms with an appropriate viewer.

\texttt{geometry.sty}: macros/latex/contrib/support/geometry
\texttt{graphics.sty}: Distributed as part of macros/latex/required/graphics
\texttt{longtable.sty}: Distributed as part of macros/latex/required/tools
\texttt{lscape.sty}: Distributed as part of macros/latex/required/graphics
\texttt{pdflscape.sty}: Distributed with Heiko Oberdiek’s packages macros/latex/contrib/support/oberdiek
\texttt{rotating.sty}: macros/latex/contrib/support/rotating
\texttt{supertabular.sty}: macros/latex/contrib/support/supertabular

134 Putting things at fixed positions on the page
\LaTeX\’s model of the world is (broadly speaking) that the author writes text, and \TeX\ and its macros decide how it all fits on the page. This is not good news for the author who has, from whatever source, a requirement that certain things go in exactly the right place on the page.

Fortunately, in the \LaTeX\ world at least, there is a fixed point on every page, to which the page header. The package \texttt{textpos} latches bits of typesetting to locations you specify, by fixing them to the page header, and thereby solves the problem.

\texttt{textpos.sty}: macros/latex/contrib/support/textpos

P.4 Spacing of characters and lines
135 Double-spaced documents in \LaTeX\n
Are you producing a thesis, and trying to obey regulations that were drafted in the typewriter era? Or are you producing copy for a journal that insists on double spacing for the submitted articles?

\LaTeX\ is a typesetting system, so the appropriate design conventions are for “real books”. If your requirement is from thesis regulations, find whoever is responsible for the regulations, and try to get the wording changed to cater for typeset theses (\emph{e.g.}, to say “if using a typesetting system, aim to make your thesis look like a well-designed book”). (If your requirement is from a journal, you’re probably even less likely to be able to get the rules changed, of course.)

If you fail to convince your officials, or want some inter-line space for copy-editing, try changing \texttt{\baselinestretch} — \texttt{\renewcommand{\baselinestretch}{1.2}} may be enough to give officials the impression you’ve kept to their regulations. Note that \texttt{\baselinestretch} changes don’t take effect until you select a new font, so make the change in the preamble before any font is selected. Don’t try changing \texttt{\baselineskip}: its value is reset at any size-changing command.

For preference, however, use a line-spacing package. The only one currently supported is \texttt{setspace} (\emph{do not} be tempted by \texttt{doublespace} — its performance under current \LaTeX\ is at best problematical). \texttt{Setspace} has the advantage that it switches off double-spacing at places where you would want it to (footnotes, figure captions, and so on); it’s very troublesome to achieve this if you’re manipulating \texttt{\baselinestretch} yourself.

\texttt{setspace.sty}: macros/latex/contrib/support/setspace/setspace.sty

136 Changing the space between letters

A common technique in advertising copy (and other text whose actual content need not actually be \emph{read}) is to alter the space between the letters (otherwise known as the tracking). As a general rule, this is a very bad idea: it detracts from legibility, which is contrary to the principles of typesetting (any respectable font you might be using should already have optimum tracking built into it).

The great type designer, Eric Gill, is credited with saying “he who would letter-space lower-case text, would steal sheep”. (The attribution is probably apocryphal: others are also credited with the remark. Stealing sheep was, in the 19th century, a capital offence.
in Britain.) As the remark suggests, though, letterspacing of upper-case text is less awful a crime; the technique used also to be used for emphasis of text set in Fraktur (or similar) fonts.

Straightforward macros (usable, in principle, with any \TeX{} macro package) may be found in \texttt{letterspacing} (which is the name of the .tex file; it also appears as the \texttt{letterspace} package in some distributions).

The \texttt{tracking} package has similar facilities.

A more comprehensive solution is to be found in the \texttt{soul} package (which is optimised for use with \LaTeX{}, but also works with Plain \TeX{}). Soul also permits hyphenation of letterspaced text; Gill’s view of such an activity is not (even apocryphally) recorded. (Spacing-out forms part of the name of \texttt{soul}; the other half is described in question 218.)

\texttt{letterspacing.tex}: macros/generic/letterspacing.tex
\texttt{soul.sty}: macros/latex/contrib/supported/soul
\texttt{tracking.sty}: macros/latex/contrib/supported/tracking/tracking.sty

137 Setting text ragged right

The trick with typesetting ragged right is to be sure you’ve told the \TeX{} engine “make this paragraph ragged, but never too ragged”. The \LaTeX{} \texttt{\raggedright} command (and the corresponding \texttt{\flushleft} environment) has a tendency to miss the “never” part, and will often create ridiculously short lines, for some minor benefit later in the paragraph. The Plain \TeX{} version of the command doesn’t suffer this failing, but is rather conservative: it is loath to create too large a gap at the end of the line, but in some circumstances (such as where hyphenation is suppressed see question 181) painfully large gaps may sometimes be required.

Martin Schröder’s \texttt{ragged2e} package offers the best of both worlds: it provides raggedness which is built on the Plain \TeX{} model, but which is easily configurable. It defines easily-remembered command (e.g., \texttt{\RaggedRight}) and environment (e.g., \texttt{\FlushLeft}) names that are simply capitalised transformations of the \LaTeX{} kernel originals. The documentation discusses the issues and explains the significance of the various parameters of \texttt{ragged2e}’s operation.

\texttt{ragged2e.sty}: Distributed as part of macros/latex/contrib/supported/ms

138 Cancelling \texttt{\ragged} commands

\LaTeX{} provides commands \texttt{\raggedright} and \texttt{\raggedleft}, but none to cancel their effect.

The following code (to be inserted in a package of your own, or as internal \LaTeX{} code —see question 206) defines a command that restores flush justification at both margins:

\begin{verbatim}
def\flushboth{%
 \let\\@normalcr
 \@rightskip\z@skip \rightskip\@rightskip
 \leftskip\z@skip \parindent 1.5em\relax}
\end{verbatim}

There’s a problem with the setting of \texttt{\parindent} in the code: it’s necessary because both the \texttt{\ragged} commands set \texttt{\parindent} to zero, but the setting isn’t a constant of nature: documents using a standard \LaTeX{} class with \texttt{twocolumn} option will have 1.0em by default, and there’s no knowing what you (or some other class) will have done.

P.5 Typesetting specialities

139 Including a file verbatim in \LaTeX{}

A good way is to use Rainer Schöpf’s \texttt{verbatim}, which provides a command \texttt{\verbatiminput} that takes a file name as argument.

Another way is to use the \texttt{alltt} environment, which requires \texttt{alltt}. \texttt{alltt} interprets its contents ‘mostly’ verbatim, but executes any \TeX{} commands it finds: so one can say:

\begin{verbatim}
70
of course, this is little use for inputting (LATEX source code).

Moreverb extends the facilities of verbatim package, providing a listing environment and a \texttt{\textbackslash listinginput} command, which line-number the text of the file. The package also has a \texttt{\textbackslash verbatimtabinput} command, that honours TAB characters in the input (the \texttt{\textbackslash listing} environment and command also both honour TAB).

The fancyverb package offers configurable implementations of everything verbatim and moreverb have, and more besides. It is nowadays the package of choice for the discerning typesetter of verbatim text, but its wealth of facilities makes it a complex beast and study of the documentation is strongly advised.

\texttt{alltt.sty}: Part of the LATEX distribution.
\texttt{fancyverb.sty}: \texttt{macros/latex/contrib/supported/fancyverb}
\texttt{moreverb.sty}: \texttt{macros/latex/contrib/supported/moreverb}
\texttt{verbatim.sty}: Distributed as part of \texttt{macros/latex/required/tools}

140 Including line numbers in typeset output

For general numbering of lines, there are two packages for use with LATEX, \texttt{lineno} (which permits labels attached to individual lines of typeset output) and \texttt{numline}.

Both of these packages play fast and loose with the LATEX output routine, which can cause problems: the user should beware...

If the requirement is for numbering verbatim text, moreverb or fancyverb (see question 139) may be used.

One common use of line numbers is in critical editions of texts, and for this the \texttt{edmac} package offers comprehensive support.

\texttt{edmac}: \texttt{macros/plain/contrib/edmac}
\texttt{fancyverb.sty}: \texttt{macros/latex/contrib/supported/fancyverb}
\texttt{lineno.sty}: \texttt{macros/latex/contrib/supported/lineno}
\texttt{moreverb.sty}: \texttt{macros/latex/contrib/supported/moreverb}
\texttt{numline.sty}: \texttt{macros/latex/contrib/supported/numline/numline.sty}

141 Code listings in LATEX

‘Pretty’ code listings are sometimes considered worthwhile by the neurotically æsthetic programmer, but they have a serious place in the typesetting of dissertations by computer science and other students who are expected to write programs. Simple verbatim listings are commonly useful, as well.

Verbatim listings are dealt with elsewhere (see question 139). ‘Pretty’ listings are generally provided by means of a pre-compiler, but the \texttt{listings} package manages to do the job within LATEX.

The \texttt{lgrind} system is a well-established pre-compiler, with all the facilities one might need and a wide repertoire of languages.

The \texttt{tiny\_c2l} system is more recent: users are encouraged to generate their own driver files for languages it doesn’t already deal with.

The C++2LaTeX system comes with strong recommendations for use with C and C++.

\texttt{C++2LaTeX}: \texttt{support/C++2LaTeX-1_1pl1}
\texttt{lgrind}: \texttt{support/lgrind}
\texttt{listings.sty}: \texttt{macros/latex/contrib/supported/listings}
\texttt{tiny\_c2l}: \texttt{support/tiny\_c2l}

142 Generating an index in (LATEX)

Making an index is not trivial; what to index, and how to index it, is difficult to decide, and uniform implementation is difficult to achieve. You will need to mark all items to be indexed in your text (typically with \texttt{\index} commands).

It is not practical to sort a large index within LATEX, so a post-processing program is used to sort the output of one \texttt{\textbackslash index} run, to be included into the document at the next run.

The following programs are available:
makeindex  Comes with most distributions — a good workhorse, but is not well-arranged to deal with other sort orders than the canonical ASCII ordering. The makeindex documentation is a good source of information on how to create your own index. Makeindex can be used with some \TeX\ macro packages other than \LaTeX, such as Eplain (see question 14), and \TeXsis (whose macros can be used independently with Plain \TeX).

\texttt{idxtex} for \LaTeX\ under VMS, which comes with a glossary-maker called \texttt{glotex}.

\texttt{texindex} A witty little shell/sed-script-based utility for \LaTeX\ under Unix. There are other programs called \texttt{texindex}, notably one that comes with the Texinfo distribution (see question 16).

\texttt{xindy} arose from frustration at the difficulty of making a multi-language version of makeindex. It is designed to be a successor to makeindex, by a team that included the then-current maintainer of makeindex. It successfully addresses many of makeindex’s shortcomings, including difficulties with collation order in different languages, and it is highly flexible. Sadly, its take-up is proving rather slow.

\texttt{idxtex}: \texttt{indexing/glo+idxtex}
\texttt{makeindex}: \texttt{indexing/makeindex}
\texttt{makeindex (Macintosh)}: \texttt{systems/mac/macmakeindex2.12.sea.hqx}
\texttt{texindex}: \texttt{support/texindex}
\texttt{texsis (system)}: \texttt{macros/texsis}
\texttt{texsis (makeindex support)}: \texttt{macros/texsis/index/index.tex}
\texttt{xindy}: \texttt{support/xindy}

143 Typesetting URLs

URLs tend to be very long, and contain characters that would naturally prevent them being hyphenated even if they weren’t typically set in \ttfamily, verbatim. Therefore, without special treatment, they often produce wildly overfull \hboxes, and their typeset representation is awful.

There are three packages that help solve this problem:

- The \texttt{path} package, which defines a \texttt{\path} command. The command defines each potential break character as a \texttt{\discretionary}, and offers the user the opportunity of specifying a personal list of potential break characters. Its chief disadvantage is fragility in \LaTeX\ moving arguments. (The Eplain macros — see question 14 — define a similar \texttt{\path} command.)

- The \texttt{url} package, which defines an \texttt{\url} command (among others, including its own \texttt{\path} command). The command gives each potential break character a maths-mode ‘personality’, and then sets the URL itself (in the user’s choice of font) in maths mode. It can produce (\LaTeX-style) ‘robust’ commands (see question 207) for use within moving arguments. Note that, because the operation is conducted in maths mode, spaces within the URL argument are ignored unless special steps are taken.

It is possible to use the \texttt{url} package in Plain \TeX, with the assistance of the \texttt{miniltx} package (which was originally developed for using the \LaTeX\ graphics package in Plain \TeX). A small patch is also necessary: the required sequence is therefore:

\begin{verbatim}
\input miniltx
\expandafter\def\expandafter\+\expandafter{\+}
\input url.sty
\end{verbatim}

- The \texttt{hyperref} package, which uses the typesetting code of \texttt{url}, in a context where the typeset text forms the anchor of a link.

The author of this answer prefers the (rather newer) \texttt{url} package (directly or indirectly); both \texttt{path} and \texttt{url} work well with Plain \TeX\ (though of course, the fancy \TeX\ facilities of \texttt{url} don’t have much place there). (\texttt{hyperref} isn’t available in a version for use with Plain \TeX.)

\texttt{hyperref.sty}: \texttt{macros/latex/contrib/supported/hyperref}
\texttt{miniltx.tex}: Distributed as part of \texttt{macros/plain/graphics}
144 Typesetting music in \TeX

In the early days, a simple music package called \texttt{mutex} was written by Angelika Schofer and Andrea Steinbach, which demonstrated that music typesetting was possible; the package was very limited, and is no longer available. Daniel Taupin took up the baton, and developed \texttt{Music\TeX}, which allows the typesetting of polyphonic and other multiple-stave music; \texttt{Music\TeX} remains available, but is most definitely no longer recommended.

\texttt{Music\TeX} has been superseded by its successor \texttt{Musix\TeX}, which is a three-pass system (with a processor program that computes values for the element spacing in the music), and achieves finer control than is possible in the unmodified \texttt{\TeX}-based mechanism that \texttt{Music\TeX} uses. Daniel Taupin’s is the only version of \texttt{Musix\TeX} currently being developed (the original author, Andreas Egler, had an alternative version, but he is now working on a different package altogether).

Input to \texttt{Musix\TeX} is extremely tricky stuff, and Don Simons’ preprocessor \texttt{pmx} is the preferred method of creating input for Taupin’s version. \texttt{pmx} greatly eases use of \texttt{Musix\TeX}, but it doesn’t support the full range of \texttt{Musix\TeX}’s facilities directly; however, it does allow in-line \texttt{Musix\TeX} code in \texttt{pmx} sources.

Dirk Laurie’s \texttt{M-Tx} allows preparation of music with lyrics; it operates “on top of” \texttt{pmx}.

Another simple notation is supported by \texttt{abc2mtex}; this is a package designed to notate tunes stored in an ASCII format (\texttt{abc} notation). It was designed primarily for folk and traditional tunes of Western European origin (such as Irish, English and Scottish) which can be written on one stave in standard classical notation, and creates input intended for \texttt{Music\TeX}. However, it should be extendable to many other types of music.

Digital music fans can typeset notation for their efforts by using \texttt{midi2tex}, which translates MIDI data files into \texttt{Music\TeX} source code.

There is a mailing list (\texttt{TeX-music@sunsite.dk}) for discussion of typesetting music in \texttt{\TeX}. To subscribe, use \url{http://sunsite.dk/mailman/listinfo/TeX-music/}

\texttt{abc2mtex}: support/abc2mtex
\texttt{M-Tx}: support/mtx
\texttt{midi2tex}: support/midi2tex
\texttt{musictex}: macros/musictex
\texttt{musixtex (Taupin’s version)}: macros/musixtex/taupin
\texttt{musixtex (Egler’s version)}: macros/musixtex/egler
\texttt{pmx}: support/pmx

145 Zero paragraph indent

The conventional way of typesetting running text has no separation between paragraphs, and the first line of each paragraph in a block of text indented.

In contrast, one common convention for typewritten text was to have no indentation of paragraphs; such a style is often required for “brutalist” publications such as technical manuals, and in styles that hanker after typewritten manuscripts, such as officially-specified dissertation formats.

Anyone can see, after no more than a moment’s thought, that if the paragraph indent is zero, the paragraphs must be separated by blank space: otherwise it is sometimes going to be impossible to see the breaks between paragraphs.

The simple-minded approach to zero paragraph indentation is thus:

\begin{verbatim}
\setlength{\parindent}{0pt}
\setlength{\parskip}{\baselineskip}
\end{verbatim}

and in the very simplest text, it’s a fine solution.

However, the non-zero \texttt{\parskip} interferes with lists and the like, and the result looks pretty awful. The \texttt{parskip} package patches things up to look reasonable; it’s not perfect, but it deals with most problems.
The Netherlands Users’ Group’s set of classes includes an article equivalent (artikel3) and a report equivalent (rapport3) whose design incorporates zero paragraph indent and non-zero paragraph skip.

**NTG classes**: macros/latex/contrib/supported/ntgclass

**parskip.sty**: macros/latex/contrib/other/misc/parskip.sty

### 146 Set specifications and Dirac brackets

One of the few glaring omissions from \TeX’s mathematical typesetting capabilities is a means of setting separators in the middle of mathematical expressions. \TeX provides primitives called \texttt{\textbackslash left} and \texttt{\textbackslash right}, which can be used to modify brackets (of whatever sort) around a mathematical expression, as in: \texttt{\left( <expression> \right)} — the size of the parentheses is matched to the vertical extent of the expression.

However, in all sorts of mathematical enterprises one may find oneself needing a \texttt{\middle} command, to be used in expressions like

\begin{verbatim}
\texttt{\left\{ x \in \mathbb{N} \middle| x \text{ even} \right\}}
\end{verbatim}

to specify the set of even natural numbers. The \$\varepsilon$-\TeX system (see question 252) defines just such a command, but users of Knuth’s original need some support. Donald Arseneau’s \textit{braket} package provides commands for set specifications (as above) and for Dirac brackets (and bras and kets). The package uses the $\varepsilon$-\TeX built-in command if it finds itself running under $\varepsilon$-\TeX.

**braket.sty**: macros/latex/contrib/other/misc/braket.sty

### 147 Big letters at the start of a paragraph

A common style of typesetting, now seldom seen except in newspapers, is to start a paragraph (in books, usually the first of a chapter) with its first letter set large enough to span several lines.

This style is known as “dropped capitals”, or (in French) “lettrines”, and \TeX’s primitive facilities for hanging indentation make its (simple) implementation pretty straightforward.

The \textit{dropping} package does the job simply, but has a curious attitude to the calculation of the size of the font to be used for the big letters. Examples appear in the package documentation, so before you process the .dtx, the package itself must already be installed. Unfortunately, \textit{dropping} has an intimate relation to the set of device drivers available in an early version of the \textit{\LaTeX} graphics package, and it cannot be trusted to work with recent offerings like PDF\LaTeX, V\LaTeX or DVIPdfm.

On such occasions, the more recent \textit{lettrine} package is more likely to succeed. It has a well-constructed array of options, and the examples (a pretty impressive set) come as a separate file in the distribution (also available in PostScript, so that they can be viewed without installing the package itself).

**dropping**: macros/latex/contrib/other/dropping

**lettrine**: macros/latex/contrib/supported/lettrine

### 148 The comma as a decimal separator

If you use a comma in maths mode, you get a small space after it; this space is inappropriate if the comma is being used as a decimal separator. An easy solution to this problem is to type (in maths mode) \texttt{3{,}14} instead of typing \texttt{3,14}. However, if your language’s typographic rules require the comma as a decimal separator, such usage can rapidly become extremely tiresome. In such cases it is probably better to use the \textit{icomma} package. The packages ensures that there will be no extra space after a comma, unless you type a space after it (as in \texttt{f(x, y)}, for instance), in which case the usual small space after the comma appears.

**icomma.sty**: Distributed as part of macros/latex/contrib/supported/was

### 149 Breaking boxes of text

(L)\TeX boxes may not be broken, in ordinary usage: once you’ve typeset something into a box, it will stay there, and the box will jut out beyond the side or the bottom of the page if it doesn’t fit in the typeset area.

If you want a substantial portion of your text to be framed (or coloured), the restriction starts to seem a real imposition. Fortunately, there are ways around the problem.
The *framed* package provides *framed* and *shaded* environments; both put their content into something which looks like a framed (or coloured) box, but which breaks as necessary at page end. The environments “lose” footnotes, marginpars and head-line entries, and will not work with *multicol* or other column-balancing macros.

The *boites* package provides a *breakbox* environment; examples of its use may be found in the distribution, and the package’s README file contains terse documentation. The environments may be nested, and may appear inside *multicols* environments; however, floats, footnotes and marginpars will be lost.

For Plain \TeX users, the facilities of the *backgrnd* package may be useful; this package subverts the output routine to provide vertical bars to mark text, and the macros are clearly marked to show where coloured backgrounds may be introduced (this requires *shade*, which is distributed as tex macros and device-independent METAFONT for the shading). The author of *backgrnd* claims that the package works with \TeX 2.09, but there are reasons to suspect that it may be unstable working with current \TeX.

*backgrnd.tex*: \texttt{macros/generic/backgrnd.tex}
*boites.sty*: \texttt{macros/latex/contrib/other/boites}
*framed.sty*: \texttt{macros/latex/contrib/other/misc/framed.sty}
*shade.tex*: \texttt{macros/generic/shade.sty}

P.6 Tables of contents and indexes

150 The format of the Table of Contents, etc.

The formats of entries in the table of contents (TOC) are controlled by a number of internal commands (discussed in section 2.4 of *The \TeX Companion* — see question 22). The commands \@pnumwidth, \@tocrmarg and \@dotsep control the space for page numbers, the indentation of the right-hand margin, and the separation of the dots in the dotted leaders, respectively. The series of commands named \@x..., where xxx is the name of a sectional heading (such as chapter or section, ...) control the layout of the corresponding heading, including the space for section numbers. All these internal commands may be individually redefined to give the effect that you want.

Alternatively, the package *tocloft* provides a set of user-level commands that may be used to change the TOC formatting. Since exactly the same mechanisms are used for the List of Figures and List of Tables, the layout of these sections may be controlled in the same way.

*tocloft.sty*: \texttt{macros/latex/contrib/supported/tocloft}

151 Unnumbered sections in the Table of Contents

The way the relevant parts of sectioning commands work is exemplified by the way the \chapter command uses the counter secnumdepth (described in Appendix C of the \TeX manual):

1. put something in the .aux file, which will appear in the .toc;
2. if secnumdepth ≥ 0, increase the counter for the chapter and write it out.
3. write the chapter title.

Other sectioning commands are similar, but with other values used in the test.

So a simple way to get headings of funny ‘sections’ such as prefaces in the table of contents is to use the counter:

\begin{verbatim}
\setcounter{secnumdepth}{-1}
\chapter{Preface}
\end{verbatim}

Of course, you have to set secnumdepth back to its usual value (which is 2 in the standard styles) before you do any ‘section’ which you want to be numbered.

Similar settings are made automatically in the \TeX book class by the \frontmatter and \backmatter commands.

The value of the counter tocdepth controls which headings will be finally printed in the table of contents. This normally has to be set in the preamble and is a constant for the document. The package *tocvsec2* package provides a convenient interface to allow you to change the secnumdepth and/or the tocdepth counter values at any point in the body of the document; this provides convenient independent controls over the sectional numbering and the table of contents.
The package *abstract* (see question 126) includes an option to add the abstract to the table of contents, while the package *tocbibind* has options to include the table of contents itself, the bibliography, index, etc., to the table of contents.

*abstract.sty*: macros/latex/contrib/support/abstract  
*tocbibind.sty*: macros/latex/contrib/support/tocbibind  
*tocvsec2.sty*: macros/latex/contrib/support/tocvsec2

### Bibliography, index, etc., in TOC

The standard \LaTeX\ classes (and many others) use \section*{} or \chapter*{} for auto-generated parts of the document (the tables of contents, figures and tables, the bibliography and the index). As a result, these items aren’t numbered (which most people don’t mind), and (more importantly) they don’t appear in the table of contents.

The correct solution (as always) is to have a class of your own that formats your document according to your requirements. The macro to do the job (\addcontentsline) is fairly simple, but there is always an issue of ensuring that the contents entry quotes the correct page:

```latex
\bibliography{frooble}
\addcontentsline{toc}{chapter}{Bibliography}
```

will produce the wrong answer if the bibliography is more than one page long. Instead, one should say:

```latex
\cleardoublepage
\addcontentsline{toc}{chapter}{Bibliography}
\bibliography{frooble}
```

(Notе that \cleardoublepage does the right thing, even if your document is single-sided — in that case, it’s a synonym for \clearpage). Ensuring that the entry refers to the right place is trickier still in a \section-based class.

