The UK TeX FAQ
Your 459 Questions Answered
version 3.25, date 2012-03-29

March 30, 2012

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

Introduction

This is a set of Frequently Asked Questions (FAQ) for English-speaking users of TeX.

The questions answered here cover a wide range of topics, but typesetting issues are mostly covered from the viewpoint of a LaTeX user. Some of the questions answered have little relevance to today's users; this is inevitable — it's easier to add information than it is to decide that information is no longer needed. The set of answered questions is therefore in a state of slow flux: new questions are answered, while old questions are deleted . . . but the whole process depends on the time available for FAQ maintenance.

You may use the FAQ

- by reading a printed document,
- by viewing a PDF file, with hyperlinks to assist browsing: copies are available formatted so that they could be printed on A4 paper or on North American "letter" paper,
- by using the FAQ's web interface (base URL: `http://www.tex.ac.uk/faq`); this version provides simple search capabilities, as well as a link to Google for a more sophisticated search restricted to the FAQ itself, or
- via Scott Pakin's Visual FAQ, which shows LaTeX constructions with links to FAQ explanations of how they may be created.

Finding the Files

Unless otherwise specified, all files mentioned in this FAQ are available from a CTAN archive, or from a mirror of CTAN — see later discussion of the CTAN archives and how to retrieve files from them.

The reader should also note that the first directory name of the path name of every file on CTAN is omitted from what follows, for the simple reason that, while it's always the same (`tex-archive/`) on the main sites, mirror sites often choose something else.

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

Origins

The 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.

The following people (at least — there are almost certainly others whose names weren't correctly recorded) have contributed help or advice on the development of the FAQ: William Adams, Donald Arseneau, Rosemary Bailey, Barbara Beeton, Karl Berry, Giuseppe Bilotta, Charles Cameron, François Charette, Damian Cugley, Michael Dewey, Michael Downes, Jean-Pierre Drucbert, David Epstein, Michael Ernst, Thomas Esser, Ulrike Fischer, Bruno Le Floch, Anthony Goreham, Norman Gray, Enrico Gregorio, Werner Grundlingh, Eitan Gurari, William Hammond, John Hammond, John Harper, Gernot Hassenpflug, Troy Henderson, Hartmut Henkel, Stephan Hennig, John Hobby, Morten Høgholm, Berthold Horn, Ian Hutchinson, Werner Icking, William Ingram, David Jansen, Alan Jeffrey, Regnor Jernsletten, David Kastrup, Oleg Katsitadze, Isaac Khabaza, Ulrich Klauer, Markus Kohm, Stefan Kottwitz David Kraus, Ryszard Kubiak, Simon Law, Uwe Lück, Daniel Luecking, Aditya Mahajan, Sanjoy Mahajan, Diego Andres Alvarez Marin, Andreas Matthias, Steve Mayer, Brooks Moses, Peter Moulder, Iain Murray, Vilar Camara Neto, Dick Nickalls, Ted Nieland, Hans Nordhaug, Pat Rau, Heiko Oberdiek, Piet van Oostrum, Scott Pakin, Oren Patashnik, Manuel Pégourié-Gonnard, Steve Peter, Sebastian Rahtz, Philip Ratcliffe, Chris Rowley, José Carlos Santos, Walter Schmidt, Hans-Peter Schröcker, Joachim Schrod, Uwe Siart, Maarten Sneep, Axel Sommerfeldt, Philipp Stephani, James Szinger, Nicola Talbot, Bob Tennent, Tomek Trzeciak, Ulrik Vieth, Mike Vulis, Chris Walker, Peter Wilson, Joseph Wright, Rick Zaccone, Gregor Zattler and Reinhard Zierke.

That list does *not* cover the many people whose ideas were encountered on various lists, newsgroups or (lately) web forums. Many such people have helped, even if simply to highlight an area in which FAQ work would be useful.

A The Background

1 Getting started

(La)TeX offers a very large number of choices, and the beginner has to navigate through that set before even taking his first steps. The aim here is to guide the beginner through a set of answers that may help the process. We assume the beginner 'knows' that (La)TeX is for them; if not, the discussion "what is TeX" may help.

To start at the very beginning, then, the beginner may wish to know what all those "things with TeX in their name are".

Armed with that knowledge, the beginner needs to decide what macro format she needs (always assuming someone hasn't already told her she needs to use "fooTeX". Learn about the alternatives in answers discussing common formats: look at writing Plain TeX, LaTeX or ConTeXt.

If no system has been provided, the beginner needs to acquire a TeX distribution appropriate to their machine. Available options are available via the answer "(La)TeX for various machines"; we assume here that the beginner can install things for herself, or has access to someone who can.

Finally, the beginner needs to get started in the chosen format. For Plain TeX, see "Online introductions: TeX"; for LaTeX, see "Online introductions: LaTeX"; and for ConTeXt, the place to start is the ConTeXt garden wiki (which is so good the present FAQs don't even try to compete).

2 What is TeX?

TeX is a typesetting system written by Donald E. Knuth, who says in the Preface to his book on TeX (see books about TeX) that it is "*intended for the creation of beautiful books — and especially for books that contain a lot of mathematics*". (If TeX were *only* good for mathematical books, much of its use nowadays would not happen: it's actually a pretty good general typesetting system.)

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

Knuth developed a system of 'literate programming' 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 is (in principle) 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 by means of which one may judge the fidelity of a TeX system. TeX and its documents are therefore highly portable.

For the interested programmer, the distribution of TeX has some fascination: it's nothing like the way one would construct such a program nowadays, yet it has lasted better than most, and has been ported to many different computer architectures and operating systems — the sorts of attributes that much modern programming practice aims for. The processed 'readable' source of TeX the program may be found in the TDS structured version of the distribution.

Knuth's source distribution: `systems/knuth/dist`

Knuth's sources in TDS layout: `macros/latex/contrib/latex-tds/knuth.tds.zip`

3 What's "writing in TeX"?

TeX is a macro processor, and offers its users a powerful programming capability. To produce a document, you write macros and text interleaved with each other. The macros define an environment in which the text is to be typeset.

However, the basic TeX engine is pretty basic, and is a pretty difficult beast to deal with. Recognising this (and not wanting to write the same things at the start of every document, himself) Knuth provided a package of macros for use with TeX, called Plain TeX; Plain TeX is a useful minimum set of macros that can be used with TeX, together with some demonstration versions of higher-level commands. When people say they're "writing (or programming) in TeX", they usually mean they're programming in Plain TeX.

4 How should I pronounce "TeX"?

The 'X' is "really" the Greek letter Chi (χ, in lower case), and is pronounced by English-speakers either a bit like the 'ch' in the Scots word 'loch' ([x] in the IPA) or (at a pinch, if you can't do the Greek sound) like 'k'. It definitely is not pronounced 'ks' (the Greek letter with that sound doesn't look remotely like the Latin alphabet 'X').

This curious usage derives from Knuth's explanation in the TeXbook that the name comes from the Greek word for 'art' or 'craft' ('$\tau\epsilon\chi\nu\eta$'), which is the root of the English word 'technology'. Knuth's logo for TeX is merely the uppercase version of the first three (Greek) letters of the word, jiggled about a bit; we don't use that logo (and logos like it) in this FAQ (see Typesetting TeX-related logos).

5 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 creates bitmaps that may be 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 (even) than is TeX macro-writing. What is more, it is a dying art: few new TeX-related fonts are produced using Metafont, nowadays. Indeed, several of the major font families (that originated in Metafont designs) are now seldom used in any other way than their conversion to an outline font format.

6 What is Metapost?

The Metapost system (by John Hobby) implements a picture-drawing language very much like that of Metafont; the difference is that Metapost 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. Metapost is able to integrate text and mathematics, marked up for use with TeX, within the graphics. (Knuth tells us that he uses nothing but Metapost for diagrams in text that he is writing.)

Although PDFLaTeX cannot ordinarily handle PostScript graphics, the output of Metapost is sufficiently simple and regular that PDFLaTeX can handle it direct, using code borrowed from ConTeXt — see graphics in PDFLaTeX.

Much of Metapost's source code was copied from Metafont's sources, with Knuth's permission.

A mailing list discussing Metapost is available; subscribe via the TUG *mailman* interface. The TUG website also hosts a Metapost summary page, and the *tex-overview* document gives you a lot more detail (and some explanatory background material).

tex-overview.pdf: info/tex-overview

7 Things with "TeX" in the name

New TeX users are often baffled by the myriad terms with "TeX" in the name. The goal of this answer is to clarify some of the more common such terms.

TeX itself TeX proper is a typesetting system based on a set of low-level control sequences that instruct TeX how to lay out text on the page. For example, `\hskip` inserts a given amount of horizontal space into the document, and `\font` makes a given font available in a document. TeX is fully programmable using an integrated macro scripting language that supports variables, scoping, conditional execution, control flow, and function (really, macro) definitions. See what is TeX? for some background information on TeX and some reference documents for pointers to descriptions of TeX control sequences, data types, and other key parts of TeX.

TeX macro packages (a.k.a. TeX formats) Some of TeX's control sequences are tedious to use directly; they are intended primarily as building blocks for higher-level — and therefore more user-friendly — abstractions. For example, there is no way in base TeX to specify that a piece of text should be typeset in a larger font. Instead, one must keep track of the current size and typeface, load a new font with the same typeface but a (specified) larger size, and tell TeX to use that new font until instructed otherwise. Fortunately, because TeX is programmable, it is possible to write a macro that hides this complexity behind a simple, new control sequence. (For example, it is possible to define `\larger{my text}` to typeset "my text" in at a font size next larger than the current one.)

While some users write their own, perfectly customized set of macros — which they then typically reuse across many documents — it is far more common to rely upon a *macro package*, a collection of TeX macros written by experts. For the user's convenience, these macro packages are often combined with the base TeX engine into a standalone executable. The following are some of that macro packages that you are likely to encounter:

Plain TeX (executable: *tex*) See Books on TeX and Plain TeX, Online introductions: TeX, Should I use Plain TeX or LaTeX? and Freely available (La)TeX books. Note that the Plain TeX executable is called *tex*; the base TeX engine is generally provided by a separate executable such as *initex* or as a `-ini` flag to *tex*.

LaTeX (executable: *latex*) See Books on TeX and its relations, (La)TeX Tutorials, etc., Online introductions: LaTeX and Specialized (La)TeX tutorials. Note that there have been two major versions of LaTeX: LaTeX2e refers to the current version of LaTeX while LaTeX 2.09 is the long-since-obsolete (since 1994) version (cf. What is LaTeX2e? for more information).

ConTeXt (executable: *texmfstart*) See What is ConTeXt?.

Texinfo (executables: *tex*, *makeinfo*) See What is Texinfo?. *makeinfo* converts Texinfo documents to HTML, DocBook, Emacs info, XML, and plain text. *Tex* (or wrappers such as *texi2dvi* and *texi2pdf*) produce one of TeX's usual output formats such as DVI or PDF. Because *tex* loads the Plain TeX macros, not the Texinfo ones, a Texinfo document must begin with

```
\input texinfo
```

explicitly load the Texinfo macro package.

Eplain — Extended Plain TeX (executable: *eplain*) See What is Eplain?.

Modified *tex* executables The original *tex* executable was produced in the late 1970s (cf. What is TeX?) and consequently lacked some features that users have come to expect from today's software. The following programs address these issues by augmenting the TeX engine with some additional useful features:

PDFTeX (executable: *pdftex*) TeX, which predates the PDF file format by a decade, outputs files in a TeX-specific format called DVI (cf. What is a DVI file?). In contrast, PDFTeX can output both DVI *and* PDF files. In PDF mode, it lets documents exploit various PDF features such as hyperlinks, bookmarks, and annotations, PDFTeX additionally supports two sophisticated micro-typographic features: character protrusion and font expansion. See What is PDFTeX?.

XeTeX (executable: *xetex*) XeTeX reads UTF-8 encoded Unicode input, and extends TeX's font support to include 'modern' formats such as TrueType and OpenType;

these extensions to its capabilities make it well-suited to multi-lingual texts covering different writing systems. See What is XeTeX?.

LuaTeX (executable: *luatex*) TeX is programmed in its own arcane, integrated, macro-based programming language. LuaTeX adds a second programming engine using a modern scripting language, Lua, which is 'embedded' in a TeX-alike engine; it too reads UTF-8 and uses TrueType OpenType fonts. See What is LuaTeX?.

e-TeX (executable: *etex*) e-TeX is an extension of TeX's programming interface; as such it's only indirectly useful to end users, but it can be valuable to package developers; there is an increasing number of macro packages that require the use of e-TeX. As well as existing in *etex*, e-TeX features are usually available in the *pdftex* executables provided in the standard distributions; XeTeX and LuaTeX also provide e-TeX's programming facilities. See What is e-TeX?.
(Note: e-TeX, which enhances the TeX engine, is not to be confused with Eplain, which enhances the Plain TeX macro package.)

Because each of the above derive from a base TeX engine, it is in principle possible to combine any of them with one of the TeX macro packages listed earlier to produce 'extended' executables. For example, the *pdflatex*, *xelatex* and *lualatex* executables each combine LaTeX with an enhanced TeX engine. Indeed, most (if not all) of the development of ConTeXt is now using LuaTeX.

Some executables combine the features of multiple enhanced TeX engines: for example, *pdftex* now (in current distributions) offers both PDFTeX and e-TeX extensions into a single executable This executable may be offered with a LaTeX format (as *latex* or *pdflatex*) or with a Plain TeX format (as *pdftex*). (*Tex* remains with an unadorned TeX executable using Plain TeX, for people such as Knuth himself, who want the certainty of the "original".)

TeX distributions A *TeX distribution* provides a structured collection of TeX-related software. Generally, a TeX distribution includes a set of "core" TeX executables such as *tex* and *latex*; various fonts optimized for use with TeX; helper programs such as the BibTeX bibliographic-database formatter, editors, integrated development environments, file-format-conversion programs; numerous LaTeX packages; configuration tools; and any other goodies the distributor chooses to include.

Commonly encountered TeX distributions include TeX Live, MiKTeX and MacTeX; older ones include ozTeX, CMacTeX and teTeX. MiKTeX is also available as the basis of the ProTeXt bundle, distributed on the TeX Live DVD mailing, as well as being available online.

Some TeX distributions target a specific operating system and/or processor architecture; others run on multiple platforms. Many TeX distributions are free; a few require payment. See (La)TeX for different machines for a list of free and shareware TeX distributions and Commercial TeX implementations for a list of commercial TeX distributions.

Summary What does it all mean? — the simple lists of objects, alone, offer no help for the beginner. The FAQ team expects this answer only to be of use for people who are seeking guidance elsewhere (possibly within these FAQs) and coming across an unexpected name like "blahTeX".

The subject matter covered by this answer is also addressed in a page on the TUG site, "the Levels of TeX".

8 What is CTAN?

The acronym stands for "Comprehensive TeX Archive Network", which more-or-less specifies what it's *for*:

- The aim is to offer a comprehensive collection of TeX resources.
- The content is to be made publicly accessible, via the internet.
- CTAN is a *network* of archives, which strive to stay in step with one another.

The basic framework was developed by a TUG working group set up to resolve the (then existing) requirement for users to *know* on which archive site a particular package might be found.

Actual implementation offers three distinct types of host:

***Core* archives** Which form a small, tightly-coupled set of machines, which perform management functions as well as serving files, and

***Mirror* archives** Which do no more than take regular copies of core archives.

Archive selector Which is a meta-service, which routes requests to an apparently "local" source (mirror or archive).

Note that there is nothing to prevent any archive from supporting other functions, so a CTAN mirror may also operate as a CPAN (Perl) mirror and as a SourceForge (general free software) mirror, and . . .

Functions that distinguish core archives are:

- Uploads: users may submit new (or updated) material, and significant changes to the archive are reported via the mailing list `ctan-ann@dante.de`
- Weak consistency: changes to the content of the archives are rapidly distributed to all core archives.
- Providing distribution (TeX Live and MiKTeX) support.
- Catalogue maintenance.
- Mirror monitoring.

Not all core archives offer all of these functions.

Users may contact the maintainers of the core archives via the mailing list `ctan@dante.de`

Users who have new material for the archive may submit it using the German or UK archive.

Users should ordinarily download material from CTAN via the archive selector: this uses the mirror monitor's database, and uses the caller's geographical location to offer an efficient choice of "sufficiently up-to-date" mirror site for you to connect to. Note that all the download links, given in the web representation of these FAQs, are set up to use the mirror selector.

9 The (CTAN) catalogue

Finding stuff on networks was always difficult, but in recent years, search engines have become amazingly good at digging out unconsidered trifles from the myriad items of information available on the net.

Sadly, in the TeX context, search engines seem to excel in locating out-of-date material, while the service users need a means of finding relevant material on CTAN — on the grounds that such material is most likely to be up-to-date. The need to find such items, thus alleviating tendency to end up with out-of-date (La)TeX material, was the motivation for developing a specialised information source: the CTAN catalogue. The aim is that they contain enough relevant information that they will appear early in a search engine's evaluation.

The basis of the catalogue is a collection of small articles; each shows basic information about a package on CTAN, and includes pointers to download address, documentation and related packages. Every CTAN mirror holds a copy of the catalogue, presented as a series of web pages; indexes (both alphabetic and category-based) are provided.

The core CTAN sites also offer a simple text search of the catalogue. This is a tolerable means of finding a package you need, but it is not really a substitute for a good information retrieval system: the problem of providing such a system has not yet been solved.

The CTAN catalogue: `help/Catalogue`

10 How can I be sure it's really TeX?

TeX (and Metafont and Metapost) are written in a 'literate' programming language called *Web* which is designed to be portable across a wide range of computer systems. How, then, is a new version of TeX checked?

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

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

Once an implementation of TeX has passed its *trip* test, or an implementation of Metafont has passed its *trap* test, then it may in principle be distributed as a working version. (In practice, any distributor would test new versions against "real" documents or fonts, too; while *trip* and *trap* test bits of pathways within the program, they don't actually test for any real world problem.)

