The UK TeX FAQ
Your 248 Questions Answered
version 3.0.1j, date 2002/07/21

Maintained by Robin Fairbains

July 22, 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

This FAQ was originated by the Committee of the UK 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 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, Daniel Luecking, Sanjoy Mahajan, Andreas Matthias, Ted Nieland, Pat Rau, Heiko Oberdiek, Piet van Oostrum, Scott Pakin, Oren Patashnik, José Carlos Santos, Walter Schmidt, Joachim Schrod, Ulrik Vieth, Mike Vulis, Peter Wilson, Rick Zaccone and Reinhard Zierke.

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 stop[1] from the end of any sentence whose last item is a path name (note that such sentences 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 (see `http://sunburn.stanford.edu/~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", but the idea proved popular and he produced a second version (in 1982) 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 Greek letter Chi (χ), 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.

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 (LA)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 PDFLATEX[2] documents (see question 241).

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.

[1] 'Full stop' (British English)=='period' (American English)
[2] PDFLATEX cannot normally handle PostScript inclusions

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: As Knuth delivers the sources, a 2-digit year is 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’s distributed source does not designate the code, which generates this 2-digit date, as a valid area where implementations may differ. However, he announced in 1998 that implementators can 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 `\year` primitive: Certification of a TeX implementation (see question 5) does not require that `\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* (see question 22) defines `\year` as “the current year of our Lord”, which is the only correct meaning for `\year` for those implementations which can supply a meaningful value, which is to say nearly all of them.

In short, TeX implementations should provide a value in `\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 `\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.

Macros: TeX macros can in principle perform calculations on the basis of the value of `\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 LaTeX are known to be Y2K compliant.

Other macros and macro packages should be individually checked.

External software: Software such as DVI translators needs to be individually checked.

7 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 What is LaTeX 2_ε?

Lamport’s last version of LaTeX (LaTeX 2.09, last updated in 1992) was superseded in 1994 by a new version (LaTeX 2_ε) written by the LaTeX team (see question 238). LaTeX 2_ε 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_ε 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_ε 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_ε 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 `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 How should I pronounce “LaTeX(2_ε)”?

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).

The ‘epsilon’ in ‘LaTeX 2_ε’ 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 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 (La)TeX 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 (see `http://www.ntg.nl/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 PDFTeX (see question 241). The default installation employs a version of TeX built with both the PDFTeX and ε-TeX (see question 240) 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. It is described in "*The Joy of TeX*" by Michael D. Spivak (second edition, AMS, 1990, ISBN 0-821-82997-1). 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.

Eplain distribution: `macros/eplain`

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 `.texi` or `.texinfo` extension. You can write and format Texinfo files into Info files within GNU *emacs*, and read them using the *emacs* Info reader. You can also format Texinfo files into Info files using *makeinfo* and read them using *info*, so you're not dependent on *emacs*. The distribution includes a *Perl* script, *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 π (at the time of writing, Knuth's latest release is version 3.14159). Knuth wants TeX to be frozen at version π 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 238) 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 239 and 240.

19 Reading (La)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 anciliary 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 VorTeX 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. VorTeX 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 244) are in progress.

Nevertheless, The TeX world has taken a long time to latch onto the idea of WYSIWYG. 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 ConTeXt 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 needs 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 *TUGboat* continues in publication (4 issues a year) with articles about TeX, METAFONT and related technologies, and about document design, processing and production. TUG holds a yearly conference, whose proceedings are published in *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 *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 TeXnical 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: see `http://www.tug.org/twg.html`).

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: see `http://www.tug.org/lugs.html`).

Contact TUG itself via:

> 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: `http://www.tug.org/`

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:

The TeXbook by Donald Knuth (Addison-Wesley, 1984, ISBN 0-201-13447-0, paperback ISBN 0-201-13448-9)

A Beginner's Book of TeX by Raymond Seroul and Silvio Levy, (Springer Verlag, 1992, ISBN 0-387-97562-4)

TeX by Example: A Beginner's Guide by Arvind Borde (Academic Press, 1992, ISBN 0-12-117650-9 — now out of print)

Introduction to TeX by Norbert Schwarz (Addison-Wesley, 1989, ISBN 0-201-51141-X — now out of print)

A Plain TeX Primer by Malcolm Clark (Oxford University Press, 1993, ISBNs 0-198-53724-7 (hardback) and 0-198-53784-0 (paperback))

TeX by Topic by Victor Eijkhout (Addison-Wesley, 1992, ISBN 0-201-56882-9 — now out of print, but see question 27)

TeX for the Beginner by Wynter Snow (Addison-Wesley, 1992, ISBN 0-201-54799-6)

TeX for the Impatient by Paul W. Abrahams, Karl Berry and Kathryn A. Hargreaves (Addison-Wesley, 1990, ISBN 0-201-51375-7)

TeX in Practice by Stephan von Bechtolsheim (Springer Verlag, 1993, 4 volumes, ISBN 3-540-97296-X for the set, or Vol. 1: ISBN 0-387-97595-0, Vol. 2: ISBN 0-387-97596-9, Vol. 3: ISBN 0-387-97597-7, and Vol. 4: ISBN 0-387-97598-5)

TeX: Starting from $\boxed{1}$[3] by Michael Doob (Springer Verlag, 1993, ISBN 3-540-56441-1 — now out of print)

The Advanced TeXbook by David Salomon (Springer Verlag, 1995, ISBN 0-387-94556-3)

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

Digital Typography by Donald Knuth (CSLI and Cambridge University Press, 1999, ISBN 1-57586-011-2, paperback ISBN 1-57586-010-4).

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:

Computers & Typesetting, Volumes A–E Boxed Set by Donald Knuth (Addison-Wesley, 2001, ISBN 0-201-73416-8).

For LaTeX, see:

LaTeX, a Document Preparation System by Leslie Lamport (second edition, Addison Wesley, 1994, ISBN 0-201-52983-1)

A guide to LaTeX Helmut Kopka and Patrick W. Daly (third edition, Addison-Wesley, 1998, ISBN 0-201-39825-7)

The LaTeX Companion by Michel Goossens, Frank Mittelbach, and Alexander Samarin (Addison-Wesley, 1993, ISBN 0-201-54199-8)

The LaTeX Graphics Companion: Illustrating documents with TeX and PostScript by Michel Goossens, Sebastian Rahtz and Frank Mittelbach (Addison-Wesley, 1997, ISBN 0-201-85469-4)

The LaTeX Web Companion Integrating TeX, HTML and XML by Michel Goossens and Sebastian Rahtz (Addison-Wesley, 1999, ISBN 0-201-43311-7)

TeX Unbound: LaTeX and TeX strategies for fonts, graphics, and more by Alan Hoenig (Oxford University Press, 1998, ISBN 0-19-509685-1 hardback, ISBN 0-19-509686-X paperback)

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: `http://www.springer-ny.com/catalog/np/jan99np/0-387-98708-8.html`

[3] That's 'Starting from Square One'

First Steps in LaTeX by George Grätzer (Birkhäuser, 1999, ISBN 0-8176-4132-7)

LaTeX: Line by Line: Tips and Techniques for Document Processing by Antoni Diller (second edition, John Wiley & Sons, 1999, ISBN 0-471-97918-X)

LaTeX for Linux: A Vade Mecum by Bernice Sacks Lipkin (Springer-Verlag, 1999, ISBN 0-387-98708-8, second printing)

A sample of George Grätzer's book, in Adobe Acrobat format, is also available (`info/mil/mil.pdf`).

Example files for the LaTeX Graphics and Web Companions are available in `info/lgc` (Graphics) and `info/lwc` (Web). Example files for George Grätzer's 'First Steps' are available in `info/FirstSteps`

The list for METAFONT is rather short:

The METAFONTbook by Donald Knuth (Addison Wesley, 1986, ISBN 0-201-13445-4, ISBN 0-201-52983-1 paperback)

Alan Hoenig's '*TeX Unbound*' includes some discussion and examples of using METAFONT.

This list only covers books in English: UK TUG cannot hope to maintain a list of books in languages other than our own.

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.

The Elements of Typographic Style by Robert Bringhurst (Hartley & Marks, 1992, ISBN 0-88179-033-8)

Finer Points in the Spacing & Arrangement of Type by Geoffrey Dowding (Hartley & Marks, 1996, ISBN 0-88179-119-9)

The Thames & Hudson Manual of Typography by Ruari McLean (Thames & Hudson, 1980, ISBN 0-500-68022-1)

The Form of the Book by Jan Tschichold (Lund Humphries, 1991, ISBN 0-85331-623-6)

Type & Layout by Colin Wheildon (Strathmore Press, 1995, ISBN 0-9624891-5-8)

The Design of Books by Adrian Wilson (Chronicle Books, 1993, ISBN 0-8118-0304-X)

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.

Alphabets Old & New by Lewis F. Day (Senate, 1995, ISBN 1-85958-160-9)

An Introduction to the History of Printing Types by Geoffrey Dowding (British Library, 1998, UK ISBN 0-7123-4563-9; USA ISBN 1-884718-44-2)

The Alphabet Abecedarium by Richar A. Firmage (David R. Godine, 1993, ISBN 0-87923-998-0)

The Alphabet and Elements of Lettering by Frederick Goudy (Dover, 1963, ISBN 0-486-20792-7)

Anatomy of a Typeface by Alexander Lawson (David R. Godine, 1990, ISBN 0-87923-338-8)

A Tally of Types by Stanley Morison (David R. Godine, 1999, ISBN 1-56792-004-7)

Counterpunch by Fred Smeijers (Hyphen, 1996, ISBN 0-907259-06-5)

Treasury of Alphabets and Lettering by Jan Tschichold (W. W. Norton, 1992, ISBN 0-393-70197-2)

A Short History of the Printed Word by Warren Chappell and Robert Bringhurst (Hartley & Marks, 1999, ISBN 0-88179-154-7)

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, at URL `http://www.tex.ac.uk/faq`, and its sources are available from CTAN.

Another excellent information source, available in English, is the (La)TeX navigator (see `http://tex.loria.fr`).

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:

German Posted regularly to `de.comp.tex`, and archived on CTAN; the FAQ also appears at `http://www.dante.de/faq/de-tex-faq/`

French Posted regularly to `fr.comp.text.tex`, and archived on CTAN.

Spanish See `http://apolo.us.es/CervanTeX/FAQ/`

Czech See `http://www.fi.muni.cz/cstug/csfaq/`

Dante FAQ: `usergrps/dante/de-tex-faq`

French FAQ: `help/LaTeX-FAQ-francaise`

Sources of this FAQ: `help/uk-tex-faq`

Obsolete `comp.text.tex` FAQ: `obsolete/help`

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 (La)TeX 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 (La)TeX 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@urz.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 (see `http://www.tuxedo.org/~esr/faqs/smart-questions.html`). 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 (La)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 *Foobar* operating system. Yes, TeX users need printers, but no, TeX users will typically *not* be *Foobar* systems managers.
   Similarly, avoid posing a question in a language that the majority of the group don't use: post in Ruritanian to `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 *mumbleTeX* distribution on the *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 *x* in Plain TeX", or "I want to do *y* in LaTeX running the *boggle* class".
4. If something's going wrong, pretend you're submitting a LaTeX bug report (see question 247), and try to generate a minimum failing example. If your example needs your local *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 *why* you're doing something, just *what* you're doing and where the problem is.

27 (La)TeX 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.

More dynamic is Tobias Oetiker's '(Not so) Short Introduction to LaTeX 2_{ε}', 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' (see `http://www.silmaril.ie/downloads/documents/beginlatex.pdf`) 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 `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 (see `http://www.tug.org/utilities/plain/cseq.html`).

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 (LA)TEX installation, of which a sample is the LATEX documentation available at http://www.tac.dk/cgi-bin/info2www?(latex)

More ambitiously, some departments have enthusiastic documenters who make public record of their (LA)TEX support. For example, Tim Love (of Cambridge University Engineering Department) maintains his deparment's excellent pages 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 http://www.giss.nasa.gov/latex/ltx-2.html

People have long argued for (LA)TEX 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 \(re)newcommand and \(re)newenvironment commands, put them in a file 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 2$_\varepsilon$ 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 source2e.tex in the LATEX distribution.

Writing good classes is not easy; it's a good idea to read some established ones (classes.dtx, for example, produces the standard classes other than *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 \LoadClass — an example of this technique may be seen in *ltxguide.cls*

classes.dtx: macros/latex/base/classes.dtx

ltxguide.cls: macros/latex/base/ltxguide.cls

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 *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 *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 PDFLATEX documents, with an emphasis on how to use appropriate fonts for any text or mathematics.

Hans Hagen (of ConTEXt fame) offers a MetaPost tutorial called MetaFun (which admittedly concentrates on the use of MetaPost with in ConTEXt). It may be found on his company's MetaPost page (see http://www.ntg.nl/context/metapost.htm).

Beginners' guide: info/metafont-for-beginners.tex

Peter Wilson's "experiences": info/metafp.ps
(PostScript) or info/metafp.pdf (PDF format)

30 BIBTEX Documentation

BIBTEX, 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 "BIBTEXing", contained in the file btxdoc.tex, gives a more complete description. *The LATEX Companion* (see question 22) also has information on BIBTEX and writing BIBTEX style files.

The document "Designing BIBTEX Styles", contained in the file btxhak.tex, explains the postfix stack-based language used to write BIBTEX styles (.bst files). The file 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]

BIB*TEX documentation*: biblio/bibtex/distribs/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 LATEX 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 181)
- Fonts for the number sets (question 180)
- Typesetting the principal value integral (question 183)

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

[4] Or submitted for inclusion in a museum dedicated to the history of computing: the stricture was conceivably reasonable in the first version of this FAQ...

32 The PICTEX manual

PICTEX is a set of macros by Michael Wichura for drawing diagrams and pictures. The macros are freely available; however, the PICTEX 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.

pictex: `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 LATEX, 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 82.

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 84) 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 84. *qdtexvpl* is another utility for creating ad-hoc virtual fonts.

fontinst: `fonts/utilities/fontinst`

Knuth on virtual fonts: `info/virtual-fonts.knuth`

qdtexvpl: `fonts/utilities/qdtexvpl`

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 2_ε has standard graphics and colour packages that make file inclusion, rotation, scaling and colour 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_ε, 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_ε *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). There are no programs yet to assist in composition.

`.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.