The common solution, therefore, is to use the \texttt{tocbibind} package, which provides many facilities to control the way these entries appear in the table of contents.

Of course, the KOMA-Script bundle of classes (\texttt{scrbook} instead of \texttt{book}, \texttt{scrreprt} instead of \texttt{report}, etc.) provide this functionality as a set of class options.

**KOMA-Script bundle**: macros/latex/contrib/support/koma-script  
*tocbibind.sty*: macros/latex/contrib/support/tocbibind

### Multiple indexes

\LaTeX\’s standard indexing capabilities (those provided by the \texttt{makeidx} package) only provide for one index in your document; even quite modest documents can be improved by indexes for separate topics.

The *multind* package provides simple and straightforward multiple indexing. You tag each \texttt{makeindex}, \texttt{index} and \texttt{printindex} command with a file name, and indexing commands are written to (or read from) the name with the appropriate (.idx or .ind) extension appended. The \texttt{printindex} command is modified from the \LaTeX\ standard so that it doesn’t create its own chapter or section heading; you therefore decide what names (or sectioning level, even) to use for the indexes, and \texttt{\indexname} (see question 203) is completely ignored.

The *index* package provides a comprehensive set of indexing facilities, including a \texttt{\newindex} command that allows the definition of new styles of index. \texttt{\newindex} takes a ‘tag’ (for use in indexing commands), replacements for the .idx and .ind file extensions, and a title for the index when it’s finally printed; it can also change the item that’s being indexed against (for example, one might have an index of artists referenced by the figure number where their work is shown).

**index.sty**: Distributed as part of macros/latex/contrib/support/camel  
*multind.sty*: macros/latex209/contrib/misc/multind.sty
Q. How do I do X in (L)\TeX

Q.1 Mathematics

154 Proof environment
It has long been thought impossible to make a proof environment which automatically includes an ‘end-of-proof’ symbol. Some proofs end in displayed maths; others do not. If the input file contains \end{proof} then \TeX finishes off the displayed maths and gets ready for a new line before it reads any instructions connected with ending the proof, so the code is very tricky. You can insert the symbol by hand, but the ntheorem package now solves the problem for \TeX users: it does indeed provide an automatic way of signalling the end of a proof.

The \AmS-\TeX package amsthm also provides a proof environment that does the job; though you need to insert a \qedhere command if the proof ends with a displayed equation.

amsthm.sty: Distributed as part of the \AmS-\TeX bundle macros/latex/required/amslatex
ntheorem: macros/latex/contrib/supported/ntheorem

155 Roman theorems
If you want to take advantage of the powerful \newtheorem command without the constraint that the contents of the theorem is in a sloped font (for example, to use it to create remarks, examples, proofs, ...) then you can use the theorem package. Alternatively, the following sets up an environment remark whose content is in roman.
\begin{verbatim}
\newcommand{preremark}{Remark}
\newenvironment{remark}{}
\begin{remark}
\end{remark}
\end{verbatim}
The ntheorem package provides roman theorems directly.

ntheorem.sty: macros/latex/contrib/supported/ntheorem
thm.sty: Distributed as part of macros/latex/required/tools

156 Defining a new log-like function in \TeX
Use the \mathop command, as in:
\begin{verbatim}
\newcommand{\diag}{\mathop{\mathrm{diag}}}
\end{verbatim}
Subscripts and superscripts on \diag will be placed as they are on \lim. If you want your subscripts and superscripts always placed to the right, do:
\begin{verbatim}
\newcommand{\diag}{\mathop{\mathrm{diag}}\nolimits}
\end{verbatim}
\AmS-\TeX (in its amsopn package) provides a command \DeclareMathOperator that satisfies the requirement.

(\textit{I}t should be noted that “log-like” was reportedly a joke on Lamport’s part; it is of course clear what was meant.)
amspn.sty: In the \AmS-\TeX distribution macros/latex/required/amslatex

Q.2 Lists

157 Fancy enumeration lists
Suppose you want your top-level enumerates to be labelled ‘I’, ‘II’, ..., then give these commands:
\begin{verbatim}
\renewcommand{\theenumi}{\Roman{enumi}}
\renewcommand{\labelenumi}{\theenumi}
\end{verbatim}
The possible styles of numbering are given in Section 6.3 of Lamport’s book (see question 22). Both \theenumi and \labelenumi must be changed, since \theenumi is used in cross-references to the list.

For lower level enumerates, replace enumi by enumii, enumiii or enumiv, according to the level. If your label is much larger than the default, you should also change \leftmargini, \leftmarginii, etc.

If you’re running \TeX, the enumerate.sty package offers similar facilities. Using it, the example above would be achieved simply by starting the enumeration \begin{enumerate}[I/].
158 How to reduce list spacing

Lamport’s book (see question 22) lists various parameters for the layout of list (things like \topsep, \itemsep and \parsep), but fails to mention that they’re set automatically within the list itself. It works by each list executes a command \@list{\(\text{depth}\)} (the depth appearing as a lower-case roman numeral); what’s more, \@listi is usually reset when the font size is changed. As a result, it’s rather tricky for the user to control list spacing; of course, the real answer is to use a document class designed with more modest list spacing, but we all know such things are hard to come by.

There are packages that provide some control of list spacing, but they seldom address the separation from surrounding text (defined by \topsep). The \texttt{expdlist} package, among its many controls of the appearance of description lists, offers a compaction parameter (see the documentation); the \texttt{mdwlist} package offers a \texttt{\makecompactlist} command for users’ own list definitions, and uses it to define compact lists \texttt{itemize*}, \texttt{enumerate*} and \texttt{description*}. In fact, you can write lists such as these commands define pretty straightforwardly — for example:

\begin{verbatim}
\newenvironment{itemize*}\
{\begin{itemize}\
  \setlength{\itemsep}{0pt}\
  \setlength{\parsep}{0pt}}\
{\end{itemize}}
\end{verbatim}

However, both packages offer other facilities for list configuration: you should certainly not try the “do-it-yourself” approach if you need a package for some other list configuration purpose.

If you want to adjust \topsep, the most sensible approach (at present) is to define your list ‘from first principles’ using the \texttt{\list} command; its invocation is \texttt{\list{(item stuff)}}{(list stuff)}: the \texttt{\list} is executed after the \@list{\(\text{depth}\)}, and can therefore be used to adjust all the parameters, including \topsep.

An alternative is to redefine \@list{\(\text{depth}\)} (and the size-changing commands that alter \@listi), but this is not recommended unless you’re building your own class or package, in which case one hopes you’re capable of analysing the way in which the standard classes do things (as recommended in question 28).

\texttt{expdlist.sty}: \texttt{macros/latex/contrib/support/expdlist}
\texttt{mdwlist.sty}: Distributed as part of \texttt{macros/latex/contrib/support/mdwtools}

Q.3 Tables, figures and diagrams

159 Fixed-width tables

There are two basic techniques for making fixed-width tables in \LaTeX: you can make the gaps between the columns stretch, or you can stretch particular cells in the table.

Basic \LaTeX can make the gaps stretch: the \texttt{tabular*} environment takes an extra argument (before the \texttt{clpr} layout one) which takes a length specification: you can say things like “15cm” or “\texttt{\columnwidth}” here. You must also have an \texttt{\extracolsep} command in the \texttt{clpr} layout argument, inside an \texttt{@{}} directive. So, for example, one might have

\begin{verbatim}
\begin{tabular*}{\columnwidth}{@{\extracolsep{\fill}}lllr}
\end{verbatim}

The \texttt{\extracolsep} applies to all inter-column gaps to its right as well; if you don’t want all gaps stretched, add \texttt{\extracolsep{0pt}} to cancel the original.

The \texttt{tabularx} package defines an extra \texttt{clpr} directive, \texttt{X}: \texttt{X} columns behave as \texttt{p} columns which expand to fill the space available. If there’s more than one \texttt{X} column in a table, the spare space is distributed between them.

The \texttt{ltxtable} combines the features of the \texttt{longtable} and \texttt{tabularx} packages: it’s important to read the documentation, since usage is distinctly odd.

\texttt{ltxtable.sty}: Distributed as part of \texttt{macros/latex/contrib/support/carlisle}
\texttt{tabularx.sty}: Distributed as part of \texttt{macros/latex/required/tools}
160 Spacing lines in tables

(\LaTeX) mechanisms for maintaining the space between lines (the “leading”) rely on \TeX’s paragraph builder, which compares the shape of consecutive lines and adjusts the space between them.

These mechanisms can’t work in exactly the same way when (\LaTeX) is building a table, because the paragraph builder doesn’t get to see the lines themselves. As a result, tables sometimes typeset with lines uncomfortably close together (or occasionally ridiculously far apart).

Traditional (moving metal type) typographers would adjust the spacing between lines of a table by use of a “strut” (a metal spacer). A \TeX user can do exactly the same thing: most macro packages define a \strut command, that defines a space appropriate to the current size of the text; placing a \strut command at the end of a troublesome row is the simplest solution to the problem — if it works. Other solutions below are \LaTeX-specific, but some may be simply translated to Plain \TeX commands.

If your table exhibits a systematic problem (i.e., every row is wrong by the same amount) use \extrarowheight, which is defined by the array package:
\begin{verbatim}
\usepackage{array}% in the preamble
\setlength{\extrarowheight}{length}
\begin{tabular}{....}
\end{verbatim}

To correct a single row whose maladjustment isn’t corrected by a \strut command, you can define your own, using \rule{0pt}{length} — which is a near approximation to the command that goes inside a \strut.

General solutions are available, however. The tabls package automatically generates an appropriately-sized strut at the end of each row. Its disadvantages are that it’s really rather slow in operation (since it gets in the way of everything within tables) and its (lack of) compatibility with other packages.

The booktabs package comes with a thought-provoking essay about how tables should be designed. Since table row-spacing problems most often appear in collisions with rules, the author’s thesis, that \LaTeX users tend too often to rule their tables, is interesting. The package provides rule commands to support the author’s scheme, but deals with inter-row spacing too. Again, booktabs is not compatible with some other packages.

array.sty: Distributed as part of macros/latex/required/tools
booktabs.sty: macros/latex/contrib/supported/booktabs
tabs.sty: macros/latex/contrib/other/misc/tabs.sty

161 Tables longer than a single page

Tables are, by default, set entirely in boxes of their own: as a result, they won’t split over a page boundary. Sadly, the world keeps turning up tables longer than a single page that we need to typeset.

For simple tables (whose shape is highly regular), the simplest solution may well be to use the tabbing environment, which is tedious to set up, but which doesn’t force the whole alignment onto a single page.

The longtable package builds the whole table (in chunks), in a first pass, and then uses information it has written to the .aux file during later passes to get the setting “right” (the package ordinarily manages to set tables in just two passes). Since the package has overview of the whole table at the time it’s doing “final” setting, the table is set “uniformly” over its entire length, with columns matching on consecutive pages. longtable has a reputation for failing to interwork with other packages, but it does work with colorbl, and its author has provided the ltxtable package to provide (most of) the facilities of tabularx (see question 159) for long tables: beware of its rather curious usage constraints — each long table should be in a file of its own, and included by \LTXtable{width}{file}. Since longtable’s multiple-page tables can’t possibly live inside floats, the package provides for captions within the longtable environment itself.

The supertabular package starts and stops a tabular environment for each page of the table. As a result, each ‘page worth’ of the table is compiled independently, and the widths of corresponding columns may differ on successive pages. However, if the
correspondence doesn’t matter, or if your columns are fixed-width, \textit{supertabular} has the great advantage of doing its job in a single run.

Both \textit{longtable} and \textit{supertabular} allow definition of head- and footlines for the table; \textit{longtable} allows distinction of the first and last head and foot.

The \textit{xtab} package fixes some infelicities of \textit{supertabular}, and also provides a “last head” facility (though this, of course, destroys \textit{supertabular}’s advantage of operating in a single run).

The \textit{stabular} package provides a simple-to-use “extension to \textit{tabular}” that allows it to typeset tables that run over the end of a page; it also has usability extensions, but doesn’t have the head- and footnote capabilities of the major packages.

\texttt{longtable.sty}: Distributed as part of \texttt{macros/latex/required/tools}
\texttt{ltxtable.sty}: Generate by running \texttt{macros/latex/contrib/supported/carlisle/ltxtable.tex}
\texttt{stabular.sty}: Distributed as part of \texttt{macros/latex/contrib/supported/sttools}
\texttt{supertabular.sty}: \texttt{macros/latex/contrib/supported/supertabular}
\texttt{xtab.sty}: \texttt{macros/latex/contrib/supported/xtab}

\section*{162 How to alter the alignment of tabular cells}

One often needs to alter the alignment of a tabular p (‘paragraph’) cell, but problems at the end of a table row are common. If we have a p cell that looks like

\begin{verbatim}
... & \centering blah ...
\end{verbatim}

one is liable to encounter errors that complain about a “misplaced \texttt{\noalign}” (or the like). The problem is that the command \texttt{\noalign} means different things in different circumstances: the \textit{tabular} environment switches the meaning to a value for use in the table, and \texttt{\centering}, \texttt{\flushright} and \texttt{\flushleft} all change the meaning to something incompatible. Note that the problem only arises in the last cell of a row: since each cell is set into a box, its settings are lost at the & (or \texttt{\}) that terminates it.

The simple (old) solution is to preserve the meaning of \texttt{\}:

\begin{verbatim}
def\PBS#1{\let\temp=\%#1\%\let\}=\temp}
\end{verbatim}

which one uses as:

\begin{verbatim}
... & \PBS\centering blah ...
\end{verbatim}

(for example).

The technique using \texttt{\PBS} was developed in the days of \LaTeXe 2.09 because the actual value of \texttt{\} that the \textit{tabular} environment used was only available as an internal command. Nowadays, the value is a public command, and you can in principle use it explicitly:

\begin{verbatim}
... & \centering blah ... \texttt{\tabularnewline}
\end{verbatim}

but the old trick has the advantage of extreme compactness.

The \texttt{\PBS} trick also serves well in \textit{array} package “field format” preamble specifications:

\begin{verbatim}
\begin{tabular}{... >{\PBS\centering}p{50mm}}...
\end{verbatim}

\texttt{array.sty}: Distributed as part of \texttt{macros/latex/required/tools}

\section*{163 Flowing text around figures in \LaTeXe}

There are several \LaTeXe packages that purport to do this, but they all have their limitations because the \TeX machine isn’t really designed to solve this sort of problem. Piet van Oostrum has conducted a survey of the available packages; he recommends:

\texttt{floatflt} is an improved version (for \LaTeXe 2.0) of \texttt{floatfig.sty}, and its syntax is:

\begin{verbatim}
\begin{floatingfigure} [options] {width of figure} figure contents \end{floatingfigure}
\end{verbatim}
There is a (more or less similar) floatingtable environment. The tables or figures can be set left or right, or alternating on even/odd pages in a double-sided document. The package works with the \multicol package, but doesn’t work well in the neighbourhood of list environments (unless you change your \LaTeX{} document).

\begin{wrapfig}
\begin{wrapfigure}
\begin{figure}
\begin{wrapfig}
\begin{wrapfigure}
\end{wrapfigure}
\end{wrapfigure}
\end{figure}
\caption{figure, caption, etc.}
\end{wrapfigure}
\end{wrapfigure}
\end{wrapfigure}
\end{wrapfigure}
\end{wrapfigure}
\end{wrapfigure}

The syntax of the \texttt{wraptable} environment is similar. Height can be omitted, in which case it will be calculated by the package; the package will use the greater of the specified and the actual width. The \{l,r,etc\} parameter can also be specified as i\texttt{(inside)} or o\texttt{(outside)} for two-sided documents, and uppercase can be used to indicate that the picture should float. The overhang allows the figure to be moved into the margin. The figure or table will entered into the list of figures or tables if you use the \texttt{\caption} command.

The environments do not work within list environments that end before the figure or table has finished, but can be used in a parbox or minipage, and in twocolumn format.

\texttt{picins} is part of a large bundle that allows inclusion of pictures (e.g., with shadow boxes, various MS-DOS formats, etc.). The command is:

\begin{verbatim}
\parpic(width, height) (x-off, y-off) \{Options\} \{Position\}
\{Picture\}
\end{verbatim}

\texttt{Paragraph text}

All parameters except the \texttt{Picture} are optional. The picture can be positioned left or right, boxed with a rectangle, oval, shadowbox, dashed box, and a caption can be given which will be included in the list of figures.

Unfortunately (for those of us whose understanding of German is not good), the documentation is in German. Piet van Oostrum has written an English summary.

\texttt{floatflt.sty}: \texttt{macros/latex/contrib/other/floatflt}
\texttt{picins.sty}: \texttt{systems/msdos/picins/picins.zip}
\texttt{picins documentation summary}: \texttt{macros/latex209/contrib/picins/picins.txt}
\texttt{wrapfig.sty}: \texttt{macros/latex/contrib/other/misc/wrapfig.sty}

164 Floats on their own on float pages

It’s sometimes necessary to force a float to live on a page by itself. (It’s sometimes even necessary for \texttt{every} float to live on a page by itself.) When the float fails to ‘set’, and waits for the end of a chapter or of the document, the natural thing to do is to declare the float as

\begin{verbatim}
\begin{figure}[p!]
\end{figure}
\end{verbatim}

but the overriding ! modifier has no effect on float page floats; so you have to make the float satisfy the parameters. Question 217 offers some suggestions, but doesn’t solve the one-float-per-page question.

The ‘obvious’ solution, using the counter \texttt{totalnumber} (“total number of floats per page”) doesn’t work: \texttt{totalnumber} only applies to floats on ‘text’ pages (pages containing text as well as one or more float). So, to allow any size float to take a whole page, set \texttt{\floatpagefraction} really small, and to ensure that no more than one float occupies a page, make the separation between floats really big:

\begin{verbatim}
\renewcommand{\floatpagefraction}{.001}
\makeatletter
\setlength{\@fpsep}{\textheight}
\makeatother
\end{verbatim}

Q.4 Footnotes

165 Footnotes in tables

The standard \LaTeX{} \texttt{\footnote} command doesn’t work in tables; the table traps the footnotes and they can’t escape to the bottom of the page.
If your table is floating, your best bet is (unfortunately) to put the table in a `minipage` environment and to put the notes underneath the table, or to use Donald Arseneau’s package `threeparttable` (which implements “table notes” proper).

Otherwise, if your table is not floating (it’s just a ‘tabular’ in the middle of some text), there are several things you can do to fix the problem.

1. Use \footnotemark to position the little marker appropriately, and then put in \footnotetext commands to fill in the text once you’ve closed the `tabular` environment. This is described in Lamport’s book, but it gets messy if there’s more than one footnote.
2. Stick the table in a `minipage` anyway. This provides all the ugliness of footnotes in a minipage with no extra effort.
3. Use `threeparttable` anyway; the package is intended for floating tables, and the result might look odd if the table is not floating, but it will be reasonable.
4. Use `tabularx` or `longtable` from the `LaTeX` tools distribution; they’re noticeably less efficient than the standard `tabular` environment, but they do allow footnotes.
5. Grab hold of `footnote`, and put your `tabular` environment inside a `savenotes` environment. Alternatively, say `\makesavenoteenv{tabular}` in the preamble of your document, and tables will all handle footnotes correctly.
6. Use `mdwtab` from the same bundle; it will handle footnotes properly, and has other facilities to increase the beauty of your tables. It may also cause other table-related packages (not the standard ‘tools’ ones, though) to become very unhappy and stop working.

`footnote.sty`: Distributed as part of `macros/latex/contrib/supported/mdwtools`

`longtable.sty`: Distributed as part of `macros/latex/required/tools`

`mdwtab.sty`: Distributed as part of `macros/latex/contrib/supported/mdwtools`

`threeparttable.sty`: `macros/latex/contrib/other/misc/threeparttable.sty`

`tabularx.sty`: Distributed as part of `macros/latex/required/tools`

### 166 Footnotes in LaTeX section headings

The `\footnote` command is fragile, so that simply placing the command in `\section`’s arguments isn’t satisfactory. Using `\protect\footnote` isn’t a good idea either: the arguments of a section command are used in the table of contents and (more dangerously) potentially also in page headings. Unfortunately, there’s no mechanism to suppress the footnote in the heading while allowing it in the table of contents, though having footnotes in the table of contents is probably unsatisfactory anyway.

To suppress the footnote in headings and table of contents:

- Take advantage of the fact that the mandatory argument doesn’t ‘move’ if the optional argument is present: `\section{title}{title}\footnote{title ftnt}`
- Use the `footmisc` package, with package option `stable` — this modifies footnotes so that they softly and silently vanish away if used in a moving argument.

`footmisc.sty`: `macros/latex/contrib/supported/footmisc`

### 167 Footnotes in captions

Footnotes in captions are especially tricky: they present problems of their own, on top of the problems one experiences with footnotes in section titles (see question 166) and with footnotes in tables (see question 165).

So as well as using the optional argument of `\caption` (or whatever) to avoid the footnote migrating to the List of .... and putting the object whose caption bears the footnote in a minipage, one also has to deal with the tendency of the `\caption` command to produce the footnote’s text twice. For this last problem, there is no tidy solution this author is aware of. If you’re suffering the problem, a well-constructed `\caption` command in a `minipage` environment within a float, such as:
can produce two copies of the footnote body “blah”. (In fact, the effect occurs with captions that are long enough to require two lines to be typeset, and so wouldn’t appear with such a short caption.) The \texttt{ccaption} package’s documentation describes a really rather awful work-around.

\texttt{ccaption.sty}: macros/latex/contrib/supported/ccaption

168 Footnotes whose texts are identical

If the \textit{same} footnote turns up at several places within a document, it’s often inappropriate to repeat the footnote in its entirety over and over again. We can avoid repetition by semi-automatic means, or by simply labelling footnotes that we know we’re going to repeat and then referencing the result. There is no completely automatic solution (that detects and suppresses repeats) available.

If you know you only have one footnote, which you want to repeat, the solution is simple: merely use the optional argument of \texttt{\footnotemark} to signify the repeats:

\begin{verbatim}
...\footnote{Repeating note}
...
...\footnotemark[1]
\end{verbatim}

... which is very easy, since we know there will only ever be a footnote number 1. A similar technique can be used once the footnotes are stable, reusing the number that \LaTeX{} has allocated. This can be tiresome, though, as any change of typesetting could change the relationships of footnote and repeat: labelling is inevitably better.

Simple hand-labelling of footnotes is possible, using a counter dedicated to the job:

\begin{verbatim}
\newcounter{fnnumber}
...
...\footnote{Text to repeat}\%
\setcounter{fnnumber}{\thefootnote}\%
...
...\footnotemark[\thefnnumber]
\end{verbatim}

but this is somewhat tedious. \LaTeX{}'s labelling mechanism can be summoned to our aid, but there are ugly error messages before the \texttt{\ref} is resolved on a second run through \LaTeX{}:

\begin{verbatim}
...\footnote{Text to repeat\label{fn:repeat}}
...
...\footnotemark[\ref{fn:repeat}]
\end{verbatim}

Alternatively, one may use the \texttt{\footref} command, which has the advantage of working even when the footnote mark isn’t expressed as a number. The command is defined in the \texttt{footmisc} package and in the \texttt{memoir} class (at least); \texttt{\footref} reduces the above example to:

\begin{verbatim}
...\footnote{Text to repeat\label{fn:repeat}}
...
...\footref{fn:repeat}
\end{verbatim}

This is the cleanest simple way of doing the job. Note that the \texttt{\label} command must be inside the argument of \texttt{\footnote}.

The \texttt{fixfoot} package takes away some of the pain of the matter: you declare footnotes you’re going to reuse, typically in the preamble of your document, using a \texttt{\DeclareFixedFootnote} command, and then use the command you’ve ‘declared’ in the body of the document:

\begin{verbatim}
\DeclareFixedFootnote{\rep}{Text to repeat}
...
...\rep{}
...\rep{}
\end{verbatim}
The package ensures that the repeated text appears at most once per page: it will usually take more than one run of \LaTeX{} to get rid of the repeats.

\texttt{fixfoot.sty}: \texttt{macros/latex/contrib/supported/fixfoot}
\texttt{footmisc.sty}: \texttt{macros/latex/contrib/supported/footmisc}
\texttt{memoir.cls}: \texttt{macros/latex/contrib/supported/memoir}

Q.5 Document management

169 What's the name of this file

One might want this so as to automatically generate a page header or footer recording what file is being processed. It’s not easy...