11 Are TeX and friends Y2K compliant?

Crashing: None of TeX, Metafont or Metapost can themselves crash due to any change whatever in the date of any sort.

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

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

The \year primitive: Certification of a TeX implementation (see trip/trap testing) 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 TeX-related books) 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 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.

12 What is e-TeX?

While Knuth has declared that TeX will never change in any substantial way, there remain things that one might wish had been done differently, or indeed implemented at all.

The NTS project set out to produce an advanced replacement for TeX, to provide a basis for developing such modifications: this "New Typesetting System" would share Knuth's aims, but would implement the work in a modern way taking account of the lessons learned with TeX. While a first demonstrator NTS did appear, it wasn't practically useful, and the project seems no longer active.

In parallel with its work on NTS itself, the project developed a set of extensions that can be used with a ("true") TeX system. Such a modified system is known as an e-TeX system, and the concept has proved widely successful. Indeed, current TeX distributions are delivered with most formats built with an e-TeX-based system (for those who don't want them, e-TeX's extensions can be disabled, leaving a functionally standard TeX system).

The extensions range from the seemingly simple (increasing the number of available registers from 256 to 32768) through to extremely subtle programming support.

ConTeXt has required e-TeX for its operation for some time, though development is now focused on the use of LuaTeX.

Some LaTeX packages already specify the use of e-TeX. Some such packages may not work at all on a non-e-TeX system; others will work, but not as well as on an e-TeX system. The LaTeX team has announced that future LaTeX packages (specifically those from the team, as opposed to those individually contributed) may require e-TeX for optimum performance.

e-TeX: `systems/e-tex`

13 What is PDFTeX?

One can reasonably say that PDFTeX is (today) the main stream of TeX distributions: most LaTeX and many ConTeXt users nowadays use PDFTeX whether they know it or not (more precisely, they use PDFTeX extended by e-TeX). So what is PDFTeX?

PDFTeX is a development of TeX that is capable of generating typeset PDF output in place of DVI. PDFTeX has other capabilities, most notably in the area of fine typographic detail (for example, its support for optimising line breaks), but its greatest impact to date has been in the area of PDF output.

PDFTeX started as a topic for Hàn Thế Thành's Master's thesis, and seems first to have been published in *TUGboat* **18**(4), in 1997 (though it was certainly discussed at the TUG'96 conference in Russia).

While the world was making good use of "pre-releases" of PDFTeX, Thành used it as a test-bed for the micro-typography which was the prime subject of his Ph.D. research. Since Thành was finally awarded his Ph.D., day-to-day maintenance and development of PDFTeX 1.0 (and later) has been in the hands of a group of PDFTeX maintainers (which includes Thành); the group has managed to maintain a stable platform for general use.

Development of PDFTeX has mostly stopped (only bug fixes, and occasional small development items are processed): future development is focused on LuaTeX.

14 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.

15 What is LaTeX2e?

Lamport's last version of LaTeX (LaTeX 2.09, last updated in 1992) was superseded in 1994 by a new version (LaTeX2e) provided by the LaTeX team. LaTeX2e 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. The "e" of the name is (in the official logo) a single-stroke epsilon (ε, supposedly indicative of no more than a small change).

LaTeX2e 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. LaTeX2e 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, and subject to voluminous complaints in the log file). Differences between LaTeX2e and LaTeX 2.09 are outlined in a series of 'guide' files that are available in every LaTeX distribution (the same directory also contains "news" about each new release of LaTeX2e).

Note that, now, LaTeX2e is "feature frozen" (the only allowed changes come from bug reports); this, too, is in pursuit of stability, and is a driving force for many of the efforts to contribute LaTeX packages.

LaTeX guides and news: `macros/latex/doc`

16 How should I pronounce "LaTeX(2e)"?

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 the rules for TeX). It is definitely *not* to be pronounced in the same way as the rubber-tree gum (which would be 'lay teks').

The LaTeX2e logo is supposed to end with an ε; nevertheless, most people pronounce the name as 'LaTeX-two-ee'.

17 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 (mental) effort.

The arguments around this topic can provoke huge amounts of noise and heat, without offering much by way of light; your best bet may be to find out what those around you are using, and to follow them in the hope of some support. 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 (La)TeX styles for journals and books, and insist that authors stick closely to their markup.

18 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 — see a later answer about translating to Plain TeX.

19 What is ConTeXt?

ConTeXt is a macro package developed by Hans Hagen of Pragma-Ade; it started as a production tool for Pragma (which is a publishing company). ConTeXt is a document-production system based, like LaTeX, on the TeX typesetting system. Whereas LaTeX insulates the writer from typographical details, ConTeXt takes a complementary approach by providing structured interfaces for handling typography, including extensive support for colors, backgrounds, hyperlinks, presentations, figure-text integration, and conditional compilation. It gives the user extensive control over formatting while making it easy to create new layouts and styles without learning the TeX macro language. ConTeXt's unified design avoids the package clashes that can happen with LaTeX.

ConTeXt also integrates MetaFun, a superset of Metapost and a powerful system for vector graphics. MetaFun can be used as a stand-alone system to produce figures, but its strength lies in enhancing ConTeXt documents with accurate graphic elements.

ConTeXt allows the users to specify formatting commands in English, Dutch, German, French, or Italian, and to use different typesetting engines (PDFTeX, XeTeX, Aleph and LuaTeX) without changing the user interface. ConTeXt continues to develop, often in response to requests from the user community.

ConTeXt has been bound to the development of LuaTeX, almost from the start of that project. Nowadays, it is distributed in two versions — mark two (files with extension `.mkii` (which runs on PDFTeX but is not under active development) and mark four files with extension `.mkiv` (which runs on LuaTeX and is where development happens).

ConTeXt has a large developer community (though possibly not as large as that of latex), but those developers who are active seem to have prodigious energy. Support is available via a WIKI site and via the mailing list; a "standalone" distribution (a TeX distribution with no macros other than ConTeXt-based ones) is available from `http://wiki.contextgarden.net/ConTeXt_Standalone` — it provides a ConTeXt system on any of a number of platforms, executing either mark ii or mark iv ConTeXt.

ConTeXt distribution: `macros/context`

20 What are the AMS packages (AMSTeX, *etc*.)?

AMSTeX is a TeX macro package, originally written by Michael Spivak for the American Mathematical Society (AMS) during 1983–1985 and is described in the book "The Joy of TeX". 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 sought to submit papers to the AMS in LaTeX, so AMSLaTeX was developed. AMSLaTeX is a collection of LaTeX packages and classes that offer authors most of the functionality of AMSTeX. The AMS no longer recommends the use of AMSTeX, and urges its authors to use AMSLaTeX instead.

AMSTeX distribution: `macros/amstex`

AMSLaTeX distribution: `macros/latex/required/amslatex`

21 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 ConTeXt, LaTeX or Texinfo. Instead, it defines macro tools that should be useful whatever commands you choose to use when you 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` does. 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. The authors of Eplain believe it is the only generally available macro package that does not force its typographic style on an author, and yet provides these capabilities.

Another useful feature of Eplain is the ability to create PDF files with hyperlinks. The cross-referencing macros can implicitly generate hyperlinks for the cross-references, but you can also create explicit hyperlinks, both internal (pointing to a destination within the current document) and external (pointing to another local document or a URL).

Several LaTeX packages provide capabilities which Plain TeX users are lacking, most notably text colouring and rotation provided by the *graphics* bundle (packages *color* and *graphics*). Although the *graphics* bundle provides a Plain TeX "loader" for some of the packages, it is not a trivial job to pass options to those packages under Plain TeX, and much of the functionality of the packages is accessed through package options. Eplain extends the loader so that options can be passed to the packages just as they are in LaTeX. The following packages are known to work with Eplain: *graphics*, *graphicx*, *color*, *autopict* (LaTeX picture environment), *psfrag*, and *url*.

Eplain documentation is available online (there's a PDF copy in the CTAN package as well), and there's also a mailing list — sign up, or browse the list archives, via `http://tug.org/mailman/listinfo/tex-eplain`

Eplain distribution: `macros/eplain`

22 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. By convention, Texinfo source file names end with a `.texi` or `.texinfo` extension.

Texinfo is a macro language, somewhat similar to LaTeX, but with slightly less expressive power. Its appearance is of course rather similar to any TeX-based macro language, except that its macros start with `@` rather than the \ that's more commonly used in TeX systems.

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: the language is a far better fit to HTML than is LaTeX, so that the breast-beating agonies of converting LaTeX to HTML are largely avoided.

Finally, of course, you can also print the files, or convert them to PDF using PDFTeX.

Texinfo distribution: `macros/texinfo/texinfo`

23 If TeX is so good, how come it's free?

It's free because Knuth chose to make it so (he makes money from royalties on his TeX books, which all sell well. 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 (GPL, sometimes referred to as 'Copyleft'), TeX itself is not subject to GPL.

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.

24 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 3.1415926). 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 2.718281.)

Knuth explains his decision, and exhorts us all to respect it, in a paper originally published in *TUGboat* **11**(4), and reprinted in the NTG journal MAPS.

There are projects (some of them long-term projects: see, for example, the LaTeX3 project) to build substantial new macro packages based on TeX. For the even longer term, there are various projects to build a *successor* to TeX. The e-TeX extension to TeX itself arose from such a project (NTS). Another pair of projects, which have delivered all the results they are likely to deliver, is the related Omega and Aleph. The XeTeX system is in principle still under development, but is widely used, and the LuaTeX project (though not scheduled to produce its final product until end 2012) has already delivered a system that is quite useful.

25 Reading (La)TeX files

So you've been sent a TeX file: what are you going to do with it?

You can, of course, "just read it", since it's a plain text file, but the markup tags in the document may prove distracting. Most of the time, even TeX experts will typeset a TeX file before attempting to read it.

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

But beware — TeX makes no attempt to look like the sort of WYSIWYG system you're probably used to (see "why is TeX not WYSIWYG"): 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 *markup*, which typically defines a logical (rather than a visual) view of what you want typeset.

So there's a balance between the simplicity of the original (marked-up) document, which can more-or-less be read in *any* editor, and the really rather large investment needed to install a system to read a document "as intended".

26 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.

The issue has of recent years started to attract attention from TeX developers, and several interesting projects addressing the "TeX document preparation environment" 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.

27 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 regular publication 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 also available.

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).

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

Contact TUG itself via `http://tug.org/contact` LastEdit2011-09-19

B Documentation and Help

28 Books relevant to TeX and friends

There are too many books for them all to appear in a single list, so the following answers aim to cover "related" books, with subject matter as follows:

- TeX itself and Plain TeX
- LaTeX
- Books on other TeX-related matters
- Books on Type

These lists only cover books in English: notices of new books, or warnings that books are now out of print are always welcome. However, these FAQs do *not* carry reviews of current published material.

29 Books on TeX, Plain TeX and relations

While Knuth's book is the definitive reference for both TeX and Plain TeX, there are many books covering these topics:

The TeXbook by Donald Knuth (Addison-Wesley, 1984, ISBN-10 0-201-13447-0, paperback ISBN-10 0-201-13448-9)

A Beginner's Book of TeX by Raymond Seroul and Silvio Levy, (Springer Verlag, 1992, ISBN-10 0-387-97562-4)

TeX by Example: A Beginner's Guide by Arvind Borde (Academic Press, 1992, ISBN-10 0-12-117650-9 — now out of print)

Introduction to TeX by Norbert Schwarz (Addison-Wesley, 1989, ISBN-10 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))

A TeX Primer for Scientists by Stanley Sawyer and Steven Krantz (CRC Press, 1994, ISBN-10 0-849-37159-7)

TeX by Topic by Victor Eijkhout (Addison-Wesley, 1992, ISBN-10 0-201-56882-9 — now out of print, but see online books; you can also now buy a copy printed, on demand, by Lulu — see `http://www.lulu.com/content/2555607`)

TeX for the Beginner by Wynter Snow (Addison-Wesley, 1992, ISBN-10 0-201-54799-6)

TeX for the Impatient by Paul W. Abrahams, Karl Berry and Kathryn A. Hargreaves (Addison-Wesley, 1990, ISBN-10 0-201-51375-7 — now out of print, but see online books)

TeX in Practice by Stephan von Bechtolsheim (Springer Verlag, 1993, 4 volumes, ISBN-10 3-540-97296-X for the set, or Vol. 1: ISBN-10 0-387-97595-0, Vol. 2: ISBN-10 0-387-97596-9, Vol. 3: ISBN-10 0-387-97597-7, and Vol. 4: ISBN-10 0-387-97598-5)

TeX: Starting from $\boxed{1}$[1] by Michael Doob (Springer Verlag, 1993, ISBN-10 3-540-56441-1 — now out of print)

The Joy of TeX by Michael D. Spivak (second edition, AMS, 1990, ISBN-10 0-821-82997-1)

The Advanced TeXbook by David Salomon (Springer Verlag, 1995, ISBN-10 0-387-94556-3)

A collection of Knuth's publications about typography is also available:

Digital Typography by Donald Knuth (CSLI and Cambridge University Press, 1999, ISBN-10 1-57586-011-2, paperback ISBN-10 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-10 0-201-73416-8).

30 Books on LaTeX

LaTeX, a Document Preparation System by Leslie Lamport (second edition, Addison Wesley, 1994, ISBN-10 0-201-52983-1)

Guide to LaTeX Helmut Kopka and Patrick W. Daly (fourth edition, Addison-Wesley, 2004, ISBN-10 0-321-17385-6)

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

LaTeX Beginner's Guide by Stefan Kottwitz (Packt Publishing, 2011, ISBN-10 1847199860, ISBN-13 978-1847199867)

The LaTeX Companion by Frank Mittelbach, Michel Goossens, Johannes Braams, David Carlisle and Chris Rowley (second edition, Addison-Wesley, 2004, ISBN-10 0-201-36299-6, ISBN-13 978-0-201-36299-2)

The LaTeX Graphics Companion: Illustrating documents with TeX and PostScript by Michel Goossens, Sebastian Rahtz, Frank Mittelbach, Denis Roegel and Herbert Voß (second edition, Addison-Wesley, 2007, ISBN-10 0-321-50892-0, ISBN-13 978-0-321-50892-8)

The LaTeX Web Companion: Integrating TeX, HTML and XML by Michel Goossens and Sebastian Rahtz (Addison-Wesley, 1999, ISBN-10 0-201-43311-7)

TeX Unbound: LaTeX and TeX strategies for fonts, graphics, and more by Alan Hoenig (Oxford University Press, 1998, ISBN-10 0-19-509685-1 hardback, ISBN-10 0-19-509686-X paperback)

*More Math into LaTeX: An Introduction to LaTeX and AMSLaTeX*by George Grätzer (fourth edition Springer Verlag, 2007, ISBN-10 978-0-387-32289-6

Digital Typography Using LaTeX Incorporating some multilingual aspects, and use of Omega, by Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniou (Springer, 2003, ISBN-10 0-387-95217-9).

First Steps in LaTeX by George Grätzer (Birkhäuser, 1999, ISBN-10 0-8176-4132-7)

LaTeX: Line by Line: Tips and Techniques for Document Processing by Antoni Diller (second edition, John Wiley & Sons, 1999, ISBN-10 0-471-97918-X)

LaTeX for Linux: A Vade Mecum by Bernice Sacks Lipkin (Springer-Verlag, 1999, ISBN-10 0-387-98708-8, second printing)

Typesetting Mathematics with LaTeX by Herbert Voß (UIT Cambridge, 2010, ISBN-10 978-1-906-86017-2)

Typesetting Tables with LaTeX by Herbert Voß, (UIT Cambridge, 2011, ISBN-10 978-1-906-86025-7)

PSTricks: Graphics and PostScript for TeX and LaTeX by Herbert Voß, (UIT Cambridge, 2011, ISBN-10 978-1-906-86013-4)

A sample of George Grätzer's "Math into LaTeX", in Adobe Acrobat format, and example files for the three LaTeX Companions, and for Grätzer's "First Steps in LaTeX", are all available on CTAN.

Examples for First Steps in LaTeX: `info/examples/FirstSteps`

Examples for LaTeX Companion: `info/examples/tlc2`

Examples for LaTeX Graphics Companion: `info/examples/lgc`

Examples for LaTeX Web Companion: `info/examples/lwc`

Sample of Math into LaTeX: `info/mil/mil.pdf`

31 Books on other TeX-related matters

There's a nicely-presented list of of "recommended books" to be had on the web: `http://www.macrotex.net/texbooks/`

The list of Metafont books is rather short:

The Metafontbook by Donald Knuth (Addison Wesley, 1986, ISBN-10 0-201-13445-4, ISBN-10 0-201-52983-1 paperback)

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

A book covering a wide range of topics (including installation and maintenance) is:

Making TeX Work by Norman Walsh (O'Reilly and Associates, Inc, 1994, ISBN-10 1-56592-051-1)

The book is decidedly dated, and is now out of print, but a copy is available via `sourceforge` and on CTAN, and we list it under "online books".