Details of the macros available for use in `.dtx` files, and the use of `.ins` files, are covered in a tutorial "dtxtut". (Its CTAN directory also contains skeleton `.dtx` and `.ins` files.)

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 "é" 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 the use of `\accent` commands in the output stream (and hence improving the quality of the hyphenation of text in inflected languages, which is interrupted by the `\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 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 "é" 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, even 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 `\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 reinstallation (see question 175) is necessary.

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

The ordinary LaTeX user need not worry about `\language`, since it is very thoroughly managed by the *babel* package; use of `\hyphenation` is discussed in question 171.

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, Polish and Sami, for example, but the Cork fonts have them. Even Cork's coverage isn't complete: it misses letters from Romanian and Welsh, at least.) 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 174).

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 the unavailability of free Adobe type 1 versions, so that PDF produced using them is inevitably bad (see question 75). (Note, however, that several commercial suppliers offer EC or EC-equivalent fonts in type 1 or TrueType form — see question 55.) 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 LaTeX 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 82), for PostScript fonts, and also by the *txfonts* and *pxfonts* font packages (see question 86).

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 (La)TeX packages for including PostScript are structured to use Encapsulated PostScript; which of course leads to much hilarity as exasperated (La)TeX 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 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 can 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`

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: see `http://www.debian.org/social_contract#guidelines`).

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 — see `http://www.tex.ac.uk/tex-archive/help/Catalogue/licenses.html`

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 (`ctan@dante.de`): 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'.

48 Finding (La)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`

`companion.ctan`: `info/companion.ctan`

49 Finding files in the CTAN archives

To find software at a CTAN site, you can use anonymous `ftp` to the host with the command '`quote site index <term>`', or the searching script at `http://www.dante.de/cgi-bin/ctan-index`

To get the best use out of the `ftp` facility you should remember that `<term>` is a *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 `dtx` rather than as package files with extension `sty`.)

One should make the regular expresion general enough to find the file you are looking for, but not too general, as the `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:

```
quote site index caption.sty
```

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

```
quote site index doc.sty
```

matches such unwanted files as `language/swedish/slatex/doc2sty/makefile`

Of course if you *know* the package is stored as `.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:

```
quote site index /caption\\.
quote site index /doc\\.
```

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 `dvips` driver for MS-DOS. You might use the command:

```
quote site index dvips
```

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.

```
quote site index msdos.*dvips
```

Which just returns relevant lines such as `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 86), 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 `http://www.tug.org/texlive.html`
- The Dutch TeX Users Group (NTG) publish the whole 4AllTeX 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 (see `http://www.ntg/nl/`) 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 (La)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 `unixtex.ftp`, though nowadays the sensible installer will take (and possibly customise) one of the packaged distributions such as teTeX (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 teTeX, you need at most one each of the `.tar.gz` files for `teTeX-src`, `teTeX-texmf` and `teTeX-texmfsrc`

Sets of binaries for many common Unix systems are to be found as part of the teTeX distribution, or on the 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, teTeX 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".

tetex: Browse `systems/unix/teTeX/1.0/distrib/sources`

tetex binaries: Browse `systems/unix/teTeX/1.0/distrib/binaries`

unixtex.ftp: `systems/unix/unixtex.ftp`

web2c: `systems/web2c`

Linux There are at least two respectable implementations of TeX to run on Linux, NTeX and teTeX (see above).

Beware the Slackware '96 CD-ROM distribution of NTeX: 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 teTeX, as do most Linux distributions. The most recent offering is a free version of the commercial VTeX (see question 55), which specialises in direct production of PDF from (LA)TeX input.

ntex: `systems/unix/ntex`

tetex: Browse `systems/unix/teTeX/1.0/distrib/sources`

tetex binaries: Browse `systems/unix/teTeX/1.0/distrib/binaries`

vtex: `systems/vtex/linux` (needs `systems/vtex/common`)

PC; MS-DOS or OS/2 EmTeX, by Eberhard Mattes, includes LaTeX, BibTeX, 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 emTeX with 386 (and better) processors and under Windows, are included in the distribution. EmTeX will operate under Windows, but Windows users are better advised to use a distribution tailored for the Windows environment.

A version of emTeX, packaged to use a TDS directory structure (see question 43), is separately available as an emTeX 'contribution'.

emtex: `systems/msdos/emtex`

emtexTDS: `systems/os2/emtex-contrib/emtexTDS`

PC; MS-DOS The most recent offering is an MS-DOS port of the Web2C 7.0 implementation, using the GNU *djgpp* compiler.

djgpp: `systems/msdos/djgpp`

PC; OS/2 OS/2 may also use a free version of the commercial VTeX (see question 55), which specialises in direct production of PDF from (LA)TeX input.

vtex: `systems/vtex/linux` (needs `systems/vtex/common`)

PC: Win32 fpTeX, by Fabrice Popineau, is a version of teTeX 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 (fpTeX can use the same `texmf` tree as a teTeX installation). fpTeX's previewer (*Windvi*) is based on *xdvi*, and takes advantage of extra facilities in the Win32 environment. *Windvi* is capable of printing directly, and a version of *dvips* is also available.

MikTeX, by Christian Schenk, is also a comprehensive distribution, developed separately from the teTeX work. It has its own previewer, YAP, which is itself capable of printing, though the distribution also includes a port of *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 "CygWin" bundle (see `http://www.cygwin.com`), 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 teTeX, too (you will need the X-server, to run *xdvi*). Of course, teTeX 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 fpTeX. TeTeX 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.

bakoma: `nonfree/systems/win32/bakoma`

fptex: `systems/win32/fptex`

miktex: Acquire `systems/win32/miktex/setup/setup.exe` and read `systems/win32/miktex/setup/install.html`

tetex: `systems/unix/teTeX/1.0/distrib/sources`

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.

miktex for AXP: `systems/win32/miktex-AXP`

miktex for Power PC: `systems/win32/miktexppc`

Mac OzTeX, by Andrew Trevorrow, is a shareware version of TeX for the Macintosh. A DVI previewer and PostScript driver are also included.

UK TUG prepays the shareware fee, so that its members may acquire the software without further payment. Questions about OzTeX may be directed to `oztex@midway.uchicago.edu`

Another partly shareware program is CMacTeX, put together by Tom Kiffe. This is much closer to the Unix TeX setup (it uses *dvips*, for instance). CMacTeX includes a port of the latest version of Omega (see question 239).

Both OzTeX and CMacTeX are additionally available on MacOS X (see `http://www.esm.psu.edu/mac-tex/versionsX.html`), but OS X users also have the option of a build of teTeX 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) TeXShop, which is an integrated TeX editor and previewer.

A useful resource for Mac users is to be found at `http://www.esm.psu.edu/mac-tex/`; it has a news and 'help' section, as well as details of systems and tools. A useful resource for Mac users (see `http://www.esm.psu.edu/mac-tex/`) has a news and 'help' section, as well as details of systems and tools.

cmactex: `systems/mac/cmactex`

oztex: `nonfree/systems/mac/oztex`

MacOS X teTeX: `ftp://ftp.nluug.nl/pub/comp/macosx/tex-gs/`

TeXShop: `http://darkwing.uoregon.edu/~koch/texshop/texshop.html`

OpenVMS TeX for OpenVMS is available.

Standard tape distribution is through DECUS (see question 45).

OpenVMS: `systems/OpenVMS/TEX97_CTAN.ZIP`

Atari TeX is available for the Atari ST.

If anonymous `ftp` is not available to you, send a message containing the line '`help`' to `atari@atari.archive.umich.edu`

Atari TeX: `systems/atari`

Amiga Full implementations of TeX 3.1 (PasTeX) and METAFONT 2.7 are available.

You can also order a CD-ROM containing this and other Amiga software from Walnut Creek CDROM, telephone +1 510-947-5997.

PasTeX: `systems/amiga`

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 `ftp` from `ftp.math.utah.edu` in `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:

Unix Try GNU *emacs* or *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 (see `http://nedit.org/`) 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:

- TeXshell is a simple, easily-customisable environment, which can be used with the editor of your choice.
- Eddi4TeX (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 4AllTeX CD-ROM (see question 52) contains another powerful Windows-based shell.

OS/2 Eddi4TeX 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 OzTeX.

Atari, Amiga and NeXT users also have nice environments. LaTeX users looking for *make*-like facilities should try *latexmk*.

There is another set of shell programs to help you manipulate BibTeX databases.

alpha: `systems/mac/support/alpha`

auctex: `support/auctex`

epmtex: `systems/os2/epmtex`

latexmk: `support/latexmk`

nedit latex support: `support/NEdit-LaTeX-Extensions`

TeXshell: `systems/msdos/texshell`

TeXtelmExtel: `systems/msdos/emtex-contrib/TeXtelmExtel`

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 TeXbook!) 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; TrueTeX 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/`

Source: Mail from Richard Kinch, October 2001.

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`
Web: `http://www.YandY.com/`

Source: Mail from Y&Y, July 2001

pcTeX Long-established: pcTeX32 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/`

Source: Mail from Personal TeX Inc, September 1997

PC; VTeX 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`
Web: `http://www.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 TrueTeX 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`

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`

Source: Mail from Christopher Mabb, May 1999

Macintosh; Textures "A TeX system 'for the rest of us' "; also gives away a METAFONT 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/`

Source: *TUGboat* **15**(1) (1994)

AmigaTeX 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.

VMS versions are available through the DECUS library (see question 45), and also as part of the CTAN distribution of TeX for VMS.

A precompiled version to work with emTeX is available on CTAN.

Macintosh users can use either the excellent drivers built into OzTeX or Textures, or a port of *dvips* in the CMacTeX package.

MS-DOS and OS/2: `systems/msdos/dviware/dvips`

VMS distribution: `systems/OpenVMS/TEX97_CTAN.ZIP`

57 DVI drivers for HP LaserJet

The emTEX 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 *dviljk* uses the same path-searching library as *web2c*. *dviljk* is no longer distributed as a separate source, but the teTEX distribution holds a copy under the name *dvilj*.

Beebe drivers: `dviware/beebe`

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`

59 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 Macintoshes.

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 83), and
- generating PDF output, and viewing that with *Acrobat Reader* or one of the substitutes for that.

xdvi: `dviware/xdvi`

xdvik: `dviware/xdvik`

H Support Packages for TEX

60 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* (see `http://tech-www.informatik.uni-hamburg.de/applets/javafig/`) is thought by many to be an acceptable substitute, written in Java.

fig2mf: `graphics/fig2mf`

xfig: `graphics/xfig`

transfig: `graphics/transfig`

61 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 `\special`s (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`

62 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 source which lacks many useful facilities 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 (LA)TEX 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 (LA)TEX 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 TEX 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 TEX.

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

TEX 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 TEX, 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 TEX 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 (LA)TEX 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*.
The DECUS TEX distribution (see question 45) also contains a program which converts *troff* to TEX.

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 Fojtik.

PC-Write *pcwritex.arc* is a print driver for PC-Write that "prints" a PC-Write V2.71 document to a TEX-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 BIBTEX and *refer/tib* formats. The collection includes a shell script converter from BIBTEX to *refer* format as well. The collection is not maintained.

RTF *Rtf2tex*, by Robert Lupton, is for converting Microsoft's Rich Text Format to TeX. There is also a convertor 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 TeX2RTF, which can produce ordinary RTF, Windows Help RTF (as well as HTML, see question 69). TeX2RTF 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.

excel2latex: `support/excel2latex/xl2latex.zip`

pcwritex.arc: `support/pcwritex`

refer and tib tools: `biblio/bibtex/utils/refer-tools`

rnototex: `support/rnototex`

rtf2latex: `support/rtf2latex`

rtf2latex2e: `support/rtf2latex2e`

rtf2tex: `support/rtf2tex`

tex2rtf: `support/tex2rtf`

tr2latex: `support/tr2latex`

troff-to-latex: `support/troff-to-latex`

wd2latex: `dviware/wd2latex`

wp2latex: `support/wp2latex`

Word processor FAQ: `http://www.tug.org/utilities/texconv/index.html`

Word processor FAQ (source): `help/wp-conv/wp-conv.zip`

67 Conversion from (La)TeX to plain ASCII

The aim here is to emulate the Unix *nroff*, which formats text as best it can for the screen, from the same input as the Unix typesetting program *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:

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

Ralph Droms provides a *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 *screen* package.

Another possibility is to use the LaTeX-to-ASCII conversion program, *l2a*, although this is really more of a de-TeXing program.

The canonical de-TeXing program is *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.

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 *lynx* to dump the resulting HTML as plain text.

catdvi: `dviware/catdvi`

crudetype: `dviware/crudetype`

detex: `support/detex`

dvi2tty: `nonfree/dviware/dvi2tty`

l2a: `support/l2a`

screen.sty: `macros/latex209/contrib/misc/screen.sty`

tex2mail: `support/tex2mail`

txt: `support/txt`

68 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, which is available from `http://www.perl.com/CPAN/modules/by-module/SGMLS`
   - Joachim Schrod and Christine Detig's *stil*, written in Common Lisp, which is available from `ftp://ftp.th-darmstadt.de/pub/text/sgml/stil`

   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 LaTeX 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).

69 (La)TeX 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 242.

For today, possible packages are:

LaTeX2HTML a *perl* script package that supports LaTeX 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 *TUGboat* **16**(2).

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 *flex*. The distribution consists of a single C source (or a compiled executable), which is easy to install and very fast-running.

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 (La)TeX 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.

TeXpider a commercial program from MicroPress (see question 55), which is described on `http://www.micropress-inc.com/webb/wbstart.htm`; it uses bitmaps for equations.

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

The World Wide Web Consortium maintains a list of "filters" to HTML, with sections on (La)TeX and BibTeX — see `http://www.w3.org/Tools/Word_proc_filters.html`

latex2html: Browse `support/latex2html`

tex4ht: `support/TeX4ht/tex4ht.zip`

tth: `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 *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 PassiveTeX uses *xmltex* to achieve this end.

xmltex: `macros/xmltex/base`

passivetex: `macros/xmltex/contrib/passivetex`

71 Retrieving (La)TeX 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 (La)TeX 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 retype 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 (La)TeX 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 (La)TeX to ASCII (see question 67) are applicable. Consider *dvi2tty*, *crudetype* and *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 *ps2ascii* tool (part of the *ghostscript* distribution) is available. One could try applying this tool to PostScript derived from an PDF file using *pdf2ps* (also from the *ghostscript* distribution), or *Acrobat Reader* itself; an alternative is *pdftotext*, which is distributed with *xpdf*.