\LaTeX{} retains what it considers the name of the \texttt{job}, only, in the special macro \texttt{\jobname}; this is the name of the file first handed to \LaTeX{}, stripped of its directory name and of any extension (such as .tex). If no file was passed (i.e., you’re using \LaTeX{} interactively), \texttt{\jobname} has the value \texttt{texput} (the name that’s given to .log files in this case).

This is fine, for the case of a small document, held in a single file; most significant documents will be held in a bunch of files, and \LaTeX{} makes no attempt to keep track of files input to the \texttt{job}. So the user has to keep track, himself — the only way is to patch the input commands and cause them to retain details of the file name. This is particularly difficult in the case of Plain \LaTeX{}, since the syntax of the \texttt{\input} command is so peculiar.

In the case of \LaTeX{}, the input commands have pretty regular syntax, and the simplest patching techniques (see question 193) can be used on them:
\begin{verbatim}
\def\ThisFile{\jobname}
\newcounter{FileStack}
\let\OrigInput\input
\renewcommand{\input}[1]{%\protect %
 \stepcounter{FileStack}
 \csname NameStack\theFileStack\endcsname
 \ThisFile
 \def\ThisFile{#1}%
 \OrigInput{#1}%
 \expandafter\let\csname NameStack\theFileStack\endcsname
 \ThisFile
 \addtocounter{FileStack}{-1}%
\end{verbatim}

(And similarly for \texttt{\include}.) The code assumes you always use \LaTeX{} syntax for \texttt{\input}, i.e., always use braces around the argument.

The FiNK (“File Name Keeper”) package provides a regular means of keeping track of the current file name (with its extension), in a macro \texttt{\finkfile}. The bundle includes a \texttt{fink.el} that provides support under \texttt{emacs} with AUC-\LaTeX{}.

\texttt{fink.sty}: \texttt{macros/latex/contrib/supported/fink}

170 All the files used by this document

When you’re sharing a document with someone else (perhaps as part of a co-development cycle) it’s as well to arrange that both correspondents have the same set of auxiliary files, as well as the document in question. Your correspondent obviously needs the same set of files (if you use the \texttt{url} package, she has to have \texttt{url} too, for example). But suppose you have a bug-free version of the \texttt{shinynew} package but her copy is still the unstable original; until you both realise what is happening, such a situation can be very confusing.

The simplest solution is the \LaTeX{} \texttt{\listfiles} command. This places a list of the files used and their version numbers in the log file. If you extract that list and transmit it with your file, it can be used as a check-list in case that problems arise.

Note that \texttt{\listfiles} only registers things that are input by the “standard” \LaTeX{} mechanisms (\texttt{\documentclass}, \texttt{\usepackage}, \texttt{\input}, \texttt{\include}, \texttt{\includegraphics} and so on). But if you use \LaTeX{} primitive syntax, as in
\input mymacros

mymacros.tex won’t be listed by \listfiles, since you’ve bypassed the mechanism that records its use.

The \texttt{snapshot} package helps the owner of a \LaTeX{} document obtain a list of the external dependencies of the document, in a form that can be embedded at the top of the document. The intended use of the package is the creation of archival copies of documents, but it has application in document exchange situations too.

The \texttt{bundledoc} system uses \texttt{\listfiles} to produce an archive (e.g., .tar.gz or .zip) of the files needed by your document; it comes with configuration files for use with \TeX{} and \texttt{mikTeX}. It’s plainly useful when you’re sending the first copy of a document.

\texttt{bundledoc}: support/bundledoc

\texttt{snapshot.sty}: macros/latex/contrib/supported/snapshot

171 Marking changed parts of your document

One often needs clear indications of how a document has changed, but the commonest technique, “change bars”, requires surprisingly much trickery of the programmer (the problem being that \TeX{} ‘proper’ doesn’t provide the programmer with any information about the “current position” from which a putative start- or end-point of a bar might be calculated; \pdf\TeX{} \textit{does} provide the information, but we’re not aware yet of any programmer taking advantage of the fact to write a \pdf\TeX{}-based changebar package).

The simplest package that offers change bars is Peter Schmitt’s \texttt{backgrnd.tex}; this was written as a Plain \TeX{} application that patches the output routine, but it appears to work at least on simple \LaTeX{} documents. Wise \LaTeX{} users will be alerted by the information that \texttt{backgrnd} patches their output routine, and will watch its behaviour very carefully (patching the \LaTeX{} output routine is not something to undertake lightly...).

The longest-established solution is the \texttt{changebar} package, which uses \texttt{\special} commands supplied by the driver you’re using. You need therefore to tell the package which driver to generate \texttt{\special}s for (in the same way that you need to tell the \texttt{graphics} package); the list of available drivers is pretty restricted, but does include \texttt{dvips}. The package comes with a shell script \texttt{chbar.sh} (for use on Unix machines) that will compare two documents and generate a third which is marked-up with \texttt{changebar} macros to highlight changes.

The \texttt{vertbars} package uses the techniques of the \texttt{lineno} package (which must be present); it’s thus the smallest of the packages for change bar marking, since it leaves all the trickery to another package.

\texttt{backgrnd.tex}: macros/generic/backgrnd.tex

\texttt{changebar.sty}: macros/latex/contrib/supported/changebar

\texttt{lineno.sty}: macros/latex/contrib/supported/lineno

\texttt{vertbars.sty}: macros/latex/contrib/supported/misc/vertbars.sty

172 Conditional compilation and “comments”

While \LaTeX{} (or any other \TeX{}-derived package) isn’t really like a compiler, people regularly want to do compiler-like things using it. Common requirements are conditional ‘compilation’ and ‘block comments’, and several \LaTeX{}-specific means to this end are available.

The simple \texttt{\newcommand{\gobble}{[1]{}}} and \texttt{\iffalse ... \fi} aren’t really satisfactory (as a general solution) for comments, since the matter being skipped is nevertheless scanned by \TeX{}. The scanning imposes restrictions one what you’re allowed to skip; this may not be a problem in today’s job, but could return to bite you tomorrow. Furthermore, \texttt{\gobble} is pretty inefficient for any but trivial arguments, since all the matter to be skipped is copied to the argument stack before being ignored.

If your requirement is for a document from which whole chapters (or the like) are missing, consider the \LaTeX{} \texttt{\include/\includeonly} system. If you ‘\texttt{\include}’ your files (rather than \texttt{\input} them — see question 228), \LaTeX{} writes macro traces of what’s going on at the end of each chapter to the .aux file; by using \texttt{\includeonly}, you can give \LaTeX{} an exhaustive list of the files that are needed. Files that don’t get \texttt{\include}d are skipped entirely, but the document processing continues as if they were there, and page, footnote, and other numbers are not disturbed. Note that you can choose which sections you want included interactively, using the \texttt{askinclude} package.
If you want to select particular pages of your document, use Heiko Oberdiek’s *pagesel* or the *selectp* packages. You can do something similar with an existing PDF document (which you may have compiled using *pdflatex* in the first place), using the *pdfpages* package. The job is then done with a document looking like:

\documentclass{article}
\usepackage[final]{pdfpages}
\begin{document}
\includepdf[pages=30-40]{yoursource.pdf}
\end{document}

(To include all of the document, you write

\includepdf[pages=-]{yoursource.pdf}

omitting the start and end pages in the optional argument.)

If you want flexible facilities for including or excluding small portions of a file, consider the *comment*, *version* or *optional* packages.

*comment* allows you to declare areas of a document to be included or excluded; you make these declarations in the preamble of your file. Its exclusion method is pretty robust, and can cope with ill-formed bunches of text (e.g., with unbalanced braces or \if commands).

*version* offers similar facilities to *comment.sty*; it’s far “lighter weight”, but is less robust (and in particular, cannot deal with very large areas of text being included/excluded).

*optional* defines a command \opt; its first argument is an ‘inclusion flag’, and its second is text to be included or excluded. Text to be included or excluded must be well-formed (nothing mismatched), and should not be too big — if a large body of text is needed, \input should be used in the argument. The documentation (in the package file itself) tells you how to declare which sections are to be included: this can be done in the document preamble, but the documentation also suggests ways in which it can be done on the command line that invokes \TeX, or interactively.

Finally, *verbatim* (which should be available in any distribution) defines a context environment, which enables the dedicated user of the source text editor to suppress bits of a \TeX source file.

*askinclude.sty*: macros/latex/contrib/other/misc/askinclude.sty

*comment.sty*: macros/latex/contrib/other/comment

*optional.sty*: macros/latex/contrib/other/misc/optional.sty

*pagesel.sty*: Distributed with Heiko Oberdiek’s packages macros/latex/contrib/supported/oberdiek

*pdfpages.sty*: macros/latex/contrib/supported/pdfpages

*selectp.sty*: macros/latex/contrib/other/misc/selectp.sty

*verbatim.sty*: Distributed as part of macros/latex/required/tools

*version.sty*: macros/latex/contrib/other/misc/version.sty

## 173 Bits of document from other directories

A common way of constructing a large document is to break it into a set of files (for example, one per chapter) and to keep everything related to each of these subsidiary files in a subdirectory.

Unfortunately, \TeX doesn’t have a changeable “current directory”, so that all files you refer to have to be specified relative to the same directory as the main file. Most people find this counter-intuitive.

It may be appropriate to use the “path extension” technique of question 110 to deal with this problem. However, if there several files with the same name in your document, such as chapter1/fig1.eps and chapter2/fig1.eps, you’re not giving \TeX any hint as to which you’re referring to when in the main chapter file you say \input {sect1}; while this is readily soluble in the case of human-prepared files (just don’t name them all the same), automatically produced files have a way of having repetitious names, and changing them is a procedure prone to error.

The *import* package comes to your help here: it defines an \import command that accepts a full path name and the name of a file in that directory, and arranges things to “work properly”. So, for example, if /home/friend/results.tex contains
then \import{/home/friend/}{results} will include both graph and explanation as one might hope. A \subimport command does the same sort of thing for a subdirectory (a relative path rather than an absolute one), and there are corresponding \includefrom and \subincludefrom commands.

\textbf{import.sty:} \url{macros/latex/contrib/supported/misc/import.sty}

\section{Version control using RCS or CVS}

If you use RCS or CVS to maintain your \LaTeX\ documents under version control, you may need some mechanism for including the RCS keywords in your document, in such a way that they can be typeset (that is, rather than just hiding them inside a comment).

The most complete solution is to use the \LaTeX\ package \texttt{rcs}, which allows you to parse and display the contents of RCS keyword fields in an extremely flexible way.

If you need a solution which works without using external packages, or which will work in plain \TeX, then you can use the following quick (but usually adequate) hack:

\begin{verbatim}
\def\RCS$#1: #2 \{\expandafter\def\csname RCS#1\endcsname{#2}}
\RCS$Revision: 1.95 \$ % or any RCS keyword
\RCS$Date: 2002/09/04 13:16:53 \$
...
\date{Revision \RCSRevision, \RCSDate}
\end{verbatim}

\textbf{rcs.sty:} \url{macros/latex/contrib/supported/rcs}

\section{Makefiles for \LaTeX\ documents}

\LaTeX\ is a tricky beast for running \texttt{make} on: the need to instruct \LaTeX\ to run several times for essentially different reasons (for example, “get the table of contents stable”, “get the labels stable”, “add the bibliography”, “add the index”) is actually rather difficult to express in the ‘ordinary’ sort of dependency graph that one constructs for \texttt{make}.

For this reason, the only \texttt{make}-like package on CTAN (for a long time) was \texttt{latexmk}, which is a Perl script that analyses your \LaTeX\ source for its dependencies, runs \BibTeX\ or \texttt{makeindex} as and when it notices that those programs’ input (parts of the \texttt{.aux} file, or the \texttt{.idx} file, respectively) has changed, and so on. \texttt{Latexmk} is a fine solution (and was used in generating printable versions of these FAQs for a long time); it has recently been upgraded and has many bells and whistles that allow it to operate as if it were a poor man’s WYSIWYG system.

The \texttt{texinfo} system (see question 16) comes with a utility called \texttt{texi2dvi}, which is capable of “converting” either \LaTeX\ or \texttt{texinfo} files into DVI (or into PDF, using PDF\LaTeX{}).

A later contribution is the bundle \texttt{latexmake}, which offers a set of \texttt{make} rules that invoke \texttt{texi2dvi} as necessary.

The curious may examine the rules employed to run the present FAQ through \LaTeX:\ we don’t present them as a complete solution, but some of the tricks employed are surely re-usable.

\begin{itemize}
\item \textit{FAQ distribution:} \url{help/uk-tex-faq}
\item \textit{latexmake:} \url{support/latexmake}
\item \textit{latexmk:} \url{support/latexmk}
\item \textit{texi2dvi:} Distributed as part of \url{macros/texinfo/texinfo}
\end{itemize}

\section{Hyphenation}

\textbf{Q.6 Hyphenation}

\textbf{176 My words aren’t being hyphenated}

Let’s assume you’ve selected the right \TeX\ ‘language’ — as explained in question 41, you’re not likely to get the correct results typesetting one language using the hyphenation rules of another. (Select the proper language, using \texttt{babel} if you’re a \LaTeX\ user. This may reveal that you need another set of hyphenation patterns; see question 180 for advice on how to install it.)

So what else can go wrong?

\begin{itemize}
\item Since \LaTeX\ version 3.0, the limits on how near to either end of a word hyphenation may take place have been programmable (see question 177), and for some reason
\end{itemize}
the values in question may have been corrupted in some macros you are using. \TeX won’t hyphenate less than \left\hyphenmin characters after the start of a word, nor less than \right\hyphenmin before the end of a word; thus it won’t hyphenate a word shorter than the sum of the two minima, at all. For example, since the minima are 2 and 3 for English, \TeX won’t hyphenate a word shorter than 5 letters long, if it believes the word to be English.

- \TeX won’t hyphenate a word that’s already been hyphenated. For example, the (caricature) English surname Smyth-Postlethwaite wouldn’t hyphenate, which could be troublesome. This is correct English typesetting style (it may not be correct for other languages), but if needs must, you can replace the hyphen in the name with a \hyphen command, defined

\def\hyph{\penalty0\hskip0pt\relax}

This is not the sort of thing this FAQ would ordinarily recommend... The \hyphenat package defines a bundle of such commands (for introducing hyphenation points at various punctuation characters).

- There may be accents in the word. The causes of and remedies for this effect are discussed in question 179.

- The hyphenation may simply not have been spotted; while \TeX’s algorithm is good, it’s not infallible, and it does miss perfectly good hyphenations in some languages. When this happens, you need to give \TeX explicit instructions on how to hyphenate.

The \hyphenation command allows you to give explicit instructions. Provided that the word will hyphenate at all (that is, it is not prevented from hyphenating by any of the other restrictions above), the command will override anything the hyphenation patterns might dictate. The command takes one or more hyphenated words as argument — \hyphenation{ana-lysis pot-able}; note that (as here, for analysis) you can use the command to overrule \TeX’s choice of hyphenation (ana-lysis is the British etymological hyphenation; some feel the American hyphenation feels ‘unfortunate’...).

\hyphenat.sty: macros/latex/contrib/supported/hyphenat

177  Weird hyphenation of words

If your words are being h-yphenated, like this, with just single letters at the beginning or the end of the word, you may have a version mismatch problem. \TeX’s hyphenation system changed between version 2.9 and 3.0, and macros written for use with version 2.9 can have this effect with a version 3.0 system. If you are using Plain \TeX, make sure your plain.tex file has a version number which is at least 3.0, and rebuild your format. If you are using \LaTeX, your best plan is to upgrade to \LaTeX 2ε. If for some reason you can’t, the last version of \LaTeX 2.09 (released on 25 March 1992) is still available (for the time being at least) and ought to solve this problem.

If you’re using \LaTeX 2ε, the problem probably arises from your hyphen.cfg file, which has to be created if you’re using a multi-lingual version.

A further source of oddity can derive from the 1995 release of Cork-encoded fonts (see question 42), which introduced an alternative hyphen character. The \LaTeX 2ε configuration files in the font release specified use of the alternative hyphen, and this could produce odd effects with words containing an explicit hyphen. The font configuration files in the December 1995 release of \LaTeX 2ε do not use the alternative hyphen character, and therefore removed this source of problems; the solution, again, is to upgrade your \LaTeX.

\LaTeX 2.09: obsolete/macros/latex209/distribs/latex209.tar

plain.tex: macros/plain/base

178  (Merely) peculiar hyphenation

You may have found that \TeX’s famed automatic word-division does not produce the break-points recommended by your dictionary. This may be because \TeX is set up for American English, whose rules for word division (as specified, for example, in Webster’s Dictionary) are completely different from the British ones (as specified, for example, in the Oxford Dictionaries). This problem is being addressed by the UK \TeX User community (see Baskerville, issue 4.4) but an entirely satisfactory solution will take time; the current status is to be found on CTAN (see question 180 for instructions on adding this new “language”).
Accented words aren’t hyphenated

\LaTeX{}’s algorithm for hyphenation gives up when it encounters an \texttt{\accent} command; there are good reasons for this, but it means that quality typesetting in non-English languages can be difficult.

For \TeX{} macro packages, you can avoid the effect by using an appropriately encoded font (for example, a Cork-encoded font — see question \ref{question-42}) which contains accented letters as single glyphs. \LaTeX{} users can achieve this end simply by adding the command
\begin{verbatim}
\usepackage[T1]{fontenc}
\end{verbatim}

to the preamble of their document. Other encodings (notably LY1, promoted by Y&Y — see question \ref{question-55}) may be used in place of T1. Indeed, most current 8-bit \TeX{} font encodings will ‘work’ with the relevant sets of hyphenation patterns.

In the future, perhaps, Omega (see question \ref{question-251}) will provide a rather different solution.

Using a new language with Babel

\textit{Babel} is capable of working with a large range of languages, and a new user often wants to use a language that her \TeX{} installation is not set up to employ. Simply asking Babel to use the language, with the command
\begin{verbatim}
\usepackage[catalan]{babel}
\end{verbatim}
provides the warning message

Package babel Warning: No hyphenation patterns were loaded for the language ‘Catalan’
\begin{verbatim}
(babel) I will use the patterns loaded for \language=0 instead.
\end{verbatim}

(The last line of the above has been wrapped to fit on the page.)

The problem is that your \TeX{} system doesn’t know how to hyphenate Catalan text: you need to tell it how before Babel can do its work properly. To do this, for \LaTeX{} installations, one needs to change \texttt{language.dat} (which is part of the Babel installation); it will contain a line
\begin{verbatim}
%catalan cahyphen.tex
\end{verbatim}
which, if you remove the comment marker, is supposed to instruct \LaTeX{} to load Catalan hyphenation patterns when you tell it to build a new format.

Unfortunately, in many Babel distributions, the line just isn’t right — you need to check the name of the file containing the patterns you’re going to use. As you can see, in the author’s system, the name is supposed to be cahyphen.tex; however the file actually present on the system is cahyph.tex — fortunately, the error should prove little more than an inconvenience (most of the files are in better distributions anyway, but an elusive one may be found on CTAN; if you have to retrieve a new file, ensure that it’s correctly installed, for which see question \ref{question-107}).

Finally, you need to regenerate the formats used (in fact, most users of Babel are using it in their \LaTeX{} documents, so regenerating the \LaTeX{}-related formats will ordinarily be enough; however, the author always generates the lot, regardless).

\TeX{}, \fpTeX{} To regenerate all formats, do:
\begin{verbatim}
fmtutil --all
\end{verbatim}
If you’re willing to think through what you’re doing (this is \textit{not} for the faint-hearted), you can select a sequence of formats and for each one, run:
\begin{verbatim}
fmtutil --byfmt ⟨formatname⟩
\end{verbatim}
where \texttt{formatname} is something like ‘latex’, or:
\begin{verbatim}
fmtutil --byhyphen ⟨hyphenfile⟩
\end{verbatim}
where \texttt{hyphenfile} is the file specifying hyphenation to the format — usually \texttt{language.dat}

\MikTeX{} On a \textit{MikTeX} distribution earlier than v2.0, do:
\begin{verbatim}
Start→Programs→MikTeX→Maintenance→Create all format files
\end{verbatim}
or get a DOS window and run:
\begin{verbatim}
inixemf --dump
\end{verbatim}
On a \textit{MikTeX} distribution v2.0 or later, the whole procedure can be done via the GUI. To select the new language, do:
Start → Programs → MiKTeX 2 → MiKTeX Options, and select the Languages tab. Select your language from the list, press the Apply button, and then the OK button. Then select the General tab and press the Update Now button.

Otherwise, edit the `language.dat` file (as outlined above), and then run:

```
initexmf --dump
```

just as for a pre-v2.0 system.

**Caveat:** It is (just) possible that your TeX system may run out of “pattern memory” while generating the new format. Most TeX implementations have fixed-size arrays for storing the details of hyphenation patterns, but although their size is adjustable in most modern distributions, actually changing the size is a fiddle. If you do find you’ve run out of memory, it may be worth scanning the list of languages in your `language.dat` file to see whether any could reasonably be removed.

*babel:* `macros/latex/required/babel`

*hyphenation patterns:* `language/hyphenation`

### 181 Stopping all hyphenation

It may seem an odd thing to want to do (after all, one of TeX’s great advertised virtues is the quality of its hyphenation) but it’s sometimes necessary. The real problem is, that the quality of TeX’s is by default largely dependent on the presence of hyphenation; if you want to abandon hyphenation, something has to give.

TeX (slightly confusingly) offers four possible mechanisms for suppressing hyphenation (there were only two prior to the extensions that arrived with TeX version 3).

First, one can set the hyphenation penalties `\hyphenpenalty` and `\exhyphenpenalty` to an ‘infinite’ value (that is to say, 10000). This means that all hyphenations will sufficiently penalise the line that would contain them, that the hyphenation won’t happen. The disadvantage of this method is that TeX will re-evaluate any paragraph for which hyphenations might help, which will slow TeX down.

Second, one can select a language for which no hyphenation patterns exist. Some distributions create a language `nohyphenation`, and the `hyphenat` package uses this technique for its `\nohyphens` command which sets its argument without any hyphenation.

Third, one can set `\lefthyphenmin` and/or `\righthyphenmin` to a sufficiently large value that no hyphenation could possibly succeed, since the minimum is larger than the the length of the longest word TeX is willing to hyphenate (the appropriate value is 62).

Fourth, one can suppress hyphenation for all text using the current font by the command

```
\hyphenchar\font=-1
```

This isn’t a particularly practical way for users to suppress hyphenation — the command has to be issued for every font the document uses — but it’s how TeX itself suppresses hyphenation in `tt` and other fixed-width fonts.

Which of the techniques you should use depends on what you actually want to do. If the text whose hyphenation is to be suppressed runs for less than a paragraph, your only choice is the no-hyphens language: the language value is preserved along with the text (in the same way that the current font is); the values for penalties and hyphen minima active at the end of a paragraph are used when hyphenation is calculated.

Contrariwise, if you are writing a multilanguage document using the `babel` package, you cannot suppress hyphenation throughout using either the no-hyphens language or the hyphen minima: all those values get changed at a `babel` language switch: use the penalties instead.

If you simply switch off hyphenation for a good bit of text, the output will have a jagged edge (with many lines seriously overfull), and your (Li)TeX run will bombard you with warnings about overfull and underfull lines. To avoid this you have two options. You may use `\sloppy` (or its environment version `sloppypar`), and have TeX stretch what would otherwise be underfull lines to fill the space offered, and wrap other lines, while prematurely wrapping overfull lines and stretching the remainder. Alternatively, you may set the text ragged right (see question 137), and at least get rid of the overfull lines; this technique is ‘traditional’ (in the sense that typists do it) and may be expected to appeal to the specifiers of eccentric document layouts (such as those for dissertations), but for once their sense conforms with typographic style. (Or at least, style constrained in this curious way.)
Q.7  Odds and ends

182  Typesetting all those \TeX-related logos

Knuth was making a particular point about the capabilities of \TeX{} when he defined the logo. Unfortunately, many believe, he thereby opened floodgates to give the world a whole range of rather silly ‘bumpy road’ logos such as \texttt{\LaTeX}\textsc{s-}, \texttt{Pic\TeX{}}, \texttt{Bib\TeX{}}, and so on, produced in a flurry of different fonts, sizes, and baselines — indeed, everything one might hope to cause them to obstruct the reading process. In particular, Lamport invented \LaTeX\textsc{s} (silly enough in itself) and marketing input from Addison-Wesley led to the even stranger current logo \LaTeX\textsc{2e}.