32 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-10 0-88179-033-8)

Finer Points in the Spacing & Arrangement of Type by Geoffrey Dowding (Hartley & Marks, 1996, ISBN-10 0-88179-119-9)

The Thames & Hudson Manual of Typography by Ruari McLean (Thames & Hudson, 1980, ISBN-10 0-500-68022-1)

The Form of the Book by Jan Tschichold (Lund Humphries, 1991, ISBN-10 0-85331-623-6)

Type & Layout by Colin Wheildon (Strathmore Press, 2006, ISBN-10 1-875750-22-3)

The Design of Books by Adrian Wilson (Chronicle Books, 1993, ISBN-10 0-8118-0304-X)

Optical Letter Spacing by David Kindersley and Lida Cardozo Kindersley (The Cardozo Kindersley Workshop 2001, ISBN-10 1-874426-139)

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-10 1-85958-160-9)

An Introduction to the History of Printing Types by Geoffrey Dowding (British Library, 1998, UK ISBN-10 0-7123-4563-9; USA ISBN-10 1-884718-44-2)

The Alphabet Abecedarium by Richard A. Firmage (David R. Godine, 1993, ISBN-10 0-87923-998-0)

The Alphabet and Elements of Lettering by Frederick Goudy (Dover, 1963, ISBN-10 0-486-20792-7)

Anatomy of a Typeface by Alexander Lawson (David R. Godine, 1990, ISBN-10 0-87923-338-8)

A Tally of Types by Stanley Morison (David R. Godine, 1999, ISBN-10 1-56792-004-7)

Counterpunch by Fred Smeijers (Hyphen, 1996, ISBN-10 0-907259-06-5)

Treasury of Alphabets and Lettering by Jan Tschichold (W. W. Norton, 1992, ISBN-10 0-393-70197-2)

A Short History of the Printed Word by Warren Chappell and Robert Bringhurst (Hartley & Marks, 1999, ISBN-10 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.

33 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`. The (long obsolete) last posted copy of that article is kept on CTAN for auld lang syne.

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

This FAQ and others are regularly mentioned, on `comp.text.tex` and elsewhere, in a "pointer FAQ" which is also saved at `http://tug.org/tex-ptr-faq`

A 2006 innovation from Scott Pakin is the "visual" LaTeX FAQ. This is a document with (mostly rubbish) text formatted so as to highlight things we discuss here, and providing Acrobat hyper-links to the relevant answers in this FAQ on the Web. The visual FAQ is provided in PDF format, on CTAN; it works best using Adobe Acrobat Reader 7 (or later); some features are missing with other readers, or with earlier versions of Acrobat Reader

Another excellent information source, available in English, is the (La)TeX navigator.

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

Herbert Voß's excellent LaTeX tips and tricks provides excellent advice on most topics one might imagine (though it's not strictly a FAQ) — highly recommended for most ordinary mortals' use.

The Open Directory Project (ODP) maintains a list of sources of (La)TeX help, including FAQs. View the TeX area at `http://dmoz.org/Computers/Software/Typesetting/TeX/`

Other non-English FAQs are available (off-CTAN):

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 A FAQ used to be posted regularly to `fr.comp.text.tex`, and is archived on CTAN — sadly, that effort seems to have fallen by the wayside.

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

Resources available on CTAN are:

Dante FAQ: `help/de-tex-faq`

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

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

Obsolete `comp.text.tex` *FAQ*: `obsolete/help/TeX,_LaTeX,_etc.:_Frequently_Asked_Questions_with_Answers`

The visual FAQ: `info/visualFAQ/visualFAQ.pdf`

34 Getting help online

We assume, here, that you have looked at all relevant FAQ answers you can find, you've looked in any books you have, and scanned relevant tutorials… and still you don't know what to do.

There are two more steps you can take before formulating a question to the TeX world at large.

First, (if you are seeking a particular package or program), start by looking on your own system: you might already have what you seek — the better TeX distributions provide a wide range of supporting material. The CTAN Catalogue can also identify packages that might help: you can search it, or you can browse it "by topic". Each catalogue entry has a brief description of the package, and links to known documentation on the net. In fact, a large proportion of CTAN package directories now include documentation, so it's often worth looking at the catalogue entry for a package you're considering using (where possible, each package link in the main body of these FAQs has a link to the relevant catalogue entry).

Failing that, look to see if anyone has solved the problem before; places where people ask are:

1. newsgroup `comp.text.tex`, whose "historical posts" are accessible via Google groups, and
2. the mailing list `texhax` via its archive, or via the 'Gmane' newsgroup `gmane.comp.tex.texhax`, which holds a *very* long history of the list. A long shot would be to search the archives of the mailing list's ancient posts on CTAN, which go back to the days when it was a digest: in those days, a question asked in one issue would only ever be answered in the next one.

If the "back question" searches fail, you must ask the world at large.

So, how do you like to ask questions? — the three available mechanisms are:

1. Mailing lists: there are various specialist mailing lists, but the place for 'general' (La)TeX queries is the `texhax` mailing list. Mail to `texhax@tug.org` to ask a question, but it's probably better to subscribe to the list (via `http://tug.org/mailman/listinfo/texhax`) first — not everyone will answer to you as well as to the list.
2. Newsgroup: to ask a question on `comp.text.tex`, you can use your own news client (if you have one), or use the "+ new post" button on `http://groups.google.com/group/comp.text.tex`.
3. Web forum: alternatives are: the "LaTeX community" site, which offers a variety of 'categories' to place your query, and the TeX, LaTeX and friends Q&A site ("Stack exchange").
   Stack exchange has a scheme for voting on the quality of answers (and hence of those who offer support). This arrangement is supposed to enable you to rank any answers that are posted.

Do **not** try mailing the LaTeX project team, the maintainers of the TeX Live or MiKTeX distributions or the maintainers of these FAQs for help; while all these addresses reach experienced (La)TeX users, no small group can possibly have expertise in every area of usage so that the "big" lists and forums are a far better bet.

`texhax` 'back copies': `digests/texhax`

35 Specialist mailing lists

The previous question, "getting help", talked of the two major forums in which (La)TeX, Metafont and Metapost are discussed; however, these aren't the only ones available.

The TUG web site offers many mailing lists other than just `texhax` via its mail list management page.

The French national TeX user group, Gutenberg, offers a Metafont (and, de facto, Metapost) mailing list, `metafont@ens.fr`: subscribe to it by sending a message

```
subscribe metafont
```

to `sympa@ens.fr`

(Note that there's also a Metapost-specific mailing list available via the TUG list server; in fact there's little danger of becoming confused by subscribing to both.)

Announcements of TeX-related installations on the CTAN archives are sent to the mailing list `ctan-ann`. Subscribe to the list via its *MailMan* web-site `https://lists.dante.de/mailman/listinfo/ctan-ann`; list archives are available at the same address. The list archives may also be browsed via `http://www.mail-archive.com/ctan-ann@dante.de/`, and an RSS feed is also available: `http://www.mail-archive.com/ctan-ann@dante.de/maillist.xml`

36 How to ask a question

You want help from the community at large; you've decided where you're going to ask your question, but how do you phrase it?

Excellent "general" advice (how to ask questions of anyone) is contained in Eric Raymond's article on the topic. Eric's an extremely self-confident person, and this comes through in his advice; but his guidelines are very good, even for us in the un-self-confident majority. It's important to remember that you don't have a right to advice from the world, but that if you express yourself well, you will usually find someone who will be pleased to help.

So how do you express yourself in the (La)TeX world? There aren't any comprehensive rules, but a few guidelines may help in the application of your own common sense.

- 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.

- 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: "I want to add the *glugtheory* package to my *mumbleTeX v12.0* distribution on the *Grobble 2024* operating system".
- 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". If you thought you knew how, but your attempts are failing, tell us what you've tried: "I've already tried installing the *elephant* in the *minicar* directory, and it didn't work, even after refreshing the filename database".
- If something's going wrong within (La)TeX, pretend you're submitting a LaTeX bug report, 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.
- Figures are special, of course. Sometimes the figure itself is *really* needed, but most problems may be demonstrated with a "figure substitute" in the form of a `\rule{width}{height}` command, for some value of ⟨*width*⟩ and ⟨*height*⟩. If the (real) figure is needed, don't try posting it: far better to put it on the web somewhere.
- Be as succinct as possible. Your helpers don't usually need to know *why* you're doing something, just *what* you're doing and where the problem is.

37 How to make a "minimum example"

Our advice on asking questions suggests that you prepare a "minimum example" (also commonly known as a "*minimal* example") of failing behaviour, as a sample to post with your question. If you have a problem in a two hundred page document, it may be unclear how to proceed from this problem to a succinct demonstration of your problem.

There are two valid approaches to this task: building up, and hacking down.

The "building up" process starts with a basic document structure (for LaTeX, this would have `\documentclass`, `\begin{document}`, `\end{document}`) and adds things. First to add is a paragraph or so around the actual point where the problem occurs. (It may prove difficult to find the actual line that's provoking the problem. If the original problem is an error, reviewing "the structure of TeX errors" may help.)

Note that there are things that can go wrong in one part of the document as a result of something in another part: the commonest is problems in the table of contents (from something in a section title, or whatever), or the list of ⟨*something*⟩ (from something in a `\caption`). In such a case, include the section title or caption (the caption probably needs the `figure` or `table` environment around it, but it *doesn't* need the figure or table itself).

If this file you've built up shows the problem already, then you're done. Otherwise, try adding packages; the optimum is a file with only one package in it, but you may find that the guilty package won't even load properly unless another package has been loaded. (Another common case is that package *A* only fails when package *B* has been loaded.)

The "hacking down" route starts with your entire document, and removes bits until the file no longer fails (and then of course restores the last thing removed). Don't forget to hack out any unnecessary packages, but mostly, the difficulty is choosing what to hack out of the body of the document; this is the mirror of the problem above, in the "building up" route.

If you've added a package (or more than one), add `\listfiles` to the preamble too: that way, LaTeX will produce a list of the packages you've used and their version numbers. This information may be useful evidence for people trying to help you.

What if your document simply doesn't fail in any of these cut-down scenarios? Whatever you do, don't post the whole of the document: if you can, it may be useful to make a copy available on the web somewhere: people will probably understand if it's impossible … or inadvisable, in the case of something confidential.

If the whole document is necessary, it could be that it's an overflow of some sort; the best you can do is to post the code "around" the error, and (of course) the full text of the error.

It may seem that all this work is rather excessive for preparing a simple post. There are two responses to that, both based on the relative inefficiency of asking a question on the internet.

First, preparing a minimum document very often leads *you* to the answer, without all the fuss of posting and looking for responses.

Second, your prime aim is to get an answer as quickly as possible; a well-prepared example stands a good chance of attracting an answer "in a single pass": if the person replying to your post finds she needs more information, you have to find that request, post again, and wait for your benefactor to produce a second response.

All things considered, a good example file can save you a day, for perhaps half an hour's effort invested.

The above advice, differently phrased, may also be read on the web at http://www.minimalbeispiel.de/mini-en.html; source of that article may be found at http://www.minimalbeispiel.de/, in both German and English.

38 Tutorials, etc., for TeX-based systems

From a situation where every (La)TeX user *had* to buy a book if she was not to find herself groping blindly along, we now have what almost amounts to an embarrassment of riches of online documentation. The following answers attempt to create lists of sources "by topic".

First we have introductory manuals, for Plain TeX and LaTeX.

Next comes a selection of "specialised" (La)TeX tutorials, each of which concentrates on a single LaTeX topic.

A small selection of reference documents (it would be good to have more) are listed in an answer of their own.

Next comes the (somewhat newer) field of TeX-related WIKIs.

A rather short list gives us a Typography style tutorial.

39 Online introductions: Plain TeX

Michael Doob's splendid 'Gentle Introduction' to Plain TeX (available on CTAN) has been stable for a very long time.

Another recommendable document is D. R. Wilkins' 'Getting started with TeX', available on the web at http://www.ntg.nl/doc/wilkins/pllong.pdf

Gentle Introduction: info/gentle/gentle.pdf

40 Online introductions: LaTeX

A pleasing little document, "Getting something out of LaTeX" is designed to give a feel of LaTeX to someone who's never used it at all. It's not a tutorial, merely helps the user to decide whether to go on to a tutorial, and thence to 'real' use of LaTeX.

Tobias Oetiker's '(Not so) Short Introduction to LaTeX2e', is regularly updated, as people suggest better ways of explaining things, etc. The introduction is available on CTAN, together with translations into a rather large set of languages.

Peter Flynn's "Beginner's LaTeX" (which started life as course material) is a pleasing read. A complete copy may be found on CTAN, but it may also be browsed over the web (http://mirrors.ctan.org/info/beginlatex/html/).

Harvey Greenberg's 'Simplified Introduction to LaTeX' was written for a lecture course, and is also available on CTAN (in PostScript only, unfortunately).

The fourth edition of George Grätzer's book "Math into LaTeX" contains a "short course" in LaTeX itself, and that course has been made publicly available on CTAN.

Philip Hirschhorn's "Getting up and running with AMSLaTeX" has a brief introduction to LaTeX itself, followed by a substantial introduction to the use of the AMS classes and the *amsmath* package and other things that are potentially of interest to those writing documents containing mathematics.

Edith Hodgen's LaTeX, a Braindump starts you from the ground up — giving a basic tutorial in the use of *Linux* to get you going (rather a large file. . .). Its parent site, David Friggens' documentation page is a useful collection of links in itself.

Andy Roberts' introductory material is a pleasing short introduction to the use of (La)TeX; some of the slides for *actual* tutorials are to be found on the page, as well.

D. R. Wilkins' 'Getting started with LaTeX' also looks good (it appears shorter — more of a primer — than some of the other offerings).

Chris Harrison's LaTeX tutorial presents basic LaTeX in a rather pleasing and straightforward way.

Nicola Talbot's LaTeX for complete novices does what it claims: the author teaches LaTeX at the University of East Anglia. That is one of several introductory tutorials, which include exercises (with solutions). Other tutorials include those for writing theses/dissertations with LaTeX, and for using LaTeX in administrative work. (The page seems to be an adjunct to her LaTeX courses — fortunate people, the people of UEA!).

Engelbert Buxbaum provides the 'slides' for his LaTeX course 'The LaTeX document preparation system'; this seems to be a departmental course at his university.

Mark van Dongen's 'LaTeXand friends' appeared as he was writing his book on the subject (soon to be published).

An interesting (and practical) tutorial about what *not* to do is *l2tabu*, or "A list of sins of LaTeX2e users" by Mark Trettin, translated into English by Jürgen Fenn. The tutorial is available from CTAN as a PDF file (though the source is also available).

Beginner's LaTeX: `info/beginlatex/beginlatex-3.6.pdf`

Getting something out of LaTeX: `info/first-latex-doc`

Getting up and running with AMSLaTeX: `info/amslatex/primer`

Slides for LaTeX course: `info/latex-course`

Not so Short Introduction: `info/lshort/english` (in English, you may browse for sources and other language versions at `info/lshort`)

Simplified LaTeX: `info/simplified-latex/simplified-intro.pdf`

Short Course in LaTeX: `info/Math_into_LaTeX-4/Short_Course.pdf`

The sins of LaTeX users: `info/l2tabu/english/l2tabuen.pdf`; source also available: `info/l2tabu/english/l2tabuen.tex`

41 (La)TeX tutorials

The AMS publishes a "Short Math Guide for LaTeX", which is available (in several formats) via `http://www.ams.org/tex/amslatex.html` (the "Additional Documentation" about half-way down the page.

Herbert Voß has written an extensive guide to mathematics in LaTeX, and a development of it has been published as a book.

Two documents written more than ten years apart about font usage in TeX are worth reading: Essential NFSS by Sebastian Rahtz, and Font selection in LaTeX, cast in the form of an FAQ, by Walter Schmidt. A general compendium of font information (including the two above) may be found on the TUG web site.

TUG India is developing a series of online LaTeX tutorials which can be strongly recommended: select single chapters at a time from `http://www.tug.org/tutorials/tugindia` — there are 17 chapters in the series, two of which are mostly introductory.

Peter Smith's "LaTeX for Logicians" page covers a rather smaller subject area, but is similarly comprehensive (mostly by links to documents on relevant topics, rather than as a monolithic document).

Keith Reckdahl's "Using Imported Graphics in LaTeX2e" (*epslatex*) is an excellent introduction to graphics use. It's available on CTAN, but not in the TeX Live or miktex distributions, for lack of sources.

Stefan Kottwitz manages a web site devoted to the use of the drawing packages PGF and TikZ, `http://www.texample.net/`

Included is examples catalogue includes examples (with output) from the package documentation as well as code written by the original site maintainer (Kjell Magne Fauske) and others.

The compendious PGF/TikZ manual is clear, but is bewildering for some beginners. The 'minimal' introduction has helped at least the present author.

Vincent Zoonekynd provides a set of excellent (and graphic) tutorials on the programming of title page styles, chapter heading styles and section heading styles. In each file, there is a selection of graphics representing an output style, and for each style, the code that produces it is shown.

An invaluable step-by-step setup guide for establishing a "work flow" through your (La)TeX system, so that output appears at the correct size and position on standard-sized paper, and that the print quality is satisfactory, is Mike Shell's *testflow*. The tutorial consists of a large plain text document, and there is a supporting LaTeX file together with correct output, both in PostScript and PDF, for each of A4 and "letter" paper sizes. The complete kit is available on CTAN (distributed with the author's macros for papers submitted for IEEE publications). The issues are also covered in a later FAQ answer.

Documentation of Japanese Ω use appears in Haruhiko Okumura's page typesetting Japanese with Omega (the parent page is in Japanese, so out of the scope of this FAQ list).

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 university departments have enthusiastic documenters who make public record of their (La)TeX support. For example, Tim Love (of Cambridge University Engineering Department) maintains his department's pages at `http://www-h.eng.cam.ac.uk/help/tpl/textprocessing/`

Graphics in LaTeX2e: `info/epslatex`

`testflow`: `macros/latex/contrib/IEEEtran/testflow`

Herbert Voß's Maths tutorial: `info/math/voss/mathmode/Mathmode.pdf`

42 Reference documents

For TeX primitive commands a rather nice quick reference booklet, by John W. Shipman, is available; it's arranged in the same way as the TeXbook. By contrast, you can view David Bausum's list of TeX primitives alphabetically or arranged by "family". Either way, the list has a link for each control sequence, that leads you to a detailed description, which includes page references to the TeXbook.

There doesn't seem to be a reference that takes in Plain TeX as well as the primitive commands.

An interesting LaTeX "cheat sheet" is available from CTAN: it's a list of (more or less) everything you 'ought to' remember, for basic LaTeX use. (It's laid out very compactly for printing on N. American 'letter'; printed on ISO A4, using Adobe Acrobat's "shrink to fit", it strains aged eyes…)

For command organised references to LaTeX, Karl Berry (et al)'s LaTeX reference manual is (to an extent) work in progress, but is generally reliable (source is available on the.archive as well).