Another avenue available to those with a PDF file they want to process is offered by Adobe *Acrobat* (version 5 or later): you can tag the PDF file into an estructured document, output thence to well-formed XHTML, and import the results into Microsoft *Word* (2000 or later). From there, one may convert to (La)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 (La)TeX source, probably on the World Wide Web, there are four technologies to consider:

1. Direct (La)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 hyperTeX conventions (standardised \special commands); there are supporting macro packages for Plain TeX and LaTeX).

The HyperTeX 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 HyperTeX guidelines; it provides general hypertext links, including those to external documents.

The HyperTeX specification says that conformant viewers/translators must recognize the following set of \special commands:

href: html:
name: html:
end: html:
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 PDFTeX (see question 241) or MicroPress's VTeX (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 PDFTeX or VTeX. PDFTeX has the advantage of availability for a wide range of platforms, VTeX (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 LaTeX cross-referencing into Acrobat links, you need a LaTeX package to suitably redefine the internal commands. There are two of these for LaTeX, both capable of conforming to the HyperTeX 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 PDFTeX primitives, the hyperTeX \specials, or DVI driver-specific \special commands. Both *dvips* and Y&Y's *DVIPSONE* translate the DVI with these \special commands into PostScript acceptable to Distiller, and *dvipdfm* has \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.

Acrobat Reader: browse `ftp://ftp.adobe.com/pub/adobe/acrobatreader`

dvipdfm: `dviware/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 required; however, most ordinary (LA)TeX documents contain at least odd characters that originate in METAFONT, and it's advisable to tell *dvips* to use type 1 versions regardless. 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 unsatisfactory. 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 174), and also recommend the use of Type 1 fonts to ensure that you get good quality PDF (see question 75). Unfortuately, 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 86.

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: `fonts/ae`

aeguill.sty: `macros/latex/contrib/supported/aeguill`

zefonts: `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).

dvistrip: Download from `http://www.yandy.com/download/dvistrip.exe`

pkfix: `support/pkfix`

L METAFONT and MetaPost

78 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 METAFONTbook* (see question 22). You will need a file, which conventionally called `local.mf`, containing all the `mode_defs` 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
Beginning to dump on file plain...	
(*output*)	

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[5]) 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 METAFONTbook* and *The TeXbook*). If `mag=<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'
```

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`[6] and `cmr10.log`).

[5]The *command_name* is symbolically linked to *virmf*, and *virmf* loads `command_name.base`

[6]Note that the file name may be transmuted by such operating systems as MS-DOS, which don't permit long file names

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, (La)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 teTeX, 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 83).

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 86).

bakoma: `fonts/cm/ps-type1/bakoma`

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

81 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 TeX 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 eperience 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 in inclusions into (La)TeX documents, so it's best to make the `prologues` command optional. Furthermore, MetaPost takes a very simple-minded approach to font encoding: since TeX 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`:

```
prologues := 2;
input MyFigure
```

or by issuing a shell command such as

```
mp '\prologues:=2; input MyFigure'
```

(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:

```
\documentclass{article}
\usepackage{graphicx}
\begin{document}
\thispagestyle{empty}
\includegraphics{MyFigure.1}
\end{document}
```

Processing the resulting DVI file with the *dvips* command

```
dvips -E -o MyFigure.eps MyFigure
```

would then give a satisfactory Encapsulated PostScript file.

M PostScript and TeX

82 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 84. 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 86).

If you use LaTeX 2_ε, 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 LaTeX3 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.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 84) 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 LaTeX there are various schemes, of which the simplest to use is probably the PSLaTeX 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 LaTeX's NFSS, and so is easy to use with PostScript fonts.

Some common problems encountered are discussed elsewhere (see question 85).

Laserwriter set of 35 fonts: `macros/latex/required/psnfss/lw35nfss.zip`

lollipop: `macros/lollipop`

psfonts: Browse `fonts/psfonts`

psnfss: `macros/latex/required/psnfss`

83 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 DVIWindo (see question 55) has long used *Adobe Type Manager* to display Type 1 fonts directl, 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 tghe 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 previewer such as *ghostscript* or *ghostscript*-based viewers such as *ghostview* or shareware offering *gsview*.
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).

ghostscript: Browse `nonfree/support/ghostscript`

ghostview: Browse `support/ghostscript/gnu/ghostview`

gsftopk: `fonts/utilities/gsftopk`

gsview: Browse `nonfree/support/ghostscript/gsview`

ps2pk: `fonts/utilities/ps2pk`

xdvi: `dviware/xdvi`

84 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_ε 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_ε'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_ε 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 *afmtotfm* program among many other goodies.

dvips: `dviware/dvips`

EdMetrics: `systems/mac/textures/utilities/EdMetrics.sea.hqx`

fontinst: `fonts/utilities/fontinst`

LY1 support: macros/latex/contrib/supported/psnfssx/ly1

PS font metrics: Browse fonts/psfonts (this directory is at the root of a really rather large tree), or http://home.vr-web.de/was/fonts.html, which contains metrics for "fonts that didn't make it on to the CTAN directories".

85 Deploying Type 1 fonts

For the LaTeX user trying to use the PSNFSS (see question 82) 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 psfonts.map file, so that *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 83.

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.

86 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 TeX 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 *afm2tfm* (possibly in combination with *vptovf*), *afmtotfm* (from Y&Y, see question 55) or *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 *TUGboat* **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.

Computer Modern (75 fonts — optical scaling) Donald E. Knuth
The CM fonts were originally designed in 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 *de facto* standard for Type 1 versions of the fonts.

AMS fonts (52 fonts, optical scaling) The AMS
This set of fonts offers adjuncts to the CM set, including two sets of symbol fonts (msam and 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 *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).

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.

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

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, math 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.

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, 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/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, LaTeX 2.09 and current LaTeX 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 `msam` and `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 LaTeX 2.09. Support under LaTeX 2_ε is provided in PSNFSS (see question 82) 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 2_ε is provided in PSNFSS (see question 82) 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 METAFONT might be used in the future: they were published simultaneously as METAFONT 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 METAFONT 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 2_ε is available in PSNFSS (see question 82); 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 `pxfonts` 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 (`cmsy`, `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 PDFTeX support files are included. The documentation (`http://www.tex.ac.uk/tex-archive/fonts/pxfonts/doc/pxfontsdocA4.pdf`) 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 (`cmsy`, `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 PDFTeX support files are included. The documentation (`http://www.tex.ac.uk/tex-archive/fonts/txfonts/doc/txfontsdocA4.pdf`) 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 and mathpple 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 and mathpple fonts are free, and the resulting PostScript files can be freely exchanged. Support under LaTeX 2_ε is available in PSNFSS (see question 82).

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 TrueTeX (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 `fonts/cm/ps-type1/bluesky`

AMS font collection: Browse `fonts/amsfonts/ps-type1`

Belleek fonts: `fonts/belleek/belleek.zip`

eulervm.sty and supporting metrics: `fonts/eulervm`

hvmath (free bitmapped version): `fonts/micropress/hvmath`

pxfonts: `fonts/pxfonts`

tmmath (free bitmapped version): `fonts/micropress/tmmath`

txfonts: `fonts/txfonts`

87 Including a PostScript figure in (LA)TEX

LATEX has a standard package for graphics inclusion, rotation, colour, and other driver-related features. The package is documented in the second edition of Lamport's LATEX book, as well as in the *LATEX Graphics Companion* (see question 22). The LATEX package is also available in a form suitable for use with Plain TEX.

The distribution itself comes with documentation, and a processed copy (`grfguide.ps`) is available in the distribution so that users can read documentation without first installing the package.

The *graphics* package comes with a relative, *graphicx*, which provides more convenient means of scaling and otherwise manipulating graphics. The packages are usually configured to the DVI processor you use by means of a `.cfg` file, but configuration by means of package options is also possible. The range of types of graphical file you may include differs according to the system you're using.

The figures themselves do not become part of the DVI file, but are only included when you use a DVI to PostScript conversion program. The `\special` commands (see question 38) used to pass pointers to the graphics files, and other information, are potentially different for every DVI processor (which is why the *graphics* package must be configured for the processor you use).

Since the `\special` commands used are typically not their own, some DVI previewers can't cope with them. More modern ones (notably *xdvi*, MikTEX's *yap* and fpTEX's *windvi*) can pass the figure to a properly configured *ghostscript* installation for rendering into a bitmap for screen viewing.

There are two rather good documents on CTAN addressing figure production, with rather different emphasis. Keith Reckdahl's *epslatex* covers the standard LATEX facilities, as well as some of the supporting packages, notably *subfigure* and *psfrag*. Anil K. Goel's "Figures in LATEX" (*figsinltx*), covers the different ways in which you might generate figures, and the old (LATEX 2.09) ways of including them into documents.

epslatex.pdf: `info/epslatex.pdf`; the document is also available in PostScript format as `info/epslatex.ps`

figsinltx: `info/figsinltx.ps`

ghostscript: Browse `nonfree/support/ghostscript`

graphics: The whole bundle for LATEX (including *graphicx.sty*) is found in `macros/latex/required/graphics`; to use the packages with Plain TEX, you should also acquire:

graphicx.tex: `macros/plain/graphics` which contains various wrappers and emulations of commands for the use of the LATEX version.

psfrag.sty: `macros/latex/contrib/supported/psfrag`

subfigure.sty: `macros/latex/contrib/supported/subfigure`

88 Weird characters in *dvips* output

You've innocently generated output, using *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 `-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 `-G1` switch: if you were using it explicitly, *don't*; if you were using `-Ppdf`, add `-G0` to suppress the implicit switch in the pseudo-printer file.

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

N Bibliographies and citations

89 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 BIBTEX 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 93).

BIBTEX documentation: `biblio/bibtex/distribs/doc`

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

90 Capitalisation in BIBTEX

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

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

```
title = {The {THE} operating system},
```

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

```
title = {Te{X}niques and tips},
```

There's more on the subject in the BIBTEX documentation (see question 30).

91 'String too long' in BIBTEX

The BIBTEX 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 — `biblio/bibtex/contrib/abstract.bst`), and rename it (e.g., on a long file-name system to `abstract-long.bst`). Now edit it: find function `output.nonnull` 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 `output.nonnull` 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:

```
@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.

92 BIBTEX doesn't understand my 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:

```
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:

```
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:

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

93 Citing URLs with BIBTEX

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, available via `http://www.nlc-bnc.ca/iso/tc46sc9/standard/690-2e.htm`; 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 `howpublished` field of the standard styles' `@misc` function. Of course, the strictures about typesetting URLs (see question 135) still apply, so the entry will look like:

```
@misc{...,
  ...,
  howpublished = "\url{http://...}"
}
```

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:

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

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

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

harvard: `macros/latex/contrib/supported/harvard`

hyperref: `macros/latex/contrib/supported/hyperref`

url: `macros/latex/contrib/other/misc/url.sty`

94 Using BIBTEX with Plain TEX

The file `btxmac.tex` contains macros and documentation for using BIBTEX with Plain TEX, either directly or with Eplain (see question 14). See question 30 for more information about BIBTEX itself.

btxmac.tex: `macros/eplain/btxmac.tex`

95 Separate bibliographies per chapter?

A separate bibliography for each 'chapter' of a document can be provided with the package *chapterbib* (which comes with a bunch of other good bibliographic things). The package allows you a different bibliography for each `\included` 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 *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 `bibunit` environment so that users can select their own structuring.

chapterbib.sty: `macros/latex/contrib/supported/cite`

bibunits.sty: `macros/latex/contrib/supported/bibunits`

96 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 95.

For more than one bibliography, there are three options.

The *multibbl* 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 188 — in a *babel*-safe way.

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

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

There is also a command `\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 `\nocite*`).

bibtopic.sty: `macros/latex/contrib/supported/bibtopic`

multibib.sty: `macros/latex/contrib/supported/multibib`

97 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 *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 *inlinebib*, which requires that you use its `inlinebib.bst`
- The package *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: `jurabib.bst`, `jhuman.bst` and two Chicago-like ones.

Options for entries in footnotes are

- The package *footbib*, and
- The package *jurabib*, again.

bibentry.sty: *Distributed with* `macros/latex/contrib/supported/natbib`

footbib.sty: `macros/latex/contrib/supported/footbib`

inlinebib.sty: `biblio/bibtex/contrib/inlinebib`

jurabib.sty: `macros/latex/contrib/supported/jurabib`

98 Sorting and compressing citations

If you give LaTeX `\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 `\cite` command, but who wants to do that sort of thing for no more improvement than "[2,3,4,6]"?

The *cite* package sorts the numbers and detects consecutive sequences, so creating "[2–4,6]". The *natbib* package, with the `numbers` and `sort&compress` options, will do the same when working with its own numeric bibliography styles (`plainnat.bst` and `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`

99 Multiple citations

A convention sometimes used in physics journals is to "collapse" a group of related citations into a single entry in the bibliography. BIBTEX, 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 pakage documentation tells you how to do that.

mcite.sty: `macros/latex/contrib/supported/mcite`

100 Listing all my BIBTEX 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.

101 Making HTML of your Bibliography

A neat solution is offered by the *noTeX* bibliography style. This style produces a `.bbl` 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 (LA)TEX.

A more conventional translator is the *awk* script *bbl2html*, which translates the `.bbl` 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 (LA)TEX files

102 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 MikTEX 2.1 offers a simpler procedure (see question 104) 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 (LA)TEX contributed "`misc`" directories on CTAN; these directories contain collections of single files, which are supposedly complete in themselves.

A small package ⟨*smallpack*⟩ might be just a single `.sty` file (typically `smallpack.sty`) with the usage instructions either included as comments in the file or in a seperate 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 ⟨*pack*⟩, 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 "`No file pack.ind`"; this is the source for the command index; if you want the index, process the raw material with:
  `makeindex -s gind.ist pack`
  and run LATEX again.
- Print and read `pack.dvi`

Sometimes a user manual is supplied seperately 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) *'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 103 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 latex-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.

teTEX, fpTEX Run:
`texhash`

web2c On a current *web2c* distribution, `texhash` ought to work; if it doesn't, run:
`mktexlsr`

MikTEX On a *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 *MikTeX* distribution v2.0 or later, do:
`Start→Programs→MikTeX 2→MikTeX Options`, and press the `Update filename database` button.