Sensible users don’t have to follow this stuff wherever it goes, but, for those who insist, a large collection of logos is defined in the \texttt{texnames} package (but note that this set of macros isn’t entirely reliable in \LaTeX\textsc{2e}). The \texttt{METAFONT} and MetaPost logos can be set in fonts that \LaTeX\textsc{2e} knows about (so that they scale with the surrounding text) using the \texttt{mflogo} package; but be aware that booby-traps surround the use of the Knuthian font for MetaPost (you might get \texttt{META O T}). You needn’t despair, however — the author himself uses just ‘MetaPost’.

For those who don’t wish to acquire the ‘proper’ logos, the canonical thing to do is to say \texttt{AMS-\TeX{}} (AMS-\TeX{}) for AMS-\TeX{}, \texttt{Pic\TeX{}} (Pic\TeX{}) for \texttt{Pic\TeX{}}, \texttt{Bib\TeX{}} (Bib\TeX{}) for \texttt{Bib\TeX{}}, and so on.

While the author of this FAQ list can’t quite bring himself to do away with the bumpy-road logos herein, he regularly advises everyone else to...

\texttt{mflogo.sty}: \texttt{macros/latex/contrib/supported/mflogo}

\texttt{texnames.sty}: \texttt{macros/eplain/texnames.sty}

183  Referring to things by their name

\LaTeX{}’s labelling mechanism is designed for the impersonal world of the academic publication, in which everything has a number: an extension is necessary if we are to record the \textit{name} of things we’ve labelled. The two packages available extend the \LaTeX{} sectioning commands to provide reference by the name of the section.

The \texttt{titleref} package is a simple extension which provides the command \texttt{\titleref}; it is a stand-alone package.

The \texttt{nameref} package employs the techniques of the \texttt{hyperref} package to define a \texttt{\nameref} command; it will work in documents that have \texttt{hyperref} loaded.

\texttt{nameref.sty}: Distributed with \texttt{macros/latex/contrib/supported/hyperref}

\texttt{titleref.sty}: \texttt{macros/latex/contrib/other/misc/titleref.sty}

184  How to do bold-tt or bold-sc

\LaTeX{}, as delivered, offers no means of handling bold “teletype” or small-caps fonts. There’s a practical reason for this (Knuth never designed such fonts), but there are typographical considerations too (the “medium weight” \texttt{cmtt} font is already pretty bold (by comparison with other fixed-width fonts), and bold small-caps is not popular with many professional typographers).

There’s a set of “extra” \texttt{METAFONT} files on that provide bold versions of both \texttt{cmtt} and \texttt{cmss} (the small caps font). With modern \TeX{} distributions, one may bring these fonts into use simply by placing them in an appropriate place in the \texttt{texmf} tree (see question \texttt{108}); \TeX{} (and friends) will automatically build whatever font files they need when you first make reference to them. There’s a jiffy package \texttt{bold-extra} that builds the necessary font data structures so that you can use the fonts within \LaTeX{}.

If you need to use Type 1 fonts, you can’t proceed with Knuth-style fonts, since there are no Type 1 versions of the \texttt{mf-extra} set. However, commercial fixed-width fonts (including the default \texttt{Courier}) almost always come with a bold variant, so that’s not a problem. Furthermore PSNFSS (see question \texttt{81}) typically provides “faked” small caps fonts, and has no compunctions about providing them in a bold form.

\texttt{bold-extra.sty}: \texttt{macros/latex/contrib/other/misc/bold-extra.sty}

\texttt{bold tt and small caps fonts}: \texttt{fonts/cm/mf-extra/bold}
Symbols for the number sets

It is a good idea to have commands such as \(\mathbb{R}\) for the real numbers and other standard number sets. Traditionally these were typeset in bold. Because mathematicians usually do not have access to bold chalk, they invented the special symbols that are now often used for \(\mathbb{R}, \mathbb{N}, \mathbb{C}, etc.\) These symbols are known as “blackboard bold”. Before insisting on using them, consider whether going back to the old system of ordinary bold might not be acceptable (it is certainly simpler).

A set of blackboard bold capitals is available in the AMS “msbm” fonts (“msbm” is available at a range of design sizes, with names such as “msbm10”). The pair of font families (the other is called “msam”) have a large number of mathematical symbols to supplement the ones in the standard \(\TeX\) distribution, and are available in Type 1 format with most modern distributions. Support files for using the fonts, both under Plain \(\TeX\) and \(\LaTeX\) (packages amssymb and amsfonts), are available.

Another complete set of blackboard bold fonts written in METAFONT is the \textit{bbold} family. This set has the interesting property of offering blackboard bold forms of lowercase letters, something rather rarely seen on actual blackboards; the font source directory also contains sources for a \(\LaTeX\) package that enables use of the fonts. The fonts are not available in Type 1 format.

An alternative source of Type 1 fonts with blackboard bold characters may be found in the steadily increasing set of complete families, both commercial and free, that have been prepared for use with \(\LaTeX\) (see question 85). Of the free sets, the \textit{txfonts} and \textit{pxfonts} families both come with replicas of \textit{msam} and \textit{msbm}, and the \textit{mathpazo} family includes a “mathematically significant” choice of blackboard bold characters.

The “lazy person’s” blackboard bold macros:

\begin{verbatim}
\newcommand{\R}{{\sf R\hspace*{-0.9ex}\rule{0.15ex}{1.5ex}\hspace*{0.9ex}}}
\newcommand{\N}{{\sf N\hspace*{-1.0ex}\rule{0.15ex}{1.3ex}\hspace*{1.0ex}}}
\newcommand{\Q}{{\sf Q\hspace*{-1.1ex}\rule{0.15ex}{1.5ex}\hspace*{1.1ex}}}
\newcommand{\C}{{\sf C\hspace*{-0.9ex}\rule{0.15ex}{1.3ex}\hspace*{0.9ex}}}
\end{verbatim}

work well at normal size if the surrounding text is \texttt{cmr10}. However, they are not part of a proper maths font, and so do not work in sub- and superscripts. Moreover, the size and position of the vertical bar can be affected by the font of the surrounding text.

AMS support files (Plain): \texttt{fonts/amsfonts/plaintex}
AMS support files (LaTeX): \texttt{fonts/amsfonts/latex}
AMS symbol fonts: \texttt{fonts/amsfonts/sources/symbols}
AMS symbol fonts in Type 1 format: Browse \texttt{fonts/amsfonts/ps-type1}
bbold fonts: \texttt{fonts/bbold}
mathpazo fonts: \texttt{fonts/mathpazo}
\textit{pxfonts}: \texttt{fonts/pxfonts}
\textit{txfonts}: \texttt{fonts/txfonts}

Better script fonts for maths

The font selected by \texttt{\textbackslash mathcal} is the only script font ‘built in’. However, there are other useful calligraphic fonts included with modern \(\TeX\) distributions.

\textbf{Euler} The \texttt{eucal} package (part of most sensible \(\TeX\) distributions; the fonts are part of the AMS font set) gives a slightly curlier font than the default. The package changes the font that is selected by \texttt{\textbackslash mathcal}.

Type 1 versions of the fonts are available in the AMS fonts distribution.

\textbf{RSFS} The \texttt{mathrsfs} package uses a really fancy script font (the name stands for “Ralph Smith’s Formal Script”) which is already part of most modern \(\TeX\) distributions. The package creates a new command \texttt{\textbackslash mathscrbold}.

Type 1 versions of the font have been made available by Taco Hoekwater.
Zapf Chancery is the standard PostScript calligraphic font. There is no package but you can easily make it available by means of the command
\DeclareMathAlphabet{\mathscr}{OT1}{pzc}%
{m}{it}
in your preamble. You may find the font rather too big; if so, you can use a scaled version of it like this:
\DeclareFontFamily{OT1}{pzc}{}
\DeclareFontShape{OT1}{pzc}{m}{it}%
{<-> s * [0.900] pzcmi7t}{}
\DeclareMathAlphabet{\mathscr}{OT1}{pzc}%
{m}{it}

Adobe Zapf Chancery (which the above examples use) is distributed in any but the most basic PostScript printers. A substantially identical font (to the extent that the same metrics may be used) is available from URW and is distributed with ghostscript.

Examples of the available styles are available on CTAN.
eucal.sty: fonts/amsfonts/latex/eucal.sty
euler fonts: fonts/amsfonts/sources/euler
euler fonts, in Type 1 format: fonts/amsfonts/ps-type1
ghostscript: Browse nonfree/support/ghostscript
mathrsfs.sty: Distributed as part of macros/latex/contrib/supported/jknappen
rsfs fonts: fonts/rsfs
rsfs fonts, in Type 1 format: fonts/rsfs/ps-type1/hoekwater
Script font examples: info/symbols/math/scriptfonts.pdf

187 Setting bold Greek letters in \LaTeX

The issue here is complicated by the fact that \textbf{} (the command for setting bold text in \LaTeX maths) affects a select few mathematical symbols (the uppercase Greek letters). However lower-case Greek letters behave differently from upper-case Greek letters (due to Knuth’s esoteric font encoding decisions). However, \textbf{} can’t be used even for upper-case Greek letters in the \texttt{AMS-\LaTeX} amsmath package, which disables this font-switching and you must use one of the techniques outlined below.

The Plain \TeX solution does work, in a limited way:
{\textbf{\theta}}
but \textbf{} may not be used in maths mode, so this ‘solution’ requires arcana such as:
$\ldots \mbox{\textbf{\theta}} \ldots$
which then causes problems in superscripts, etc.

These problems may be addressed by using a bold mathematics package.
• The \texttt{bm} package, which is part of the \texttt{AMSTeX} tools distribution, defines a command \texttt{\bm{}} which may be used anywhere in maths mode.

• The \texttt{amsbsy} package (which is part of \texttt{AMS-\LaTeX}) defines a command \texttt{\boldsymbol{}} which (though slightly less comprehensive than \texttt{\bm{}}) covers almost all common cases.

All these solutions cover all mathematical symbols, not merely Greek letters.
bm.sty: Distributed as part of macros/latex/required/tools
amsbsy.sty: Distributed as part of the \texttt{AMS-\LaTeX} distribution macros/latex/required/amslatex
amsmath.sty: Distributed as part of the \texttt{AMS-\LaTeX} distribution macros/latex/required/amslatex

93
The Principal Value Integral symbol

This symbol (an integral sign, ‘crossed’) does not appear in any of the fonts ordinarily available to (L)\TeX users, but it can be created by use of the following macros:
\def\Xint#1{\mathchoice
{\XXint\displaystyle\textstyle{#1}}%
{\XXint\textstyle\scriptstyle{#1}}%
{\XXint\scriptstyle\scriptscriptstyle{#1}}%
{\XXint\scriptscriptstyle\scriptscriptstyle{#1}}%
\!\int}
\def\XXint#1#2#3{{\setbox0=\hbox{$#1\{#2#3\}$}\vcenter\hbox{$#2#3$}}\kern-.5\wd0}
\def\ddashint{\Xint=}
\def\dashint{\Xint-}
\dashint gives a single-dashed integral sign, \ddashint a double-dashed one.

How to use the underscore character

The underscore character \_ is ordinarily used in \TeX to indicate a subscript in maths mode; if you type \_ in the course of ordinary text, \TeX will complain. If you’re writing a document which will contain a large number of underscore characters, the prospect of typing \_ (or, worse, \textunderscore) for every one of them will daunt most ordinary people.

Moderately skilled macro programmers can readily generate a quick hack to permit typing \_ to mean ‘text underscore’. However, the code is somewhat tricky, and more importantly there are significant points where it’s easy to get it wrong. There is therefore a package underscore which provides a general solution to this requirement.

There is a problem, though: OT1 text fonts don’t contain an underscore character, unless they’re in the typewriter version of the encoding (used by fixed-width fonts such as cmtt). So either you must ensure that your underscore characters only occur in text set in a typewriter font, or you must use a fuller encoding, such as T1, which has an underscore character in every font.

If the requirement is only for occasional uses of underscores, it may be acceptable to use the following construct:
\def\us{\char\_}
\texttt{create\us process}

The construction isn’t in the least robust (in the normal English sense of the word), but it is robust under expansion (i.e., the \TeX sense of the word); so use it with care, but don’t worry about section headings and the like.

underscore.sty: macros/latex/contrib/other/misc/underscore.sty

How to type an ‘@’ sign?

Long ago, some packages used to make the ‘@’ sign active, so that special measures were needed to type it. While those packages are still in principle available, few people use them, and those that do use them have ready access to rather good documentation.

Ordinary people (such as the author of this FAQ) need only type ‘@’.

Typesetting the Euro sign

The European currency “Euro” is represented by a symbol of somewhat dubious design, but it’s an important currency and (L)\TeX users need to typeset it.

Note that the Commission of the European Community at first deemed that the Euro symbol should always be set in a sans-serif font; fortunately, this eccentric ruling has now been rescinded, and one may apply best typesetting efforts to making it appear at least slightly “respectable” (typographically).

The TS1-encoded fonts provided as part of the EC font distribution provide Euro glyphs. The fonts are called Text Companion (TC) fonts, and offer the same range of faces as do the EC fonts themselves. The textcomp package provides a \texteuro command for accessing the symbol, which selects a symbol to match the surrounding text. The design of the symbol in the TC fonts is not universally loved... Nevertheless, use the TC font version of the symbol if you are producing documents using Knuth’s Computer Modern Fonts.
The \texttt{latin9} input encoding defined by the \texttt{inputenc} package has a euro character defined (character position 164, occupied in other ISO Latin character sets by the “currency symbol”). The encoding uses the command \texttt{\texteuro} for the character; at present that command is only available from the \texttt{textcomp} package. There is a Microsoft code page position, too, but standardisation of such things proceeds via rather different routes and the \LaTeX{} project hasn’t yet been given details of the change.

Outline fonts which contain nothing but Euro symbols are available (free) from Adobe — the file is packaged as a \texttt{Windows} self-extracting executable, but it may be decoded as a \texttt{.zip} format archive on other operating systems. The \texttt{euro} bundle contains metrics, \texttt{dvips} map files, and macros (for Plain \LaTeX{} and \LaTeX{}), for using these fonts in documents. \LaTeX{} users will find two packages in the bundle: \texttt{eurosans} only offers the sans-serif version (to conform with the obsolete ruling about sans-serif-only symbols; the package provides the command \texttt{\euro}), whereas \texttt{euro} matches the Euro symbol with the surrounding text (providing the command \texttt{\EUR}). To use either package with the \texttt{latin9} encoding, you need to define \texttt{\texteuro} as an alias for the euro command the package defines.

The Adobe fonts are probably the best bet for use in non-Computer Modern environments. They are apparently designed to fit with Adobe Times, Helvetica and Courier, but can probably fit with a wider range of modern fonts.

The \texttt{eurofont} package provides a compendious analysis of the “problem of the euro symbol” in its documentation, and offers macros for configuring the source of the glyphs to be used; however, it seems rather large for everyday use.

Euro symbols are found in several other places, which we list here for completeness.

The \texttt{marvosym} fonts contain a Euro symbol among many other good things; the font on CTAN is not Adobe \textsc{ATM} compatible, but a compatible version is available free from Y\&Y. The font on CTAN comes with a set of macros to typeset all the symbols it contains.

Other \textsc{METAFONT}-based bundles containing Euro symbols are to be found in \texttt{china2e} (whose primary aim is Chinese dates and suchlike matters) and the \texttt{eurosym} fonts.

\texttt{china2e bundle: macros/latex/contrib/supported/china2e}

\texttt{EC fonts: fonts/ec}

\texttt{euro fonts: fonts/euro}

\texttt{eurofont.sty: macros/latex/contrib/supported/eurofont}

\texttt{eurosym fonts: fonts/eurosym}

\texttt{marvosym fonts: fonts/psfonts/marvosym}

\texttt{textcomp.sty: Part of the \LaTeX{} distribution}.

\section*{S Macro programming}

\subsection*{S.1 “Generic” macros}

\texttt{192 Finding the width of a letter, word, or phrase}

Put the word in a box, and measure the width of the box. For example,

\begin{verbatim}
\newdimen\stringwidth
\setbox0=\hbox{hi}
\stringwidth=\wd0
\end{verbatim}

Note that if the quantity in the \texttt{\hbox} is a phrase, the actual measurement only approximates the width that the phrase will occupy in running text, since the inter-word glue can be adjusted in paragraph mode.

The same sort of thing is expressed in \LaTeX{} by:

\begin{verbatim}
\newlength{\gnat}
\settowidth{\gnat}{\textbf{small}}
\end{verbatim}

This sets the value of the length command \texttt{\gnat} to the width of “small” in bold-face text.
193  Patching existing commands

In the general case (possibly sticking something in the middle of an existing command) this is difficult. However, the common requirement, to add some code at the beginning or the end of an existing command, is conceptually quite easy. Suppose we want to define a version of a command that does some small extension of its original definition: we would naturally write:

\renewcommand{\splat}{\mumble\splat}

However, this would not work: a call to \splat would execute \mumble, and the call the redefined \splat again; this is an infinite recursive loop, that will quickly exhaust \TeX’s memory.

Fortunately, the \TeX primitive \let command comes to our rescue; it allows us to take a “snapshot” of the current state of a command, which we can then use in the redefinition of the command. So:

\let\OldSmooth\smooth
\renewcommand{\smooth}{\mumble\OldSmooth}

effects the required patch, safely. Adding things at the end of a command works similarly. If \smooth takes arguments, one must pass them on:

\renewcommand{\smooth}[2]{\mumble\OldSmooth{#1}{#2}}

The general case may be achieved in two ways. First, one can use the \LaTeX command \CheckCommand; this compares an existing command with the definition you give it, and issues a warning if two don’t match. Use is therefore:

\CheckCommand{\complex}{⟨original definition⟩}
\renewcommand{\complex}{⟨new definition⟩}

This technique is obviously somewhat laborious, but if the original command comes from a source that’s liable to change under the control of someone else, it does at least warn you that your patch is in danger of going wrong.

Otherwise, \LaTeX users may use one of the patch or patchcmd systems.

Patch gives you an ingenious (and difficult to understand) mechanism, and comes as an old-style \LaTeX documented macro file. Sadly the old-style doc macros are no longer available, but the file (patch.doc) may be input directly, and the documentation may be read (un-typeset). Roughly speaking, one gives the command a set of instructions analogous to sed substitutions, and it regenerates the command thus amended. The author of this FAQ has (slightly reluctantly) given up using patch...

The patchcmd package addresses a slightly simpler task, by restricting the set of commands that you may patch: you mayn’t patch any command that has an optional argument, though it does deal with the case of commands defined with \DeclareRobustCommand. The package defines a \patchcommand command, that takes three arguments: the command to patch, stuff to stick at the front of its definition, and stuff to stick on the end of its definition. So, if \b contains “b”, then \patchcommand{\b}{a}{c} will produce a new version of \b that contains “abc”:

patch.doc: macros/generic/patch.doc
patchcommand.sty: macros/latex/contrib/supported/patchcmd

194  Comparing the “job name”

The token \jobname amusingly produces a sequence of characters whose category code is 12 (‘other’), regardless of what the characters actually are. Since one inevitably has to compare a macro with the contents of another macro (using \ifx, somewhere) one needs to create a macro whose expansion looks the same as the expansion of \jobname. We find we can do this with \meaning, if we strip the “\show command” prefix.

The full command looks like:

\def\StripPrefix#1>{}
\def\jobs#1{FF\fi
  \def\predicate{#1}%
  \edef\predicate{\expandafter\StripPrefix\meaning\predicate}%
  \edef\job{\jobname}%
  \ifx\job\predicate%
And it’s used as:
Note that the command \StripPrefix need not be defined if you’re using \TeX — there’s already an internal command (see question 206) \strip@prefix that you can use.

195 Is the argument a number?

\TeX’s own lexical analysis doesn’t offer the macro programmer terribly much support: while category codes will distinguish letters (or what \TeX currently thinks of as letters) from everything else, there’s no support for analysing numbers.

The simple-minded solution is to compare numeric characters with the characters of the argument, one by one, by a sequence of direct tests, and to declare the argument “not a number” if any character fails all comparisons:
\begin{verbatim}
\ifx1#1\else\ifx2#1\else\ifx9#1\else\isanumfalse\fi\fi\fi
\end{verbatim}
which one would then use in a tail-recursing macro to gobble an argument. One could do slightly better by assuming (pretty safely) that the digits’ character codes are consecutive:
\begin{verbatim}
\ifnum'#1<0 \isanumfalse\else\ifnum'#1>'9 \isanumfalse\fi\fi
\end{verbatim}
again used in tail-recursion. However, these forms aren’t very satisfactory: getting the recursion “right” is troublesome (it has a tendency to gobble spaces in the argument), and in any case \TeX itself has mechanisms for reading numbers, and it would be nice to use them.

Donald Arseneau’s cite package offers the following test for an argument being a strictly positive integer:
\begin{verbatim}
\def\IsPositive#1{\TT\ifcat_\ifnum0<0#1 _\else A\fi\}
\end{verbatim}
which can be adapted to a test for a non-negative integer thus:
\begin{verbatim}
\def\IsNonNegative{\ifcat_\ifnum9<1#1 _\else A\fi\}
\end{verbatim}
or a test for any integer:
\begin{verbatim}
\def\gobble#1{}\def\gobbleminus{\futurelet\temp\gobm}\def\gobm{\ifx-\temp\expandafter\gobble\fi}\def\IsInteger#1{\TT\ifcat_\ifnum9<1\gobbleminus#1 _\else A\fi\}
\end{verbatim}
but this surely stretches the technique further than is reasonable.

If we don’t care about the sign, we can use \TeX to remove the entire number (sign and all) from the input stream, and then look at what’s left:
\begin{verbatim}
\def\testnum#1{\afterassignment\testresult\count255=#1 \end}
\def\testresult#1\end{verbatim}
(whch technique is due to David Kastrup). In a later thread on the same topic, Michael Downes offered:
which relies on \romannumeral producing an empty result if its argument is zero.

All the complete functions above are designed to be used in \TeX conditions written “naturally” — for example:
\begin{verbatim}
\if\IsInteger{<subject of test>}%
<deal with integer>%
\else
<deal with non-integer>%
\fi
\end{verbatim}

196 Defining macros within macros

The way to think of this is that ## gets replaced by # in just the same way that #1 gets replaced by ‘whatever is the first argument’.

So if you define a macro and use it as:
\begin{verbatim}
\def\a#1{+++#1+++#1+++#1+++
\a{b}
\end{verbatim}

the macro expansion produces ‘+++/b+++b+++b+++’, which people find normal. However, if we now replace part of the macro:
\begin{verbatim}
\def\a#1{+++#1+++
\def\x #1{xxx#1}\a{b}
\end{verbatim}

\a{b} will expand to ‘+++b++++\def\x b{xxxb}’. This defines \x to be a macro delimited by b, and taking no arguments, which people may find strange, even though it is just a specialisation of the example above. If you want \a to define \x to be a macro with one argument, you need to write:
\begin{verbatim}
\def\a#1{+++#1+++
\def\x #1{xxx#1}\a{b}
\end{verbatim}

and \a{b} will expand to ‘+++b++++\def\x #1{xxx#1}’, because #1 gets replaced by ‘b’ and ## gets replaced by #.

To nest a definition inside a definition inside a definition then you need ####1, as at each stage ## is replaced by #. At the next level you need 8 #s each time, and so on.

197 Spaces in macros

It’s very easy to write macros that produce space in the typeset output where it’s neither desired nor expected. Spaces introduced by macros are particularly insidious because they don’t amalgamate with spaces around the macro (in the way that consecutive spaces that you type do), so your output can have a single bloated space that proves to be made up of two or even more spaces that haven’t amalgamated. And of course, your output can also have a space where none was wanted at all.

Spaces are produced, inside a macro as elsewhere, by space or tab characters, or by end-of-line characters. There are two basic rules to remember when writing a macro: first, the rules for ignoring spaces when you’re typing macros are just the same as the rules that apply when you’re typing ordinary text, and second, rules for ignoring spaces do not apply to spaces produced while a macro is being obeyed (“expanded”).

Spaces are ignored in vertical mode (between paragraphs), at the beginning of a line, and after a command name. Since sequences of spaces are collapsed into one, it ‘feels as if’ spaces are ignored if they follow another space. Space can have syntactic meaning after certain sorts of non-braced arguments (e.g., count and dimen variable assignments in Plain \TeX) and after certain control words (e.g., in \hbox to, so again we have instances where it ‘feels as if’ spaces are being ignored when they’re merely working quietly for their living.