Martin Scharrer's "List of internal LaTeX macros" is a help to those aiming to write a class or package.

The reference provided by the Emerson Center of Emory University), LaTeXe help also looks good.

Cheat sheet: `info/latexcheat/latexcheat/latexsheet.pdf`

LaTeX reference manual: `info/latex2e-help-texinfo`

LaTeX internal macros: `info/macros2e`

43 WIKI books for TeX and friends

The *WIKI* concept can be a boon to everyone, if used sensibly. The "general" WIKI allows *anyone* to add stuff, or to edit stuff that someone else has added: while there is obvious potential for chaos, there is evidence that a strong user community can keep a WIKI under control.

Following the encouraging performance of the ConTeXt WIKI, valiant efforts have been made generating "WIKI books" for (La)TeX users. Thus we have (Plain) TeX WIKI and LaTeX WIKI — both well established. Neither would seem exactly of "publication quality" (though that isn't really the aim), but they are both useful reference sources.

44 Typography tutorials

Peter Wilson's article *memdesign* has a lengthy introductory section on typographic considerations, which is a fine tutorial, written by someone who is aware of the issues as they apply to (La)TeX users. (*Memdesign* now distributed separately from the manual for his *memoir* class, but was originally part of that manual)

There's also (at least one) typographic style tutorial available on the Web, the excellent "Guidelines for Typography in NBCS". In fact, its parent page is also worth a read: among other things, it provides copies of the "guidelines" document in a wide variety of primary fonts, for comparison purposes. The author is careful to explain that he has no ambition to supplant such excellent books as Bringhurst's, but the document (though it does contain its Rutgers-local matter) is a fine introduction to the issues of producing readable documents.

memdesign: `info/memdesign`

45 Freely available (La)TeX books

People have long argued for (La)TeX books to be made available on the web, and until relatively recently this demand went un-answered.

The first to appear was Victor Eijkhout's excellent "TeX by Topic" in 2001 (it had been published by Addison-Wesley, but was long out of print). The book is now available on CTAN; it's not a beginner's tutorial but it's a fine reference. It's also available, as a printed copy, via the on-line publishers Lulu (not quite free, of course, but not a *bad* deal. . .).

Addison-Wesley have also released the copyright of "TeX for the Impatient" by Paul W. Abrahams, Karl Berry and Kathryn A. Hargreaves, another book whose unavailability many have lamented. The authors have re-released the book under the GNU Free Documentation Licence, and it is available from CTAN.

Norm Walsh's "Making TeX Work" (originally published by O'Reilly) is also available (free) on the Web, at `http://makingtexwork.sourceforge.net/mtw/`; the sources of the Web page are on CTAN. The book was an excellent resource in its day, but while it is now somewhat dated, it still has its uses, and is a welcome addition to the list of on-line resources. A project to update it is believed to be under way.

Making TeX Work: `info/makingtexwork/mtw-1.0.1-html.tar.gz`

TeX by Topic: `info/texbytopic`

TeX for the Impatient: `info/impatient`

46 Documentation of packages

These FAQs regularly suggest packages that will "solve" particular problems. In some cases, the answer provides a recipe for the job. In other cases, or when the solution needs elaborating, how is the poor user to find out what to do?

If you're lucky, the package you need is already in your installation. If you're particularly lucky, you're using a distribution that gives access to package documentation and the documentation is available in a form that can easily be shown. For example, on many current distributions, the *texdoc* command often helps, as in:

```
    texdoc footmisc
```

which opens a window showing documentation of the *footmisc* package. According to the type of file *texdoc* finds, it will launch a DVI viewer or a PDF reader.

If *texdoc* can't find any documentation, it may launch a Web browser to look at its copy of the CTAN catalogue. The catalogue has an entry for package documentation, and since the CTAN team asks authors documentation of their packages (and members of the team even occasionally generate documentation that seems useful), you will more often than not find documentation that way.

If your luck (as defined above) doesn't hold out, you've got to find documentation by other means. That is, you have to find the documentation for yourself. The rest of this answer offers a range of possible techniques.

The commonest form of documentation of LaTeX add-ons is within the `.dtx` file in which the code is distributed (see documented LaTeX sources). Such files are supposedly processable by LaTeX itself, but there are occasional hiccups on the way to readable documentation. Common problems are that the package itself is needed to process its own documentation (so must be unpacked before processing), and that the `.dtx` file will *not* in fact process with LaTeX. In the latter case, the `.ins` file will usually produce a `.drv` (or similarly-named) file, which you process with LaTeX instead. (Sometimes the package author even thinks to mention this wrinkle in a package `README` file.)

Another common form is the separate documentation file; particularly if a package is "conceptually large" (and therefore needs a lot of documentation), the documentation would prove a cumbersome extension to the `.dtx` file. Examples of such cases are the *memoir* class, the *KOMA-script* bundle (whose developers take the trouble to produce detailed documentation in both German and English), the *pgf* documentation (which would make a substantial book in its own right) and the *fancyhdr* package (whose documentation derives from a definitive tutorial in a mathematical journal). Even if the documentation is not separately identified in a `README` file, it should not be too difficult to recognise its existence.

Documentation within the package itself is the third common form. Such documentation ordinarily appears in comments at the head of the file, though at least one eminent author regularly places it after the `\endinput` command in the package. (This is desirable, since `\endinput` is a 'logical' end-of-file, and (La)TeX doesn't read beyond it: thus such documentation does not 'cost' any package loading time.)

The above suggestions cover most possible ways of finding documentation. If, despite your best efforts, you can't find it in any of the above places, there's the awful possibility that the author didn't bother to document his package (on the "if it was hard to write, it should be hard to use" philosophy). Most ordinary mortals will seek support from some more experienced user at this stage, though it *is* possible to proceed in the way that the original author apparently expected... by reading his code.

47 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 'LaTeX2e for class and package writers' (`clsguide`, part of the LaTeX documentation distribution). Beyond this, a good knowledge of TeX itself is valuable: thus books such as the TeXbook or TeX by topic are relevant. With good TeX 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. Such processing is noticeably tedious, but Heiko Oberdiek has prepared a well-linked PDF version, which is in the file `base.tds.zip` of his *latex-tds* distribution. Individual files in the LaTeX distribution may be processed separately with LaTeX, like any well-constructed `.dtx` file.

Writing good classes is not easy; it's a good idea to read some established ones (classes.dtx, for example, is the documented source of 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*

An annotated version of *article*, as it appears in classes.dtx, was published in *TUGboat* **28**(1). The article, by Peter Flynn, is a good guide to understanding classes.dtx

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

clsguide.pdf: macros/latex/doc/clsguide.pdf

latex-tds rmfamilycollection: macros/latex/contrib/latex-tds

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

LaTeX documentation: macros/latex/doc

source2e.tex: macros/latex/base/source2e.tex

48 Metafont and Metapost Tutorials

Apart from Knuth's book, there seems to be only one publicly-available tutorial for Metafont, by Christophe Grandsire (a copy in PDF form may be downloaded). Geoffrey Tobin's *Metafont for Beginners* (see using Metafont) describes how the Metafont system works and how to avoid some of the potential pitfalls.

There is also an article on how to run both Metafont and Metapost (the programs). 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). For Metafont the article is geared towards testing and installing new Metafont fonts, while its Metapost section 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 within ConTeXt). It may be found on his company's 'manuals' page.

Other Metapost tutorials that have appeared are two in English by André Heck and Urs Oswald, and one in French (listed here because it's clearly enough written that even this author understands it), by Laurent Chéno; both have been recommended for inclusion in the FAQ

Urs Oswald's tutorial is accompanied by a super tool (by Troy Henderson) for testing little bits of Metapost, which is an invaluable aid to the learner: http://www.tlhiv.org/mppreview

Vincent Zoonekynd's massive set of example Metapost files is available on CTAN; the set includes a *Perl* script to convert the set to html, and the set may be viewed on the web. While these examples don't exactly constitute a "tutorial", they're most certainly valuable learning material. Urs Oswald presents a similar document, written more as a document, and presented in PDF.

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

Peter Wilson's "experiences": info/metafont/metafp/metafp.pdf

Vincent Zoonekynd's examples: info/metapost/examples

49 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. The document "BibTeXing", in the BibTeX distribution (look for btxdoc), expands on the chapter in Lamport's book. *The LaTeX Companion* also has information on BibTeX and writing BibTeX style files. (See LaTeX books for details of both books.)

The document "Designing BibTeX Styles", also in the BibTeX distribution (look for btxhak), explains the postfix stack-based language used to write BibTeX styles (.bst files). The filebtxbst.doc, also in the BibTeX distribution, is the template for the four standard styles (*plain*, *abbrv*, *alpha*, and *unsrt*); it also contains their documentation.

A useful tutorial of the whole process of using BibTeX is Nicolas Markey's "*Tame the BeaST (The B to X of BibTeX)*", which may also be found on CTAN. A summary and FAQ by Michael Shell and David Hoadley, is also to be recommended.

BibTeX distribution: `biblio/bibtex/base`

Shell and Hoadley's FAQ: `biblio/bibtex/contrib/doc/btxFAQ.pdf`

Tame the BeaST: `info/bibtex/tamethebeast/ttb_en.pdf`

50 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 the LaTeX packages needed to produce them.

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

- Script fonts for mathematics
- Fonts for the number sets
- Typesetting the principal value integral

A useful new symbol search web site is available. It provides you a scratch area on which you draw the symbol you're thinking of, with your mouse; when you've finished drawing, the classifier tries to match your sketch with symbols it knows about. The matching process is pretty good, even for the sketches of a *really* poor draughtsman, and it's often worth trying more than once. Given the huge size of the symbol list, this tool can be very helpful.

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

51 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.

However, there *is* a summary of PiCTeX commands available on CTAN, which is a great aide-memoire for those who basically know the package to start with.

pictex: `graphics/pictex`

PiCTeX summary: `info/pictex/summary`

C Bits and pieces of (La)TeX

52 What is a DVI file?

A DVI file (that is, a file with the type or extension `.dvi`) is TeX's main output file, when we use "original" TeX; later TeX systems, such as PDFTeX may use other formats. XeTeX (released some time after PDFTeX) uses an "extended DVI format" (XDV) to send its output to its close-coupled DVI driver, *xdvipdfmx*.

'DVI' is supposed to be an acronym for DeVice-Independent, meaning that the file can be processed for printing or viewing on most kinds of typographic output device or display. The DVI file may be processed by a DVI driver to produce further output designed specifically for a particular printer, or it may be used by a previewer for display on a computer screen. The character encoding of a DVI file is determined by the document itself — see "what are encodings" for an explanation of what 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.

The canonical reference for the structure of a DVI file is the source of Knuth's program *dvitype* (whose original purpose, as its name implies, was to view the content of a DVI file), and a partially complete "standard" for the way they should be processed may offer further enlightenment.

rmfamilyDVI processing standard: `dviware/driv-standard`

dvitype: `systems/knuth/dist/texware/dvitype.web`

53 What is a DVI driver?

A DVI driver is a program that takes as input a DVI file and (usually) produces a file in a format that something *other* than a TeX-related program can process.

A driver may be designed for producing output for printing (e.g., PostScript), for later processing (e.g., PostScript for inclusion in a later document), or for document exchange (e.g., PDF).

As well as the DVI file, the driver typically also needs font information. Font information may be held as bitmaps or as outlines, or simply as a set of pointers into the fonts that a printer itself provides. Each driver will expect the font information in a particular form.

For more information on the forms of font information, see PK files, TFM files, virtual fonts and Using PostScript fonts with TeX.

54 What are PK files?

PK files (packed raster) are the canonical form of font bitmaps. The output from Metafont 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 of each design (point) size for each weight for each font in 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.

While this menagerie of bitmaps can (in principle) provide fonts that are closely matched to the capabilities of each printer, the size of the collection (and the resulting difficulty of maintaining it) has been a potent driver to the move towards outline fonts such as Adobe Type 1 fonts.

55 What are TFM files?

TFM is an acronym for 'TeX Font Metrics'; 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; each TFM file serves for all magnifications of 'its' font, so that there are (typically) fewer TFM files than there are PK files. TeX (LaTeX, etc.) itself needs only to know about the sizes of characters and their interactions with each other, but not what characters look like. By contrast, TFM files are not, in principle, needed by the DVI driver, which only needs to know about the glyphs that each character selects, so as to print or display them.

Note that TrueType and OpenType fonts contain the necessary metrics, so that XeTeX and LuaTeX, using such fonts, have no need of TFM files. A corollary of this is that setting up fonts for use by these engines is far *easier*.

56 Virtual fonts

Virtual fonts provide a means of collecting bits and pieces together to make the glyphs of a font: the bits and pieces may be other glyphs, rules and other "basic" typesetting commands, and the positioning information that specifies how everything comes together.

An early instance of something like virtual fonts for TeX was implemented by David Fuchs to use an unusual printer. However, for practical purposes for the rest of us, virtual fonts date from when Knuth specified a format and wrote some support software, in 1989 (he published an article in *TUGboat* at the time; a plain text 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. TeX reads a TFM file of the font, just as before, but the DVI processor will read the VF and use its content to specify how each glyph is to be processed.

The virtual font may contain commands:

- to 'open' one or more (real) fonts for subsequent use,
- to remap a glyph from one of the (real) fonts,
- to build up a more complicated effect (using DVI commands).

In practice, the most common use of virtual fonts is to remap Adobe Type 1 fonts (see font metrics), though there has also been useful useful work building 'fake' maths fonts (by bundling glyphs from several fonts into a single virtual font). Virtual Computer Modern fonts, making a Cork encoded font from Knuth's originals by using remapping and fragments of DVI for single-glyph 'accented characters', were the first "Type 1 format" Cork-encoded Computer Modern fonts available.

Virtual fonts are normally created in a single ASCII VPL (Virtual Property List) file, which includes two sets of information. The *vptovf* utility will use the VPL file to create the binary TFM and VF files.

A "how-to" document, explaining how to generate a VPL, describes the endless hours of fun that may be had, doing the job by hand. Despite the pleasures to be had, the commonest way (nowadays) of generating an VPL file is to use the *fontinst* package, which is described in more detail PostScript font metrics. *Qdtexvpl* is another utility for creating ad-hoc virtual fonts (it uses TeX to parse a description of the virtual font, and *qdtexvpl* itself processes the resulting DVI file).

fontinst: `fonts/utilities/fontinst`

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

Virtual fonts "how to": `info/virtualfontshowto/virtualfontshowto.txt`

qdtexvpl: `fonts/utilities/qdtexvpl`

57 What are (TeX) macros

TeX is a *macro processor*: this is a computer-science-y term meaning "text expander" (more or less); TeX typesets text as it goes along, but *expands* each macro it finds. TeX's macros may include instructions to TeX itself, on top of the simple text generation one might expect.

Macros are a *good thing*, since they allow the user to manipulate documents according to context. For example, the macro `\TeX` is usually defined to produce "TEX" with the 'E' lowered (the original idea was Knuth's), but in these FAQs the default definition of the macro is overridden, and it simply expands to the letters "TeX". (*You* may not think this a good thing, but the author of the macros has his reasons – see TeX-related logos.)

Macro names are conventionally built from a \ followed by a sequence of letters, which may be upper or lower case (as in `\TeX`, mentioned above). They may also be \⟨*any single character*⟩, which allows all sorts of oddities (many built in to most TeX macro sets, all the way up from the apparently simple '\ ' meaning "insert a space here").

Macro programming can be a complicated business, but at their very simplest they need little introduction — you'll hardly need to be told that:

```
\def\foo{bar}
```

replaces each instance of `\foo` with the text "bar". The command `\def` is Plain TeX syntax for defining commands; LaTeX offers a macro `\newcommand` that goes some way towards protecting users from themselves, but basically does the same thing:

```
\newcommand{\foo}{bar}
```

Macros may have "arguments" , which are used to substitute for marked bits of the macro expansion:

```
\def\foo#1{This is a #1 bar}
...
\foo{2/4}.
```

which produces:

> This is a 2/4 bar.

or, in LaTeX speak:

```
\newcommand{\foo}[1]{This is a #1 bar}
...
\foo{3/4}.
```

which produces:

> This is 3/4 bar.

(LaTeX users waltz through life, perhaps?)

You will have noticed that the arguments, above, were enclosed in braces (`{...}`); this is the normal way of typing arguments, though TeX is enormously flexible, and you may find all sorts of other ways of passing arguments (if you stick with it).

Macro writing can get very complicated, very quickly. If you are a beginner (La)TeX programmer, you are well advised to read something along the lines of the TeXbook; once you're under way, TeX by Topic is possibly a more satisfactory choice. Rather a lot of the answers in these FAQs tell you about various issues of how to write macros.

58 `\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.

Instructions for 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 human-friendly 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. LaTeX2e has standard graphics and colour packages that make figure inclusion, rotation and scaling, and colour typesetting relatively straightforward, despite the rather daunting `\special` commands involved. (ConTeXt provides similar support, though not by way of packages.)

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.

Note that `\special` behaves rather differently in PDFTeX, since there is no device driver around. There *is* a concept of PDF specials, but in most cases `\special` will provoke a warning when used in PDFTeX.

59 Writing (text) files from TeX

TeX allows you to write to output files from within your document. The facility is handy in many circumstances, but it is vital for several of the things LaTeX (and indeed almost any higher-level TeX-based macro package) does for you.

The basic uses of writing to an external file are "obvious" — remembering titles of sections for a table of contents, remembering label names and corresponding section or figure numbers, all for a later run of your document. However, the "non-obvious" thing is easy to forget: that page numbers, in TeX, are slippery beasts, and have to be captured with some care. The trick is that `\write` operations are only executed as the page is sent to the DVI or PDF file. Thus, if you arrange that your page-number macro

(\thepage, in LaTeX) is not expanded until the page is written, then the number written is correct, since that time is where TeX guarantees the page number tallies with the page being sent out.