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

103 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 teTeX, fpTeX or 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:

```
teTeX:       /usr/share/texmf-local/
fpTeX:       c:\fptex\texmf.local\
mikTeX:      c:\localtexmf\
```

Let's write `$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:

```
.sty, .cls or .fd: $TEXMF/tex/latex/<package>/
.dvi, .ps or .pdf: $TEXMF/doc/latex/<package>/
.mf:   $TEXMF/source/tfm/<supplier>/<font>/
.tfm:  $TEXMF/fonts/tfm/<supplier>/<font>/
.vf:   $TEXMF/fonts/vf/<supplier>/<font>/
.afm:  $TEXMF/fonts/afm/<supplier>/<font>/
.pfb:  $TEXMF/fonts/type1/<supplier>/<font>/
.ttf:  $TEXMF/fonts/truetype/<supplier>/<font>/
```

Where of course ⟨*package*⟩, ⟨*font*⟩ and ⟨*supplier*⟩ depend upon what's appropriate for the individual file.

104 Installing MikTeX "known packages"

MikTeX 2.1 maintains a database of packages it "knows about", together with (coded) installation instructions that enable it to install with with minimal user intervention.

If MikTeX does know about a package you need installed, it's worth using the system.

First, open the MikTeX packages window: click on `Start`→`Programs`→`MikTeX`→`MikTeX 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, "`MikTeX Packages`", and select "`Search`": enter the name of the package you're interested in, and press the "`Search`" button. If MikTeX knows about your package, it will open up the tree to show you a tick box for you package: check that box.

If MikTeX *doesn't* know about the package you're interested in, you have to use the long-winded procedure (see question 102) outlined elsewhere in this FAQ.

If necessary, repeat to select other packages, and then press "`OK`"; MikTeX tells you how many packages you have selected — if you're happy, press "`OK`" again. MikTeX 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.

105 "Temporary" installation of (La)TeX files

Operating systems and applications need to know where to find files: many files that they need are "just named" — the user doesn't necessarily know *where* they are, but knows to ask for them. The commonest 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 TeX 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 102). To do this, we need to change the relevant path as TeX perceives it. However, we don't want to throw away TeX's built-in path (all of a sudden, TeX won't know how to deal with all sorts of things).

To *extend* a TeX path, we define an operating system environment variable in 'path format', but leaving a gap which TeX will fill with its built-in value for the path. The commonest 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:

```
export TEXINPUTS=/tmp:
```

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

```
set TEXINPUTS=C:/temp;
```

In either case, we're asking TeX 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 TeX that are started from the shell where the environment variable was set. If you run TeX 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

106 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 128. 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 æsthetically pleasing document anyway!)

`UC thesis style`: `macros/latex/contrib/supported/ucthesis`

107 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 (LA)TEX, 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.

108 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 thing to a canned solution is Peter Wilson's *combine* class; it 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`s and `\author`s (etc.) of the `\import`ed documents to appear in the global table of contents.

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 PDFLaTeX. 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`

109 *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.

curve.cls: `macros/latex/contrib/supported/curve`

vita.cls: `macros/latex/contrib/other/vita`

110 Letters and the like

LaTeX itself provides a *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:

```
\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 TeXbook), 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.

The largest, most comprehensive, class is *newlfm*; the `lfm` part of the name implies that the class can create letters, faxes and memoranda. The documentation is voluminous, and the package seems very flexible.

Axel Kielhorn's *akletter* class is the only other one, recommended for inclusion in this FAQ, whose documentation is available in English.

The *dinbrief* class, while recommended, is only documented in German.

There are letter classes in each of the excellent *koma-script* (*scrlettr*: documentation is available in English) and *ntgclass* (*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.

akletter.cls: `macros/latex/contrib/supported/akletter`

brief.cls: Part of the `macros/latex/contrib/supported/ntgclass` bundle

dinbrief.cls: `macros/latex/contrib/supported/dinbrief`

newlfm.cls: `macros/latex/contrib/supported/newlfm` *scrlettr.cls*: Part of the `macros/latex/contrib/supported/koma-script` bundle

P.2 Document structure

111 The style of document titles

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

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

titling.sty: `macros/latex/contrib/supported/titling`

112 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 *The LaTeX Companion* (see question 22; the programming itself is discussed in question 192). 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 redefinition. 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, *titletoc*, 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`

113 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`

114 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 `\subsubsubsection`s 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 (`ltsect.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`

115 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 188.) 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.

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

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

116 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 123), by enabling you to define different page styles for initial and for body pages.

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

117 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_ε), 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, *i.e.*, `\evensidemargin`, `\oddsidemargin`, `\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, *i.e.*, before the `\begin{document}` statement. To adjust text width within a document we define an environment:

```
\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 `\begin{changemargin}{-1cm}{-1cm}` narrows the left and right margins by 1cm.

The *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.

chngpage.sty: `macros/latex/contrib/supported/misc/chngpage.sty`

geometry.sty: `macros/latex/contrib/supported/geometry`

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

vmargin.sty: `macros/latex/contrib/supported/vmargin`

118 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, `figure*` provides a floating page-wide figure (and `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 — `h` or `b` float placement directives are simply ignored.

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

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

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

119 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:

```
\documentclass[twocolumn]{article}
...
\begin{document}
... % \author, etc
\twocolumn[
  \begin{@twocolumnfalse}
    \maketitle
    \begin{abstract}
      ...
    \end{abstract}
  \end{@twocolumnfalse}
]
```

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:

```
\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
```

120 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 191 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`

121 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 *twocolumn* 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 191.

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 packages 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

122 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/supported/carlisle/nopageno.sty`

123 `\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/supported/fancyhdr`

nopageno.sty: `macros/latex/contrib/supported/carlisle/nopageno.sty`

124 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/supported/crop`

125 '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 LaTeX 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/supported/draftcopy`

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

126 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 *lscape* package. 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 157, 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 *lscape* 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 *dviwindo* (see question 55), and the fpTeX previewer 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.

geometry.sty: `macros/latex/contrib/supported/geometry`

graphics.sty: Distributed as part of `macros/latex/required/graphics`

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

lscape.sty: Distributed as part of `macros/latex/required/graphics`

rotating.sty: `macros/latex/contrib/supported/rotating`

supertabular.sty: `macros/latex/contrib/supported/supertabular`

127 Putting things at fixed positions on the page

TeX'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 whit the page header. The package *textpos* latches bits of typesetting to locations you specify, by fixing them to the page header, and thereby solves the problem.

textpos.sty: `macros/latex/contrib/supported/textpos`

P.4 Spacing of characters and lines

128 Double-spaced documents in LaTeX

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 (*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 `\baselinestretch` — `\renewcommand{\baselinestretch}{1.2}` may be enough to give officials the impression you've kept to their regulations. Note that `\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 `\baselineskip`: its value is reset at any size-changing command.

For preference, however, use a line-spacing package. The only one currently supported is *setspace* (do *not* be tempted by *doublespace* — its performance under current LaTeX is at best problematical). *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 `\baselinestretch` yourself.

setspace.sty: `macros/latex/contrib/supported/setspace/setspace.sty`

129 Changing the space between letters

A common technique in advertising copy (and other text whose actual content need not actually be *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 letterspace 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 *letterspacing* (which is the name of the `.tex` file; it also appears as the *letterspace* package in some distributions).

The *tracking* package has similar facilities.

A more comprehensive solution is to be found in the *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 *soul*; the other half is described in question 208.)

letterspacing.tex: `macros/generic/letterspacing.tex`

soul.sty: `macros/latex/contrib/supported/soul`

tracking.sty: `macros/latex/contrib/supported/tracking/tracking.sty`

130 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 `\raggedright` command (and the corresponding `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 176) painfully large gaps may sometimes be required.

Martin Schröder's *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., `\RaggedRight`) and environment (e.g., `FlushLeft`) names that are simply capitalised transformations of the LaTeX kernel originals. The documentation discusses the issues and explains the signficance of the various parameters of *ragged2e*'s operation.

ragged2e.sty: Distributed as part of
`macros/latex/contrib/supported/ms`

131 Cancelling `\ragged` commands

LaTeX provides commands `\raggedright` and `\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 192) defines a command that restores flush justification at both margins:

```
\def\flushboth{%
  \let\\\@normalcr
  \@rightskip\z@skip \rightskip\@rightskip
  \leftskip\z@skip
  \parindent 1.5em\relax}
```

There's a problem with the setting of `\parindent` in the code: it's necessary because both the `\ragged` commands set `\parindent` to zero, but the setting isn't a constant of nature: documents using a standard LaTeX class with `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

132 Including a file verbatim in LaTeX

A good way is to use Rainer Schöpf's *verbatim*, which provides a command `\verbatiminput` that takes a file name as argument.

Another way is to use the `alltt` environment, which requires *alltt*. `alltt` interprets its contents 'mostly' verbatim, but executes any TeX commands it finds: so one can say:

```
\begin{alltt}
\input{verb.txt}
\end{alltt}
```

of course, this is little use for inputting (La)TeX source code...

Moreverb extends the facilities of *verbatim* package, providing a `listing` environment and a `\listinginput` command, which line-number the text of the file. The package also has a `\verbatimtabinput` command, that honours TAB characters in the input (the `listing` environment and command also both honour TAB).

The *fancyvrb* 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.

alltt.sty: Part of the LaTeX distribution.

fancyvrb.sty: `macros/latex/contrib/supported/fancyvrb`

moreverb.sty: `macros/latex/contrib/supported/moreverb`

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

133 Including line numbers in typeset output

For general numbering of lines, there are two packages for use with LaTeX, *lineno* (which permits labels attached to individual lines of typeset output) and *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 *fancyvrb* (see question 132) may be used.

One common use of line numbers is in critical editions of texts, and for this the *edmac* package (`macros/plain/contrib/edmac`) offers comprehensive support.

fancyvrb.sty: `macros/latex/contrib/supported/fancyvrb`

lineno.sty: `macros/latex/contrib/supported/lineno`

moreverb.sty: `macros/latex/contrib/supported/moreverb`

numline.sty: `macros/latex/contrib/supported/numline/numline.sty`

134 Generating an index in (La)TeX

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 `\index` commands).

It is not practical to sort a large index within TeX, so a postprocessing program is used to sort the output of one TeX 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).

idxtex for LaTeX under VMS, which comes with a glossary-maker called *glotex*.

texindex A witty little shell/*sed*-script-based utility for LaTeX under Unix.

There are other programs called *texindex*, notably one that comes with the Texinfo distribution (see question 16).

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.

idxtex: `indexing/glo+idxtex`

makeindex: `indexing/makeindex`

makeindex (Macintosh): `systems/mac/macmakeindex2.12.sea.hqx`

texindex: `support/texindex`

texsis (system): `macros/texsis`

texsis (makeindex support): `macros/texsis/index/index.tex`

xindy: `support/xindy`

135 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 `\hbox`es, and their typeset representation is awful.

There are three packages that help solve this problem:

- The *Path* package, which defines a `\path` command. The command defines each potential break character as a `\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 `\path` command.)
- The *url* package, which defines an `\url` command (among others, including its own `\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 193) 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 *url* package in Plain TeX, with the assistance of the *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:

```
\input miniltx
\expandafter\def\expandafter\+%
                  \expandafter{\+}
\input url.sty
```

- The *hyperref* package, which uses the typesetting code of *url*, in a context where the typeset text forms the anchor of a link.

The author of this answer prefers the (rather newer) *url* package (directly or indirectly); both *path* and *url* work well with Plain TeX (though of course, the fancy LaTeX facilities of *url* don't have much place there). (*hyperref* isn't available in a version for use with Plain TeX.)

hyperref.sty: `macros/latex/contrib/supported/hyperref`

miniltx.tex: Distributed as part of `macros/plain/graphics`

path.sty: `macros/latex/contrib/other/misc/path.sty`

url.sty: `macros/latex/contrib/other/misc/url.sty`

136 Typesetting music in TeX

In the early days, a simple music package called *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 MusicTeX, which allows the typesetting of polyphonic and other multiple-stave music; MusicTeX remains available, but is most definitely no longer recommended.

MusicTeX has been superseded by its successor MusiXTeX, 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 TeX-based mechanism that MusicTeX uses. Daniel Taupin's is the only version of MusiXTeX currently being developed (the original author, Andreas Egler, had an alternative version, but he is now working on a different package altogether).

Input to MusixTeX is extremely tricky stuff, and Don Simons' preprocessor *pmx* is the preferred method of creating input for Taupin's version. *Pmx* greatly eases use of MusixTeX, but it doesn't support the full range of MusixTeX's facilities directly; however, it does allow in-line MusixTeX code in *pmx* sources.

Dirk Laurie's *M-Tx* allows preparation of music with lyrics; it operates "on top of" *pmx*

Another simple notation is supported by *abc2mtex*; this is a package designed to notate tunes stored in an ASCII format (`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 MusicTeX. However, it should be extendable to many other types of music.

Digital music fans can typeset notation for their efforts by using *midi2tex*, which translates MIDI data files into MusicTeX source code.

There is a mailing list (`TeX-music@sunsite.dk`) for discussion of typesetting music in TeX. To subscribe, use `http://sunsite.dk/mailman/listinfo/tex-music/`

abc2mtex: `support/abc2mtex`

M-Tx: `support/mtx`

midi2tex: `support/midi2tex`

musictex: `macros/musictex`

musixtex (Taupin's version): `macros/musixtex/taupin`

musixtex (Egler's version): `macros/musixtex/egler`

pmx: `support/pmx`

137 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 `\tiny` to `\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 *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.

extsizes bundle: `macros/latex/contrib/other/extsizes`

138 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:

```
\setlength{\parindent}{0pt}
\setlength{\parskip}{\baselineskip}
```

and in the very simplest text, it's a fine solution.

However, the non-zero `\parskip` interferes with lists and the like, and the result looks pretty awful. The *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`

139 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 `\left` and `\right`, which can be used to modify brackets (of whatever sort) around a mathematical expression, as in: `\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 `\middle` command, to be used in expressions like

```
\left\{ x\in N \middle| x\mbox{ even} \right\}
```

to specify the set of even natural numbers. The ε-TeX system (see question 240) defines just such a command, but users of Knuth's original need some support. Donald Arseneau's *braket* package provides commands for set specifications (as above) and for Dirac brackets (and bras and kets). The package uses the ε-TeX built-in command if it finds itself running under ε-TeX.

braket.sty: `macros/latex/contrib/other/misc/braket.sty`

140 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 *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.