Consider the following macro, fairly faithfully adapted from one that appeared on comp.text.tex:

\begin{verbatim}
\newcommand\stline[1]{ \bigskip \makebox[2cm]{ \textbf{#1} } }
\end{verbatim}

The macro definition contains five spaces:
• after the opening \{ of the macro body; this space will be ignored, not because “because the macro appears at the start of a line”, but rather because the macro was designed to operate between paragraphs
• after \bigskip; this space will be ignored (while the macro is being defined) because it follows a command name
• after the \{ of the mandatory argument of \makebox; even though this space will inevitably appear at the start of an output line, it will not be ignored
• after the \} closing the argument of \textbf; this space will not be ignored, but may be overlooked if the argument is well within the 2cm allowed for it
• after the \} closing the mandatory argument of \makebox; this space will not be ignored, and will appear in the argument

The original author of the macro had been concerned that the starts of his lines with this macro in them were not at the left margin, and that the text appearing after the macro wasn’t always properly aligned. These problems arose from the space at the start of the mandatory argument of \makebox and the space immediately after the same argument. He had written his macro in that way to emphasise the meaning of its various parts; unfortunately the meaning was rather lost in the problems the macro caused.

The principal technique for suppressing spaces is the use of % characters: everything after a % is ignored, even the end of line itself (so that not even the end of line can contribute an unwanted space). The secondary technique is to ensure that the end of line is preceded by a command name (since the end of line behaves like a space, it will be ignored following a command name). Thus the above command would be written (by an experienced programmer with a similar eye to emphasising the structure):
\newcommand{\stline}[1]{% 
\bigskip 
\makebox[2cm]{% 
\textbf{#1}\relax 
}% 
}%

Care has been taken to ensure that every space in the revised definition is ignored, so none appears in the output. The revised definition takes the “belt and braces” approach, explicitly dealing with every line ending (although, as noted above, a space introduced at the end of the first line of the macro would have been ignored in actual use of the macro. This is the best technique, in fact — it’s easier to blindly suppress spaces than to analyse at every point whether you actually need to. Three techniques were used to suppress spaces:
• placing a % character at the end of a line (as in the 1st, 3rd and 5th lines),
• ending a line ‘naturally’ with a control sequence, as in line 2, and
• ending a line with an ‘artificial’ control sequence, as in line 4; the control sequence in this case (\relax) is a no-op in many circumstances (as here), but this usage is deprecated — a % character would have been better.

Beware of the (common) temptation to place a space before a % character: if you do this you might as well omit the % altogether.

In “real life”, of course, the spaces that appear in macros are far more cryptic than those in the example above. The most common spaces arise from unprotected line ends, and this is an error that occasionally appears even in macros written by the most accomplished programmers.

198 How to break the 9-argument limit

If you think about it, you will realise that Knuth’s command definition syntax:
\def\blah#1#2 \ldots #9{<macro body>}

is intrinsically limited to just 9 arguments. There’s no direct way round this: how would you express a 10th argument? — and ensure that the syntax didn’t gobble some other valid usage?

If you really must have more than 9 arguments, the way to go is:
\def\blah#1#2 \ldots #9{% 
\def\ArgI{{#1}}% 
\def\ArgII{{#2}}% 
\ldots 

\def\ArgIX{{#9}}%
\BlahRelay
}
\def\BlahRelay#1#2#3{%  
% arguments 1-9 are now in
% \ArgI-\ArgIX
% arguments 10-12 are in
% #1-#3
<macro body>}%

This technique is easily extendible by concert pianists of the \TeX keyboard, but is really hard to recommend.

\LaTeX users have the small convenience of merely giving a number of arguments in the "newcommand" that defines each part of the relaying mechanism: Knuth’s restriction applies to \newcommand just as it does to \def. However, \LaTeX users also have the way out of such barbarous command syntax: the keyval package. With keyval, and a bit of programming, one can write really quite sophisticated commands, whose invocation might look like:

\flowerinstance{species=Primula veris, 
  family=Primulaceae, 
  location=Coldham’s Common, 
  locationtype=Common grazing land, 
  date=1995/04/24, 
  numplants=50, 
  soiltype=alkaline
}

The merit of such verbosity is that it is self-explanatory: the typist doesn’t have to remember that argument twelve is soiltype, and so on: the commands may be copied from field notes quickly and accurately.

keyval.sty: Distributed as part of macros/latex/required/graphics

199 Defining characters as macros

Both Plain \TeX and \LaTeX define a character as a macro: the character ~ expands to a “non-breakable space”. Since Knuth wrote the first such macro ever, no-one expects anything other than that ~ will do such a thing; the slightly surprising thing is how difficult it is to get such things right for oneself.

A character’s category code (catcode) must be set to “active” before you can define a meaning of the character as a command. There is no reason in principle that one shouldn’t set any character ‘active’, and apply a definition to it as a macro. The problem is that by so doing, you preclude the character’s use for other purposes, and there are few characters “free” to be subverted in this way.

Some packages facilitate such definitions — for example, the shortvrb package in its \MakeShortVerb command. The doc package, which processes .dtx files (see question 39) uses shortvrb to define \|...\| as a shorthand for \verb|...|. But \| is also used in the preambles of tabular environments, so that tables in .dtx files can only have vertical line separation between columns by employing special measures of some sort — a typical example of the sort of problem that unconstrained use of active characters can produce.

To define the character \z as a command, one would therefore say something like:

\catcode’z=\active
\def z{Yawn, I’m tired}%

and each subsequent \z in the text would become a yawn. This would be an astoundingly bad idea, for any but the most specialised document. (Note that, in \def z, z is no longer a letter; the space is therefore not necessary — \defz would do; we choose to retain the space, for what little clarity we can manage.)

However, the definition of the command persists even if the character’s catcode reverts to its original value; the definition becomes accessible again if the character once again becomes active. This fact is the basis of most “safe” uses of character definitions.
To define a character as a command “safely” (i.e., storing it for later use), we need to isolate the definition in a group. The simple way of doing this is:

```latex
\% 
\catcode`\z=\active 
\gdef z{Yawn, I'm tired}\%
```

Use of `\gdef` (global definition) ensures that the definition persists after the group is closed, even though the catcode reverts.

The alternative (“tricksy”) way of creating such an isolated definition depends on the curious properties of `\lowercase`, which changes characters without altering their catcodes. Since there is always one active character (~), we can fool `\lowercase` into patching up a definition without ever explicitly changing a catcode:

```latex
\begingroup 
\lccode`\~=`\z 
\lowercase{\endgroup 
\def~{Yawn, I'm tired}\%}
```

The two definitions have the same overall effect (the character is defined as a command, but the character does not remain active), except that the first defines a `\global` command.

200 Active characters in command arguments

Occasionally, it’s nice to make one or two characters active in the argument of a command, to make it easier for authors to code the arguments.

Active characters can be used safely in such situations; but care is needed.

An example arose while this answer was being considered: an aspirant macro writer posted to comp.text.tex asking for help to make # and b produce musical sharp and flat signs, respectively, in a macro for specifying chords.

The first problem is that both # and b have rather important uses elsewhere in \TeX (to say the least!), so that the characters can only be made active while the command is executing.

Using the techniques discussed in question 199, we can define:

```latex
\begingroup 
\catcode`\#=\active 
\gdef#{$\sharp$}\endgroup
```

and:

```latex
\begingroup 
\lccode`\~=`\b 
\lowercase{\endgroup 
\def~{$\flat$}}
```

The second problem is one of timing: the command has to make each character active before its arguments are read: this means that the command can’t actually “have” arguments itself, but must be split in two. So we write:

```latex
\def\chord{% 
\begingroup 
\catcode`\#=\active 
\catcode`\b=\active 
\Xchord 
\} 
\def\Xchord#1{% 
\chordfont#1\% 
\endgroup 
}
```

and we can use the command as `\chord{F#}` or `\chord{Bb minor}`.

Two features of the coding are important:

- `\begingroup` in `\chord` opens a group that is closed by `\endgroup` in `\Xchord`; this group limits the change of category codes, which is the raison
Although \# is active while \%chord is executed, it’s not active when it’s being defined, so that the use of \#1 doesn’t require any special attention.

Note that the technique used in such macros as \chord, here, is analogous to that used in such commands as \verb; and, in just the same way as \verb (see question 226), \chord won’t work inside the argument of another command (the error messages, if they appear at all, will probably be rather odd).

### 201 Defining a macro from an argument

It’s common to want a command to create another command: often one wants the new command’s name to derive from an argument. \LaTeX{} does this all the time: for example, \newenvironment creates start- and end-environment commands whose names are derived from the name of the environment command.

The (seemingly) obvious approach:

\begin{verbatim}
\def\relay#1#2{\def#1{#2}}
\end{verbatim}

doesn’t work (the \TeX{} engine interprets it as a rather strange redefinition of \#). The trick is to use \csname, which is a \TeX{} primitive for generating command names from random text, together with \expandafter. The definition above should read:

\begin{verbatim}
\def\relay#1#2{\expandafter\def\csname #1\endcsname{#2}}
\end{verbatim}

With this definition, \relay{blah}{bleah} is equivalent to \def{blah}{bleah}.

Note that the definition of \relay omits the braces round the ‘command name’ in the \newcommand it executes. This is because they’re not necessary (in fact they seldom are), and in this circumstance they make the macro code slightly more tedious.

The name created need not (of course) be just the argument:

\begin{verbatim}
\def\newrace#1#2#3{% \expandafter\def\csname start#1\endcsname{% #2} \expandafter\def\csname finish#1\endcsname{% #3} }
\end{verbatim}

With commands

\begin{verbatim}
\def\start#1{\csname start#1\endcsname}
\def\finish#1{\csname finish#1\endcsname}
\end{verbatim}

these ‘races’ could behave a bit like \LaTeX{} environments.

### 202 Transcribing \LaTeX{} command definitions

At several places in this FAQ, questions are answered in terms of how to program a \LaTeX{} macro. Sometimes, these macros might also help users of Plain \TeX{} or other packages; this answer attempts to provide a rough-and-ready guide to transcribing such macro definitions for use in other packages.

The reason \LaTeX{} has commands that replace \def, is that there’s a general philosophy within \LaTeX{} that the user should be protected from himself: the user has different commands according to whether the command to be defined exists (\renewcommand) or not (\newcommand), and if its status proves not as the user expected, an error is reported. A third definition command, \providecommand, only defines if the target is not already defined; \LaTeX{} has no direct equivalent of \def, which ignores the present state of the command. The final command of this sort is \DeclareRobustCommand, which creates a command which is “robust” (i.e., will not expand if subjected to \LaTeX{} “protected expansion”); from the Plain \TeX{} user’s point of view, \DeclareRobustCommand should be treated as a non-checking version of \newcommand.

\LaTeX{} commands are, by default, defined \long; an optional * between the \newcommand and its (other) arguments specifies that the command is not to be defined \long. The * is detected by a command \@ifstar which uses \futurelet to switch between two branches, and gobbles the *: \LaTeX{} users are encouraged to think of the * as part of the command name.
LaTeX’s checks for unknown command are done by `\ifx` comparison of a `\csname` construction with `\relax`; since the command name argument is the desired control sequence name, this proves a little long-winded. Since `#1` is the requisite argument, we have:

\begin{verbatim}
\expandafter\ifx\csname\expandafter\@gobble\string#1\endcsname\relax...
\end{verbatim}

(`\@gobble` simply throws away its argument).

The arguments of a LaTeX command are specified by two optional arguments to the defining command: a count of arguments (0–9: if the count is 0, the optional count argument may be omitted), and a default value for the first argument, if the defined command’s first command is to be optional. So:

\begin{verbatim}
\newcommand\foo{...}
\newcommand\foo[0]{...}
\newcommand\foo[1]{...#1...}
\newcommand\foo[2][boo]{...#1...#2...}
\end{verbatim}

In the last case, `\foo` may be called as `\foo{goodbye}`, which is equivalent to `\foo[boo]{goodbye}` (employing the default value given for the first argument), or as `\foo[hello]{goodbye}` (with an explicit first argument).

Coding of commands with optional arguments is exemplified by the coding of the last `\foo` above:

\begin{verbatim}
\def\foo{\futurelet\next\@r@foo}
\def\@r@foo{\ifx\next[\let\next\@x@foo\else\def\next{\@x@foo[boo]}\fi\next}
\def\@x@foo[#1][#2]{...#1...#2...}
\end{verbatim}

S.2 LaTeX macros

203 How to change LaTeX’s “fixed names”

LaTeX document classes define several typographic operations that need ‘canned text’ (text not supplied by the user). In the earliest days of LaTeX 2.09 these bits of text were built into the body of LaTeX’s macros and were rather difficult to change, but “fixed name” macros were introduced for the benefit of those wishing to use LaTeX in languages other than English. For example, the special section produced by the `\tableofcontents` command is always called `\contentsname` (or rather, what `\contentsname` is defined to mean). Changing the canned text is now one of the easiest customisations a user can do to LaTeX.

The canned text macros are all of the form `\langle thing\rangle name`, and changing them is simplicity itself. Put:

\begin{verbatim}
\renewcommand{\langle thing\rangle name}{Res minor}
\end{verbatim}

in the preamble of your document, and the job is done. (However, beware of the babel package, which requires you to use a different mechanism: be sure to check question 204 if you’re using it.)

The names that are defined in the standard LaTeX classes (and the `makeidx` package) are listed below. Some of the names are only defined in a subset of the classes (and the `letter` class has a set of names all of its own); the list shows the specialisation of each name, where appropriate.

\begin{verbatim}
\abstractname Abstract
\alsoname see also (makeidx package)
\appendixname Appendix
\bibname Bibliography (report,book)
\ccname cc (letter)
\chaptername Chapter (report,book)
\contentsname Contents
\end{verbatim}
204 Changing the words babel uses

\LaTeX uses symbolic names for many of the automatically-generated text it produces (special-purpose section headers, captions, etc.). As noted in question 203 (which includes a list of the names themselves), this enables the user to change the names used by the standard classes, which is particularly useful if the document is being prepared in some language other than \LaTeX’s default English. So, for example, a Danish author may wish that her table of contents was called “Indholdsfortegnelse”, and so would expect to place a command

\begin{verbatim}
\renewcommand{\contentsname}{Indholdsfortegnelse}
\end{verbatim}

in the preamble of her document.

However, it’s natural for a user of a non-English language to use babel, because it offers many conveniences and typesetting niceties for those preparing documents in those languages. In particular, when babel is selecting a new language, it ensures that \LaTeX’s symbolic names are translated appropriately for the language in question. Unfortunately, babel’s choice of names isn’t always to everyone’s choice, and there is still a need for a mechanism to replace the ‘standard’ names.

Whenever a new language is selected, babel resets all the names to the settings for that language. In particular, babel selects the document’s main language when \{document\} is executed, which immediately destroys any changes to these symbolic names made in the prologue of a document that uses babel.

Therefore, babel defines a command to enable users to change the definitions of the symbolic names, on a per-language basis: \addto\captions\{language\} is the thing (\{language\} being the language option you gave to babel in the first place). For example:

\begin{verbatim}
\addto\captionsdanish{%
\renewcommand{\contentsname}{Indholdsfortegnelse}%
}
\end{verbatim}

205 Running equation, figure and table numbering

Many \LaTeX classes (including the standard book class) number things per chapter; so figures in chapter 1 are numbered 1.1, 1.2, and so on. Sometimes this is not appropriate for the user’s needs.

Short of rewriting the whole class, one may use one of the removefr and remreset packages; both define a \removefromreset{figure} command, and having included the package one writes something like:

\begin{verbatim}
\makeatletter
\removefromreset{figure}{chapter}
\makeatother
\end{verbatim}

and the automatic renumbering stops. You then need to redefine the way in which the figure number (in this case) is printed:

\begin{verbatim}
\renewcommand{\thefigure}{$\arabic{\@figure}$}
\makeatother
\end{verbatim}

(remember to do the whole job, for every counter you want to manipulate, within \makeatletter ... \makeatother).

The technique may also be used to change where in a multilevel structure a counter is reset. Suppose your class numbers figures as \{chapter\}.\{section\}.\{figure\}, and you want figures numbered per chapter, try:
@removefromreset{figure}{section}
@addtoreset{figure}{chapter}
\renewcommand{\thefigure}{\thechapter.\@arabic\c@figure}
(the command @addtoreset is a part of \LaTeX itself).

The chngcntr package provides a simple means to access the two sorts of changes discussed, defining \counterwithin and \counterwithout commands.

chngcntr.sty: macros/latex/contrib/supported/misc/chngcntr.sty
removefr.tex: macros/latex/contrib/other/fragments/removefr.tex
(note, this is constructed as a “fragment” for use within other packages: load by \input{removefr})

remreset.sty: Distributed as part of macros/latex/contrib/supported/carlisle

206 \@ and @ in macro names

Macro names containing @ are internal to \LaTeX, and without special treatment just don’t work in ordinary use. An exemplar of the problems caused is discussed in question 245.

The problems users see are caused by copying bits of a class (.cls file) or package (.sty file) into a document, or by including a class or package file into a \LaTeX document by some means other than \documentclass or \usepackage. \LaTeX defines internal commands whose names contain the character @ to avoid clashes between its internal names and names that we would normally use in our documents. In order that these commands may work at all, \documentclass and \usepackage play around with the meaning of @.

If you’ve included a file wrongly, you solve the problem by using the correct command.

If you’re using a fragment of a package or class, you may well feel confused: books such as The \LaTeX Companion (see question 22) are full of fragments of packages as examples for you to employ.

For example, there’s a lengthy section in The Companion about \@startsection and how to use it to control the appearance of section titles. Page 15 discusses the problem; and suggests that you make such changes in the document preamble, between \makeatletter and \makeatother. So the redefinition of \subsection on page 26 could be:

\makeatletter
\renewcommand{\subsection}{\@startsection
{subsection}{% % name
{\normalfont\normalsize\itshape}{% style
\makeatother

The alternative is to treat all these fragments as a package proper, bundling them up into a .sty file and including them with \usepackage. (This approach is marginally preferable, from the \LaTeX purist’s point of view.)

207 What’s the reason for ‘protection’?

Sometimes \LaTeX saves data it will reread later. These data are often the argument of some command; they are the so-called moving arguments. (‘Moving’ because data are moved around.) Places to look for are all arguments that may go into table of contents, list of figures, etc.; namely, data that are written to an auxiliary file and read in later. Other places are those data that might appear in head- or footlines. Section headers and figure captions are the most prominent examples; there’s a complete list in Lamport’s book (see question 22).

What’s going on really, behind the scenes? The commands in the moving arguments are already expanded to their internal structure during the process of saving. Sometimes this expansion results in invalid \LaTeX code when processed again. “\protect\cmd” tells \LaTeX to save \cmd as \cmd, without expansion.

What is a ‘fragile command’? It’s a command that expands into illegal \LaTeX code during the save process.

What is a ‘robust command’? It’s a command that expands into legal \LaTeX code during the save process.
No-one (of course) likes this situation; the \LaTeX{} team have removed the need for protection of some things in the production of \LaTeX{}\textsuperscript{2\varepsilon}, but the techniques available to them within current \LaTeX{} mean that this is an expensive exercise. It remains a long-term aim of the team to remove all need for these things.

208 \texttt{\edef} does not work with \texttt{\protect}

Robust \LaTeX{} commands are either “naturally robust” — meaning that they never need \texttt{\protect}, or “self-protected” — meaning that they have \texttt{\protect} built in to their definition in some way. Self-protected commands are robust only in a context where the \texttt{\protect} mechanism is properly handled. The body of an \texttt{\edef} definition doesn’t handle \texttt{\protect} properly, since \texttt{\edef} is a \TeX{} primitive rather than a \LaTeX{} command.

This problem is resolved by a \LaTeX{} internal command \texttt{\protected@edef}, which does the job of \texttt{\edef} while keeping the \texttt{\protect} mechanism working. There’s a corresponding \texttt{\protected@xdef} which does the job of \texttt{\xdef}.

Of course, these commands need to be tended carefully, since they’re internal: see question 206.

209 Optional arguments like \texttt{\section}

Optional arguments, in macros defined using \texttt{\newcommand}, don’t quite work like the optional argument to \texttt{\section}. The default value of \texttt{\section}’s optional argument is the value of the mandatory argument, but \texttt{\newcommand} requires that you ‘know’ the value of the default beforehand.

The requisite trick is to use a macro in the optional argument:
\begin{verbatim}
\newcommand\thing[2][\DefaultOpt]{\def\DefaultOpt{#2} ...}
\end{verbatim}

210 Making labels from a counter

Suppose we have a \LaTeX{} counter, which we’ve defined with \texttt{\newcounter{foo}}. We can increment the value of the counter by \texttt{\addtocounter{foo}{1}}, but that’s pretty clunky for an operation that happens so often… so there’s a command \texttt{\stepcounter{foo}} that does this special case of increasing-by-one.

There’s an internal \LaTeX{} variable, the “current label”, that remembers the last ‘labellable’ thing that \LaTeX{} has processed. You could (if you were to insist) set that value by the relevant \TeX{} command (having taken the necessary precautions to ensure that the internal command worked) — but it’s not necessary. If, instead of either of the stepping methods above, you say \texttt{\refstepcounter{foo}}, the internal variable is set to the new value, and (until something else comes along), \texttt{\label} will refer to the counter.

211 Finding if you’re on an odd or an even page

Question 232 discusses the issue of getting \texttt{\marginpar} commands to put their output in the correct margin of two-sided documents. This is an example of the general problem of knowing where a particular bit of text lies: the output routine is asynchronous, and (\LaTeX{}) will usually process quite a bit of the “next” page before deciding to output any page. As a result, the page counter (known internally in \LaTeX{} as \texttt{\c@page}) is normally only reliable when you’re actually in the output routine.

The solution is to use some version of the \texttt{\label} mechanism to determine which side of the page you’re on; the value of the page counter that appears in a \texttt{\pageref} command has been inserted in the course of the output routine, and is therefore safe.

However, \texttt{\pageref} itself isn’t reliable: one might hope that
\begin{verbatim}
\ifthenelse{\isodd{\pageref{foo}}}{odd}{even}
\end{verbatim}

would do the necessary, but both the \texttt{babel} and \texttt{hyperref} packages have been known to interfere with the output of \texttt{\pageref}; be careful!

The \texttt{chngpage} package needs to provide this functionality for its own use, and therefore provides a command \texttt{\checkoddpage}; this sets a private-use label, and the page reference part of that label is then examined (in a \texttt{hyperref}-safe way) to set a conditional \texttt{ifcpoddpage} true if the command was issued on an odd page. Of course, the \texttt{\label} contributes to \LaTeX{}’s “Rerun to get cross-references right” error messages...

\texttt{chngpage.sty}: macros/latex/contrib/supported/misc/chngpage.sty
212 How to change the format of labels

By default, when a label is created, it takes on the appearance of the counter labelled: specifically, it is set to \the<counter> — what would be used if you asked to typeset the counter in your text. This isn’t always what you need: for example, if you have nested enumerated lists with the outer numbered and the inner labelled with letters, one might expect to want to refer to items in the inner list as “2(c)”. (Remember, you can change the structure of list items — see question 157.) The change is of course possible by explicit labelling of the parent and using that label to construct the typeset result — something like

\ref{parent-item}({\ref{child-item}})

which would be both tedious and error-prone. What’s more, it would be undesirable, since you would be constructing a visual representation which is inflexible (you couldn’t change all the references to elements of a list at one fell swoop).

\LaTeX{} in fact has a label-formatting command built into every label definition; by default it’s null, but it’s available for the user to program. For any label ⟨counter⟩ there’s a \LaTeX{} internal command \p@<counter>; for example, a label definition on an inner list item is done (in effect) using the command \p@enumii{\theenumii}. So to change the labels on all inner lists, put the following patch in your preamble:

\makeatletter
\renewcommand{\p@enumii}[1]{\theenumi(#1)}
\makeatother

The analogous change works for any counter that gets used in a \label command.

213 A command with two optional arguments

If you’ve already read “breaking the 9-argument limit” (question 198) you can probably guess the solution to this problem: command relaying.

\LaTeX{} allows commands with a single optional argument thus:

\newcommand{\blah}[1][Default]{...}

You may legally call such a command either with its optional argument present, as \blah[nonDefault] or as \blah; in the latter case, the code of \blah will have an argument of Default.

To define a command with two optional arguments, we use the relaying technique, as follows:

\newcommand{\blah}[1][Default1]{% 
  \def\ArgI{(#1)}% 
  \BlahRelay
} \newcommand{\BlahRelay}[1][Default2]{% 
  % the first optional argument is now in \ArgI
  % the second is in #1
  ...
}

Of course, \BlahRelay may have as many mandatory arguments as are allowed, after allowance for the one taken up with its own optional argument — that is, 8.