Now, there are times when you want to write something straight away: for example, to interact with the user. TeX captures that requirement, too, with the primitive command \immediate:

```
\immediate\write\terminal{I'm waiting...}
```

writes a "computer-irritates-user" message, to the terminal.

Which brings us to the reason for that \terminal. TeX can "\write" up to 16 streams simultaneously, and that argument to \write says which is to be used. Macro packages provide the means of allocating streams for your use: Plain TeX provides a macro \newwrite (used as "\newwrite\streamname", which sets \streamname as the stream number). In fact, \terminal (or its equivalent) is the first output stream ever set up (in most macro packages): it is never attached to a file, and if TeX is asked to write to *any* stream that isn't attached to a file it will send the output to the terminal (and the log).

60 Spawning programs from (La)TeX: \write18

The TeX \write primitive instruction is used to write to different file 'streams'; TeX refers to each open file by a number, not by a file name (although most of the time we hide this). Originally, TeX would write to a file connected to a stream numbered 0–15. More recently, a special "stream 18" has been implemented: it is not writing to a file, but rather tells TeX to ask the operating system to do something. To run a command, we put it as the argument to \write18. So to run the *epstopdf* utility on a file with name stored as \epsfilename, we would write:

```
\write18{epstopdf \epsfilename}
```

When using something like the *epstopdf* package, the 'stream' write operation is hidden away and you don't need to worry about the exact way it's done.

However, there is a security issue. If you download some (La)TeX code from the Internet, can you be sure that there is not some command in it (perhaps in a hidden way) to do stuff that might be harmful to your computer (let's say: delete everything on the hard disk!)? In the face of this problem, both MiKTeX and TeX Live have, for some time, disabled \write18 by default. To turn the facility on, both distributions support an additional argument when starting TeX from the command shell:

```
(pdf)(la)tex --shell-escape <file>
```

The problem with this is that many people use (La)TeX via a graphical editor, so to use \write18 for a file the editor's settings must be changed. Of course, the settings need restoring after the file is processed: you defeat the point of the original protection, that way.

The latest MiKTeX (version 2.9), and TeX Live (2010 release) get around this by having a special "limited" version of \write18 enabled 'out of the box'. The idea is to allow only a pre-set list of commands (for example, BibTeX, *epstopdf*, TeX itself, and so on). Those on the list are regarded as safe enough to allow, whereas anything else (for example deleting files) still needs to be authorised by the user. This seems to be a good balance: most people most of the time will not need to worry about \write18 at all, but it will be available for things like *epstopdf*.

Note that the TeX system may tell you that the mechanism is in use:

```
This is pdfTeX, Version 3.1415926-1.40.11 (TeX Live 2010)
 restricted \write18 enabled.
```

when it starts.

epstopdf.sty: Distributed with Heiko Oberdiek's packages macros/latex/contrib/oberdiek

61 How does hyphenation work in TeX?

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

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

TeX uses a pretty good system for hyphenation (originally designed by Frank Liang — you may view his Ph.D. thesis online) 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 sport to be played by ordinary mortals).

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 hyphenation patterns recognised by a TeX-based or XeTeX system, a partial reinstallation is necessary (note that LuaTeX relaxes this constraint).

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 the context of hyphenation failure.

62 What are LaTeX classes and packages?

LaTeX aims to be a general-purpose document processor. Such an aim could be achieved by a selection of instructions which would enable users to use TeX primitives, but such a procedure is considered too inflexible (and probably too daunting for ordinary users), and so the designers of LaTeX created a model which offered an *abstraction* of the design of documents. Obviously, not all documents can look the same (even with the defocussed eye of abstraction), so the model uses *classes* of document. Base LaTeX offers five classes of document: *book*, *report*, *report*, *article* and *letter*. For each class, LaTeX provides a *class file*; the user arranges to use it via a `\documentclass` command at the top of the document. So a document starting

```
\documentclass{article}
```

may be called "an *article* document".

This is a good scheme, but it has a glaring flaw: the actual typographical designs provided by the LaTeX class files aren't widely liked. The way around this is to *refine* the class. To refine a class, a programmer may write a new class file that loads an existing class, and then does its own thing with the document design.

If the user finds such a refined class, all is well, but if not, the common way is to load a *package* (or several).

LaTeX provides rather few package files, but there are lots of them to be found on the archives by a wide variety of authors; several packages are designed just to adjust the design of a document — using such packages achieves what the programmer might have achieved by refining the class.

Other packages provide new facilities: for example, the *graphics* package (actually provided by the LaTeX team) allows the user to load externally-provided graphics into a document, and the *hyperref* package enables the user to construct hyper-references within a document.

On disc, class and package files only appear different by virtue of their name "extension" — class files are called `*.cls` while package files are called `*.sty`. Thus we find that the LaTeX standard *article* class is represented on disc by a file called `article.cls`, while the *hyperref* package is represented on disc by a file called `hyperref.sty`.

The class vs. package distinction was not clear in LaTeX 2.09 — everything was called a style ("document style" or "document style option"). It doesn't really matter that the nomenclature has changed: the important requirement is to understand what other people are talking about.

63 Documented LaTeX sources (.dtx files)

LaTeX2e, and most contributed macro packages, are now written in a literate programming style, with source and documentation in the same file. This format, known as 'doc', in fact originated before the days of the LaTeX project as one of the "Mainz" series of packages. A documented source file conventionally has the suffix `.dtx`, and will normally be 'stripped' before use with LaTeX; an installation file (`.ins`) is normally provided, to automate this process of removing comments for speed of loading. To read the comments, you can run LaTeX on the `.dtx` file to produce a nicely formatted version of the documented code. Several packages can be included in one `.dtx` file (they're sorted out by the `.ins` file), with conditional sections, and there are facilities for indexes of macros, etc.

Anyone can write `.dtx` files; the format is explained in The LaTeX Companion, and a tutorial is available from CTAN (which comes with skeleton `.dtx` and `.ins` files).

Composition of `.dtx` files is supported in *emacs* by AUC-TeX.

Another useful way of generating `.dtx` files is to write the documentation and the code separately, and then to combine them using the *makedtx* system. This technique has particular value in that the documentation file can be used separately to generate HTML output; it is often quite difficult to make LaTeX to HTML conversion tools deal with `.dtx` files, since they use an unusual class file.

The *sty2dtx* system goes one step further: it attempts to create a `.dtx` file from a 'normal' `.sty` file with comments. It works well, in some circumstances, but can become confused by comments that aspire to "structure" (e.g., tabular material, as in many older packages' file headers).

The `.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.

An interesting sideline to the story of `.dtx` files is the *docmfp* package, which extends the model of the *doc* package to Metafont and Metapost, thus permitting documented distribution of bundles containing code for Metafont and Metapost together with related LaTeX code.

AUC-TeX: `support/auctex`

clsguide.pdf: `macros/latex/doc/clsguide.pdf`

docmfp.sty: `macros/latex/contrib/docmfp`

docstrip.tex: Part of the LaTeX distribution

DTX tutorial: `info/dtxtut`

makedtx: `support/makedtx`

sty2dtx: `support/sty2dtx`

64 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 its input (however encoded) to something regular. Nowadays, the operating system translates keystrokes into a code appropriate for the user's language: the encoding used is usually a national or international standard, though some operating systems use "code pages" (as defined by Microsoft). These standards and code pages often contain characters that may not 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 virtual font). 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 appeared, because of the possibility of suppressing `\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 "how does hyphenation work"). To take advantage of the diacriticised characters represented in the fonts, it is necessary to arrange that whenever the command sequence "`\'e`" has been input (explicitly, or implicitly via the sort of mapping of input mentioned above), the character that codes the position of the "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.

65 What are the EC fonts?

A font provides a number of *glyphs*. In order that the glyphs may be printed, they are *encoded*, 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 drivers for the eccentricity of Knuth's encodings went away, and at TUG's Cork meeting, an encoding for a set of 256 glyphs, for use in TeX text, was defined. The intention was that these glyphs should cover 'most' European languages that use Latin alphabets, in the sense of including all accented letters needed. (Knuth's CMR fonts missed things necessary for Icelandic and Polish, for example, which the Cork fonts do have, though even Cork encoding's coverage isn't complete.) 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 hyphenation of accented words).

The first Metafont-fonts to conform to the Cork encoding were the EC fonts. They look CM-like, though their metrics differ from CM-font metrics in several areas. They have long been regarded as 'stable' (in the same sense that the CM fonts are stable: their metrics are unlikely ever to change). Each EC font is, of course, roughly twice the size of the corresponding CM font, and there are far more of them than there are CM fonts. The simple number of fonts proved problematic in the production of Type 1 versions of the fonts, but EC or EC-equivalent fonts in Type 1 or TrueType form (the latter only from commercial suppliers). Free auto-traced versions — the CM-super and the LGC fonts, and the Latin Modern series (rather directly generated from Metafont sources), are available.

Note that the Cork encoding doesn't cover mathematics (so that no "T1-encoded" font families can not support it). If you're using Computer-Modern-alike fonts, this doesn't actually matter: your system will have the original Computer Modern mathematical fonts (or the those distributed with the Latin Modern set), which cover 'basic' TeX mathematics; more advanced mathematics are likely to need separate fonts anyway. Suitable mathematics fonts for use with other font families are discussed in "choice of scalable fonts".

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 necessarily as 'stable' are the EC fonts are. Note that modern distributions tend not to distribute the EC fonts in outline format, but rather to provide Latin Modern for T1-encoded Computer Modern-style fonts. This can sometimes cause confusion when users are recompiling old documents.

The Cork encoding is also implemented by virtual fonts provided in the PSNFSS system, for Adobe Type 1 fonts, and also by most other such fonts that have been developed (or otherwise made available) for use with (La)TeX.

Note that T1 (and other eight-bit font encodings) are superseded in the developing TeX-family members XeTeX and LuaTeX, which use Unicode as their base encoding, and use Unicode-encoded fonts (typically in `ttf` or `otf` formats). The *cm-unicode* fonts carry the flag in this arena, along with the Latin Modern set.

CM-super fonts: `fonts/ps-type1/cm-super`

CM-LGC fonts: `fonts/ps-type1/cm-lgc`

CM unicode fonts: `fonts/cm-unicode`

EC and TC fonts: `fonts/ec`

Latin Modern fonts: `fonts/lm`

66 Unicode and TeX

Unicode is a character code scheme that has the capacity to express the text of the languages of the world, as well as important symbols (including mathematics). Any coding scheme that is directly applicable to TeX may be expressed in single bytes (expressing up to 256 characters); Unicode characters may require several bytes, and the scheme may express a very large number of characters.

For "old-style" applications (TeX or PDFTeX) to deal with Unicode input, the sequence of bytes to make up Unicode character is processed by a set of macros, and converted to an 8-bit number representing a character in an appropriate font (in practice, only the standard UTF-8 byte sequences are supported). TeX code that reads these bytes is complicated, but works well enough; there is an `utf8` option for the LaTeX distribution *inputenc* package. The separate package *ucs* provides wider, but less robust, coverage via an *inputenc* option `utf8x`. Broadly, the difference is that `utf8` deals with "standard LaTeX fonts" (those for which LaTeX has a defined encoding), while `utf8x` deals with pretty much anything for which it knows a mapping of a Unicode range to a font. As a general rule, you should never use *ucs*/`utf8x` until you have convinced yourself that *inputenc*/`utf8` can not do the job for you.

‘Modern’ TeX-alike applications, XeTeX and LuaTeX read their input using UTF-8 representations of Unicode as standard. They also use TrueType or OpenType fonts for output; each such font has tables that tell the application which part(s) of the Unicode space it covers; the tables enable the engines to decide which font to use for which character (assuming there is any choice at all).

inputenc.sty: Part of the `macros/latex/base` distribution

ucs.sty: `macros/latex/contrib/unicode`

67 What is the TDS?

TDS is an acronym for “TeX Directory Structure”; it specifies a standard way of organising all the TeX-related files on a computer system.

Most modern distributions arrange their TeX files in conformance with the TDS, using both a ‘distribution’ directory tree and a (set of) ‘local’ directory trees, each containing TeX-related files. The TDS recommends the name `texmf` for the name of the root directory (folder) of an hierarchy; in practice there are typically several such trees, each of which has a name that compounds that (e.g., `texmf-dist`, `texmf-var`).

Files supplied as part of the distribution are put into the distribution’s tree, but the location of the distribution’s hierarchy is system dependent. (On a Unix system it might be at `/usr/share/texmf` or `/opt/texmf`, or a similar location.)

There may be more than one ‘local’ hierarchy in which additional files can be stored. An installation will also typically offer a local hierarchy, while each user may have an individual local hierarchy.

The TDS itself is published as the output of a TUG Technical Working Group. You may browse an on-line version of the standard, and copies in several other formats (including source) are available on CTAN.

TDS specification: `tds`

68 What is “Encapsulated PostScript” (“EPS”)?

PostScript has been for many years a *lingua franca* of powerful printers (though modern high-quality printers now tend to require some constrained form of Adobe Acrobat, instead); 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” (EPS) 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.

69 Adobe font formats

Adobe has specified a number of formats for files to represent fonts in PostScript files; this question doesn’t attempt to be encyclopaedic, so we only discuss the two formats most commonly encountered in the (La)TeX context, types 1 and 3. In particular, we don't discuss the OpenType format, whose many advantages now becoming accessible to most (La)TeX users (by means of the widely-used XeTeX and the more experimental LuaTeX).

Adobe Type 1 format specifies a means to represent outlines of the glyphs in a font. The 'language' used is closely restricted, to ensure that the font is rendered as quickly as possible. (Or rather, as quickly as possible with Adobe's technology at the time the specification was written: the structure could well be different if it were specified now.) The format has long been the basis of the digital type-foundry business, though nowadays most new fonts are released in OpenType format.

In the (La)TeX context, Type 1 fonts are extremely important. Apart from their simple availability (there are thousands of commercial Type 1 text fonts around), the commonest reader for PDF files has long (in effect) *insisted* on their use (see below).

Type 3 fonts have a more forgiving specification. A wide range of PostScript operators is permissible, including bitmap specifiers. Type 3 is therefore the natural format to be used for programs such as *dvips* when they auto-generate something to represent Metafont-generated fonts in a PostScript file. It's Adobe Acrobat Viewer's treatment of bitmap Type 3 fonts that has made direct Metafont output increasingly unattractive, in recent years. If you have a PDF document in which the text looks fuzzy and uneven in Acrobat Reader, ask Reader for the `File`→`Document Properties`→ `Fonts ...`, and it will likely show some font or other as "Type 3" (usually with encoding "Custom"). The problem has disappeared with version 6 of Acrobat Reader. See PDF quality for a discussion of the issue, and for ways of addressing it.

Type 3 fonts should not entirely be dismissed, however. Acrobat Reader's failure with them is entirely derived from its failure to use the anti-aliasing techniques common in TeX-ware. Choose a different set of PostScript graphical operators, and you can make pleasing Type 3 fonts that don't "annoy" Reader. For example, you may not change colour within a Type 1 font glyph, but there's no such restriction on a Type 3 font, which opens opportunities for some startling effects.

70 What are "resolutions"?

"Resolution" is a word that is used with little concern for its multiple meanings, in computer equipment marketing. The word suggests a measure of what an observer (perhaps the human eye) can resolve; yet we regularly see advertisements for printers whose resolution is 1200dpi — far finer than the unaided human eye can distinguish. The advertisements are talking about the precision with which the printer can place spots on the printed image, which affects the fineness of the representation of fonts, and the accuracy of the placement of glyphs and other marks on the page.

In fact, there are two sorts of "resolution" on the printed page that we need to consider for (La)TeX's purposes:

- the positioning accuracy, and
- the quality of the fonts.

In the case where (La)TeX output is being sent direct to a printer, in the printer's "native" language, it's plain that the DVI processor must know all such details, and must take detailed account of both types of resolution.

In the case where output is being sent to an intermediate distribution format, that has potential for printing (or displaying) we know not where, the final translator, that connects to directly to the printer or display, has the knowledge of the device's properties: the DVI processor need not know, and should not presume to guess.

Both PostScript and PDF output are in this category. While PostScript is used less frequently for document distribution nowadays, it is regularly used as the source for distillation into PDF; and PDF is the workhorse of an enormous explosion of document distribution.

Therefore, we need DVI processors that will produce "resolution independent" PostScript or PDF output; of course, the independence needs to extend to both forms of independence outlined above.

Resolution-independence of fonts was for a long time forced upon the world by the feebleness of Adobe's *Acrobat Reader* at dealing with bitmap files: a sequence of answers starting with one aiming at the quality of PDF from PostScript addresses the problems that arise.

Resolution-independence of positioning is more troublesome: *dvips* is somewhat notorious for insisting on positioning to the accuracy of the declared resolution of the printer. One commonly-used approach is to declare a resolution of 8000 ("better than any device"), and this is reasonably successful though it does have its problems.

71 What is the "Berry naming scheme"?

In the olden days, (La)TeX distributions were limited by the feebleness of file systems' ability to represent long names. (The MS-DOS file system was a particular bugbear: fortunately any current Microsoft system allows rather more freedom to specify file names. Sadly, the ISO 9660 standard for the structure of CD-ROMs has a similar failing, but that too has been modified by various extension mechanisms.)

One area in which these short file names posed a particular problem was that of file names for Type 1 fonts. These fonts are distributed by their vendors with pretty meaningless short names, and there's a natural ambition to change the name to something that identifies the font somewhat precisely. Unfortunately, names such as "BaskervilleMT" are already far beyond the abilities of the typical feeble file system, and add the specifier of a font shape or variant, and the difficulties spiral out of control. Font companies deal with the issue by inventing silly names, and providing a map file to show what the "real" names. Thus the Monotype Corporation provides the translations:

```
bas_____ BaskervilleMT
basb____ BaskervilleMT-Bold
basbi___ BaskervilleMT-BoldItalic
```

and so on. These names could be used within (La)TeX programs, except that they are not unique: there's nothing to stop Adobe using '`bas_____`' for *their* Baskerville font.