The *lettrine* package has a well-constructed array of options, and the examples (a pretty impressive set) come as a separate file (also distributed 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`

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 132). 'Pretty' listings are generally provided by means of a pre-compiler, but the *listings* package manages to do the job within LaTeX.

The *lgrind* system is a well-established pre-compiler, with all the facilities one might need and a wide repertoire of languages.

The *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++.

C++2LaTeX: `support/C++2LaTeX-1_1pl1`

lgrind: `support/lgrind`

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

tiny_c2l: support/tiny_c2l

142 The comma as a decimal separator

If you use a comma in maths mode, you get a small space after it; this space is inappropriete if the comma is being used as a decimal separator. An easy solution to this problem is to type (in maths mode) `3{,}14` instead of typing `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 *icomma* package. The packages ensures that there will be no extra space after a comma, unless you type a space after it (as in `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

143 Breaking boxes of text

(LA)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 LaTeX 2.09, but there are reasons to suspect that it may be unstable working with current LaTeX.

backgrnd.tex: macros/generic/backgrnd.tex

boites.sty: macros/latex/contrib/other/boites

framed.sty: macros/latex/contrib/other/misc/framed.sty

shade.tex: macros/generic/shade.sty

P.6 Tables of contents and indexes

144 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 LaTeX 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 seperation of the dots in the dotted leaders, respectively. The series of commands named `\l@xxx`, 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: macros/latex/contrib/supported/tocloft

145 Unnumbered sections in the Table of Contents

The easiest way to get headings of funny 'sections' such as prefaces in the table of contents is to use the counter `secnumdepth` described in Appendix C of the LaTeX manual. For example:

```
\setcounter{secnumdepth}{-1}
\chapter{Preface}
```

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 LaTeX book class by the `\frontmatter` and `\backmatter` commands.

This is why it works: `\chapter` without the star does

1. put something in the `.toc` file;
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.

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 119) 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/supported/abstract

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

tocvsec2.sty: macros/latex/contrib/supported/tocvsec2

146 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 ite,s 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:

```
\bibliography{frooble}
\addcontentsline{toc}{chapter}{Bibliography}
```

will produce the *wrong* answer if the bibliography is more than one page long. Instead, one should say:

```
\cleardoublepage
\addcontentsline{toc}{chapter}{Bibliography}
\bibliography{frooble}
```

(Note 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 *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 (*scrbook* instead of *book*, *scrreprt* instead of *report*, etc.) provide this functionality as a set of class options.

KOMA-Script bundle: `macros/latex/contrib/supported/koma-script`

tocbibind.sty: `macros/latex/contrib/supported/tocbibind`

147 Multiple indexes

LaTeX's standard indexing capabilities (those provided by the *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 `\makeindex`, `\index` and `\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 `\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 `\indexname` (see question 188) is completely ignored.

The *index* package provides a comprehensive set of indexing facilities, including a `\newindex` command that allows the definition of new styles of index. `\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/supported/camel`

multind.sty: `macros/latex209/contrib/misc/multind.sty`

Q How do I do *X* in (La)TeX

Q.1 Mathematics

148 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 LaTeX 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 LaTeX users: it does indeed provide an automatic way of signalling the end of a proof.

The AMS-LaTeX 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-LaTeX bundle `macros/latex/required/amslatex`

ntheorem: `macros/latex/contrib/supported/ntheorem`

149 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.

```
\newtheorem{preremark}{Remark}
\newenvironment{remark}%
  {\begin{preremark}\upshape}{\end{preremark}}
```

The *ntheorem* package provides roman theorems directly.

ntheorem.sty: `macros/latex/contrib/supported/ntheorem`

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

150 Defining a new log-like function in LaTeX

Use the `\mathop` command, as in:

```
\newcommand{\diag}{\mathop{\mathrm{diag}}}
```

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:

```
\newcommand{\diag}%
  {\mathop{\mathrm{diag}}\nolimits}
```

AMS-LaTeX (in its *amsopn* package) provides a command `\DeclareMathOperator` that satisfies the requirement.

(It should be noted that "log-like" was reportedly a *joke* on Lamport's part; it is of course clear what was meant.)

amsopn.sty: In the AMS-LaTeX distribution `macros/latex/required/amslatex`

Q.2 Lists

151 Fancy enumeration lists

Suppose you want your top-level `enumerates` to be labelled 'I/', 'II/', . . . , then give these commands:

```
\renewcommand{\theenumi}{\Roman{enumi}}
\renewcommand{\labelenumi}{\theenumi/}
```

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 LaTeX 2_ε, the *enumerate.sty* package offers similar facilities. Using it, the example above would be achieved simply by starting the enumeration `\begin{enumerate}[I/]`.

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

152 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`⟨*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 *expdlist* package, among its many controls of the appearance of `description` lists, offers a compaction parameter (see the documentation); the *mdwlist* package offers a `\makecompactlist` command for users' own list definitions, and uses it to define compact lists `itemize*`, `enumerate*` and `description*`. In fact, you can write lists such as these commands define pretty straightforwardly — for example:

```
\newenvironment{itemize*}%
  {\begin{itemize}%
    \setlength{\itemsep}{0pt}%
    \setlength{\parsep}{0pt}}%
  {\end{itemize}}
```

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 `\list` command; its invocation is `\list{`⟨*item stuff*⟩`}{`⟨*list stuff*⟩`}`; the ⟨*list stuff*⟩ is executed *after* the `\@list`⟨*depth*⟩, and can therefore be used to adjust all the parameters, including `\topsep`.

An alternative is to redefine `\@list`⟨*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).

expdlist.sty: `macros/latex/contrib/supported/expdlist`

mdwlist.sty: Distributed as part of
  `macros/latex/contrib/supported/mdwtools`

Q.3 Tables, figures and diagrams

153 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 `tabular*` environment takes an extra argument (before the `clpr` layout one) which takes a length specification: you can say things like "`15cm`" or "`\columnwidth`" here. You must also have an `\extracolsep` command in the `clpr` layout argument, inside an `@{}` directive. So, for example, one might have

```
\begin{tabular*}{\columnwidth}%
                 {@{\extracolsep{\fill}}lllr}
```

The `\extracolsep` applies to all inter-column gaps to its right as well; if you don't want all gaps stretched, add `\extracolsep{0pt}` to cancel the original.

The *tabularx* package defines an extra `clpr` directive, `X`; `X` columns behave as `p` columns which expand to fill the space available. If there's more than one `X` column in a table, the spare space is distributed between them.

The *ltxtable* combines the features of the *longtable* and *tabularx* packages: it's important to read the documentation, since usage is distinctly odd.

ltxtable.sty: Distributed as part of
  `macros/latex/contrib/supported/carlisle`

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

154 Spacing lines in tables

(La)TeX 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 (La)TeX 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:

```
\usepackage{array}% in the preamble
...
\setlength{\extrarowheight}{length}
\begin{tabular}{....}
```

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`

tabls.sty: `macros/latex/contrib/other/misc/tabls.sty`

155 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 aligment 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 *colortbl*, and its author has provided the *ltxtable* package to provide (most of) the facilities of *tabularx* (see question 153) 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, *supertabular* has the great advantage of doing its job in a single run.

Both *longtable* and *supertabular* allow definition of head- and footlines for the table; *longtable* allows distinction of the first and last head and foot.

The *xtab* package fixes some infelicities of *supertabular*, and also provides a "last head" facility (though this, of course, destroys *supertabular*'s advantage of operating in a single run).

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

ltxtable.sty: Generate by running `macros/latex/contrib/supported/carlisle/ltxtable.tex`

supertabular.sty: `macros/latex/contrib/supported/supertabular`

xtab.sty: `macros/latex/contrib/supported/xtab`

156 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

```
... & \centering blah ... \\
```

one is liable to encounter errors that complain about a "misplaced `\noalign`" (or the like). The problem is that the command `\\` means different things in different circumstances: the `tabular` environment switches the meaning to a value for use in the table, and `\centering`, `\flushright` and `\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 `\\`) that terminates it.

The simple (old) solution is to preserve the meaning of `\\`:

```
\def\PBS#1{\let\temp=\\%
  #1%
  \let\\=\temp
}
```

which one uses as:

```
... & \PBS\centering blah ... \\
```

(for example).

The technique using `\PBS` was developed in the days of LaTeX 2.09 because the actual value of `\\` that the `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:

```
... & \centering blah ... \tabularnewline
```

but the old trick has the advantage of extreme compactness.

The `\PBS` trick also serves well in *array* package "field format" preamble specifications:

```
\begin{tabular}{... >{\PBS\centering}p{50mm}}
...
```

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

157 Flowing text around figures in LaTeX

There are several LaTeX 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:

`floatflt` *floatflt* is an improved version (for LaTeX 2$_\varepsilon$) of `floatfig.sty`, and its syntax is:

`\begin{floatingfigure}[`*options*`]{`*width of figure*`}`
figure contents
`\end{floatingfigure}`

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).

`wrapfig` *wrapfig* has syntax:

`\begin{wrapfigure}[`*height of figure in lines*`]`
`{l,r,`*etc*`}[`*overhang*`]{`*width*`}`
figure, caption, etc.
`\end{wrapfigure}`

The syntax of the `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`*(nside)* or `o`*(utside)* 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 `\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.

`picins` *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:

`\parpic(`*width*`,`*height*`)(`*x-off*`,`*y-off*`)`
`[`*Options*`][`*Position*`]`
`{`*Picture*`}`
Paragraph text

All parameters except the *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.

floatflt.sty: `macros/latex/contrib/other/floatflt`

picins.sty: `systems/msdos/picins/picins.zip`

picins documentation summary: `macros/latex209/contrib/picins/picins.txt`

wrapfig.sty: `macros/latex/contrib/other/misc/wrapfig.sty`

158 Drawing with TeX

There are many packages to do pictures in (La)TeX 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 PiCTeX. Depending on your type of drawing, and setup, four systems should be at the top of your list to look at:

1. *pstricks*; this gives you access to all the power of PostScript from TeX itself, by sophisticated use of `\specials`. 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 (La)TeX distributions. Knuth uses it for all his work...
3. *Mfpic*; you liked METAFONT, but can't understand the language? The package makes up METAFONT or MetaPost code for you within using familiar-looking TeX macros. Not *quite* the full power of METAFONT, but a friendlier interface.
4. You liked PiCTeX but don't have enough memory or time? Look at Eitan Gurari's *dratex*, which is as powerful as most other TeX drawing packages, but is an entirely new implementation, which is not as hard on memory, is much more readable (and is fully documented).

dratex: `graphics/dratex`

mfpic: `graphics/mfpic`

pstricks: `graphics/pstricks`

159 Drawing Feynman diagrams in LaTeX

Michael Levine's *feynman* bundle for drawing the diagrams in LaTeX 2.09 is still available.

Thorsten Ohl's *feynmf* is designed to work with current LaTeX, and works in combination with METAFONT (or MetaPost). The *feynmf* or *feynmp* package reads a description of the diagram written in TeX, and writes out code. METAFONT (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 Vermaseren'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`

160 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 *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{figure}[p!]
```

but the overriding `!` modifier has no effect on float page floats; so you have to make the float satisfy the parameters. Question 207 offers some suggestions, but doesn't solve the one-float-per-page question.

The 'obvious' solution, using the counter `totalnumber` ("total number of floats per page") doesn't work: `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 `\floatpagefraction` really small, and to ensure that no more than one float occupies a page, make the separation between floats really big:

```
\renewcommand\floatpagefraction{.001}
\makeatletter
\setlength\@fpsep{\textheight}
\makeatother
```

Q.4 Footnotes

161 Footnotes in tables

The standard LaTeX `\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`

162 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`

163 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 162) and with footnotes in tables (see question 161).

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:

```
\begin{figure}
  \begin{minipage}{\textwidth}
    ...
    \caption[Caption for LOF]%
      {Real caption\footnote{blah}}
  \end{minipage}
\end{figure}
```

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 *ccaption* package's documentation describes a really rather awful work-around.

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

164 Footnotes whose texts are identical

If the *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 `\footnotemark` to signify the repeats:

```
...\footnote{Repeating note}
...
...\footnotemark[1]
```

... 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:

```
\newcounter{fnnumber}
...
...\footnote{Text to repeat}%
\setcounter{fnnumber}{\thefootnote}%
...
...\footnotemark[\thefnnumber]
```

but this is somewhat tedious. LaTeX's labelling mechanism can be summoned to our aid, but there are ugly error messages before the `\ref` is resolved on a second run through LaTeX:

```
...\footnote{Text to repeat\label{fn:repeat}}
...
...\footnotemark[\ref{fn:repeat}]
```

This is the cleanest simple way of doing the job. Note that the `\label` command *must* be inside the argument of `\footnote`.

The *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 `\DeclareFixedFoot` command, and then use the command you've 'declared' in the body of the document:

```
\DeclareFixedFootnote{\rep}{Text to repeat}
...
...\rep{}
...\rep{}
```

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.

fixfoot.sty: `macros/latex/contrib/supported/fixfoot`

Q.5 Document management

165 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...

TeX retains what it considers the name of the *job*, only, in the special macro `\jobname`; this is the name of the file first handed to TeX, stripped of its directory name and of any extension (such as `.tex`). If no file was passed (i.e., you're using TeX interactively), `\jobname` has the value `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 TeX makes no attempt to keep track of files input to the *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 TeX, since the syntax of the `\input` command is so peculiar.

In the case of LaTeX, the input commands have pretty regular syntax, and the simplest patching techniques (see question 191) can be used on them:

```
\def\ThisFile{\jobname}
\newcounter{FileStack}
\let\OrigInput\input
\renewcommand{\input}[1]{%
  \stepcounter{FileStack}
  \expandafter\let
    \csname NameStack\theFileStack\endcsname
    \ThisFile
  \def\ThisFile{#1}%
  \OrigInput{#1}%
  \expandafter\let\expandafter
    \ThisFile
    \csname NameStack\theFileStack\endcsname
  \addtocounter{FileStack}{-1}%
}
```

(And similarly for `\include`.) The code assumes you always use LaTeX syntax for `\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 `\finkfile`. The bundle includes a `fink.el` that provides support under *emacs* with AUC-TeX.

fink.sty: `macros/latex/contrib/supported/fink`

166 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 *url* package, she has to have *url* too, for example). But suppose you have a bug-free version of the *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 `\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 `\listfiles` only registers things that are input by the "standard" LaTeX mechanisms (`\documentclass`, `\usepackage`, `\input`, `\include`, `\includegraphics` and so on). But if you use TeX 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 *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 *bundledoc* system uses `\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 *teTeX* and *mikTeX*. It's plainly useful when you're sending the first copy of a document.

bundledoc: `support/bundledoc`

snapshot.sty: `macros/latex/contrib/supported/snapshot`

167 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; PDFTeX *does* provide the information, but we're not aware yet of any programmer taking advantage of the fact to write a PDFTeX-based changebar package).

The simplest package that offers change bars is Peter Schmitt's *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 *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 *changebar* package, which uses `\special` commands supplied by the driver you're using. You need therefore to tell the package which driver to generate `\specials` for (in the same way that you need to tell the *graphics* package); the list of available drivers is pretty restricted, but does include *dvips*. The package comes with a shell script *chbar.sh* (for use on Unix machines) that will compare two documents and generate a third which is marked-up with *changebar* macros to highlight changes.

The *vertbars* package uses the techniques of the *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.

backgrnd.tex: `macros/generic/backgrnd.tex`

changebar.sty: `macros/latex/contrib/supported/changebar`

lineno.sty: `macros/latex/contrib/supported/lineno`

vertbars.sty: `macros/latex/contrib/supported/misc/vertbars.sty`

168 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.

If your requirement is for a document from which whole chapters (or the like) are missing, consider the LaTeX `\include`/`\includeonly` system. If you '`\include`' your files (rather than `\input` them — see question 218), LaTeX writes macro traces of what's going on at the end of each chapter to the `.aux` file; by using `\includeonly`, you can give LaTeX an exhaustive list of the files that are needed. Files that don't get `\included` 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 *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.)

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 LaTeX, or interactively.

Finally, *verbatim* (which should be available in any distribution) defines a `comment` environment, which enables the dedicated user of the source text editor to suppress bits of a LaTeX 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

169 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 105 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

```
Graph: \includegraphics{picture}
\input{explanation}
```

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.

import.sty: macros/latex/contrib/supported/misc/import.sty

170 Version control using RCS or CVS

If you use RCS or CVS to maintain your (La)TeX 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 *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:

```
\def\RCS$#1: #2 ${\expandafter
  \def\csname RCS#1\endcsname{#2}%
}
\RCS$Revision: 1.92 $ % or any RCS keyword
\RCS$Date: 2002/06/26 14:04:01 $
...
\date{Revision \RCSRevision, \RCSDate}
```

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

Q.6 Hyphenation

171 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 *babel* if you're a LaTeX user. This may reveal that you need another set of hyphenation patterns; see question 175 for advice on how to install it.)

So what else can go wrong?

- Since TeX version 3.0, the limits on how near to either end of a word hyphenation may take place have been programmable (see question 172), and for some reason the values in question may have been corrupted in some macros you are using. TeX won't hyphenate less than `\lefthyphenmin` characters after the start of a word, nor less than `\righthyphenmin` 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 `\hyph` 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 174.
- 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

172 Weird hyphenation of words

If your words are being h-yphenated, like this, with jus-t 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 2.09 your best plan is to upgrade to LaTeX 2_ε. 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 2_ε, 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 2_ε 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 2_ε 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`

173 (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 175 for instructions on adding this new "language").

UK patterns: `language/hyphenation/ukhyphen.tex`

174 Accented words aren't hyphenated

TeX's algorithm for hyphenation gives up when it encounters an `\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 avoiding the effect by using an appropriately encoded font (for example, a Cork-encoded font — see question 42) which contains accented letters as single glyphs. LaTeX users can achieve this end simply by adding the command

```
\usepackage[T1]{fontenc}
```

to the preamble of their document. Other encodings (notably LY1, promoted by Y&Y — see 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 239) will provide a rather different solution.

175 Using a new language with Babel

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

```
\usepackage[catalan]{babel}
```

provokes the warning message

```
Package babel Warning: No hyphenation patterns
... were loaded for the language 'Catalan'
... I will use the patterns loaded for
... \language=0 instead.
```

(The layout of the error message isn't quite like this; the language name *is* capitalised, though.)

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 `language.dat` (which is part of the Babel installation); it will contain a line

```
%catalan          cahyphen.tex
```

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 102).

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).

teTeX, fpTeX To regenerate all formats, do:
`fmtutil --all`
If you're willing to think through what you're doing (this is *not* for the faint-hearted), you can select a sequence of formats and for each one, run:
`fmtutil --byfmt ⟨formatname⟩`
where *formatname* is something like '`latex`', or:
`fmtutil --byhyphen ⟨hyphenfile⟩`
where *hyphenfile* is the file specifying hyphenation to the format — usually `language.dat`

MikTeX On a *MikTeX* distribution earlier than v2.0, do:
`Start→Programs→MikTeX→Maintenance→Create all format files`
or get a DOS window and run:
`initexmf --dump`
On a *MikTeX* distribtution 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` to see whether any could reasonably be removed.

babel: `macros/latex/required/babel`

hyphenation patterns: `language/hyphenation`

176 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 `\left-` 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 LaTeX 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 (La)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 130), 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.)

hyphenat.sty: `macros/latex/contrib/supported/hyphenat`

Q.7 Odds and ends

177 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 AMS-TeX, PiCTeX, BibTeX, 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 (silly enough in itself) and marketing input from Addison-Wesley led to the even stranger current logo LaTeX 2_ε.

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 *texnames* package (but note that this set of macros isn't entirely reliable in LaTeX 2_ε). The METAFONT and MetaPost logos can be set in fonts that LaTeX 2_ε knows about (so that they scale with the surrounding text) using the *mflogo* package; but be aware that booby-traps surround the use of the Knuthian font for MetaPost (you might get 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 `AMS-\TeX{}` (AMS-TeX) for AMS-TeX, `Pic\TeX{}` (PicTeX) for PiCTeX, `Bib\TeX{}` (BibTeX) for BibTeX, 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...

mflogo.sty: `macros/latex/contrib/supported/mflogo`

texnames.sty: `macros/eplain/texnames.sty`

178 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 *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 *titleref* package is a simple extension which provides the command `\titleref`; it is a stand-alone package.

The *nameref* package employs the techniques of the *hyperref* package to define a `\nameref` command; it will work in documents that have *hyperref* loaded.

nameref.sty: Distributed with `macros/latex/contrib/supported/hyperref`

titleref.sty: `macros/latex/contrib/other/misc/titleref.sty`

179 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" `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" METAFONT files on that provide bold versions of both `cmtt` and `cmcsc` (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 *texmf* tree (see question 103); TeX (and friends) will automatically build whatever font files they need when you first make reference to them. There's a jiffy package *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 *mf-extra* set. However, commercial fixed-width fonts (including the default *Courier*) almost always come with a bold variant, so that's not a problem. Furthermore PSNFSS (see question 82) typically provides "faked" small caps fonts, and has no compunctions about providing them in a bold form.

bold-extra.sty: `macros/latex/contrib/other/misc/bold-extra.sty`

bold tt and small caps fonts: `fonts/cm/mf-extra/bold`

R Symbols, etc.

180 Symbols for the number sets

It is a good idea to have commands such as `\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 `\R`, `\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 *bbold* family. This set has the interesting property of offering blackboard bold forms of lower-case 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 (La)TeX (see question 86). Of the free sets, the *txfonts* and *pxfonts* families both come with replicas of *msam* and *msbm*, and the *mathpazo* family includes a "mathematically significant" choice of blackboard bold characters.

The "lazy person's" blackboard bold macros:

```
\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}}}
```

work well at normal size if the surrounding text is `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): `fonts/amsfonts/plaintex`

AMS support files (LaTeX): `fonts/amsfonts/latex`

AMS symbol fonts: `fonts/amsfonts/sources/symbols`

AMS symbol fonts in Type 1 format: Browse `fonts/amsfonts/ps-type1`

bbold fonts: `fonts/bbold`

mathpazo fonts: `fonts/mathpazo`

pxfonts: `fonts/pxfonts`

txfonts: `fonts/txfonts`

181 Better script fonts for maths

The font selected by `\mathcal` is the only script font 'built in'. However, there are other useful calligraphic fonts included with modern TeX distributions.

Euler The *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 `\mathcal`.
Type 1 versions of the fonts are available in the AMS fonts distribution.

RSFS The *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 `\mathscr`.
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 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

182 Setting bold Greek letters in LaTeX

The issue here is complicated by the fact that \mathbf (the command for setting bold *text* in TeX 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, \mathbf *can't* be used even for upper-case Greek letters in the 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:

```
{\boldmath$\theta$}
```

but \boldmath may not be used in maths mode, so this 'solution' requires arcana such as:

```
$... \mbox{\boldmath$\theta$} ...$
```

which then causes problems in superscripts, etc.

These problems may be addressed by using a bold mathematics package.

- The *bm* package, which is part of the LaTeX tools distribution, defines a command \bm which may be used anywhere in maths mode.
- The *amsbsy* package (which is part of AMS-LaTeX) defines a command \boldsymbol, which (though slightly less comprehensive than \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 AMS-LaTeX distribution macros/latex/required/amslatex

amsmath.sty: Distributed as part of the AMS-LaTeX distribution macros/latex/required/amslatex

183 The Principal Value Integral symbol

This symbol (an integral sign, 'crossed') does not appear in any of the fonts ordinarily available to (La)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{$#1{#2#3}{\int}$}
     \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.

184 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.

underscore.sty: macros/latex/contrib/other/misc/underscore.sty

185 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 '@'.

186 Typesetting the Euro sign

The European currency "Euro" is represented by a symbol of somewhat dubious design, but it's an important currency and (LA)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 *latin9* input encoding defined by the *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 `\texteuro` for the character; at present that command is *only* available from the *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 (see `ftp://ftp.adobe.com/pub/adobe/type/win/all/eurofont.exe`) — the file is packaged as a *Windows* self-extracting executable, but it may be decoded as a `.zip` format achive on other operating systems. The *euro* bundle contains metrics, *dvips* map files, and macros (for Plain TEX and LATEX), for using these fonts in documents. LATEX users will find two packages in the bundle: *eurosans* only offers the sans-serif version (to conform with the obsolete ruling about sans-serif-only symbols; the package provides the command `\euro`), whereas *europs* matches the Euro symbol with the surrounding text (providing the command `\EUR`). To use either package with the *latin9* encoding, you need to define `\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 *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 *marvosym* fonts contain a Euro symbol among many other good things; the font on CTAN is not Adobe *ATM* compatible, but a compatible version is available free from Y&Y (see `http://www.YandY.com/download/marvosym.zip`). The font on CTAN comes with a set of macros to typeset all the symbols it contains.

Other METAFONT-based bundles containing Euro symbols are to be found in *china2e* (whose primary aim is Chinese dates and suchlike matters) and the *eurosym* fonts.

china2e bundle: `macros/latex/contrib/supported/china2e`

EC fonts: `fonts/ec`

euro fonts: `fonts/euro`

eurofont.sty: `macros/latex/contrib/supported/eurofont`

eurosym fonts: `fonts/eurosym`

marvosym fonts: `fonts/psfonts/marvosym`

textcomp.sty: Part of the LATEX distribution.

S Macro programming

187 Finding the width of a letter, word, or phrase

Put the word in a box, and measure the width of the box. For example,

```
\newdimen\stringwidth
\setbox0=\hbox{hi}
\stringwidth=\wd0
```

Note that if the quantity in the `\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:

```
\newlength{\gnat}
\settowidth{\gnat}{\textbf{small}}
```

This sets the value of the length command `\gnat` to the width of "small" in bold-face text.

188 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 in to 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 `\`⟨*thing*⟩`name`, and changing them is simplicity itself. Put:

```
\renewcommand{\⟨thing⟩name}{Res minor}
```

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 189 if you're using it.)

The names that are defined in the standard LATEX classes (and the *makeidex* 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.

`\abstractname`	Abstract
`\alsoname`	see also (*makeidx* package)
`\appendixname`	Appendix
`\bibname`	Bibliography (*report*,*book*)

`\ccname`	cc (*letter*)
`\chaptername`	Chapter (*report*,*book*)
`\contentsname`	Contents
`\enclname`	encl (*letter*)
`\figurename`	Figure (for captions)
`\headtoname`	To (*letter*)
`\indexname`	Index
`\listfigurename`	List of Figures
`\listtablename`	List of Tables
`\pagename`	Page (*letter*)
`\partname`	Part
`\refname`	References (*article*)
`\seename`	see (*makeidx* package)
`\tablename`	Table (for captions)

189 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 188 (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

```
\renewcommand{\contentsname}%
    {Indholdsfortegnelse}
```

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 `\begin{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:

```
\addto\captionsdanish{%
  \renewcommand{\contentsname}%
    {Indholdsfortegnelse}%
}
```

190 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` command, and having included the package one writes something like:

```
\makeatletter
\@removefromreset{figure}{chapter}
```

and the automatic renumbering stops. You then need to redefine the way in which the figure number (in this case) is printed:

```
\renewcommand{\thefigure}{\@arabic\c@figure}
\makeatother
```

(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`

191 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, you 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 analagous 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`

192 `\@` 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 235.

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.)

193 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 TeX 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 TeX code during the save process.

What is a 'robust command'? It's a command that expands into legal TeX code during the save process.

No-one (of course) likes this situation; the LaTeX3 team have removed the need for protection of some things in the production of LaTeX 2_ε, 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.

194 `\edef` does not work with `\protect`

Robust LaTeX commands are either "naturally robust" — meaning that they never need `\protect`, or "self-protected" — meaning that they have `\protect` built in to their definition in some way. Self-protected commands are robust only in a context where the `\protect` mechanism is properly handled. The body of an `\edef` definition doesn't handle `\protect` properly, since `\edef` is a TeX primitive rather than a LaTeX command.

This problem is resolved by a LaTeX internal command `\protected@edef`, which does the job of `\edef` while keeping the `\protect` mechanism working. There's a corresponding `\protected@xdef` which does the job of `\xdef`.

Of course, these commands need to be tended carefully, since they're internal: see question 192.

195 Optional arguments like `\section`

Optional arguments, in macros defined using `\newcommand`, don't quite work like the optional argument to `\section`. The default value of `\section`'s optional argument is the value of the mandatory argument, but `\newcommand` requires that you 'know' the value of the default beforehand.

The requisite trick is to use a macro in the optional argument:

```
\newcommand\thing[2][\DefaultOpt]{%
  \def\DefaultOpt{#2}%
  ...%
}
```

196 Making labels from a counter

Suppose we have a LaTeX counter, which we've defined with \newcounter{foo}. We can increment the value of the counter by \addtocounter{foo}{1}, but that's pretty clunky for an operation that happens so often ... so there's a command \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 \refstepcounter{foo}, the internal variable is set to the new value, and (until something else comes along), \label will refer to the counter.

197 Finding if you're on an odd or an even page

Question 222 discusses the issue of getting \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 (La)TeX 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 \c@page) is normally only reliable when you're actually *in* the output routine.

The solution is to use some version of the \label mechanism to determine which side of the page you're on; the value of the page counter that appears in a \pageref command has been inserted in the course of the output routine, and is therefore safe.

However, \pageref itself isn't reliable: one might hope that

```
\ifthenelse{\isodd{\pageref{foo}}}{odd}{even}
```

would do the necessary, but both the *babel* and *hyperref* packages have been known to interfere with the output of \pageref; be careful!

The *chngpage* package needs to provide this functionality for its own use, and therefore provides a command \checkoddpage; this sets a private-use label, and the page reference part of that label is then examined (in a *hyperref*-safe way) to set a conditional \ifcpoddpage true if the command was issued on an odd page. Of course, the \label contributes to LaTeX's "Rerun to get cross-references right" error messages...

chngpage.sty: macros/latex/contrib/supported/misc/chngpage.sty

198 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 151.) 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 analagous change works for any counter that gets used in a \label command.

199 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\jobis#1{FF\fi
  \def\predicate{#1}%
  \edef\predicate{\expandafter\StripPrefix
                  \meaning\predicate}%
  \edef\job{\jobname}%
  \ifx\job\predicate
}
```

And it's used as:

```
\if\jobis{mainfile}%
  \message{YES}%
\else
  \message{NO}%
\fi
```

Note that the command \StripPrefix need not be defined if you're using LaTeX — there's already an internal command (see question 192) \strip@prefix that you can use.