A command with two optional arguments strains the limit of what’s sensible: obviously you can extend the technique to provide as many optional arguments as your fevered imagination can summon. However, see the comments on the use of the keyval package in “breaking the 9-argument limit” (question 198), which offer an alternative way forward.

214 Adjusting the presentation of section numbers

The general issues of adjusting the appearance of section headings are pretty complex, and are covered in question 119.

However, people regularly want merely to change the way the section number appears in the heading, and some such people don’t mind writing out a few macros. This answer is for them.

The way the section number is typeset is determined by the \@seccntformat command, which is given the “name” (section, subsection, ...) of the heading, as argument. Ordinarily, it merely outputs the section number, and then a quad of space. Suppose
you want to put a stop after every section (subsection, subsubsection, ...) number, a
trivial change may be implemented by simple modification of the command:
\renewcommand*{%
\csname the\#1\endcsname \quad
\}

Most people (for some reason) just want a stop after a section number. To do this,
one must make \@seccntformat switch according to its argument. The following
technique for doing the job is slightly wasteful, but is efficient enough:
\let\@@seccntformat\@seccntformat
\renewcommand*{%
\expandafter\let\@tempa\expandafter
\csname @seccntformat@#1\endcsname
\ifx\@tempa\relax
\expandafter\@@seccntformat
\else
\expandafter\@tempa
\fi
\{#1\%
\}

which looks to see if a second-level command has been defined, and uses it if so;
otherwise it uses the original. The second-level command to define stops after section
numbers (only) has the same definition as the original "all levels alike" version:
\newcommand*{%
\csname the\#1\endcsname \quad
\}

Note that all the command definitions of this answer are dealing in \LaTeX
\LaTeX\ internal commands (see question 206), so the above code should be in a package file, for
preference.

\section{Things are Going Wrong...}

\subsection{Getting things to fit}

\section{Enlarging \TeX}

The \TeX error message 'capacity exceeded' covers a multitude of problems. What has
been exhausted is listed in brackets after the error message itself, as in:
\inserterror
! \TeX\ capacity exceeded, sorry
... [main memory size=263001].

Most of the time this error can be fixed \emph{without} enlarging \TeX. The most common
causes are unmatched braces, extra-long lines, and poorly-written macros. Extra-long
lines are often introduced when files are transferred incorrectly between operating sys-
tems, and line-endings are not preserved properly (the tell-tale sign of an extra-long
line error is the complaint that the 'buf_size' has overflowed).

If you really need to extend your \TeX's capacity, the proper method depends on
your installation. There is no need (with modern \TeX implementations) to change the
defaults in Knuth's \TeX source; but if you do need to do so, use a change file to modify
the values set in module 11, recompile your \TeX and regenerate all format files.

Modern implementations allow the sizes of the various bits of \TeX's memory to be
changed semi-dynamically. Some (such as \emTeX) allow the memory parameters to be
changed in command-line switches when \TeX is started; most frequently, a configura-
tion file is read which specifies the size of the memory. On \web\text{-}based systems, this
file is called \texttt{texmf.cnf}: see the documentation that comes with the distribution for
other implementations. Almost invariably, after such a change, the format files need to
be regenerated after changing the memory parameters.

\section{Why can't I load \pictex?}

\pictex is a resource hog; fortunately, most modern \TeX implementations offer gener-
ous amounts of space, and most modern computers are pretty fast, so users aren't too
badly affected by its performance.
However, \TeX{} has the further unfortunate tendency to fill up \TeX{}’s fixed-size arrays — notably the array of 256 ‘dimension’ registers. This is a particular problem when you’re using \texttt{pictex.sty} with \TeX{} and some other packages that also need dimension registers. When this happens, you will see the \TeX{} error message:

\begin{verbatim}
! No room for a new \texttt{\dimen}.
\end{verbatim}

There is nothing that can directly be done about this error: you can’t extend the number of available \texttt{\dimen} registers without extending \TeX{} itself. (e-\TeX{} and \TeX{} — see questions 252 and 251 respectively — both do this, as does MicroPress Inc’s \VT{}X — see question 55.) Since you can’t (ordinarily) extend \TeX{}, you need to change \TeX{} itself; unfortunately \TeX{}’s author is no longer active in the \TeX{} world, so one must resort to patching. There are two solutions available.

The \texttt{Con\TeX} module \texttt{m-pictex.tex} (for Plain \TeX{} and variants) or the corresponding \texttt{\LaTeX} \texttt{m-pictex} package provide an ingenious solution to the problem based on hacking the code of \texttt{\newdimen} itself.

Alternatively, Andreas Schell’s \texttt{pictexwd} and related packages replace \TeX{} with a version that uses 33 fewer \texttt{\dimen} registers; so use \texttt{pictexwd} in place of \texttt{pictex} (either as a \LaTeX{} package, or as a file to read into Plain \TeX{}).

And how does one use \TeX{} anyway, given that the manual is so hard to come by (see question 32)? Fortunately for MS-DOS and Windows users, the \texttt{MathsPic} system may be used to translate a somewhat different language into \TeX{} commands; and the \texttt{MathsPic} manual is free (and part of the distribution). \texttt{MathsPic} is written in \texttt{Basic}; a version written in \texttt{Perl} is under development, and should be available soon.

\texttt{m-pictex.sty}: Distributed as part of \texttt{macros/context/cont-tmf.zip}

\texttt{m-pictex.tex}: Distributed as part of \texttt{macros/context/cont-tmf.zip}

\texttt{MathsPic}: \texttt{graphics/pictex/mathspic}

\texttt{pictexwd.sty}: Distributed as part of \texttt{graphics/pictex/addon}

\section{Making things stay where you want them}

\subsection{Moving tables and figures in \LaTeX{}X}

Tables and figures have a tendency to surprise, by floating away from where they were specified to appear. This is in fact perfectly ordinary document design; any professional typesetting package will float figures and tables to where they’ll fit without violating the certain typographic rules. Even if you use the placement specifier \texttt{h} for ‘here’, the figure or table will not be printed ‘here’ if doing so would break the rules; the rules themselves are pretty simple, and are given on page 198, section C.9 of the \LaTeX{} manual. In the worst case, \LaTeX{}’s rules can cause the floating items to pile up to the extent that you get an error message saying “Too many unprocessed floats” (see question 244).

What follows is a simple checklist of things to do to solve these problems (the checklist talks throughout about figures, but applies equally well to tables, or to “non-standard” floats defined by the \texttt{float} or other packages).

\begin{itemize}
  \item Do your figures need to float at all? If not, consider the \texttt{[H]} placement option offered by the \texttt{float} package: figures with this placement are made up to look as if they’re floating, but they don’t in fact float. Beware outstanding floats, though: the \texttt{\caption} commands are numbered in the order they appear in the document, and a \texttt{[H]} float can ‘overtake’ a float that hasn’t yet been placed, so that figures numbers get out of order.
  \item Are the placement parameters on your figures right? The default (\texttt{tbp}) is reasonable, but you can reasonably change it (for example, to add an \texttt{h}). Whatever you do, \texttt{don’t} omit the \texttt{p}: doing so could cause \TeX{} to believe that if you can’t have your figure \texttt{here}, you don’t want it \texttt{anywhere}. (\TeX{} does try hard to avoid being confused in this way…)
  \item \TeX{}’s own float placement parameters could be preventing placements that seem entirely “reasonable” to you — they’re notoriously rather conservative. To encourage \TeX{} not to move your figure, you need to loosen its demands. (The most important ones are the ratio of text to float on a given page, but it’s sensible to have a fixed set that changes the whole lot, to meet every eventuality.)
\end{itemize}

\texttt{\renewcommand{\topfraction}{.85}}
\texttt{\renewcommand{\bottomfraction}{.7}}
The meanings of these parameters are described on pages 199–200, section C.9 of the \LaTeX{} manual.

- Are there places in your document where you could ‘naturally’ put a \texttt{\clearpage} command? If so, do: the backlog of floats is cleared after a \texttt{\clearpage}. (Note that the \texttt{\chapter} command in the standard \texttt{book} and \texttt{report} classes implicitly executes \texttt{\clearpage}, so you can’t float past the end of a chapter.)

- Try the \texttt{placeins} package: it defines a \texttt{\FloatBarrier} command beyond which floats may not pass. A package option allows you to declare that floats may not pass a \texttt{\section} command, but you can place \texttt{\FloatBarriers} wherever you choose.

- If you are bothered by floats appearing at the top of the page (before they are specified in your text), try the \texttt{flafter} package, which avoids this problem by insisting that floats should always appear after their definition.

- Have a look at the \LaTeX{}2ε \texttt{afterpage} package. Its documentation gives as an example the idea of putting \texttt{\clearpage} \texttt{after} the current page (where it will clear the backlog, but not cause an ugly gap in your text), but also admits that the package is somewhat fragile. Use it as a last resort if the other two possibilities below don’t help.

- If you would actually like great blocks of floats at the end of each of your chapters, try the \texttt{morefloats} package; this ‘simply’ increases the number of floating inserts that \LaTeX{} can handle at one time (from 18 to 36).

- If you actually wanted all your figures to float to the end (e.g., for submitting a draft copy of a paper), don’t rely on \LaTeX{}’s mechanism: get the \texttt{endfloat} package to do the job for you.

\texttt{afterpage.sty}: Distributed as part of \texttt{macros/latex/required/tools}
\texttt{endfloat.sty}: \texttt{macros/latex/contrib/supported/endfloat}
\texttt{flafter.sty}: Part of the \LaTeX{} distribution
\texttt{float.sty}: \texttt{macros/latex/contrib/supported/float}
\texttt{morefloats.sty}: \texttt{macros/latex/contrib/other/misc/morefloats.sty}
\texttt{placeins.sty}: \texttt{macros/latex/contrib/other/misc/placeins.sty}

218 Underlined text won’t break

Knuth made no provision for underlining text: he took the view that underlining is not a typesetting operation, but rather one that provides emphasis on typewriters, which typically offer but one face. The corresponding technique in typeset text is to switch from upright to italic text (or vice-versa): the \LaTeX{} command \texttt{\emph} does just that to its argument.

Nevertheless, typographically illiterate people (such as those that specify double-spaced thesis styles — see question 135) continue to require underlining of us, so \LaTeX{} as distributed defines an \texttt{\underline} command that applies the mathematical ‘underbar’ operation to text. This technique is not entirely satisfactory, however: the text gets stuck into a box, and won’t break at line end.

Two packages are available that solve this problem. The \texttt{ulem} package redefines the \texttt{\emph} command to underline its argument; the underlined text thus produced behaves as ordinary emphasised text, and will break over the end of a line. (The package is capable of other peculiar effects, too: read its documentation, contained within the file itself.) The \texttt{soul} package defines an \texttt{\ul} command (after which the package is, in part, named) that underlines running text.

\texttt{ulem.sty}: \texttt{macros/latex/contrib/other/misc/ulem.sty}
\texttt{soul.sty}: \texttt{macros/latex/contrib/supported/soul}
Controlling widows and orphans

Widows (the last line of a paragraph at the start of a page) and orphans (the first line of paragraph at the end of a page) interrupt the reader’s flow, and are generally considered “bad form”; (\LaTeX) takes some precautions to avoid them, but completely automatic prevention is often impossible. If you are typesetting your own text, consider whether you can bring yourself to change the wording slightly so that the page break will fall differently.

The page maker, when forming a page, takes account of \widowpenalty and \clubpenalty (which relates to orphans!). These penalties are usually set to the moderate value of 150; this offers mild discouragement of bad breaks. You can increase the values by saying (for example) \widowpenalty=500; however, vertical lists (such as pages are made of) typically have rather little stretchability or shrinkability, so if the page maker has to balance the effect of stretching the unstretchable and being penalised, the penalty will seldom win. This dichotomy can be avoided by allowing the pagemaker to run pages short, by using the \raggedbottom directive; however, many publishers insist on the default \flushbottom; it is seldom acceptable to introduce stretchability into the vertical list, except at points (such as section headings) where the document design explicitly permits it.

Once you’ve exhausted the automatic measures, and have a final draft you want to “polish”, you have to indulge in manual measures. To get rid of an orphan is simple: precede the paragraph with \clearpage and the paragraph can’t start in the wrong place.

Getting rid of a widow can be more tricky. If the paragraph is a long one, it may be possible to set it ‘tight’: say \looseness=-1 immediately after the last word of the paragraph. If that doesn’t work, try adjusting the page size: \enlargethispage \baselineskip may do the job, and get the whole paragraph on one page. Reducing the size of the page by \enlargethispage-\baselineskip may produce a (more-or-less) acceptable “two-line widow”. (Note: \looseness=1, increasing the line length by one, seldom seems to work — the looser paragraph typically has a one-word final line, which doesn’t look much better than the straight widow.)

Things have “gone away”

Old \LaTeX font references such as \tenrm

\LaTeX 2.09 defined a large set of commands for access to the fonts that it had built in to itself. For example, various flavours of \rm could be found as \fivrm, \sixrm, \sevrm, \gtrm, \ninem, \tenrm, \elvrm, \twlrm, \frtnrm, \svtnrm, \twtyrm and \twfvm. These commands were never documented, but certain packages nevertheless used them to achieve effects they needed.

Since the commands weren’t public, they weren’t included in \LaTeX 2\epsilon; to use the unconverted \LaTeX 2.09 packages under \LaTeX 2\epsilon, you need also to include the rawfonts package (which is part of the \LaTeX 2\epsilon distribution).

Missing symbol commands

You’re processing an old document, and some symbol commands such as \Box and \lhd appear no longer to exist. These commands were present in the core of \LaTeX 2.09, but are not in current \LaTeX. They are available in the latexsym package (which is part of the \LaTeX distribution), and in the amsfonts package (which is part of the AMS distribution, and requires AMS symbol fonts).

\amsfonts.sty: fonts/amsfonts/latex

AMS symbol fonts: fonts/amsfonts/sources/symbols

Where are the msa and mab fonts?

The mab and mae fonts were designed by the American Mathematical Society in the very early days of \TeX, for use in typesetting papers for mathematical journals. They were designed using the ‘old’ METAFONT, which wasn’t portable and is no longer available; for a long time they were only available in 300dpi versions which only imperfectly matched modern printers. The AMS has now redesigned the fonts, using the current version of METAFONT, and the new versions are called the msa and mab families.
Nevertheless, \texttt{msx} and \texttt{msy} continue to turn up to plague us. There are, of course, still sites that haven’t got around to upgrading; but, even if everyone upgraded, there would still be the problem of old documents that specify them.

If you have a .\texttt{tex} source that requests \texttt{msx} and \texttt{msy}, the best technique is to edit it so that it requests \texttt{msa} and \texttt{msb} (you only need to change the single letter in the font names).

If you have a DVI file that requests the fonts, there is a package of virtual fonts (see question 37) to map the old to the new series.

\texttt{msa} and \texttt{msb} fonts: \texttt{fonts/amsfonts/sources/symbols}  
\texttt{virtual font set: fonts/vf-files/msx2msa}

\section{Where are the am fonts?}

One still occasionally comes across a request for the \texttt{am} series of fonts. The initials stood for ‘Almost [Computer] Modern’, and they were the predecessors of the Computer Modern fonts that we all know and love (or hate). There’s not a lot one can do with these fonts; they are (as their name implies) almost (but not quite) the same as the \texttt{cm} series; if you’re faced with a document that requests them, all you can reasonably do is to edit the document. The appearance of DVI files that request them is sufficiently rare that no-one has undertaken the mammoth task of creating a translation of them by means of virtual fonts; however, most drivers let you have a configuration file in which you can specify font substitutions. If you specify that every \texttt{am} font should be replaced by its corresponding \texttt{cm} font, the output should be almost correct.

\section{Why does it do that?}

\subsection{Common errors}

224 \LaTeX gets cross-references wrong

Sometimes, however many times you run \LaTeX, the cross-references are just wrong. Remember that the \texttt{\textbackslash label} command must come \textit{after} the \texttt{\textbackslash caption} command, or be part of it. For example,

\begin{verbatim}
\begin{figure}
\begin{figure}
\caption{A Figure}
\label{fig} \label{fig}
\end{figure}
\end{figure}
\end{verbatim}

225 Start of line goes awry

This answer concerns two sorts of problems: errors of the form

\begin{verbatim}
! Missing number, treated as zero.
<to be read again>
\texttt{g}
\end{verbatim}

and those where a single asterisk at the start of a line mysteriously fails to appear in the output.

Both problems arise because \texttt{\textbackslash} takes optional arguments. The command \texttt{\textbackslash*} means “break the line here, and inhibit page break following the line break”; the command \texttt{\textbackslash[\langle\textit{dimen}\rangle]} means “break the line here and add \texttt{\langle\textit{dimen}\rangle} extra vertical space afterwards”.

So why does \texttt{\textbackslash} get confused by these things at the start of a line? It’s looking for the first non-blank thing, and in the test it uses ignores the end of the line in your input text.

The solution is to enclose the stuff at the start of the new line in braces:

\begin{verbatim}
\texttt{\ttfamily}
/* C-language comment*/
{[\texttt{[grump]}]} I don’t like this format
*/
\end{verbatim}

\footnote{The fonts acquired their label ‘Almost’ following the realisation that their first implementation in \texttt{METAFONT79} still wasn’t quite right; Knuth’s original intention had been that they were the final answer}
(The above text derives from an actual post to \texttt{comp.text.tex}; this particular bit of typesetting could plainly also be done using the \texttt{verbatim} environment.)

The problem also appears in maths mode, in arrays and so on. In this case, large-scale bracketing of things is \textit{not} a good idea; the \LaTeX primitive \texttt{\relax} (which does nothing except to block searches of this nature) may be used. From another \texttt{comp.text.tex} example:

\begin{verbatim}
\begin{eqnarray}
  [a] &=& b \\
  \relax[a] &=& b
\end{eqnarray}
\end{verbatim}

\section{Why doesn't \texttt{\verb} work within...?}

The \LaTeX verbatim commands work by changing category codes. Knuth says of this sort of thing “Some care is needed to get the timing right...”, since once the category code has been assigned to a character, it doesn’t change. So \texttt{\verb} has to assume that it is getting the first look at its parameter text; if it isn’t, \LaTeX has already assigned category codes so that \texttt{\verb} doesn’t have a chance. For example:

\begin{verbatim}
\verb+\error+
\end{verbatim}

will work (typesetting ‘\texttt{\error}’), but

\begin{verbatim}
newcommand{\unbrace}[1]{#1}
unbrace{\verb+\error+}
\end{verbatim}

will not (it will attempt to execute \texttt{\error}). Other errors one may encounter are ‘\verb ended by end of line’, or even ‘\verb illegal in command argument’.

This is why the \LaTeX book insists that verbatim commands must not appear in the argument of any other command; they aren’t just fragile, they’re quite unusable in any command parameter, regardless of \texttt{\protect} (see question 207).

The first question to ask yourself is: “is \texttt{\verb} actually necessary?”.

\begin{itemize}
  \item If \texttt{your text} produces the same result as \texttt{\verb+your text+}, then there’s no need of \texttt{\verb} in the first place.
  \item If you’re using \texttt{\verb} to typeset a URL or email address or the like, then the \texttt{url} package will help: it doesn’t suffer from the problems of \texttt{\verb}.
  \item If you’re putting \texttt{\verb} into the argument of a boxing command (such as \texttt{\fbox}), consider using the \texttt{lrbox} environment:

\begin{verbatim}
\newsavebox{\mybox}
\begin{lrbox}{\mybox}
\verb!VerbatimStuff!
\end{lrbox}
\fbox{\usebox{\mybox}}
\end{verbatim}

Otherwise, there are three partial solutions to the problem.

\begin{itemize}
  \item The \texttt{fancyvrb} package defines a command \texttt{\VerbatimFootnotes}, which redefines the \texttt{\footnotetext} (and hence the \texttt{\footnote}) commands in such a way that you can include \texttt{\verb} commands in its argument. This approach could in principle be extended to the arguments of other commands, but it can clash with other packages: for example, \texttt{\VerbatimFootnotes} interacts poorly with the \texttt{paratext} package.
  \item The \texttt{fancyvrb} package defines a command \texttt{\SaveVerb}, with a corresponding \texttt{\UseVerb} command, that allow you to save and then to reuse the content of its argument; for details of this extremely powerful facility, see the package documentation.
  \item Rather simpler is the \texttt{verbdef} package, which defines a (robust) command which expands to the verbatim argument given.
  \item If you have a single character that is giving trouble (in its absence you could simply use \texttt{\texttt{your text}}), consider using \texttt{\string}. \texttt{\texttt{my\string_name}} typesets the same as \texttt{\verb+my_name+}, and will work in the argument of a command. It won’t, however, work in a moving argument, and no amount of \texttt{\protect} (see question 207) will make it work in such a case.
\end{itemize}
The error
! LaTeX Error: There's no line here to end.

See the LaTeX manual or LaTeX Companion for explanation.

\begin{description}
  \item[Very long label] Text...
\end{description}

is actually a rather bad command to use in this case (even if it worked), since it would force the ‘paragraph’ that’s made up of the text of the item to terminate a line which has nothing on it but the label. This would lead to an “Underfull $\hbox$” warning message (usually with ‘infinite’ badness of 10000); while this message doesn’t do any actual harm other than slowing down your \TeX run, any message that doesn’t convey any information distracts for no useful purpose.

The proper solution to the problem is to write a new sort of \texttt{description} environment, that does just what you’re after. (The \texttt{\LaTeX Companion} — see question \ref{question:22} — offers a rather wide selection of variants of these things.)

The quick-and-easy solution, which avoids the warning, is to write:
\begin{description}
  \item[Very long label] \hspace*{\fill} Text...
\end{description}

which fills out the under-full line before forcing its closure. The \texttt{expdlist} package provides the same functionality with its \texttt{\breaklabel} command, and \texttt{mdwlist} provides it via its \texttt{\desclabelstyle} command.

The other common occasion for the message is when you’re using the \texttt{center} (or \texttt{flushleft} or \texttt{flushright}) environment, and have decided you need extra separation between lines in the environment:
\begin{center}
  First (heading) line\\
  body of the centred text...
\end{center}

The solution here is plain: use the command in the way it’s supposed to be used, to provide more than just a single line break space. \\ takes an optional argument, which specifies how much extra space to add; the required effect in the text above can be had by saying:
\begin{center}
  First (heading) line\[
  body of the centred text...
\end{center}

The original \TeX provided the \texttt{\include} command to address the problem of long documents: with the relatively slow computers of the time, the companion \texttt{\includeonly} facility was a boon. With the vast increase in computer speed,
\texttt{\textbackslash includeonly} is less valuable (though it still has its place in some very large projects). Nevertheless, the facility is retained in current \TeX, and causes some confusion to those who misunderstand it.

In order for \texttt{\textbackslash includeonly} to work, \texttt{\include} makes a separate .aux file for each included file, and makes a ‘checkpoint’ of important parameters (such as page, figure, table and footnote numbers); as a direct result, it must clear the current page both before and after the \texttt{\include} command. What’s more, this mechanism doesn’t work if a \texttt{\include} command appears in a file that was \texttt{\included} itself: \TeX diagnoses this as an error.

So, we can now answer the two commonest questions about \texttt{\include}:

- Why does \TeX throw a page before and after \texttt{\include} commands? Answer: because it has to. If you don’t like it, replace the \texttt{\include} command with \texttt{\input} — you won’t be able to use \texttt{\includeonly} any more, but you probably don’t need it anyway, so don’t worry.

- Why can’t I nest \texttt{\include} files? — I always used to be able to under \TeX 2.09. Answer: in fact, you couldn’t, even under \TeX 2.09, but the failure wasn’t diagnosed. However, since you were happy with the behaviour under \TeX 2.09, replace the \texttt{\include} commands with \texttt{\input} commands (with \texttt{\clearpage} as appropriate).

\section{229 Why does it ignore paragraph parameters?}

When \TeX is laying out text, it doesn’t work from word to word, or from line to line; the smallest complete unit it formats is the paragraph. The paragraph is laid down in a buffer, as it appears, and isn’t touched further until the end-paragraph marker is processed. It’s at this point that the paragraph parameters have effect; and it’s because of this sequence that one often makes mistakes that lead to the paragraph parameters not doing what one would have hoped (or expected).