Thus arose the Berry naming scheme.

The basis of the scheme is to encode the meanings of the various parts of the file's specification in an extremely terse way, so that enough font names can be expressed even in impoverished file name-spaces. The encoding allocates one character to the font "foundry" (Adobe, Monotype, and so on), two to the typeface name (Baskerville, Times Roman, and so on), one to the weight, shape, and encoding and so on.

The whole scheme is outlined in the *fontname* distribution, which includes extensive documentation and a set of tables of fonts whose names have been systematised.

`fontname distribution`: `info/fontname`

D Acquiring the Software

72 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 each answer).

The CTAN sites are Dante (Germany) and Cambridge (UK); the links are to the root of the CTAN tree, above which the layout is identical at each site.

The TeX files at each CTAN node may also be accessed using `ftp` at URLs `ftp://dante.ctan.org/tex-archive` and `ftp://ftp.tex.ac.uk/tex-archive` respectively.

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 is available as file `CTAN.sites` on the archives themselves.

Better still, the script behind http://mirror.ctan.org/ will cunningly choose a "nearby" mirror for you, using information from the database 'behind' CTAN.sites; it will offer a connection using http (web) protocol if one is available, otherwise one using ftp protocol.

To access a particular thing through the mirror.ctan.org mechanism, simply place the CTAN path after the base URL; so http://mirror.ctan.org/macros/latex/contrib/footmisc/ will connect you to the *footmisc* directory at some CTAN site (or a mirror) — note that the tex-archive part of the CTAN path isn't needed.

For details of how to find files at CTAN sites, see "finding (La)TeX files".

The TeX user who has no access to any sort of network may buy a copy of the archive as part of the TeX Live distribution.

73 Ready-built installation files on the archive

The TDS is a simple structure, and almost all files can be installed simply by putting them in the "right" place, and updating a single index. (Note, this simple idea typically doesn't work for fonts, unless they're distributed as Metafont source.)

The CTAN network is therefore acquiring "TDS-ZIP" files, which have a built-in directory structure that matches the TDS. These files have to be built, and the CTAN team has asked that package authors supply them (the team will advise, of course, if the author has trouble). The CTAN team hopes that the extra work involved will contribute to happier lives for package users, which in turn must surely help keep the TeX community lively and active.

At the time of writing, there are rather few .tds.zip files (by comparison with the huge number of packages that are available). As packages are updated, the number of files is steadily increasing, but it will be a long time before the whole set is covered.

Use of the files is discussed in "installing using ready-built ZIP files".

74 What was the CTAN nonfree tree?

When CTAN was founded, in the 1990s, it was unusual to publish the terms under which a TeX-related package was distributed (or, at any rate, to publish those terms formally).

With the advent of the TeX *distributions*, however, people started to realise the need for such information, to protect those who create, distribute or sell the discs that hold the packages, etc. With the licence information available, the distributors can decide which packages may be distributed.

The CTAN team decided that it would be useful for users (and distributors, not to say package authors) to separate packages that were candidates for distribution, and those that were in some sense "not free". Thus was the nonfree tree born.

From the start, the nonfree tree was controversial: the terms under which a package would be placed on the tree were hotly contested, and the CTAN team were only able slowly to populate the tree. It became obvious to the team that the project would never have been completed.

The CTAN catalogue now records the nature of the licences of a good proportion of the packages it describes (though there remain several for which the licence is unknown, which is as good, for the distributors, as a licence forbidding distribution). Since the catalogue's coverage of CTAN is good (and slowly improving), the general rule for distributors has become

> "if the package is listed in the catalogue, check there to see whether you should distribute; if the package is not listed in the catalogue, don't think of distributing it".

(The catalogue only has a modest list of licences, but it covers the set used by packages on CTAN, with a wild-card "other-free" which covers packages that the CTAN administrators believe to be free even though the authors haven't used a standard licence.)

There is a corollary to the 'general rule': if you notice something that ought to be in the distributions, for which there is no catalogue entry, please let the CTAN team (ctan@dante.de) know. It may well be that the package has simply been missed, but some aren't catalogued because there's no documentation and the team just doesn't understand the package.

In the light of the above, the `nonfree` tree is being dismantled, and its contents moved (or moved *back*) to the main CTAN tree. So the answer to the question is, now, "the nonfree tree was a part of CTAN, whose contents are now in the main tree".

75 Contributing a file to the archives

You have something to submit to the archive — great! Before we even start, here's a check-list of things to sort out:

1. Licence: in the spirit of TeX, we hope for free software; in the spirit of today's lawyer-enthralled society, CTAN provides a list of "standard" licence statements. Make sure that there's a formal statement of the licence of your package, somewhere in the files you upload; beyond the CTAN installation, your package is a candidate for inclusion in (La)TeX distributions ... and thereafter, also in operating system distributions ... and the people who bundle all these things up need a clear statement of your intent.
2. Documentation: it's good for users to be able to browse documentation before downloading a package. You need at least a plain text `README` file (exactly that name, upper case and no `.txt` extension); in addition a PDF file of the package documentation, prepared for screen reading, is highly desirable.
3. Name: endless confusion is caused by name clashes. If your package has the same name as one already on CTAN, or if your package installation generates files of the same name as something in a "normal" distribution, the CTAN team will delay installation while they check that you're doing the right thing: they may nag you to change the name, or to negotiate a take-over with the author of the original package. Browse the archive to ensure uniqueness.
   The name you choose should also (as far as possible) be somewhat descriptive of what your submission actually *does*; while "descriptiveness" is to some extent in the eye of the beholder, it's clear that names such as `mypackage` or `jiffy` aren't suitable.

If you are able to use anonymous *ftp*, you can upload that way. The file `README.uploads` on CTAN 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 principal CTAN sites offers an upload page — choose from `http://www.dante.de/CTAN/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. This method enforces one file per upload: if you had intended to upload lots of files, you need to bundle them into an 'archive' file of some sort; acceptable formats are `.zip` and `.tar.gz` (or `.tar.bz2`). Most uploads are packed in `.zip` format.

If you can use neither of these methods, or if you find something confusing, ask advice of the CTAN management

If your package is large, or regularly updated, it may be appropriate to *mirror* your contribution direct into CTAN. Mirroring is only practical using `ftp` or `rsync` protocols, so this facility is limited to those who can establish their own server for one of those protocols.

`README.uploads`: `README.uploads`

76 Finding (La)TeX files

Modern TeX distributions contain a huge array of various sorts of files, but sooner or later most people need to find something that's not in their present system (if nothing else, because they've heard that something has been updated).

But how to find the files?

Modern distributions (TeX Live and MiKTeX, at least) provide the means to update your system "over the net". This is the minimum effort route to getting a new file: 'simply' find which of the distributions 'packages' holds the file in question, and ask the distribution to update it. The mechanisms are different (the two distributions exhibit the signs of evolutionary divergence in their different niches), but neither is difficult — see "using MiKTeX for installing" and "using TeX Live for installing".

There are packages, though, that aren't in the distribution you use (or for which the distribution hasn't yet been updated to offer the version you need).

Some sources, such as these FAQ answers, provide links to files: so if you've learnt about a package here, you should be able to retrieve it without too much fuss.

Otherwise, the CTAN sites provide searching facilities, via the web: at Dante with `http://dante.ctan.org/search.html`, at Cambridge with `http://www.tex.ac.uk/search.html`.

Two search mechanisms are offered: the simpler search, locating files by name, simply scans a list of files (`FILES.byname` — see below) and returns a list of matches, arranged neatly as a series of links to directories and to individual files.

The more sophisticated search looks at the contents of each catalogue entry, and returns a list of catalogue entries that mention the keywords you ask for.

An alternative way to scan the catalogue is to use the catalogue's "by topic" index; this lists a series of topics, and (La)TeX projects that are worth considering if you're working on matters related to the topic.

In fact, *Google*, and other search engines, can be useful tools. Enter your search keywords, and you may pick up a package that the author hasn't bothered to submit to CTAN. If you're using *Google*, you can restrict your search to CTAN by entering

```
site:ctan.org tex-archive ⟨search term(s)⟩
```

in *Google*'s "search box". You can also enforce the restriction using *Google*'s "advanced search" mechanism; other search engines (presumably) have similar facilities.

Many people avoid the need to go over the network at all, for their searches, by downloading the file list that the archives' web file searches use. This file, `FILES.byname`, presents a unified listing of the archive (omitting directory names and cross-links). Its companion `FILES.last07days` is also useful, to keep an eye on the changes on the archive. Since these files are updated only once a day, a nightly automatic download (perhaps using *rsync*) makes good sense.

FILES.byname: `FILES.byname`

FILES.last07days: `FILES.last07days`

77 Finding new fonts

Nowadays, new fonts are seldom developed by industrious people using Metafont, but if such do appear, they will nowadays be distributed in the same way as any other part of (La)TeX collections. (An historical review of Metafont fonts available is held on CTAN as "Metafont font list".) Nowadays, most new fonts that appear are only available in some scalable outline form, and a large proportion is distributed under commercial terms.

Such fonts often make their way to the free distributions (at least TeX Live and MiKTeX) if their licensing is such that the distributions can accept them. Commercial fonts do not get to distributions, though support for some of them is held by CTAN.

Arranging for a new font to be usable by (La)TeX is very different, depending on which type of font it is, and which TeX-alike engine you are using; roughly speaking:

- MetaFont fonts will work without much fuss (provided their sources are in the correct place in the installation's tree); TeX-with-*dvips*, and PDFTeX are "happy" with them.
- Adobe Type 1 fonts can be made to work, after `.tfm` and (usually) `.vf` files have been created from their metric (`.afm`) files.
- TrueType fonts can be made to work with PDFTeX, using the techniques discussed in ANSWER TO BE WRITTEN
- TrueType and OpenType fonts are the usual sort used by XeTeX and LuaTeX; they "just work" with those engines.

The answer "choice of scalable fonts" discusses fonts that are configured for general (both textual and mathematical) use with (La)TeX. The list of such fonts is sufficiently short that they *can* all be discussed in one answer.

Metafont font list: `info/metafont-list`

78 The TeX collection

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

The TeX collection provides this, together with ready-to-run TeX systems for various architectures. The collection is distributed on DVD, and contains:

- The TeX Live distribution, including the programs themselves compiled for a variety of architectures; TeX Live may be run from the DVD or installed on hard disc.
- MacTeX: an easy to install TeX system for MacOS/X, based on TeX Live; this distribution includes a native Mac installer, the TeXShop front-end and other Mac-specific tools;
- ProTeXt: an easy to install TeX system for Windows, based on MiKTeX, based on an 'active' document that guides installation; and
- A snapshot of CTAN.

A fair number of national TeX User Groups, as well as TUG, distribute copies to their members at no extra charge. Some user groups are also able to sell additional copies: contact your local user group or TUG.

You may also download disc images from CTAN; the installation disc, ProTeXt and MacTeX are all separately available. Beware: they are all pretty large downloads. TeX Live, once installed, may be updated online.

More details of the collection are available from its own web page on the TUG site.

MacTeX: `systems/mac/mactex`

ProTeXt: `systems/win32/protext`

TeX Live install image: `systems/texlive`

E TeX Systems

79 (La)TeX for different machines

We list here the free or shareware packages; discusses commercial packages.

The list is provided in four answers:

- TeX systems for use with Unix and GNU Linux systems
- TeX systems for use with Modern Windows systems
- TeX systems for use with Macintosh systems
- TeX systems for Other sorts of systems

80 Unix and GNU Linux systems

> Note that Mac OS/X, though it is also a Unix-based system, has different options; users should refer to the information in Mac systems.

The TeX distribution of choice, for Unix systems (including GNU/Linux and most other free Unix-like systems) is TeX Live, which is distributed as part of the TeX collection.

TeX Live may also be installed "over the network"; a network installer is provided, and once you have a system (whether installed from the network or installed off-line from a disc) a manager (*tlmgr*) can both keep your installation up-to-date and add packages you didn't install at first.

TeX Live may be run with no installation at all; the web page TeX Live portable usage describes the options for installing TeX Live on a memory stick for use on another computer, or for using the TeX Live DVD with no installation at all.

TeX-gpc is a "back-to-basics" distribution of TeX utilities, *only* (unlike TeX Live, no 'tailored' package bundles are provided). It is distributed as source, and compiles with GNU Pascal, thereby coming as close as you're likely to get to Knuth's original distribution. It is known to work well, but the omission of e-TeX and PDFTeX will rule it out of many users' choices.

A free version of the commercial VTeX extended TeX system is available for use under Linux. VTeX, among other things, specialises in direct production of PDF from (La)TeX input. Sadly, it's no longer supported, and the ready-built images are made for use with a rather ancient Linux kernel.

Finally, the developer of MiKTeX has released a port of the PDFTeX executable from of that excellent (Windows) system for use on Linux; get it via the MiKTeX site. The MiKTeX package manager (*mpm*), for Linux, is also available from the same place.

tex-gpc: `systems/unix/tex-gpc`

texlive: Browse `systems/texlive`

texlive installer (Unix): `systems/texlive/tlnet/install-tl-unx.tar.gz`

vtex: `systems/vtex/linux`

vtex required common files: `systems/vtex/common`

81 (Modern) Windows systems

Windows users nowadays have a real choice, between two excellent distributions, MiKTeX and TeX Live. TeX Live on windows has only in recent years been a real challenger to the long-established MiKTeX, and even now MiKTeX has features that TeX Live lacks. Both are comprehensive distributions, offering all the established TeX variants (TeX, PDFTeX — both with e-TeX variants — as well as XeTeX and LuaTeX), together with a wide range of support tools.

Both MiKTeX and TeX Live offer management tools, including the means of keeping an installation "up-to-date", by reinstalling packages that have been updated on CTAN (the delay between a package update appearing, and it being available to the distribution users) can be as short as a day (and is never very long).

MiKTeX, by Christian Schenk, is the longer-established of the pair, and has a large audience of satisfied users; TeX Live is the dominant distribution in use in the world of Unix-like systems, and so its Windows version may be expected to appeal to those who use both Unix-like and Windows systems. The latest release of MiKTeX — version 2.9 — requires Windows XP, or later (so it does not work on Windows 2000 or earlier).

Both distributions may be used in a configuration which involves no installation at all. MiKTeX's "portable" distribution may be unpacked on a memory stick, and used on any windows computer without making any direct use of the hard drive. The web page TeX Live portable usage describes the options for installing TeX Live on a memory stick, or for using the TeX Live DVD with no installation at all.

Both MiKTeX and TeX Live may be downloaded and installed, package by package, over the net. This is a mammoth undertaking, only to be undertaken by those with a good network connection (and a patient disposition!).

A ready-to-run copy of the MiKTeX distribution, on DVD may be bought via the MiKTeX web site. MiKTeX may also be installed using ProTeXt, on the TeX Collection DVD.

The TeX Collection DVD also provides an offline installer for TeX Live.

XEmTeX, by Fabrice Popineau (he who created the excellent, but now defunct, fpTeX distribution), is another integrated distribution of TeX, LaTeX, ConTeXt, *XEmacs* and other friends for Windows. All programs have been compiled natively to take the best advantage of the Windows environment. Configuration is provided so that the resulting set of programs runs out-of-the-box, but the distribution is not actively promoted.

A further (free) option is available thanks to the CygWin bundle, which presents a Unix-like environment in Windows systems (and also provides an X-windows server). The (now obsolete) teTeX distribution is provided as part of the CygWin distribution, but there is a CygWin build of TeX Live so you can have a current TeX system. TeX under CygWin is reputedly somewhat slower than native Win32 implementations such as MiKTeX, and of course the TeX applications behave like Unix-system applications.

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: `systems/win32/bakoma`

miktex: `systems/win32/miktex`; acquire `systems/win32/miktex/setup/setup.exe` (also available from the MiKTeX web site), and read installation instructions from the MiKTeX installation page

Portable miktex: `systems/win32/miktex/setup/miktex-portable.exe`

protext.exe: `systems/win32/protext`

texlive: Browse `systems/texlive`

texlive installer (Windows): `systems/texlive/tlnet/install-tl.zip`

82 Macintosh systems

The TeX collection DVD includes MacTeX, which is a Mac-tailored version of TeX Live; details may be found on the TUG web site. If you don't have the disc, you can download the distribution from CTAN (but note that it's pretty big). MacTeX is an instance of TeX Live, and has a Mac-tailored graphical TeX Live manager, so that you can keep your distribution up-to-date.

Note that installing MacTeX requires `root` privilege. This is a pity, since it offers several extras that aren't available via a standard TeX Live, which aren't therefore available to "ordinary folk" at work. For those who don't have `root` privilege, the option is to install using the TeX Live `tlinstall` utility.

OzTeX, by Andrew Trevorrow, is a shareware version of TeX for the Macintosh. A DVI previewer and PostScript driver are also included. OzTeX is a Carbon app, so will run under Mac OS/X (see `http://www.trevorrow.com/oztex/ozosx.html` for details), but it is *not* a current version: it doesn't even offer PDFTeX. A mailing list is provided by TUG: sign up via `http://tug.org/mailman/listinfo/oztex`

Another partly shareware program is CMacTeX, put together by Tom Kiffe. CMacTeX is much closer than OzTeX to the Unix TeX model of things (it uses *dvips*, for instance). CMacTeX runs natively under Mac OS/X; it includes a port of a version of Omega.

Further information may be available in the MacTeX wiki. The MacTeX-on-OS X mailing list is another useful resource for users; subscribe via the list home page

cmactex: `systems/mac/cmactex`

mactex: `systems/mac/mactex`

oztex: `systems/mac/oztex`

83 Other systems' TeX availability

For PCs, running MS-DOS or OS/2, EmTeX (by Eberhard Mattes) offers LaTeX, BibTeX, previewers, and drivers. It is available as a series of zip archives, with documentation in both German and English. Appropriate memory managers for using emTeX with 386 (and better) processors and under pre-'9x Windows, are included in the distribution. (EmTeX *can* be made to operate under Windows, but even back when it was "current", such use wasn't very actively encouraged.)