200 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:

```
\ifx1#1
\else\ifx2#1
...
\else\ifx9#1
\else\isanumfalse
```

```
\fi\fi...\fi
```

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:

```
\ifnum`#1<`0 \isanumfalse
\else\ifnum`#1>`9 \isanumfalse
     \fi
\fi
```

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:

```
\def\IsPositive#1{%
  TT\fi
  \ifcat_\ifnum0<0#1 _\else A\fi
}
```

which can be adapted to a test for a non-negative integer thus:

```
\def\IsNonNegative{%
  \ifcat_\ifnum9<1#1 _\else A\fi
}
```

or a test for any integer:

```
\def\gobble#1{}
\def\gobbleminus{\futurelet\temp\gobm}
\def\gobm{\ifx-\temp\expandafter\gobble\fi}
\def\IsInteger#1{%
  TT\fi
  \ifcat_\ifnum9<1\gobbleminus#1 _\else A\fi
}
```

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:

```
\def\testnum#1{\afterassignment\testresult
               \count255=#1 \end}
\def\testresult#1\end{\ifx\end#1\end}
\def\IsInteger#1{TT\fi \testnum{#1}}
```

(which technique is due to David Kastrup). In a later thread on the same topic, Michael Downes offered:

```
\def\IsInteger#1{%
  TT\fi
  \begingroup \lccode`\-=`\0 \lccode`+=`\0
    \lccode`\1=`\0 \lccode`\2=`\0 \lccode`\3=`\0
    \lccode`\4=`\0 \lccode`\5=`\0 \lccode`\6=`\0
    \lccode`\7=`\0 \lccode`\8=`\0 \lccode`\9=`\0
  \lowercase{\endgroup
    \expandafter\ifx\expandafter\delimiter
    \romannumeral0\string#1}\delimiter
}
```

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 conditionals written "naturally" — for example:

```
\if\IsInteger{<subject of test>}%
  <deal with integer>%
\else
  <deal with non-integer>%
\fi
```

201 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:

```
\def\a#1{+++#1+++#1+++#1+++}  \a{b}
```

the macro expansion produces '+++b+++b+++b+++', which people find normal. However, if we now replace part of the macro:

```
\def\a#1{+++#1+++\def\x #1{xxx#1}}
```

`\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:

```
\def\a#1{+++#1+++\def\x ##1{xxx##1}}
```

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.

202 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`:

```
\newcommand{\stline}[1]
      { \bigskip \makebox[2cm]{ \textbf{#1} } }
```

(the original appeared on a single line: it's wrapped here to fit in the printed FAQ's narrow columns).

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.

203 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 ... #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 ... #9{%
  \def\ArgI{{#1}}%
  \def\ArgII{{#2}}%
  ...
  \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`

204 A command with two optional arguments

If you've already read "breaking the 9-argument limit" (question 203) 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 203), which offer an alternative way forward.

T Things are Going Wrong…

T.1 Getting things to fit

205 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:

```
  ! TeX capacity exceeded, sorry
..                 [main memory size=263001].
```

Most of the time this error can be fixed *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 systems, 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 WEB 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 configuration file is read which specifies the size of the memory. On *web2c*-based systems, this file is called `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.

206 Why can't I load PiCTeX?

PiCTeX is a resource hog; fortunately, most modern TeX implementations offer generous amounts of space, and most modern computers are pretty fast, so users aren't too badly affected by its performance.

However, PiCTeX 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 `pictex.sty` with LaTeX and some other packages that also need dimension registers. When this happens, you will see the TeX error message:

```
! No room for a new \dimen.
```

There is nothing that can directly be done about this error: you can't extend the number of available `\dimen` registers without extending TeX itself. (ε-TeX and Ω — see questions 240 and 239 respectively — both do this, as does MicroPress Inc's VTeX — see question 55.) Since you can't (ordinarily) extend TeX, you need to change PiCTeX; unfortunately PiCTeX's author is no longer active in the TeX world, so one must resort to patching. There are two solutions available.

The ConTeXt module `m-pictex.tex` (for Plain TeX and variants) or the corresponding LaTeX *m-pictex* package provide an ingenious solution to the problem based on hacking the code of `\newdimen` itself.

Alternatively, Andreas Schell's *pictexwd* and related packages replace PiCTeX with a version that uses 33 fewer `\dimen` registers; so use *pictexwd* in place of *pictex* (either as a LaTeX package, or as a file to read into Plain TeX).

And how does one use PiCTeX anyway, given that the manual is so hard to come by (see question 32)? Fortunately for MS-DOS and Windows users, the *MathsPic* system may be used to translate a somewhat different language into PiCTeX commands; and the *MathsPic* manual is free (and part of the distribution). *MathsPic* is written in *Basic*; a version written in *Perl* is under development, and should be available soon.

m-pictex.sty: Distributed as part of
    `macros/context/cont-tmf.zip`

m-pictex.tex: Distributed as part of
    `macros/context/cont-tmf.zip`

MathsPic: `graphics/pictex/mathspic`

pictexwd.sty: Distributed as part of
    `graphics/pictex/addon`

T.2 Making things stay where you want them

207 Moving tables and figures in LaTeX

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 `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 234). 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 *float* or other packages).

- Do your figures need to float at all? If not, consider the [H] placement option offered by the *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 `\caption` commands are numbered in the order they appear in the document, and a [H] float can 'overtake' a float that hasn't yet been placed, so that figures numbers get out of order.
- Are the placement parameters on your figures right? The default (tbp) is reasonable, but you can reasonably change it (for example, to add an h). Whatever you do, *don't* omit the 'p': doing so could cause LaTeX to believe that if you can't have your figure *here*, you don't want it *anywhere*. (LaTeX does try hard to avoid being confused in this way...)
- LaTeX's own float placement parameters could be preventing placements that seem entirely "reasonable" to you — they're notoriously rather conservative. To encourage LaTeX 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.)

```
\renewcommand{\topfraction}{.85}
\renewcommand{\bottomfraction}{.7}
\renewcommand{\textfraction}{.15}
\renewcommand{\floatpagefraction}{.66}
\renewcommand{\dbltopfraction}{.66}
\renewcommand{\dblfloatpagefraction}{.66}
\setcounter{topnumber}{9}
\setcounter{bottomnumber}{9}
\setcounter{totalnumber}{20}
\setcounter{dbltopnumber}{9}
```

  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 `\clearpage` command? If so, do: the backlog of floats is cleared after a `\clearpage`. (Note that the `\chapter` command in the standard *book* and *report* classes implicitly executes `\clearpage`, so you can't float past the end of a chapter.)
- Try the *placeins* package: it defines a `\FloatBarrier` command beyond which floats may not pass. A package option allows you to declare that floats may not pass a `\section` command, but you can place `\FloatBarrier`s wherever you choose.
- Have a look at the LaTeX 2_ε `afterpage` package. Its documentation gives as an example the idea of putting `\clearpage` *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 *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 *endfloat* package to do the job for you.

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

endfloat.sty: `macros/latex/contrib/supported/endfloat`

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

morefloats.sty: `macros/latex/contrib/other/misc/morefloats.sty`

placeins.sty: `macros/latex/contrib/other/misc/placeins.sty`

208 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 typeface. The corresponding technique in typeset text is to switch from upright to italic text (or vice-versa): the LaTeX command `\emph` does just that to its argument.

Nevertheless, typographically illiterate people (such as those that specify double-spaced thesis styles — see question 128) continue to require underlining of us, so LaTeX as distributed defines an `\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 *ulem* package redefines the `\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 *soul* package defines an `\ul` command (after which the package is, in part, named) that underlines running text.

ulem.sty: `macros/latex/contrib/other/misc/ulem.sty`

soul.sty: `macros/latex/contrib/supported/soul`

209 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"; (La)TeX 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.)

T.3 Things have "gone away"

210 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 `cmr` could be found as `\fivrm`, `\sixrm`, `\sevrm`, `\egtrm`, `\ninrm`, `\tenrm`, `\elvrm`, `\twlrm`, `\frtnrm`, `\svtnrm`, `\twtyrm` and `\twfvrm`. 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_ε; to use the unconverted LaTeX 2.09 packages under LaTeX 2_ε, you need also to include the *rawfonts* package (which is part of the LaTeX 2_ε distribution).

211 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`

212 Where are the `msx` and `msy` fonts?

The `msx` and `msy` 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 `msb` families.

Nevertheless, `msx` and `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 `.tex` source that requests `msx` and `msy`, the best technique is to edit it so that it requests `msa` and `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.

msa and msb fonts: `fonts/amsfonts/sources/symbols`

virtual font set: `fonts/vf-files/msx2msa`

213 Where are the `am` fonts?

One *still* occasionally comes across a request for the `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)[7]. 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 `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 `am` font should be replaced by its corresponding `cm` font, the output should be almost correct.

U Why does it *do* that?

U.1 Common errors

214 LaTeX gets cross-references wrong

Sometimes, however many times you run LaTeX, the cross-references are just wrong. Remember that the `\label` command must come *after* the `\caption` command, or be part of it. For example,

```
\begin{figure}              \begin{figure}
\caption{A Figure}    or    \caption{A Figure%
\label{fig}                        \label{fig}}
\end{figure}                \end{figure}
```

215 Start of line goes awry

This answer concerns two sorts of problems: errors of the form

```
! Missing number, treated as zero.
<to be read again>
                   g
<*> [grump]
```

and those where a single asterisk at the start of a line mysteriously fails to appear in the output.

Both problems arise because `\\` takes optional arguments. The command `\\*` means "break the line here, and inhibit page break following the line break"; the command `\\[`⟨*dimen*⟩`]` means "break the line here and add ⟨*dimen*⟩ extra vertical space afterwards".

[7] The fonts acquired their label 'Almost' following the realisation that their first implementation in METAFONT79 still wasn't quite right; Knuth's original intention had been that they were the final answer

So why does \\ 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:

```
{\ttfamily
  /* C-language comment\\
  {[grump]} I don't like this format\\
  {*}/
}
```

(The above text derives from an actual post to `comp.text.tex`; this particular bit of typesetting could plainly also be done using the `verbatim` environment.)

The problem also appears in maths mode, in arrays and so on. In this case, large-scale bracketing of things is *not* a good idea; the TeX primitive `\relax` (which does nothing except to block searches of this nature) may be used. From another `comp.text.tex` example:

```
\begin{eqnarry}
  [a] &=& b \\
  \relax[a] &=& b
\end{eqnarry}
```

216 Why doesn't `\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 `\verb` has to assume that it is getting the first look at its parameter text; if it isn't, TeX has already assigned category codes so that `\verb` doesn't have a chance. For example:

```
\verb+\error+
```

will work (typesetting '`\error`'), but

```
\newcommand{\unbrace}[1]{#1}
\unbrace{\verb+\error+}
```

will not (it will attempt to execute `\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 `\protection` (see question 193).

The first question to ask yourself is: "is `\verb` actually necessary?".

- If `\texttt{`*your text*`}` produces the same result as `\verb+`*your text*`+`, then there's no need of `\verb` in the first place.
- If you're using `\verb` to typeset a URL or email address or the like, then the `\url` command from the *url* package will help: it doesn't suffer from the problems of `\verb`.
- If you're putting `\verb` into the argument of a boxing command (such as `\fbox`), consider using the `lrbox` environment:

  ```
  \newsavebox{\mybox}
  ...
  \begin{lrbox}{\mybox}
    \verb!VerbatimStuff!
  \end{lrbox}
  \fbox{\usebox{\mybox}}
  ```

Otherwise, there are three partial solutions to the problem.

- The *fancyvrb* package defines a command `\VerbatimFootnotes`, which redefines the `\footnotetext` (and hence the `\footnote`) commands in such a way that you can include `\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, `\VerbatimFootnotes` interacts poorly with the `para` option to the *footmisc* package.
- The *fancyvrb* package defines a command `\SaveVerb`, with a corresponding `\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.