Consider the following sequence of \TeX:

\begin{verbatim}
{\raggedright % declaration for ragged text
Here’s text to be ranged left in our output, but it’s the only such paragraph, so we now end the group.}

Here’s more that needn’t be ragged...
\end{verbatim}

\TeX will open a group, and set the ragged-setting parameters within that group; it will then save a couple of sentences of text and close the group (thus restoring the previous value of the ragged-setting parameters). Then it encounters a blank line, which it knows to treat as a \texttt{\par} token, so it typesets the two sentences; but because the enclosing group has now been closed, the parameter settings have been lost, and the paragraph will be typeset normally.

The solution is simple: close the paragraph inside the group, so that the setting parameters remain in place. An appropriate way of doing that is to replace the last three lines above with:

\begin{verbatim}
end the group.\par
Here’s more that needn’t be ragged...
\end{verbatim}

In this way, the paragraph is completed while the setting parameters are still in force within the enclosing group.

Another alternative is to define an environment that does the appropriate job for you. For the above example, \TeX already defines an appropriate one:

\begin{verbatim}
\begin{flushleft}
Here’s text to be ranged left...
\end{flushleft}
\end{verbatim}

\section{230 Case-changing oddities}

\TeX provides two primitive commands \texttt{\uppercase} and \texttt{\lowercase} to change the case of text; they’re not much used, but are capable creating confusion.

The two commands do not expand the text that is their parameter — the result of \texttt{\uppercase{abc}} is ‘ABC’, but \texttt{\lowercase{\abc}} is always ‘\abc’, whatever the meaning of ‘\abc’. The commands are simply interpreting a table of equivalences
between upper- and lowercase characters. They have (for example) no mathematical sense, and

```
\uppercase{\text{About $y=f(x)$}}
```

will produce

```
ABOUT $Y=F(X)$
```

which is probably not what is wanted.

In addition, \texttt{\uppercase} and \texttt{\lowercase} do not deal very well with non-American characters, for example \texttt{\uppercase{\æ}} is the same as \æ.

\LaTeX{} provides commands \texttt{\MakeUppercase} and \texttt{\MakeLowercase} which fixes the latter problem. These commands are used in the standard classes to produce upper case running heads for chapters and sections.

Unfortunately \texttt{\MakeUppercase} and \texttt{\MakeLowercase} do not solve the other problems with \texttt{\uppercase}, so for example a section title containing \texttt{\begin{tabular} \end{tabular}} will produce a running head containing \texttt{\begin{tabular}{TABULAR}}. The simplest solution to this problem is using a user-defined command, for example:

```
\newcommand{\mytable}{\begin{tabular} \end{tabular}}
\section{A section title \protect\mytable{} with a table}
```

Note that \texttt{\mytable} has to be protected, otherwise it will be expanded and made uppercase; you can achieve the same result by declaring it with \texttt{\DeclareRobustCommand}, in which case the \texttt{\protect} won’t be necessary.

David Carlisle’s \texttt{textcase} package addresses many of these problems in a transparent way. It defines commands \texttt{\MakeTextUppercase} and \texttt{\MakeTextLowercase} which do upper- or lowercase, with the fancier features of the \LaTeX{} standard \texttt{\Make*} commands but without the problems mentioned above. Load the package with \texttt{\usepackage[overload]{textcase}}, and it will redefine the standard commands, so that section headings and the like don’t produce broken page headings.

\texttt{textcase.sty}: Distributed as part of \texttt{macros/latex/contrib/supported/carlisle}

231 Why does \LaTeX{} split footnotes across pages?
\LaTeX{} splits footnotes when it can think of nothing better to do. Typically, when this happens, the footnote mark is at the bottom of the page, and the complete footnote would overfill the page. \LaTeX{} could try to salvage this problem by making the page short of both the footnote and the line with the footnote mark, but its priorities told it that splitting the footnote would be preferable.

As always, the best solution is to change your text so that the problem doesn’t occur in the first place. Consider whether the text that bears the footnote could move earlier in the current page, or on to the next page.

If this isn’t possible, you might want to change \LaTeX{}’s perception of its priorities: they’re controlled by \texttt{\interfootnotelinepenalty} — the larger it is, the less willing \LaTeX{} is to split footnotes.

Setting

```
\interfootnotelinepenalty=10000
```

inhibits split footnotes altogether, which will cause ‘Underfull \vbox’ messages unless you also specify \texttt{\raggedbottom}. The default value of the penalty is 100, which is rather mild.

An alternative technique is to juggle with the actual size of the pages. \texttt{\enlargethispage} changes the size of the current page by its argument (for example, you might say \texttt{\enlargethispage{\baselineskip}} to add a single line to the page, but you can use any ordinary \TeX{} length such as \texttt{15mm} or \texttt{-20pt} as argument). Reducing the size of the current page could force the offending text to the next page; increasing the size of the page may allow the footnote to be included in its entirety. It may be necessary to change the size of more than one page.

232 Getting \texttt{\marginpar on the right side}
In \tex\twoside{} documents, \LaTeX{} makes sterling attempts to put \texttt{\marginpars} in the correct margin (the outer or the gutter margin, according to the user’s command). However, a booby-trap arises because \TeX{} runs its page maker asynchronously. If a
\texttt{\textbackslash marginpar} is processed while page \( n \) is being built, but doesn’t get used until page \( n+1 \), then the \texttt{\textbackslash marginpar} will turn up on the wrong side of the page. This is an instance of a general problem: see question 211.

The solution to the problem is for \LaTeX{} to ‘remember’ which side of the page each \texttt{\textbackslash marginpar} should be on. The \texttt{mparhack} package does this, using marks stored in the \texttt{.aux} file.

\texttt{mparhack.sty}: \texttt{macros/latex/contrib/supported/mparhack}

### 233 Where have my characters gone?

You’ve typed some apparently reasonable text and processed it, but the result contains no sign of some of the characters you typed. A likely reason is that the font you selected just doesn’t have a representation for the character in question.

For example, if I type “that will be £44.00” into an ordinary \LaTeX{} document, or if I select the font \texttt{rsfs10} (which contains uppercase letters only) and type pretty much anything, the £ sign, or any lowercase letters or digits will not appear in the output. There’s no actual error message, either: you have to read the log file, where you’ll find cryptic little messages like:

\begin{verbatim}
Missing character: There is no "a3 in font cmr10!
Missing character: There is no 3 in font rsfs10!
\end{verbatim}

(the former demonstrating my \TeX{}’s unwillingness to deal in characters which have the eighth bit set, while the \texttt{rsfs10} example shows that \TeX{} will log the actual character in error, if it thinks it’s possible).

Somewhat more understandable are the diagnostics you may get from \texttt{dvips} when using the OT1 and T1 versions of fonts that were supplied in Adobe standard encoding: \texttt{dvips}: Warning: missing glyph ‘Delta’

The process that generates the metrics for using the fonts generates an instruction to \texttt{dvips} to produce these diagnostics, so that their non-appearance in the printed output is less surprising than it might be. Quite a few glyphs available in Knuth’s text encodings and in the Cork encoding are not available in the Adobe fonts. In these cases, there is a typeset sign of the character: \texttt{dvips} produces a black rectangle of whatever size the concocted font file has specified.

### 234 “Rerun” messages won’t go away

The \LaTeX{} message “Rerun to get crossreferences right” is supposed to warn the user that the job needs to be processed again, since labels seem to have changed since the previous run. (\LaTeX{} compares the labels it has created this time round with what it found from the previous run when it started; it does this comparison at \texttt{\end{document}}.)

Sometimes, the message won’t go away: however often you reprocess your document, \LaTeX{} still tells you that “Label(s) may have changed”. This can sometimes be caused by a broken package: both \texttt{footmisc} (with the \texttt{perpage} option) and \texttt{hyperref} have been known to give trouble, in the past: if you are using either, check you have the latest version, and upgrade if possible.

However, there is a rare occasion when this can happen as a result of pathological structure of the document itself. Suppose you have pages numbered in roman, and you add a reference to a label on page “ix” (9). The presence of the reference pushes the thing referred to onto page “x” (10), but since that’s a shorter reference the label moves back to page “ix” at the next run. Such a sequence can obviously not terminate.

The only solution to this problem is to make a small change to your document (something as small as adding or deleting a comma will often be enough).

\texttt{footmisc.sty}: \texttt{macros/latex/contrib/supported/footmisc}

\texttt{hyperref.sty}: \texttt{macros/latex/contrib/supported/hyperref}

### 235 Commands gobble following space

People are forever surprised that simple commands gobble the space after them: this is just the way it is. The effect arises from the way \TeX{} works, and Lamport describes a solution (place a pair of braces after a command’s invocation) in the description of \LaTeX{} syntax. Thus the requirement is in effect part of the definition of \LaTeX{}.

This FAQ, for example, is written with definitions that \texttt{require} one to type \texttt{fred{}} for almost all macro invocations, regardless of whether the following space is required:
however, this FAQ is written by highly dedicated (and, some would say, eccentric) people. Many users find all those braces become very tedious very quickly, and would really rather not type them all.

An alternative structure, that doesn’t violate the design of \LaTeX, is to say \fred — the \ command is “self terminating” (like \) and you don’t need braces after it. Thus one can reduce to one the extra characters one needs to type.

If even that one character is too many, the package xspace defines a command \xspace that guesses whether there should have been a space after it, and if so introduces that space. So “fred\xspace jim” produces “fred jim”, while “fred\xspace. jim” produces “fred. jim”. Which usage would of course be completely pointless; but you can incorporate \xspace in your own macros:
\begin{verbatim}
\usepackage{xspace}
\end{verbatim}
... \begin{verbatim}
\newcommand{\restenergy}{\ensuremath{mc^2}\xspace}
\end{verbatim}
... and we find \restenergy available to us...

The \xspace command must be the last thing in your macro definition (as in the example); it’s not completely foolproof, but it copes with most obvious situations in running text.

The xspace package doesn’t save you anything if you only use a modified macro once or twice within your document, and in any case be careful with usage of \xspace — it offers a change in your input syntax which can be confusing, particularly if you retain some commands which don’t use it. (Of course, any command built into \LaTeX or into any class or package you use won’t use \xspace: you need to think every time you use such a command.) And of course, be careful to explain what you’re doing to any collaborating author!

xspace.sty: Distributed as part of macros/latex/required/tools

236 (B)\LaTeX makes overfull lines

When \LaTeX is building a paragraph, it can make several attempts to get the line-breaking right; on each attempt it runs the same algorithm, but gives it different parameters. You can affect the way \LaTeX’s line breaking works by adjusting the parameters: this answer deals with the “tolerance” and stretchability parameters. The other vital ‘parameter’ is the set of hyphenations to be applied: see question 176 (and the questions it references) for advice.

If you’re getting an undesired “overfull box”, what has happened is that \LaTeX has given up: the parameters you gave it don’t allow it to produce a result that doesn’t overfill. In this circumstance, Knuth decided the best thing to do was to produce a warning, and to allow the user to solve the problem. (The alternative, silently to go beyond the envelope of “good taste” defined for this run of \LaTeX, would be distasteful to any discerning typographer.) The user can almost always address the problem by rewriting the text that’s provoking the problem — but that’s not always possible, and in some cases it’s impossible to solve the problem without adjusting the parameters. This answer discusses the approaches one might take to resolution of the problem, on the assumption that you’ve got the hyphenation correct.

The simplest case is where a ‘small’ word fails to break at the end of a line; pushing the entire word to a new line isn’t going to make much difference, but it might make things just bad enough that \LaTeX won’t do it by default. In such a case on can try the \LaTeX \texttt{\linebreak} command: it may solve the problem, and if it does, it will save an awful lot of fiddling. Otherwise, one needs to adjust parameters: to do that we need to recap the details of \LaTeX’s line breaking mechanisms.

\LaTeX’s first attempt at breaking lines is performed without even trying hyphenation: \LaTeX sets its “tolerance” of line breaking oddities to the internal value \texttt{\pretolerance}, and sees what happens. If it can’t get an acceptable break, \LaTeX adds the hyphenation points allowed by the current patterns, and tries again using the internal \texttt{\tolerance} value. If this pass also fails, and the internal \texttt{\emergencystretch} value is positive, \LaTeX will try a pass that allows \texttt{\emergencystretch} worth of extra stretchability to the spaces in each line.

In principle, therefore, there are three parameters (other than hyphenation) that you can change: \texttt{\pretolerance}, \texttt{\tolerance} and \texttt{\emergencystretch}. Both the
tolerance values are simple numbers, and should be set by TeX primitive count assignment — for example
\pretolerance=150

For both, an “infinite” tolerance is represented by the value 10000, but infinite tolerance is rarely appropriate, since it can lead to very bad line breaks indeed.
\emergencystretch is a TeX-internal ‘dimen’ register, and can be set as normal for dimens in Plain TeX; in LATEX, use \setlength — for example:
\setlength{\emergencystretch}{3em}

The choice of method has time implications — each of the passes takes time, so adding a pass (by changing \emergencystretch) is less desirable than suppressing one (by changing \pretolerance). However, it’s unusual nowadays to find a computer that’s slow enough that the extra passes are really troublesome.

In practice, \pretolerance is rarely used other than to manipulate the use of hyphenation; Plain TeX and LATEX both set its value to 100. To suppress the first scan of paragraphs, set \pretolerance to -1.
\tolerance is often a good method for adjusting spacing; Plain TeX and LATEX both set its value to 200. LATEX’s \sloppy command sets it to 9999, as does the sloppypar environment. This value is the largest available, this side of infinity, and can allow pretty poor-looking breaks (this author rarely uses \sloppy “bare”, though he does occasionally use sloppypar — that way, the change of \tolerance is confined to the environment). More satisfactory is to make small changes to \tolerance, incrementally, and then to look to see how the change affects the result; very small increases can often do what’s necessary. Remember that \tolerance is a paragraph parameter, so you need to ensure it’s actually applied — see question 229. LATEX users could use an environment like:
\newenvironment{tolerant}[1]{%  
\par\tolerance=#1\relax
}{%  
\par
}

enclosing entire paragraphs (or set of paragraphs) in it.
\emergencystretch is a slightly trickier customer to understand. The example above set it to 3em; the Computer Modern fonts ordinarily fit three space skips to the em, so the change would allow anything up to the equivalent of nine extra spaces in each line. In a line with lots of spaces, this could be reasonable, but with (say) only three spaces on the line, each could stretch to four times its natural width.

V The joy of TeX errors

237 How to approach errors
Since TeX is a macroprocessor, its error messages are often difficult to understand; this is a (seemingly invariant) property of macroprocessors. Knuth makes light of the problem in the TeXbook, suggesting that you acquire the sleuthing skills of a latter-day Sherlock Holmes; while this approach has a certain romantic charm to it, it’s not good for the ‘production’ user of (L)ATEX. This answer (derived, in part, from an article by Sebastian Rahtz in TUGboat 16(4)) offers some general guidance in dealing with TeX error reports, and other answers in this section deal with common (but perplexing) errors that you may encounter. There’s a long list of “hints” in Sebastian’s article, including the following:

• Look at TeX errors; those messages may seem cryptic at first, but they often contain a straightforward clue to the problem. See question 238 for further details.
• Read the .log file; it contains hints to things you may not understand, often things that have not even presented as error messages.
• Be aware of the amount of context that TeX gives you. The error messages gives you some bits of TeX code (or of the document itself), that show where the error “actually happened”; it’s possible to control how much of this ‘context’ TeX actually gives you. LATEX (nowadays) instructs TeX only to give you one line of context, but you may tell it otherwise by saying
in the preamble of your document. (If you’re not a confident macro programmer, don’t be ashamed of cutting that 999 down a bit; some errors will go on and on, and spotting the differences between those lines can be a significant challenge.)

- As a last resort, tracing can be a useful tool; reading a full \LaTeX{} trace takes a strong constitution, but once you know how, the trace can lead you quickly to the source of a problem. You need to have read the \TeX{}book (see question 22) in some detail, fully to understand the trace.

The command \texttt{\tracingall} sets up maximum tracing; it also sets the output to come to the interactive terminal, which is somewhat of a mixed blessing (since the output tends to be so vast — all but the simplest traces are best examined in a text editor after the event).

The \LaTeX{} trace package (first distributed with the 2001 release of \LaTeX{}) provides more manageable tracing. Its \texttt{\traceon} command gives you what \texttt{\tracingall} offers, but suppresses tracing around some of the truly verbose parts of \LaTeX{} itself. The package also provides a \texttt{\traceoff} command (there’s no “off” command for \texttt{\tracingall}), and a package option (\texttt{logonly}) allows you to suppress output to the terminal.

The best advice to those faced with \TeX{} errors is not to panic: most of the common errors are plain to the eye when you go back to the source line that \TeX{} tells you of. If that approach doesn’t work, the remaining answers in this section deal with some of the odder error messages you may encounter. You should not ordinarily need to appeal to the wider public (question 25) for assistance, but if you do, be sure to report full backtraces (see \texttt{errorcontextlines} above) and so on.

\texttt{trace.sty}: Distributed as part of \texttt{macros/latex/required/tools}

238 The structure of \TeX{} error messages

\TeX{}’s error messages are reminiscent of the time when \TeX{} itself was conceived (the 1970s): they’re not terribly user-friendly, though they do contain all the information that \TeX{} can offer, usually in a pretty concise way.

\TeX{}’s error reports all have the same structure:

- An error message
- Some ‘context’
- An error prompt

The error message will relate to the \TeX{} condition that is causing a problem. Sadly, in the case of complex macro packages such as \ifTeX{}, the underlying \TeX{} problem may be superficially difficult to relate to the actual problem in the “higher-level” macros. Many \ifTeX{}-detected problems manifest themselves as ‘generic’ errors, with error text provided by \ifTeX{} itself (or by a \ifTeX{} class or package).

The context of the error is a stylised representation of what \TeX{} was doing at the point that it detected the error. As noted in question 237, a macro package can tell \TeX{} how much context to display, and the user may need to undo what the package has done. Each line of context is split at the point of the error; if the error actually occurred in a macro called from the present line, the break is at the point of the call. (If the called object is defined with arguments, the “point of call” is after all the arguments have been scanned.) For example:

\begin{verbatim}
\texttt{\newcommand{\textbackslash blah}[1]{\textbackslash bleah #1}}
\texttt{\textbackslash blah(to you), folks}
\end{verbatim}

produces the error report

! Undefined control sequence.
1.4 \texttt{\textbackslash blah}

and so on

while:

\begin{verbatim}
\texttt{\textbackslash newcommand{\textbackslash blah}[1]{\textbackslash bleah #1}}
\texttt{\textbackslash bleah(to you), folks}
\end{verbatim}

produces the error report

! Undefined control sequence.
\texttt{\textbackslash bleah #1->\textbackslash bleah #1}
If the argument itself is in error, we will see things such as
\newcommand{\blah}[1]{#1 to you}
\blah{\bleah}, folks
producing
! Undefined control sequence.
<argument> \bleah

1.5 \blah\{\bleah\}
        , folks

The prompt accepts single-character commands: the list of what’s available may be had by typing \?. One immediately valuable command is h, which gives you an expansion of \TeX’s original précis message, sometimes accompanied by a hint on what to do to work round the problem in the short term. If you simply type ‘return’ (or whatever else your system uses to signal the end of a line) at the prompt, \TeX will attempt to carry on (often with rather little success).

239 An extra ‘}’??
You’ve looked at your \LaTeX source and there’s no sign of a misplaced } on the line in question.

    Well, no: this is \TeX’s cryptic way of hinting that you’ve put a fragile command in a moving argument (see question 207).

    For example, \footnote is fragile, and if we put that in the moving argument of a \section command, as
\section{Mumble\footnote{I couldn’t think of anything better}}
we get told
! Argument of \@sect has an extra }.

    The solution is usually to use a robust command in place of the one you are using, or to force your command to be robust by prefixing it with \protect, which in the above case would show as
\section{Mumble\protect\footnote{I couldn’t think of anything better}}

    Note that, in some cases, simple \protect is not the answer; question 166 deals specifically with this case.

240 Capacity exceeded [semantic nest…]
! \TeX capacity exceeded, sorry [semantic nest
    size=100].
...
If you really absolutely need more capacity, you can ask a wizard to enlarge me.

Even though \TeX suggests (as always) that enlargement by a wizard may help, this message usually results from a broken macro or bad parameters to an otherwise working macro.

The “semantic nest” \TeX talks about is the nesting of boxes within boxes. A stupid macro can provoke the error pretty easily:
\def\silly{\hbox{here’s \silly being executed}}
\silly

The extended traceback (see question 237) does help, though it does rather run on. In the case above, the traceback consists of
\silly ->\hbox {
        here’s \silly being executed}
followed by 100 instances of
\silly ->\hbox {here’s \silly
        being executed}
The repeated lines are broken at exactly the offending macro; of course the loop need not be as simple as this — if \silly calls \dopy which boxes \silly, the effect is just the same and alternate lines in the traceback are broken at alternate positions.

241 No room for a new ‘thing’

The technology available to Knuth at the time \TeX was written is said to have been particularly poor at managing dynamic storage; as a result much of the storage used within \TeX is allocated as fixed arrays, in the reference implementations. Many of these fixed arrays are expandable in modern \TeX implementations, but size of the arrays of “registers” is written into the specification as being 256 (usually); this number may not be changed if you still wish to call the result \TeX (see question 5).

If you fill up one of these register arrays, you get a \TeX error message saying

! No room for a new \texttt{\langle\text{thing}\rangle}.

The \texttt{\langle\text{things}\rangle} in question may be \count (the object underlying \LaTeX’s \newcounter command), \skip (the object underlying \LaTeX’s \newlength command), \box (the object underlying \LaTeX’s \newsavebox command), or \dimen, \muskip, \toks, \read, \write or \language (all types of object whose use is “hidden” in \LaTeX; the limit on the number of \read or \write objects is just 16).

There is nothing that can directly be done about this error, as you can’t extend the number of available registers without extending \TeX itself. (Of course, \Omega and \epsilon-\TeX — see questions 251 and 252 respectively — both do this, as does MicroPress Inc’s \VT\TeX — see question 55.)

The commonest way to encounter one of these error messages is to have broken macros of some sort, or incorrect usage of macros (an example is discussed in question 242). However, sometimes one just needs more than \TeX can offer, and when this happens, you’ve just got to work out a different way of doing things. An example is the difficulty of loading \P\TeX with \LaTeX (see question 216).

242 epsf gives up after a bit

Some copies of the documentation of \texttt{epsf.tex} seem to suggest that the command

\begin{verbatim}
\input epsf
\end{verbatim}

is needed for every figure included. If you follow this suggestion too literally, you get an error

! No room for a new \texttt{\langle\text{read}\rangle}.

after a while; this is because each time \texttt{epsf.tex} is loaded, it allocates itself a new file-reading handle to check the figure for its bounding box, and there just aren’t enough of these things (see question 241).

The solution is simple — this is in fact an example of misuse of macros; one only need read \texttt{epsf.tex} once, so change

\begin{verbatim}
...\input epsf
\epsffile{...}
...\input epsf
\epsffile{...}
\end{verbatim}

(and so on) with a single

\begin{verbatim}
\input epsf
\epsffile{...}
\end{verbatim}

somewhere near the start of your document, and then decorate your \texttt{\epsffile} statements with no more than adjustments of \texttt{\epsfxsize} and so on.

243 Improper \hyphenation will be flushed

For example

! Improper \texttt{\hyphenation{will be flushed}}.

\begin{verbatim}
\'\#{1}->{
\accent 19 \#1}
\texttt{\hyphenation{Ji-m'e
-mez}}
\end{verbatim}

(in Plain \TeX) or

122
Improper \hyphenation will be flushed.  
\leavevmode ->\unhbox \voidb@x
\hyphenation{Jim\`e-nez}
in L\TeX.  

As mentioned in question 176, words with accents in them may not be hyphenated. As a result, any such word is deemed improper in a \hyphenation command.  
The solution is to use a font that contains the character in question, and to express the \hyphenation command in terms of that character; this “hides” the accent from the hyphenation mechanisms.  L\TeX\ users can be achieved this by use of the fontenc package (part of the L\TeX\ distribution). If you select an 8-bit font with the package, as in \usepackage[T1]{fontenc}, accented-letter commands such as the \^e in \hyphenation{Jim\^e-nez} automatically become the single accented character by the time the hyphenation gets a to look at it.  

244 “Too many unprocessed floats”  
If L\TeX\ responds to a \begin{figure} or \begin{table} command with the error message  
! LaTeX Error: Too many unprocessed floats.  

See the LaTeX manual or LaTeX Companion for explanation.  
your figures (or tables) are failing to be placed properly. L\TeX\ has a limited amount of storage for ‘floats’ (figures, tables, or floats you’ve defined yourself with the float package); if you don’t let it ever actually typeset any floats, it will run out of space.  

This failure usually occurs in extreme cases of floats moving “wrongly” (see question 217); L\TeX\ has found it can’t place a float, and floats of the same type have piled up behind it. L\TeX\’s idea is to ensure that caption numbers are sequential in the document: the caption number is allocated when the figure (or whatever) is created, and can’t be changed, so that placement out of order would mean figure numbers appearing out of order in the document (and in the list of figures, or whatever). So a simple failure to place a figure means that no subsequent figure can be placed; and hence (eventually) the error.  

Techniques for solving the problem are discussed in the floats question (217) already referenced.  
The error also occurs in a long sequence of figure or table environments, with no intervening text. Unless the environments will fit “here” (and you’ve allowed them to go “here”), there will never be a page break, and so there will never be an opportunity for L\TeX\ to reconsider placement. (Of course, the floats can’t all fit “here” if the sequence is sufficiently polonged: once the page fills, L\TeX\ won’t place any more floats. Techniques for resolution may involve redefining the floats using the float package’s \[H\] float qualifier, but you are unlikely to get away without using \clearpage from time to time.  

\texttt{float.sty: macros/latex/contrib/supported/float}  
245 \spacefactor complaints  
The errors  
! You can’t use ‘\spacefactor’ in vertical mode. \@->\spacefactor \@m  
or  
! Improper \spacefactor.  
...  
come of using a L\TeX\ internal command without taking the precaution of fooling L\TeX\ that you’re inside it. The problem is discussed in detail in question 206, together with solutions.  

246 \end occurred inside a group  
The actual error we observe is:  
(\end occurred inside a group at level <n>)
and it tells us that something we started in the document never got finished before we ended the document itself. The things involved (‘groups’) are what \TeX uses for restricting the scope of things: you see them, for example, in the “traditional” font selection commands: {\it stuff} — if the closing brace is left off such a construct, the effect of {\it} will last to the end of the document, and you’ll get the diagnostic.

\TeX itself doesn’t tell you where your problem is, but you can often spot it by looking at the typeset output in a previewer. Otherwise, you can usually find mismatched braces using an intelligent editor (at least \texttt{emacs} and \texttt{winedt} offer this facility). However, groups are not only created by matching \{ with \}: other grouping commands are discussed elsewhere in these FAQs, and are also a potential source of unclosed group.

\begin{⟨environment⟩} encloses the environment’s body in a group, and establishes its own diagnostic mechanism. If you end the document before closing some other environment, you get the ‘usual’ \TeX diagnostic

\begin{verbatim}
! LaTeX Error: \begin{blah} on input line 6 ended by \end{document},
\end{verbatim}

which (though it doesn’t tell you which \texttt{file} the \begin{blah} was in) is usually enough to locate the immediate problem. If you press on past the LaTeX error, you get the “occurred inside a group” message before LaTeX finally exits.

In the absence of such information from \TeX, you need to use “traditional” binary search to find the offending group. Separate the header from the body of your \texttt{file}, and process each half on its own with the header; this tells you which half of the \texttt{file} is at fault. Divide again and repeat. The process needs to be conducted with care (it’s obviously possible to split a correctly-written group by chopping in the wrong place), but it will usually find the problem fairly quickly.

\TeX (and \texttt{elatex} — \LaTeX run on \TeX) gives you further diagnostics after the traditional infuriating \TeX one — it actually keeps the information in a similar way to LaTeX:

\begin{verbatim}
(\end occurred inside a group at level 3)

### semi simple group (level 3) entered at line 6 (\begingroup)
### simple group (level 2) entered at line 5 (\)
### simple group (level 1) entered at line 4 (\)
### bottom level
\end{verbatim}

The diagnostic not only tells us where the group started, but also the way it started: \begingroup or \{} (which is an alias of \bgroup, and the two are not distinguishable at the \TeX-engine level).

247 “Missing number, treated as zero”

In general, this means you’ve tried to assign something to a count, dimension or skip register that isn’t (in \TeX’s view of things) a number. Usually the problem will become clear using the ordinary techniques of examining errors (see question 237).

Two \TeX-specific errors are commonly aired on the newsgroups.

The first arises from a misconfiguration in a system that has been upgraded from \TeX 2.09: the document uses \usepackage{times}, and the error appears at \begin{document}: the likely cause is lurking files that remain from the \TeX 2.09 version of the \times package. The \times package in \psnfss is a very simple beast, but the \TeX 2.09 version is quite complicated, and loads another package: this is clear in the log if you have been “bitten” this way. The resolution is to clear out all the old PostScript-related packages, and then to install \psnfss.

The second arises from attempting to use an example describe in \texttt{The \TeX Companion} (see question 22), and is exemplified by the following error text:

\begin{verbatim}
! Missing number, treated as zero.
\end{verbatim}

\begin{verbatim}
\relax
1.21 \begin{Ventry}{Return values}
\end{verbatim}

The problem arises because the Companion’s examples always assume that the \texttt{calc} package is loaded: this fact is mentioned in the book, but often not noticed. The remedy is to load the \texttt{calc} package in any document using such examples from the Companion.

\texttt{calc.sty}: Distributed as part of \texttt{macros/latex/required/tools}

\texttt{The psnfss bundle}: \texttt{macros/latex/required/psnfss}
Sometimes, when you are running \LaTeX, it will abruptly stop and present you with a prompt (by default, just a * character). Many people (including this author) will reflexively hit the ‘return’ key, pretty much immediately, and of course this is no help at all — \TeX just says:

\texttt{(Please type a command or say ‘\textbackslash end’)}

and prompts you again.

What’s happened is that your \LaTeX file has finished prematurely, and \TeX has fallen back to a supposed including file, from the terminal. This could have happened simply be because you’ve omitted the \texttt{\bye} (Plain \TeX), \texttt{\end{document}} (\LaTeX), or whatever. Other common errors are failure to close the braces round a command’s argument, or (in \LaTeX) failure to close a verbatim environment: in such cases you’ve already read and accepted an error message about encountering end of file while scanning something.

If the error is indeed because you’ve forgotten to end your document, you can insert the missing text: if you’re running Plain \TeX, the advice, to “say \textbackslash end” is good enough: it will kill the run; if you’re running \LaTeX, the argument will be necessary: \texttt{\end{document}}.

However, as often as not this isn’t the problem, and (short of debugging the source of the document before ending) brute force is probably necessary. Excessive force (killing the job that’s running \TeX) is to be avoided: there may well be evidence in the \texttt{.log} file that will be useful in determining what the problem is — so the aim is to persuade \TeX to shut itself down and hence flush all log output to file.

If you can persuade \TeX to read it, an end-of-file indication (control-D under Unix, control-Z under Windows) will provoke \TeX to report an error and exit immediately. Otherwise you should attempt to provoke an error dialogue, from which you can exit (using the x ‘command’). An accessible error could well be inserting an illegal character: what it is will depend on what macros you are running. If you can’t make that work, try a silly command name or two.

\textbf{249 “Unknown graphics extension”}

\LaTeX graphics package deals with several different types of DVI (or other) output drivers; each one of them has a potential to deal with a different selection of graphics formats. The package therefore has to be told what graphics file types its output driver knows about; this is usually done in the \texttt{⟨driver⟩.def} file corresponding to the output driver you’re using.

The error message arises, then, if you have a graphics file whose extension doesn’t correspond with one your driver knows about. Most often, this is because you’re being optimistic: asking \texttt{dvips} to deal with a \texttt{.png} file, or PDF\TeX to deal with a \texttt{.eps} file: the solution in this case is to transform the graphics file to a format your driver knows about.

If you happen to know that your device driver deals with the format of your file, you are probably falling foul of a limitation of the file name parsing code that the graphics package uses. Suppose you want to include a graphics file whose extension doesn’t correspond with one your drivers knows about. Most often, this is because you’re being optimistic: asking \texttt{dvips} to deal with a \texttt{.png} file, or PDF\TeX to deal with a \texttt{.eps} file: the solution in this case is to transform the graphics file to a format your driver knows about.

If you happen to know that your device driver deals with the format of your file, you are probably falling foul of a limitation of the file name parsing code that the graphics package uses. Suppose you want to include a graphics file \texttt{home.bedroom.eps} using the \texttt{dvips} driver; the package will conclude that your file’s extension is \texttt{.bedroom.eps}, and will complain. To get around this limitation, you have three alternatives:

\begin{itemize}
  \item Rename the file — for example \texttt{home.bedroom.eps}→\texttt{home-bedroom.eps}
  \item Mask the first dot in the file name:
  \begin{verbatim}
\newcommand*{\DOT}{.}
\includegraphics{home\DOT bedroom.eps}
\end{verbatim}
  \item Tell the graphics package what the file is, by means of options to the \texttt{\includegraphics} command:
  \begin{verbatim}
\includegraphics[type=eps,ext=.eps,read=.eps]{home.bedroom}
\end{verbatim}
\end{itemize}

\section{Current \TeX Projects}

\textbf{250 The \LaTeX3 project}

The \LaTeX3 project team (see \url{http://www.latex-project.org/latex3.html}) is a small group of volunteers whose aim is to produce a major new document processing
The system is based on the principles pioneered by Leslie Lamport in the current \LaTeX. It will remain freely available and it will be fully documented at all levels.

The \LaTeXX team’s first product (\LaTeXX\ 2\epsilon) was delivered in 1994 (it’s now properly called “\LaTeX”, since no other version is current).

\LaTeXX\ 2\epsilon was intended as a consolidation exercise, unifying several sub-variants of \LaTeX while changing nothing whose change wasn’t absolutely necessary. This has permitted the team to support a single version of \LaTeXX, in parallel with development of \LaTeXX3.

Some of the older discussion papers about directions for \LaTeXX3 are to be found at info/ltx3pub; other (published) articles are to be found on the project web site (see http://www.latex-project.org/articles.html), as is some of the project’s experimental code (http://www.latex-project.org/experimental). You can participate in discussions of the future of \LaTeXX through the mailing list latex-l. Subscribe to the list by sending a message ‘subscribe latex-l <your name>’ to listserv@urz.Uni-Heidelberg.de

251 The Omega project

Omega (Ω) is a program built by extension of the \TeX sources which works internally with ‘wide’ characters (it is capable of dealing with all of Unicode version 3); this allows it to work with most scripts in the world with few difficulties from coding schemes. Ω also has a powerful concept of input and output filters to allow the user to work with existing transliteration schemes, etc.

An email discussion list is available: subscribe by sending a message ‘subscribe’ to omega-request@omega.cse.unsw.edu.au

Ω was first released in November 1996 by the project originators, John Plaice and Yannis Haralambous; a recent version is maintained on CTAN. However, Ω is now an open source project, and details of a cvs repository, as well as papers and other information, are available via the project’s web site.

Implementations of Ω are available as part of the te\TeX, mik\TeX, fp\TeX and CMac\TeX distributions (see question 53); Ω is also distributed on the \TeX Live CD-ROM (see question 52).

CTAN distribution: systems/omega

252 The \NTS project

The \NTS project was established in 1992, to produce a typesetting system that’s even better than \TeX. The project is not simply enhancing \TeX, for two reasons: first, that \TeX itself has been frozen by Knuth (see question 18), and second, even if they were allowed to develop the program, some members of the \NTS team feel that \TeX in its present form is simply unsuited to further development. While all those involved in the project are committed to \TeX, they recognise that the end product may very well have little in common with \TeX other than its philosophy.

The group’s first product was nevertheless a set of extensions and enhancements to \TeX, implemented through the standard medium of a change-file. The extended system is known ε-\TeX, and is 100% compatible with \TeX; furthermore, ε-\TeX can construct a format that is “\TeX”, with no extensions or enhancements present.

The most recent base source of ε-\TeX (i.e., the Web change file) is available on CTAN. Implementations of ε-\TeX are also distributed on the \TeX Live CD-ROM (see question 52), and with most other modern free \TeX distributions.

The project has now produced a β-version of \TeX written (from scratch) in Java. Since it isn’t \TeX (it remains slightly incompatible in microscopic ways), it’s known as \NTS. As might be expected, this first re-implementation runs rather slowly, but its operation has been demonstrated in public, and the β-release is available on CTAN.

e-\TeX: Browse systems/e-tex

\NTS: systems/nts

253 The PDF\TeX project

PDF\TeX (formerly known as \TeX2PDF) arose from Hán Thế Thành’s post-graduate research at Masaryk University, Brno, Czech Republic. The basic idea is very simple: to provide a version of \TeX that can output PDF as an alternative format to DVI. PDF\TeX implements a small number of new primitives, to switch to PDF output, and to control various PDF features. Hán Thế Thành has worked on PDF\TeX throughout his Ph.D.
research into typesetting, and the latest release includes facilities written to support novel typesetting techniques.

The latest version of PDF\TeX{} is available on CTAN, and implementations are available as part the tc\TeX{}, mikt\TeX{}, fp\TeX{}, and CMac\TeX{} distributions (see question 53); it is also distributed on the \TeX{} Live CD-ROM (see question 52). A version (by the author of CMac\TeX{}) for use with Oz\TeX{} is also available on CTAN.

A mailing list discussing PDF\TeX{} is available; send a message containing (just) subscribe pdftex to majordomo@tug.org (you will be required to confirm your subscription).

**pdftex**: Browse systems/web2c for current sources

**pdftex for Oz\TeX{}**: nonfree/systems/mac/pdftex/pdftex_for_oztex.sit.bin

### 254 Future WEB technologies and (\LaTeX)\TeX{}

An earlier question (69) addresses the issue of converting existing (\LaTeX)\TeX{} documents for viewing on the Web as HTML. All the present techniques are somewhat flawed: the answer explains why.

However, things are changing, with better font availability, cunning HTML programming and the support for new Web standards.

**Font technologies** Direct representation of mathematics in browsers has been hampered up to now by the limited range of symbols in the fonts one can rely on being available. In the near future, we can expect rather wide availability of Unicode fonts with better coverage of symbols.

**XML** The core of the range of new standards is XML, which provides a framework for better structured markup; limited support for it has already appeared in some browsers.

Conversion of (\LaTeX)\TeX{} source to XML is already available (through \TeX{}4ht at least), and work continues in that arena. The alternative, authoring in XML (thus producing documents that are immediately Web-friendly, if not ready) and using (\LaTeX)\TeX{} to typeset is also well advanced. One useful technique is transforming the XML to IS\TeX{}, using XSLT, and then simply using IS\TeX{}; alternatively, one may typeset direct from the XML source (see question 70).

**Direct representation of mathematics** MathML is a standard for representing maths on the Web; its original version is distinctly limited, but efforts to give it greater richness (approaching that of \TeX{}) are under way. Browser support for MathML (e.g., in amaya, a version of the Netscape ‘Open Source’ browser mozilla and in specially extended versions of Internet Explorer) is becoming available. There’s evidence that (\LaTeX)\TeX{} users are starting to use such browsers.

Work in both the \TeX{}4ht and TtH projects, to produce MathML is well advanced.

**Graphics** SVG is a standard for graphics representation on the web. While the natural use is for converting existing figures, representations of formulas are also possible, in place of the separate bitmaps that have been used in the past (and while we wait for the wide deployment of MathML).

Browser plug-ins, that deal with SVG are already available (Adobe offer one, for example).

### 255 The \TeX{}Xtrace project

\TeX{}Xtrace is a bundle of Unix scripts that use a freeware boundary tracing package to generate Type 1 outline fonts from \METAFONT bitmap font outputs. The result is unlikely ever to be of the quality of the commercially-produced Type 1 font, but there remain fonts which many people find useful and which fail to attract the paid experts.

The project was started by Péter Szabó, and its current state is available via the project’s entry on Sourceforge and there are already a few sets of fonts on CTAN generated using \TeX{}Xtrace: Péter Szabó’s own EC/TC font set, Vladimir Volovich’s CM-Super set, which covers the EC, TC, and the Cyrillic LH font sets, and Takanori Uchiyama’s set of the Musix\TeX{} fonts.

A package that says it’s “inspired” by \TeX{}Xtrace is pktrace: this is a small Python program that does the same job — see http://www.cs.ruu.nl/~hanwen/pktrace/; it has not yet been used to generate fonts that have been installed on CTAN.

**CM-Super fonts**: fonts/ps-type1/cm-super
256 The \TeX{} document preparation environment

"Why \TeX{} is not WYSIWYG" (see question 20) outlines the reasons (or excuses) for the huge disparity of user interface between "typical" \TeX{} environments and commercial word processors.

Nowadays, at last, there is a range of tools available that try either to bridge or to close the gap. One range modestly focuses on providing the user with a legible source document. At the other extreme we have TeXmacs, a document processor using \TeX{}'s algorithms and fonts for both editor display and printing. TeXmacs does not use the \TeX{} language itself (though among other formats, \LaTeX{} may be exported and imported). A bit closer to \LaTeX{} is LyX, which has its own editor display and file formats as well, but does its print output by exporting to \LaTeX{}. The editor display merely resembles the printed output, but you have the possibility of entering arbitrary \LaTeX{} code. If you use constructs that LyX does not understand, it will just display them as source text marked red, but will properly export them.

Since a lot of work is needed to create an editor from scratch that actually is good at editing (as well as catering for \TeX{}), it is perhaps no accident that several approaches have been implemented using the extensible emacs editor. The low end of the prettifying range is occupied by syntax highlighting: marking \TeX{} tokens, comments and other stuff with special colours. Many free editors (including emacs) can cater for \TeX{} in this way. Under Windows, one of the more popular editors with such support is the Shareware product winedt. Continuing the range of tools prettifying your input, we have the emacs package x-symbol, which does the WYSIWYG part of its work by replacing single \TeX{} tokens and accented letter sequences with appropriately looking characters on the screen.

A different type of tool focuses on making update and access to previews of the typeset document more immediate. A recent addition in several viewers, editors and \TeX{} executables are so-called 'source specials' for cross-navigation. When \TeX{} compiles a document, it will upon request insert special markers for every input line into the typeset output. The markers are interpreted by the DVI previewer which can be made to let its display track the page corresponding to the editor input position, or to let the editor jump to a source line corresponding to a click in the preview window.

An emacs package that combines this sort of editor movement tracking with automatic fast recompliations (through the use of dumped formats) is Whizzy\TeX{} which is best used with a previewer by the same author. A simpler package in a similar spirit is called InstantPreview and makes use of a continuously running copy of \TeX{} (under the moniker of \TeX{} daemon) instead of dumping formats to achieve its speed.

Another emacs package called preview-latex tries to solve the problem of visual correlation between source and previews in a more direct way: it uses a \LaTeX{} package to chop the document source into interesting fragments (like figures, text or display math) which it runs through \LaTeX{} and replaces the source text of those fragments with the corresponding rendered output images. Since it does not know about the structure of the images, at the actual cursor position the source text is displayed while editing rather than the preview. This approach is more or less a hybrid of the source prettifying and fast preview approaches since it works in the source buffer but uses actual previews rendered by \LaTeX{}.

A more ambitious contender is called \TeX{}lite. This system is only available on request from its author; it continuously updates its screen with the help of a special version of \TeX{} dumping its state in a compressed format at each page and using hooks into \TeX{}'s line breaking mechanism for reformatting paragraphs on the fly. That way, it can render the output from the edited \TeX{} code with interactive speed on-screen, and it offers the possibility of editing directly in the preview window.

That many of these systems occupy slightly different niches can be seen by comparing the range of the emacs-based solutions ranging from syntax highlighting to instant previewing: all of them can be activated at the same time without actually interfering in their respective tasks.

The different approaches offer various choices differing in the immediacy of their response, the screen area they work on (source or separate window), degree of correspondence of the display to the final output, and the balance they strike between visual
aid and visual distraction.

preview-latex: Browse support/preview-latex
tezmacs: Browse systems/unix/TeXmacs

X You’re still stuck?

257 You don’t understand the answer

While the FAQ maintainers don’t offer a ‘help’ service, they’re very keen that you understand the answers they’ve already written. They’re (almost) written “in a vacuum”, to provide something to cover a set of questions that have arisen; it’s always possible that they’re written in a way that a novice won’t understand them.

Which is where you can help the community. Mail the maintainers to report the answer that you find unclear. Time permitting (the team is small and all its members are busy), we’ll try and clarify the answer. This way, with a bit of luck, we can together improve the value of this resource to the whole community.

(We need hardly say that we look forward to hearing from none of you: but we’re not so arrogant as to be confident that we won’t!)

258 Submitting new material for the FAQ

The FAQ will never be complete, and we always expect that there will be people out there who know better than we do about something or other. We always need to be put right about whatever we’ve got wrong, and suggestions for improvements, particularly covering areas we’ve missed, are always needed: mail anything you have to the maintainers.

If you have actual material to submit, your contribution is more than ever welcome. Submission in plain text is entirely acceptable, but if you’re really willing, you may feel free to mark up your submission in the form needed for the FAQ itself. The markup is a strongly-constrained version of \LaTeX — the constraints come from the need to translate the marked-up text to HTML on the fly (and hence pretty efficiently). There is a file markup-syntax in the FAQ distribution that describes the structure of the markup, but there’s no real substitute for reading at least some of the source (faqbody.tex) of the FAQ itself. If you understand perl, you may also care to look at the translation code in texfaq2file and sanitize.pl in the distribution: this isn’t the code actually used on the Web site, but it’s a close relation and is kept up to date for development purposes.

FAQ distribution: help/uk-tex-faq

259 Reporting a \LaTeX bug

The \LaTeX team supports \LaTeX, and will deal with bona fide bug reports. However, you need to be slightly careful to produce a bug report that is usable by the team. The steps are:

1. Are you still using current \LaTeX? Maintenance is only available for sufficiently up-to-date versions of \LaTeX — if your \LaTeX is more than two versions out of date, the bug reporting mechanisms will reject your report.

2. Has your bug already been reported? Browse the \LaTeX bugs database, to find any earlier instance of your bug. In many cases, the database will list a work-around.

3. Prepare a “minimum” file that exhibits the problem. Ideally, such a file should contain no contributed packages — the \LaTeX team as a whole takes no responsibility for such packages (if they’re supported at all, they’re supported by their authors). The “minimum” file should be self-sufficient: if a member of the team should run it in a clean directory, on a system with no contributed packages, it should replicate your problem.

4. Run your file through \LaTeX: the bug system needs the .log file that this process creates.

You now have two possible ways to proceed: either create a mail report to send to the bug processing mechanism (5, below), or submit your bug report via the web (7, below).

5. Process the bug-report creation file, using \LaTeX itself:

    \latex \latexbug
latexbug asks you some questions, and then lets you describe the bug you’ve found. It produces an output file `latexbug.msg`, which includes the details you’ve supplied, your “minimum” example file, and the log file you got after running the example. (I always need to edit the result before submitting it: typing text into latexbug isn’t much fun.)

6. Mail the resulting file to latex-bugs@latex-project.org; the subject line of your email should be the same as the bug title you gave to latexbug. The file `latexbug.msg` should be included into your message in-line: attachments are likely to be rejected by the bug processor.

7. Connect to the latex bugs processing web page and enter details of your bug — category, summary and full description, and the two important files (source and log file); note that members of the LATEX team need your name and email address, as they may need to discuss the bug with you, or to advise you of a work-around.

What to do if you find a bug

For a start, make entirely sure you have found a bug. Double-check with books about \TeX, \LaTeX, or whatever you’re using; compare what you’re seeing against the other answers above; ask every possible person you know who has any \TeX-related expertise. The reasons for all this caution are various.

If you’ve found a bug in \TeX itself, you’re a rare animal indeed. Don Knuth is so sure of the quality of his code that he offers real money prizes to finders of bugs; the cheques he writes are such rare items that they are seldom cashed. If you think you have found a genuine fault in \TeX itself (or METAFONT, or the CM fonts, or the \TeXbook), don’t immediately write to Knuth, however. He only looks at bugs once or twice a year, and even then only after they are agreed as bugs by a small vetting team. In the first instance, contact Barbara Beeton at the AMS (bnb@math.ams.org), or contact TUG (see question 21).

If you’ve found a bug in \LaTeX{} 2\epsilon{}, you may report it (see question 259) using mechanisms supplied in the \LaTeX{} distribution.

If you’ve found a bug in \LaTeX{} 2.09, or some other such unsupported software, there’s not a lot you can do about it. You may find help or de facto support on a newsgroup such as comp.text.tex or on a mailing list such as texhax@tex.ac.uk, but posting non-bugs to any of these forums can lay you open to ridicule! Otherwise you need to go out and find yourself a willing \TeX-consultant — TUG maintains a register of \TeX{} consultants (see http://www.tug.org/consultants.html).