  Rather simpler is the *verbdef* package, which defines a (robust) command which expands to the verbatim argument given.
- If you have a single character that is giving trouble (in its absence you could simply use `\texttt`), consider using `\string`. `\texttt{my\string_name}` typesets the same as `\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 `\protection` (see question 193) will make it work in such a case.

fancyvrb.sty: `macros/latex/contrib/supported/fancyvrb`

url.sty: `macros/latex/contrib/other/misc/url.sty`

verbdef.sty: `macros/latex/contrib/other/misc/verbdef.sty`

217 No line here to end

The error

```
! LaTeX Error: There's no line here to end.

See the LaTeX manual or LaTeX Companion
...                              for explanation.
```

comes in reaction to you giving LaTeX a \\ command at a time when it's not expecting it. The commonest case is where you've decided you want the label of a list item to be on a line of its own, so you've written (for example):

```
\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 LaTeX 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 `description` environment, that does just what you're after. (The *LaTeX Companion* — see 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 *expdlist* package provides the same functionality with its `\breaklabel` command, and *mdwlist* provides it via its `\desclabelstyle` command.

The other common occasion for the message is when you're using the `center` (or `flushleft` or `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\\[\baselineskip]
  body of the centred text...
\end{center}
```

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

mdwlist.sty: Distributed as part of macros/latex/contrib/supported/mdwtools

U.2 Common misunderstandings

218 What's going on in my `\include` commands?

The original LaTeX provided the `\include` command to address the problem of long documents: with the relatively slow computers of the time, the companion `\includeonly` facility was a boon. With the vast increase in computer speed, `\includeonly` is less valuable (though it still has its place in some very large projects). Nevertheless, the facility is retained in current LaTeX, and causes some confusion to those who misunderstand it.

In order for `\includeonly` to work, `\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 `\include` command. What's more, this mechanism doesn't work if a `\include` command appears in a file that was `\include`d itself: LaTeX diagnoses this as an error.

So, we can now answer the two commonest questions about `\include`:

- Why does LaTeX throw a page before and after `\include` commands?
  Answer: because it has to. If you don't like it, replace the `\include` command with `\input` — you won't be able to use `\includeonly` any more, but you probably don't need it anyway, so don't worry.
- Why can't I nest `\include`d files? — I always used to be able to under LaTeX 2.09.
  Answer: in fact, you couldn't, even under LaTeX 2.09, but the failure wasn't diagnosed. However, since you were happy with the behaviour under LaTeX 2.09, replace the `\include` commands with `\input` commands (with `\clearpage` as appropriate).

219 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 LaTeX:

```
{\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...
```

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 `\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:

```
end the group.\par}
Here's more that needn't be ragged...
```

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, LaTeX already defines an appropriate one:

```
\begin{flushleft}
  Here's text to be ranged left...
\end{flushleft}
```

220 Case-changing oddities

TeX provides two primitive commands `\uppercase` and `\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 `\uppercase{abc}` is '`ABC`', but `\uppercase{\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{About $y=f(x)$}
```

will produce

```
ABOUT $Y=F(X)$
```

which is probably not what is wanted.

In addition, \uppercase and \lowercase do not deal very well with non-American characters, for example \uppercase{\ae} is the same as \ae.

LaTeX provides commands \MakeUppercase and \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 \MakeUppercase and \MakeLowercase do not solve the other problems with \uppercase, so for example a section title containing \begin{tabular} ... \end{tabular} will produce a running head containing \begin{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 \mytable has to be protected, otherwise it will be expanded and made upper case; you can achieve the same result by declaring it with \DeclareRobustCommand, in which case the \protect won't be necessary.

David Carlisle's *textcase* package addresses many of these problems in a transparent way. It defines commands \MakeTextUppercase and \MakeTextLowercase which do upper- or lowercase, with the fancier features of the LaTeX standard \Make*-commands but without the problems mentioned above. Load the package with \usepackage[overload]{textcase}, and it will redefine the standard commands, so that section headings and the like don't produce broken page headings.

textcase.sty: Distributed as part of macros/latex/contrib/supported/carlisle

221 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 \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 \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. \enlargethispage changes the size of the current page by its argument (for example, you might say \enlargethispage{\baselineskip} to add a single line to the page, but you can use any ordinary TeX length such as 15mm or -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.

222 Getting \marginpar on the right side

In twoside documents, LaTeX makes sterling attempts to put \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 \marginpar is processed while page n is being being built, but doesn't get used until page n+1, then the \marginpar will turn up on the wrong side of the page. This is an instance of a general problem: see question 197.

The solution to the problem is for LaTeX to 'remember' which side of the page each \marginpar *should* be on. The *mparhack* package does this, using marks stored in the .aux file.

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

223 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 (La)TeX document, or if I select the font 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

```
Missing character:
                There is no ^^a3 in font cmr10!
Missing character:
                  There is no 3 in font rsfs10!
```

(the former demonstrating my TeX's unwillingness to deal in characters which have the eighth bit set, while the 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 *dvips* when using the OT1 and T1 versions of fonts that were supplied in Adobe standard encoding:

```
dvips: Warning: missing glyph 'Delta'
```

The process that generates the metrics for using the fonts generates an instruction to *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: *dvips* produces a black rectangle of whatever size the concocted font file has specified.

224 "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 `\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 *footmisc* (with the `perpage` option) and *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 error 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).

footmisc.sty: `macros/latex/contrib/supported/footmisc`

hyperref.sty: `macros/latex/contrib/supported/hyperref`

225 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 *require* one to type `\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:

```
\usepackage{xspace}
...
\newcommand{\restenergy}%
           {\ensuremath{mc^2}\xspace}
...
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`

226 (La)TeX makes overfull lines

When TeX 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 TeX'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 171 (and the questions it references) for advice.

If you're getting an undesired "overfull box", what has happened is that TeX 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 TeX, 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 TeX won't do it by default. In such a case on can *try* the LaTeX `\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 TeX's line breaking mechanisms.

TeX's first attempt at breaking lines is performed without even trying hyphenation: TeX sets its "tolerance" of line breaking oddities to the internal value `\pretolerance`, and sees what happens. If it can't get an acceptable break, TeX adds the hyphenation points allowed by the current patterns, and tries again using the internal `\tolerance` value. If this pass also fails, and the internal `\emergencystretch` value is positive, TeX will try a pass that allows `\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: `\pretolerance`, `\tolerance` and `\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 `10 000`, 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 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 219. 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

227 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 (La)TeX. 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 228 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

  ```
  \setcounter{errorcontextlines}{999}
  ```

  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 (La)TeX 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 TeXbook (see question 22) in some detail, fully to understand the trace.

  The command `\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 `\traceon` command gives you what `\tracingall` offers, but suppresses tracing around some of the truly verbose parts of LaTeX itself. The package also provides a `\traceoff` command (there's no "off" command for `\tracingall`), and a package option (`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 `errorcontextlines` above) and so on.

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

228 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 LaTeX, the underlying TeX problem may be superficially difficult to relate to the actual problem in the "higher-level" macros. Many LaTeX-detected problems manifest themselves as 'generic' errors, with error text provided by LaTeX itself (or by a LaTeX 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 227, 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:

```
\blah and so on
```

produces the error report

```
! Undefined control sequence.
l.4 \blah
          and so on
```

while:

```
\newcommand{\blah}[1]{\bleah #1}
\blah{to you}, folks
```

produces the error report

```
! Undefined control sequence.
\blah #1->\bleah
                 #1
l.5 \blah{to you}
                 , folks
```

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

l.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 TeXs 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).

229 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 193).

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 `\protection` is *not* the answer; question 162 deals specifically with this case.

230 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 227) *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.

231 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 \<thing>.
```

The `\thing`s 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, Ω and ε-TeX — see questions 239 and 240 respectively — both do this, as does MicroPress Inc's VTeX — 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 232). 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 PICTeX with LaTeX (see question 206).

232 `epsf` gives up after a bit

Some copies of the documentation of `epsf.tex` seem to suggest that the command

```
\input epsf
```

is needed for every figure included. If you follow this suggestion too literally, you get an error

```
! No room for a new \read .
```

after a while; this is because each time `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 231).

The solution is simple — this is in fact an example of misuse of macros; one only need read `epsf.tex` once, so change

```
...
\input epsf
\epsffile{...}
...
\input epsf
\epsffile{...}
```

(and so on) with a single

```
\input epsf
```

somewhere near the start of your document, and then decorate your `\epsffile` statements with no more than adjustments of `\epsfxsize` and so on.

233 Improper `\hyphenation` will be flushed

For example

```
! Improper \hyphenation will be flushed.
\'#1->{
       \accent 19 #1}
<*> \hyphenation{Ji-m\'e
                         -nez}
```

(in Plain TeX) or

```
! Improper \hyphenation will be flushed.
\leavevmode ->\unhbox
                      \voidb@x
<*> \hyphenation{Ji-m\'e
                         -nez}
```

in LaTeX.

As mentioned in question 171, 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. LaTeX users can be achieved this by use of the *fontenc* package (part of the LaTeX 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{Ji-m\'e-nez}` automatically become the single accented character by the time the hyphenation gets a to look at it.

234 "Too many unprocessed floats"

If LaTeX 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. LaTeX 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 207); LaTeX has found it can't place a float, and floats of the same type have piled up behind it. LaTeX'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 (207) 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 LaTeX to reconsider placement. (Of course, the floats can't all fit "here" if the sequence is sufficiently polonged: once the page fills, LaTeX 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.

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

235 `\spacefactor` complaints

The errors

```
! You can't use '\spacefactor' in vertical mode.
\@->\spacefactor
                 \@m
```

or

```
! Improper \spacefactor.
...
```

come of using a LaTeX internal command without taking the precaution of fooling LaTeX that you're inside it. The problem is discussed in detail in question 192, together with solutions.

236 `\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 emacs and *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' LaTeX diagnostic

```
! LaTeX Error: \begin{blah} on input line 6
                         ended by \end{document}.
```

which (though it doesn't tell you which *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 LaTeX, you need to use "traditional" binary search to find the offending group. Separate the header from the body of your file, and process each half on its own with the header; this tells you which half of the 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.

ε-TeX (and *elatex* — LaTeX run on ε-TeX) gives you further diagnostics after the traditional infuriating TeX one — it actually keeps the information in a similar way to LaTeX:

```
(\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
```

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).

237 "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 227).

Two LaTeX-specific errors are commonly aired on the newsgroups.

The first arises from a misconfiguration in a system that has been upgraded from LaTeX 2.09: the document uses `\usepackage{times}`, and the error appears at `\begin {document}`: the likely cause is lurking files that remain from the LaTeX 2.09 version of the *times* package. The *times* package in *psnfss* is a very simple beast, but the LaTeX 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 *The LaTeX Companion* (see question 22), and is exemplified by the following error text:

```
! Missing number, treated as zero.
<to be read again>
                   \relax
l.21 \begin{Ventry}{Return values}
```

The problem arises because the *Companion*'s examples always assume that the *calc* package is loaded: this fact is mentioned in the book, but often not noticed. The remedy is to load the *calc* package in any document using such examples from the *Companion*.

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

The psnfss bundle: `macros/latex/required/psnfss`

W Current TeX Projects

238 The LaTeX3 project

The LaTeX3 project team (see `http://www.latex-project.org/latex3.html`) is a small group of volunteers whose aim is to produce a major new document processing system 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 LaTeX3 team's first product (LaTeX 2_ε) was delivered in 1994 (it's now properly called "LaTeX", since no other version is current).

LaTeX 2_ε 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 LaTeX, in parallel with development of LaTeX3.

Some of the older discussion papers about directions for LaTeX3 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 LaTeX 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`

239 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 (`http://omega.cse.unsw.edu.au`).

Implementations of Ω are available as part of the teTeX, mikTeX, fpTeX and CMacTeX distributions (see question 53);

Ω is also distributed on the TeX Live CD-ROM (see question 52).

CTAN distribution: systems/omega

240 The $\mathcal{NTS}$ project

The $\mathcal{NTS}$ 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 $\mathcal{NTS}$ 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 $\mathcal{NTS}$. 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

241 The PDFTeX project

PDFTeX (formerly known as TeX2PDF) arose from Hán Thế Thánh'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. PDFTeX implements a small number of new primitives, to switch to PDF output, and to control various PDF features. Hán Thế Thánh has worked on PDFTeX throughout his Ph.D. research into typesetting, and the latest release includes facilities written to support novel typesetting techniques.

The latest version of PDFTeX is available on CTAN, and implementations are available as part the teTeX, mikTeX, fpTeX, and CMacTeX distributions (see question 53); it is also distributed on the TeX Live CD-ROM (see question 52). A version (by the author of CMacTeX) for use with OzTeX is also available on CTAN.

A mailing list discussing PDFTeX 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 OzTeX: nonfree/systems/mac/pdftex/pdftex_for_oztex.sit.bin

242 Future WEB technologies and (LA)TeX

An earlier question (69) addresses the issue of converting existing (LA)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 for it has already appeared in some browswers.

Conversion of (LA)TeX source to XML is already available (through TeX4ht 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 (LA)TeX to typeset is also well advanced. One useful technique is *transforming* the XML to LaTeX, using XSLT, and then simply using LaTeX; alternatively, one may typeset direct from the XML source (see question 70).

Direct represention 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 (LA)TeX users are starting to use such browsers.

Work in both the TeX4ht 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).

243 The TeXtrace project

TeXtrace 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 (see http://sourceforge.net/projects/textrace/) and there are already a few sets of fonts on CTAN generated using TeXtrace: 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 MusixTeX fonts.

A package that says it's "inspired" by TeXtrace 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

Type 1 versions of EC and TC fonts: fonts/ps-type1/ec

musixtex fonts: fonts/musixtex/ps-type1/musixps-unix.tar.gz

244 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* (see http://www.texmacs.org), 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 (see http://www.lyx.org/), 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* (see http://www.winedt.com/). Continuing the range of tools prettifying your input, we have the *emacs* package *x-symbol* (see http://x-symbol.sourceforge.net), 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 recompilations (through the use of dumped formats) is *WhizzyTeX* (see http://pauillac.inria.fr/whizzytex/) which is best used with a previewer by the same author. A simpler package in a similar spirit is called *InstantPreview* (see http://www.activetex.org/) 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* (see http://preview-latex.sourceforge.net) 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 TeXlite. 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 correspondance of the display to the final output, and the balance they strike between visual aid and visual distraction.

preview-latex: Browse support/preview-latex

texmacs: Browse systems/unix/TeXmacs

X You're still stuck?

245 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 (see mailto:faq@tex.ac.uk) 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!)

246 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 (see `mailto:faq@tex.ac.uk`).

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 `texfaq2html` in the distribution: this isn't (any longer) the program actually used on the Web site, but it's kept up to date with that version, for testing purposes.

FAQ distribution: `help/uk-tex-faq`

247 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 (see `http://www.latex-project.org/cgi-bin/ltxbugs2html?introduction=yes`), 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 (see `http://www.latex-project.org/bugs-upload.html`) and enter details of your bug — category, summary and full description, and the two important files (source and log file); note that 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.

248 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$_\varepsilon$, you may report it (see question 247) 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.tex.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`).