A version of emTeX, packaged to use a TDS directory structure, is separately available as an emTeX 'contribution'. Note that neither emTeX itself, nor emTeX-TDS, is maintained. Users of Microsoft operating systems, who want an up-to-date (La)TeX system, need Win32-based systems.

For PCs, running MS-DOS, a further option is a port of the Web2C 7.0 implementation, using the GNU *djgpp* compiler. While this package is more recent than emTeX, it nevertheless also offers a rather old instance of (La)TeX.

For PCs running OS/2, users also have the option of the free version of the commercial VTeX, which specialises in direct production of PDF from (La)TeX input. (This version is, like the Linux version, unfortunately no longer supported.)

For VAX systems running OpenVMS, a TeX distribution is available CTAN, but is almost certainly not the latest (it is more than 10 years old). Whether a version is even available for current VMS (which typically runs on Intel 64-bit processors) is not clear, but it seems unlikely.

For the Atari ST and TT, CS-TeX is available from CTAN; it's offered as a set of ZOO archives.

Amiga users have the option of a full implementations of TeX 3.1 (PasTeX) and Metafont 2.7.

It's less likely that hobbyists would be running TOPS-20 machines, but since TeX was originally written on a DEC-10 under WAITS, the TOPS-20 port is another near approach to Knuth's original environment. Sources are available by anonymous `ftp` from `ftp://ftp.math.utah.edu/pub/tex/pub/web`

Atari TeX: `systems/atari/cs-tex`

djgpp: `systems/msdos/djgpp`

emtex: `systems/msdos/emtex`

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

OpenVMS: `systems/OpenVMS/TEX97_CTAN.ZIP`

PasTeX: `systems/amiga`

vtex for Linux: `systems/vtex/linux`

vtex for OS/2: `systems/vtex/os2`

vtex required common files: `systems/vtex/common`

84 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 The commonest choices are *[X]Emacs* or *vim*, though several others are available.
GNU *emacs* and *XEmacs* are supported by the AUC-TeX bundle (available from CTAN). 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.
Vim is also highly configurable (also available for Windows and Macintosh systems). Many plugins are available to support the needs of the (La)TeX user, including syntax highlighting, calling TeX programs, auto-insertion and -completion of common (La)TeX structures, and browsing LaTeX help. The scripts `auctex.vim` and `bibtex.vim` seem to be the most common recommendations.
The editor *NEdit* is also free and programmable, and is available for Unix systems. An AUC-TeX-alike set of extensions for *NEdit* is available from CTAN.
LaTeX4Jed provides much enhanced LaTeX support for the *jed* editor. *LaTeX4Jed* is similar to AUC-TeX: menus, shortcuts, templates, syntax highlighting, document outline, integrated debugging, symbol completion, full integration with external programs, and more. It was designed with both the beginner and the advanced LaTeX user in mind.
The *Kile* editor that is provided with the KDE window manager provides GUI "shell-like" facilities, in a similar way to the widely-praised *Winedt* (see below); details (and downloads) are available from the project's home on SourceForge.
TUG is sponsoring the development of a cross-platform editor and shell, modelled on the excellent *TeXshop* for the Macintosh. While this program, *TeXworks*, is

still under development, it is already quite usable: if you're looking for a (La)TeX development environment, and are willing to step beyond the boundaries, it may be for you.

Windows-32 *TeXnicCenter* is a (free) TeX-oriented development system, uniting a powerful platform for executing (La)TeX and friends with a configurable editor.
TeXworks (see above) is also available for Windows systems.
Winedt, a shareware package, is also highly spoken of. It too provides a shell for the use of TeX and related programs, as well as a powerful and well-configured editor. The editor can generate its output in UTF-8 (to some extent), which is useful when working with XeTeX (and other "next-generation" (La)TeX applications).
Both *emacs* and *vim* are available in versions for Windows systems.

Macintosh For Mac OS/X users, the free tool of choice appears to be *TeXshop*, which combines an editor and a shell with a coherent philosophy of dealing with (La)TeX in the OS X environment. TeXShop is distributed as part of the MacTeX system, and will therefore be available out of the box on machines on which MacTeX has been installed.
Vim is also available for use on Macintosh systems.
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 also works well with OzTeX. From release 2.2.0 (at least), Textures works under Mac OS/X.

OS/2 *epmtex* offers an OS/2-specific shell.

Atari, Amiga and NeXT users also have nice environments. LaTeX users looking for *make*-like facilities should review the answer on Makefiles for LaTeX documents.

While many (La)TeX-oriented editors can support work on BibTeX files, there are many systems that provide specific "database-like" access to your BibTeX files — see "creating a bibliography file".

alpha: `systems/mac/support/alpha`

auctex: `support/auctex`

epmtex: `systems/os2/epmtex`

LaTeX4Jed: `support/jed`

Nedit LaTeX support: `support/NEdit-LaTeX-Extensions`

TeXnicCenter: `systems/win32/TeXnicCenter`

TeXshell: `systems/msdos/texshell`

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

winedt: `systems/win32/winedt`

85 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
7890 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, August 2004.

pcTeX Long-established: pcTeX32 is a Windows implementation.

Personal TeX Inc
725 Greenwich Street, Suite 210
San Francisco, CA 94133
USA

Tel: 800-808-7906 (within the USA)
Tel: +1 415-296-7550
Fax: +1 415-296-7501
Email: sales@pctex.com
Web: http://www.pctex.com/

Source: Personal TeX Inc web site, December 2004

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.
990 Anlaby Road,
Hull,
East Yorkshire,
HU4 6AT
UK

Tel: 0845 766 0340 (within the UK)
Fax: 0845 603 9443 (within the UK)
Email: christopher@sciword.demon.co.uk
Web: http://www.sciword.demon.co.uk

MacKichan Software Inc.
19307 8th Avenue, Suite C
Poulsbo WA 98370-7370
USA

Tel: +1 360 394 6033
Tel: 877 724 9673 (within the USA) Fax: +1 360 394 6039
Email: info@mackichan.com
Web: http://www.mackichan.com

Source: Mail from Christopher Mabb, August 2007

Macintosh; Textures "A TeX system 'for the rest of us' ". A beta release of Textures for Mac OS/X is available — see http://www.bluesky.com/news/news_

frames.html
(Blue Sky also gives away a Metafont implementation and some font manipulation tools for Macs.)

Blue Sky Research
PO Box 80424
Portland, OR 97280
USA

Tel: 800-622-8398 (within the USA)
Tel/Fax: +1 503-222-9571
Fax: +1 503-246-4574
Email: sales@bluesky.com
Web: http://www.bluesky.com/

Source: Mail from Gordon Lee, April 2007

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

Note that the company Y&Y has gone out of business, and Y&Y TeX (and support for it) is therefore no longer available. Users of Y&Y systems may care to use the self-help mailing list that was established in 2003; the remaining usable content of Y&Y's web site is available at http://www.tug.org/yandy/

F DVI Drivers and Previewers

86 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, and versions are available in all TeX distributions that recognise the use of PostScript.

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

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

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

VMS distribution: systems/OpenVMS/TEX97_CTAN.ZIP

87 DVI drivers for HP LaserJet

The emTeX distribution (see TeX systems) 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*.

Beebe drivers: dviware/beebe

`dviljk`: dviware/dviljk

88 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 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, 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 `support/ghostscript/GPL`

89 DVI previewers

EmTeX for PCs running MS-DOS or OS/2, MiKTeX and XEmTeX 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 commercial vendors) 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; it is now built from the same distribution as *xdvi*. The TeX Live distributions for Unix systems include a version of *xdvik*.

Alternatives to previewing include

- conversion to 'similar' ASCII text (see converting to ASCII) 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 previewing PostScript files), and
- generating PDF output, and viewing that with *Acrobat Reader* or one of the substitutes for that.

`xdvi`: `dviware/xdvi`

90 Generating bitmaps from DVI

In the last analysis, any DVI driver or previewer is generating bitmaps: bitmaps for placing tiny dots on paper via a laser- or inkjet-printer, or bitmaps for filling some portion of your screen. However, it's usually difficult to extract any of those bitmaps any way other than by screen capture, and the resolution of *that* is commonly lamentable.

Why would one want separate bitmaps? Most often, the requirement is for something that can be included in HTML generated from (La)TeX source — not everything that you can write in (La)TeX can be translated to HTML (at least, portable HTML that may be viewed in 'most' browsers), so the commonest avoiding action is to generate a bitmap of the missing bit. Examples are maths (a maths extension to the '*ML' family is available but not universally supported by browsers), and 'exotic' typescripts (ones that you cannot guarantee your readers will have available). Other common examples are generation of sample bitmaps, and generation for insertion into some other application's display — to insert equations into Microsoft PowerPoint, or to support the enhanced-*emacs* setup called *preview-latex*.

In the past, the commonest way of generating bitmaps was to generate a PostScript file of the DVI and then use *ghostscript* to produce the required bitmap format (possibly by way of PNM format or something similar). This is an undesirable procedure (it is very slow, and requires two or three steps) but it has served for a long time.

(La)TeX users may now take advantage of two bitmap 'drivers'. The longer-established, *dvi2bitmap*, will generate XBM and XPM formats, the long-deprecated GIF format (which is now obsolescent, but has finally been relieved of the patent protection of the LZW compression it uses), and also the modern (ISO-standardised) PNG format.

Dvipng started out as a PNG renderer; from version 1.2 it can also render to the GIF format. It is designed for speed, in environments that generate large numbers of PNG files: the `README` mentions *preview-latex*, *LyX*, and a few web-oriented environments. Note that *dvipng* gives high-quality output even though its internal operations are optimised for speed.

`dvi2bitmap`: `dviware/dvi2bitmap`

`dvipng`: `dviware/dvipng`

G Support Packages for TeX

91 (La)TeX-friendly drawing packages

(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*. The list of export formats is very long, and includes Metafont and Metapost, Encapsulated PostScript and PDF, as well as combinations that wrap a graphics format in a LaTeX import file, which may include LaTeX commands to place text (compiled by LaTeX itself) as labels, etc., in the figures.

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

Asymptote is a widely-praised development of the Metapost language, which can draw 2D or 3D diagrams, and can also label diagrams with LaTeX text; copious documentation is available via *asymptote*'s web site.

asymptote: `graphics/asymptote`

xfig: `graphics/xfig`

transfig: `graphics/transfig`

92 TeXCAD, a drawing package for LaTeX

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

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

emtex: `systems/msdos/emtex`

xtexcad: `graphics/xtexcad/xtexcad-2.4.1.tar.gz`

93 Spelling checkers for work with TeX

For Unix, *ispell* was long the program of choice; it is well integrated with *emacs*, and deals with some TeX syntax. However, it has more-or-less been replaced everywhere, by *aspell*, which was designed as a successor, and certainly performs better on most metrics; there remains some question as to its performance with (La)TeX sources.

For Windows, there is a good spell checker incorporated into many of the shell/editor combinations that are available. The spell checker from the (now defunct) 4AllTeX shell remains 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`

aspell: Browse `support/aspell` — choose just those language dictionaries (under subdirectory `dict/`) that you need.

excalibur: `systems/mac/support/excalibur/Excalibur-4.0.2.sit.hqx`

ispell: `support/ispell/ispell-3.2.06.tar.gz`

jspell: `support/jspell`

VMS spell: `support/vmspell`

winedt: `systems/win32/winedt`

94 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
```

The technique is beguilingly simple, but it's not terribly accurate

The *latexcount* script does the same sort of job, in one "step"; being a *perl* script, it is in principle rather easily configured (see documentation inside the script). Several editors and shells offer something similar.

TeXcount goes a long way with heuristics for counting, starting from a LaTeX file; the documentation is comprehensive, and you may try the script on-line via the package home page.

However, even quite sophisticated stripping of (La)TeX markup can never be entirely reliable: markup itself may contribute typeset words, or even consume words that appear in the text.

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, if it works for you.

detex: `support/detex`

latexcount.pl: `support/latexcount/latexcount.pl`

TeXcount: `support/texcount`

wordcount: `macros/latex/contrib/wordcount`

H Literate programming

95 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 () 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/comp.programming.literate_FAQ`

96 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.

FWEB, by John Krommes, is a version for Fortran, Ratfor,C, C++, working with LaTeX; it was derived from *CWEB*.

Spidery WEB, by Norman Ramsey, supports many languages including Ada, `awk`, and C and, while not in the public domain, is usable without charge. It is now superseded by *noweb* (also by Norman Ramsay) which incorporates the lessons learned in implementing spidery WEB, and which is a simpler, equally powerful, tool.

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).

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/dist/web`

tweb: `web/tweb`

weave: `systems/knuth/dist/web`

I Format conversions

97 Conversion from (La)TeX to plain text

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* and
- *catdvi*, which is capable of generating Latin-1 (ISO 8859-1) or UTF-8 encoded output. *Catdvi* was conceived as a replacement for *dvi2tty*, but development seems to have stopped before the authors were willing to declare the work complete.

A common problem is the hyphenation that TeX inserts when typesetting something: since the output is inevitably viewed using fonts that don't match the original, the hyphenation usually looks silly.

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.

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, 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: dviware/dvi2tty

l2a: support/l2a

tex2mail: support/tex2mail

txt: support/txt

98 Conversion from SGML or HTML to TeX

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

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

   - David Megginson’s *sgmlspm*, written in *Perl* 5.
   - Joachim Schrod and Christine Detig’s *STIL*, (‘SGML Transformations in Lisp’).

   Both of these allow the user to write ‘handlers’ for every SGML element, with plenty of access to attributes, entities, and information about the context within the document tree.
   If these packages don’t meet your needs for an average SGML typesetting job, you need the big commercial stuff.

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

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).

html2latex source: support/html2latex

99 Conversion from (La)TeX 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 markup. TeX is capable of extremely detailed page layout; HTML is not, because HTML is a functional markup 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 future Web technologies.

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 now sports an illustrious list of co-authors and is 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).

A mailing list for users may be found via `http://tug.org/mailman/listinfo/latex2html`

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.

plasTeX a Python-based LaTeX document processing framework. It gives DOM-like access to a LaTeX document, as well as the ability to generate mulitple output formats (e.g. HTML, DocBook, tBook, etc.).

TeXpider a commercial program from Micropress, 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/`

An interesting set of samples, including conversion of the same text by the four free programs listed above, is available at `http://www.mayer.dial.pipex.com/samples/example.htm`; a linked page gives lists of pros and cons, by way of comparison.

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`

`plasTeX`: Browse `support/plastex`

`tex4ht`: `obsolete/support/TeX4ht/tex4ht-all.zip` (but see `http://tug.org/tex4ht/`)

`tth`: `support/tth/dist`

100 Other conversions to and from (La)TeX

troff *Tr2latex*, assists in the translation of a *troff* document into LaTeX 2.09 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. . .). *Tr2latex* is an enhanced version of the earlier *troff-to-latex* (which is no longer available).

WordPerfect *wp2latex* is actively maintained, and is available either for MS-DOS or for Unix systems.

RTF *Rtf2tex*, by Robert Lupton, is for converting Microsoft's Rich Text Format to TeX. There is also a converter to LaTeX by Erwin Wechtl, called *rtf2latex*. The latest converter, by Ujwal Sathyam and Scott Prahl, is *rtf2latex2e* which seems rather good, though development of it seems to have stalled.

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, conversion to HTML). TeX2RTF is supported on various Unix platforms and under Windows 3.1

Microsoft Word A rudimentary (free) program for converting MS-Word to LaTeX is *wd2latex*, which runs on MS-DOS; it probably produces output destined for an archaic version of MS-Word (the program itself was archived in 1991).

The current preferred free-software method is a two-stage process:

- Convert LaTeX to *OpenOffice* format, using the *tex4ht* command *oolatex*;
- open the result in *OpenOffice* and 'save as' a MS-Word document.

(Note that *OpenOffice* itself is *not* on CTAN; see `http://www.openoffice.org/`, though most *linux* systems offer it as a ready-to-install bundle.)

Word2TeX and *TeX2Word* are shareware translators from Chikrii Softlab; positive users' reports have been noted (but not recently).

The excellent *tex4ht* will generate OpenOffice ODT format, which can be used as an intermediate to producing Word format files.

If cost is a constraint, the best bet is probably to use an intermediate format such as RTF or HTML. *Word* outputs and reads both, so in principle this route may be useful.

You can also use PDF as an intermediate format: Acrobat Reader for Windows (version 5.0 and later) will output rather feeble RTF that *Word* can read.

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.

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.

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.

Wilfried Hennings' FAQ, which deals specifically with conversions between TeX-based formats and word processor formats, offers much detail as well as tables that allow quick comparison of features.

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`

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

tex4ht: obsolete/support/TeX4ht/tex4ht-all.zip (but see http://tug.org/tex4ht/)

tr2latex: support/tr2latex

wd2latex: support/wd2latex

wp2latex: support/wp2latex

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

101 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 solution (of sorts) for 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.

However, modern usage would proceed via XSL or XSLT2 to produce a formattable version.

xmltex: macros/xmltex/base

passivetex: macros/xmltex/contrib/passivetex

102 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 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 "converting to and from (La)TeX".

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: dviware/dvi2tty

ghostscript: Browse support/ghostscript/GPL

xpdf: Browse support/xpdf

103 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" 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.

Some of this work is has (in a sense) been done, in the port of the LaTeX graphics package to Plain TeX. However, while *graphics* is available, other complicated packages (notably *hyperref*) are not. The aspiring translator may find the Eplain system a useful source of code. (In fact, a light-weight system such as Eplain might reasonably be adopted as an alternative target of translation, though it undoubtedly gives the user more than the "bare minimum" that Plain TeX is designed to offer.)

The `eplain` system: `macros/eplain`

'Plain TeX' `graphics`: `macros/plain/graphics`

J Installing (La)TeX files

104 Installing things on a (La)TeX system

Installing (or replacing) things on your (La)TeX system has the potential to be rather complicated; the following questions attempt to provide a step-by-step approach, starting from the point where you've decided what it is that you want to install:

- You must find the file you need;
- If you are going to install a LaTeX package, you may need to unpack the distributed files;
- It may be necessary to generate some documentation to read;
- You need to decide where to install the files;
- You must now install the files; and finally
- You may need to tidy up after the installation.

105 Finding packages to install

How did you learn about the package?

If the information came from these FAQs, you should already have a link to the file (there are lists of packages at the end of each answer). Click on one of the links associated with the package, and you can get the package (which may be one file or several).

If you heard about the file somewhere else, it's possible that the source told you where to look; if not, try the CTAN searching facilities, such as `http://www.tex.ac.uk/search/`. That (rather simple) search engine can return data from a search of the CTAN catalogue (which covers most useful packages), or data from a search of the names of files on the archive.

Packages come in a variety of different styles of distribution; the very simplest will actually offer just `package.sty` — in this case, just download the file and get on with installation.

You will regularly find that the file you want (e.g., `foo.sty`) is distributed in a LaTeX documented source file `foo.dtx`; thus you should search just for *foo* — `foo.sty` won't be visible anywhere on the archive or in the catalogue.

Since most packages are distributed in this `.dtx`/`.ins` way, they usually occupy their own directory on the archive. Even if that directory contains other packages, you should download everything in the directory: as often as not, packages grouped in this way depend on each other, so that you really *need* the other ones.

Having acquired the package distribution, "unpacking LaTeX packages" outlines your next step.

106 Unpacking LaTeX packages

As discussed elsewhere, the 'ordinary' way to distribute a LaTeX package is as a pair of files `package.dtx` and `package.ins`. If you've acquired such a pair, you simply process `package.ins` with LaTeX, and the files will appear, ready for installation.

Other sorts of provision should ordinarily be accompanied by a `README` file, telling you what to do; we list a few example configurations.

Sometimes, a directory comes with a bunch of `.dtx` files, but fewer (often only one) `.ins` files (LaTeX itself comes looking like this). If there is more than one `.ins` file, and in the absence of any instruction in the `README` file, simply process the `.ins` file(s) one by one.

If you're missing the `package.ins` altogether, you need to play around until something works. Some `.dtx` files are "self-extracting" — they do without an `.ins` file, and once you've processed the `package.dtx`, `package.sty` has automagically appeared. Various other oddities may appear, but the archivists aim to have `README` file in every package, which should document anything out of the ordinary with the distribution.

107 Generating package documentation

We are faced with a range of "normal" provision, as well as several oddities. One should note that documentation of many packages is available on CTAN, without the need of any further effort by the user — such documentation can usually be browsed *in situ*.

However, if you find a package that does not offer documentation on the archive, or if you need the documentation in some other format than the archive offers, you can usually generate the documentation yourself from what you download from the archive.

The standard mechanism, for LaTeX packages, is simply to run LaTeX on the `package.dtx` file, as you would any ordinary LaTeX file (i.e., repeatedly until the warnings go away).

A variant is that the unpacking process provides a file `package.drv`; if such a thing appears, process it in preference to the `package.dtx` (it seems that when the documented LaTeX source mechanism was first discussed, the `.drv` mechanism was suggested, but it's not widely used nowadays).

Sometimes, the LaTeX run will complain that it can't find `package.ind` (the code line index) and/or `package.gls` (the list of change records, not as you might imagine, a glossary). Both types of file are processed with special *makeindex* style files; appropriate commands are:

```
makeindex -s gind package
makeindex -s gglo -o package.gls package.glo
```

This author finds that the second (the change record) is generally of limited utility when reading package documentation; it is, however, valuable if you're part of the package development team. If you don't feel you need it, just leave out that step

Another common (and reasonable) trick performed by package authors is to provide a separate file `package-doc.tex` or even simply `manual.tex`; if the file `package.dtx` doesn't help, simply look around for such alternatives. The files are treated in the same way as any "ordinary" LaTeX file.

108 Installing files "where (La)TeX can find them"

In the past, package documentation used always to tell you to put your files "where LaTeX can find them"; this was always unhelpful — if you knew where that *was*, you didn't need telling, but if you *didn't* know, you were completely stuck.

It was from this issue that the whole idea of the TDS sprang; "where to put" questions now come down to "where's the TDS tree?".

We therefore answer the question by considering:

- what tree to use, and
- where in the tree to put the files.

Once we know the answer to both questions, and we've created any directories that are needed, we simply copy files to their rightful location.

This has done what the old requirement specified: LaTeX (or whatever) *can* (in principle) find the files. However, in order that the software *will* find the files, we need to update an index file.

On a MiKTeX system, open the window `Start`→`All Programs`→`MiKTeX` ⟨*version*⟩→`Settings`, and click on `Refresh FNDB`. The job may also be done in a command window, using the command:

```
initexmf --update-fndb
```

The MiKTeX documentation gives further details about `initexmf`.

On a teTeX or TeX Live-based system, use the command `texhash` (or if that's not available, `mktexlsr` — they ought to be different names for the same program).

Having done all this, the new package will be available for use.

109 Which tree to use

In almost all cases, new material that you install should go into the "local" tree of your (La)TeX installation. (A discussion of reasons *not* to use the local tree appears below.)

On a Unix(-alike) system, using teTeX or TeX Live, the root directory will be named something like `/usr/share/texmf-local/` or `/usr/local/share/texmf/` You can ask such a system where it believes a local tree should be:

```
kpsewhich -var-value TEXMFLOCAL
```

the output being the actual path, for example (on the workstation the author is using today):

```
/usr/local/share/texmf
```

In a MiKTeX installation, the location will in fact typically be something you specified yourself when you installed MiKTeX in the first place, but you may find you need to create one. The MiKTeX "Settings" window (`Start`→`Programs`→`MiKTeX`→`Settings`) has a tab "`Roots`"; that tab gives a list of current TDS roots (they're typically not called `texmf`-anything). If there's not one there with "`local`" in its name, create an appropriate one (see below), and register it using the window's "`Add`" button.

The MiKTeX FAQ suggests that you should create "`C:\Local TeX Files`", which is good if you manage your own machine, but often not even possible in corporate, or similar, environments — in such situations, the user may have no control over the hard disc of the computer, at all.

So the real criterion is that your local tree should be somewhere that *you*, rather than the system, control. Restrictive systems often provide a "home directory" for each user, mounted as a network drive; this is a natural home for the user's local tree. Other (often academic) environments assume the user is going to provide a memory stick, and will assign it a defined drive letter — another good candidate location. Note that the semantics of such a tree are indistinguishable from those of a "home" TEXMF tree.

There are circumstances when you might not wish to use the 'local' tree:

- if the package, or whatever, is "personal" (for example, something commercial that has been licensed to you alone, or something you're developing yourself), it should go in your "home" TEXMF tree;
- if you *know* that the package you are installing is a replacement for the copy on the TEXMF tree of your (La)TeX distribution, you should replace the existing copy in the TEXMF tree.

The reason one would put things on a local tree is to avoid their disappearance if the system is upgraded (or otherwise re-installed).

The reason one might place upgrades the distribution's main tree is to avoid confusion. Suppose you were to place the file on the local tree, and then install a new version of the distribution — you might have an effect like:

- distribution comes with package version n;
- you install package version $n+1$ on the local tree; and

- the updated distribution provides package version $n+2$.

In such a situation, you will find yourself using version $n+1$ (from the local tree) *after* the new distribution has been installed.

If you install in the local tree, the only way to avoid such problems is to carefully purge the local tree when installing a new distribution. This is tedious, if you're maintaining a large installation.

110 Where to install packages

We assume here that you have decided what tree to put your files in, after reading "choosing a TDS tree". We will therefore write `$TEXMF` for it, and you need to substitute the tree you decided on.

The basic idea is to imitate the directory structure in your existing tree(s). Here are some examples of where various sorts of files should go:

```
.sty, .cls or .fd: $TEXMF/tex/<format>/<package>/
.mf:   $TEXMF/fonts/source/<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>/
.otf:  $TEXMF/fonts/opentype/<supplier>/<font>/
.pool, .fmt, .base or .mem: $TEXMF/web2c
```

and for modern systems (those distributed in 2005 or later, using TDS v1.1 layouts):

```
.map:  $TEXMF/fonts/map/<syntax>/<bundle>/
.enc:  $TEXMF/fonts/enc/<syntax>/<bundle>/
```

(Map and encoding files went to directories under `$TEXMF/dvips/`, in earlier distributions.)

In the lists above ⟨*format*⟩ identifies the format the macros are designed for — it can be things such as `plain`, `generic` (i.e., any format), `latex` or `context` (or several less common formats).

For fonts, ⟨*font*⟩ refers to the font family (such as `cm` for Knuth's Computer Modern, `times` for Adobe's Times Roman). The supplier is usually obvious — the supplier "public" is commonly used for free fonts.

The ⟨*syntax*⟩ (for `.map` and `.enc` files) is a categorisation based on the way the files are written; candidates are names of programs such as *dvips* or *pdftex*.

"Straight" (La)TeX input can take other forms than the `.sty`, `.cls` or `.fd` listed above, too (apart from the 'obvious' `.tex`). Examples are (the obvious) `.tex`, `.lfd` for *babel* language definitions, `.sto` and `.clo` for package and class options, `.cfg` for configuration information, `.def` for variants (such as the types of devices *graphics* drives). The `README` of the package should tell you of any others, though sometimes that information is printed when the package's comments are stripped. All of these files should live together with the main package files.

Note that ⟨*font*⟩ may stand for a single font or an entire family: for example, files for all of Knuth's Computer Modern fonts are to be found in `.../public/cm`, with various prefixes as appropriate.

The font "supplier" *public* is a sort of hold-all for "free fonts produced for use with (La)TeX": as well as Knuth's fonts, *public*'s directory holds fonts designed by others (originally, but no longer exclusively, in Metafont).

Documentation for each package should all go, undifferentiated, into a directory on the `doc/` subtree of the TDS. The layout of the subtree is slightly different: `doc/latex` hosts all LaTeX documentation directories, but more fundamental things are covered, e.g., by `doc/etex` or `doc/xetex`.

111 Tidying up after installation

There's not usually a lot to do after you've completed the steps above — indeed, if you're merely installed files direct from the archive, or whatever, there will be precisely nothing left, by way of debris.

Things you might care to clean up are:

- the archive file, if you retrieved your data from the archive as a `.zip` file, or the like;
- the `.dtx` and `.ins` files, if you chose not to install them with the documentation; and
- the `.aux`, `.log`, `.idx`, etc., from building the documentation.

A simple way of achieving all this is to download to a working directory that was created for the purpose, and then to delete that directory and all its contents after you've finished installing.

112 Shortcuts to installing files

There are circumstances in which most of the fuss of installation can be bypassed:

- If you are a MiKTeX user, the MiKTeX package management system can usually help;
- Similarly, if you are a TeX Live user, the TeX Live manager can usually help;
- The package you want may already exist as a ZIP file formatted for direct installation.

113 Installation using MiKTeX package manager

Packages for use with MiKTeX are maintained very efficiently by the project managers (new packages and updates on CTAN ordinarily make their way to the MiKTeX package repository within a week). Thus it makes sense for the MiKTeX user to take advantage of the system rather than grinding through the steps of installation.

MiKTeX maintains a database of packages it "knows about", together with (coded) installation instructions that enable it to install the packages with minimal user intervention; you can update the database over the internet.

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 your package: check that box.

If you prefer a command-line utility, there's *mpm*. Open a command shell, and type:

```
mpm --install=<package>
```

(which of course assumes you know the name by which MiKTeX refers to your package).

114 Installation using TeX Live manager

TeX Live manager (*tlmgr*) is, by default, a shell (or Windows terminal window) command. There is voluminous documentation about it from the command

```
tldoc tlmgr
```

but basic operation is pretty straightforward. The manager needs to know where to download stuff from; the canonical setup is

```
tlmgr option repository http://mirror.ctan.org/systems/texlive/tlnet
```

which passes the decision to the mirror selector. You can (of course) specify a particular archive or mirror that you "trust", or even a local disc copy that you keep up-to-date (disc space and bandwidth are so cheap nowadays, that a "home mirror" of CTAN is a feasible proposition).

To update a single package, use:

```
tlmgr update <package>
```

To update everything you already have in your installation, use:

```
tlmgr update --all
```

115 Installing using ready-built ZIP files

Installing packages, as they ("traditionally") appear on CTAN, involves:

- identifying where to put the various files on an TDS tree,
- installing them, and
- a few housekeeping operations.

Most people, for most packages, find the first two steps onerous, the last being easy (unless it is forgotten!).

Ready-built ZIP files — also known as TDS-ZIP files — are designed to lighten the load of performing the first two steps of installation: they contain all the files that are to be installed for a given package, in their "correct" locations in a TDS tree.

To install such a file on a Unix system (we assume that you'll install into the local TEXMF tree, at `$TEXMFLOCAL`):

```
cd $TEXMFLOCAL
unzip $package.tds.zip
```

On a Windows system that is modern enough that it has a built-in ZIP unpacker, simply double-click on the file, and browse to where it's to be unpacked. (We trust that those using earlier versions of Windows will already have experience of using *WinZIP* or the like.)

Having unpacked the `.zip` archive, in most cases the only remaining chore is to update the file indexes — as in normal installation instructions. However, if the package provides a font, you also need to enable the font's map, which is discussed in "Installing a Type 1 font"

116 "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'. 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.

117 "Private" installations of files

It sometimes happens that you need a new version of some macro package or font, but that the machine you use is maintained by someone who's unwilling to update and won't give you privileges to do the job yourself. A "temporary" installation is sometimes the correct approach, but if there's the slightest chance that the installation will be needed on more than one project, temporary installations aren't right.

In circumstances where you have plenty of quota on backed-up media, or adequate local scratch space, the correct approach is to create a private installation of (La)TeX which includes the new stuff you need; this is the ideal, but is not generally possible.

So, since you can't install into the public `texmf` tree, you have to install into a `texmf` tree of your own; fortunately, the TDS standard allows for this, and modern distributions allow you to do it. The most modern distributions refer to the tree as `$TEXMFHOME`, but it used to be called `$HOMETEXMF`; so to check that your TeX system does indeed support the mechanism you should start with

```
kpsewhich -var-value TEXMFHOME
```

(for example). This will almost invariably return a pointer to a subdirectory `texmf` of your home directory; the commonest exception is Macintoshes, using MacTeX, where the diretory is conventionally `Library/texmf` in your home directory.

If you can confirm that the technique does indeed work, install your new package (or whatever) in the correct place in a tree based on `$HOME/texmf`, and generate an index of that tree

```
texhash $HOME/texmf
```

(the argument specifies which tree you are indexing: it's necessary since you don't, by hypothesis, have access to the main tree, and *texhash* without the argument would try to write the main tree.

There are two wrinkles to this simple formula: first, the installation you're using may *not* define a home TEXMF directory, and second, there may be some obstruction to using `$HOME/texmf` as the default name. In either case, a good solution is to have your own `texmf.cnf` — an idea that sounds more frightening that it actually is. The installation's existing file may be located with the command:

```
kpsewhich texmf.cnf
```

Take a copy of the file and put it into a directory of your own; this could be any directory, but an obvious choice is the `web2c` directory of the tree you want to create, i.e., `$HOME/texmf/web2c` or the like. Make an environment variable to point to this directory:

```
TEXMFCNF=$HOME/texmf/web2c
export TEXMFCNF
```

(for a Bourne shell style system), or

```
setenv TEXMFCNF $HOME/texmf/web2c
```

(for a C-shell style system). Now edit the copy of `texmf.cnf`

There will be a line in the existing file that defines the tree where everything searches; the simplest form of the line is:

```
TEXMF = !!$TEXMFMAIN
```

but, there are likely to be several alternative settings behind comment markers ("%"), and the person who installed your system may have left them there. Whatever, you need to modify the line that's in effect: change the above to three lines:

```
HOMETEXMF = $HOME/texmf
TEXMF = {$HOMETEXMF,!!$TEXMFMAIN}
% TEXMF = !!$TEXMFMAIN
```

the important point being that `$HOMETEXMF` must come before whatever was there before, inside the braces. For example, if the original was

```
TEXMF = {!!$LOCALTEXMF,!!$TEXMFMAIN}
```

it should be converted to:

```
HOMETEXMF = $HOME/texmf
TEXMF = {$HOMETEXMF,!!$LOCALTEXMF,!!$TEXMFMAIN}
% TEXMF = {!!$LOCALTEXMF,!!$TEXMFMAIN}
```

(retaining the original, as a comment, is merely an aide-memoir in case you need to make another change, later). The `!!` signs tell the file-searching library that it should insist on a *texhash*-ed directory tree; if you can count on yourself remembering to run *texhash* on your new tree every time you change it, then it's worth adding the marks to your tree:

```
TEXMF = {!!$HOMETEXMF,!!$LOCALTEXMF,!!$TEXMFMAIN}
```

as this will make (La)TeX find its files marginally faster.

Having made all these changes, (La)TeX should "just use" files in your new tree, in preference to anything in the main tree — you can use it for updates to packages in the main tree, as well as for installing new versions of things.

118 Installing a new font

Fonts are really "just another package", and so should be installed in the same sort of way as packages. However, fonts tend to be more complicated than the average package, and as a result it's sometimes difficult to see the overall structure.

Font files may appear in any of a large number of different formats; each format has a different function in a TeX system, and each is stored in a directory its own sub-tree in the installation's TDS tree; all these sub-trees have the directory `$TEXMF/fonts` as their root. A sequence of answers, below, describes the installation of fonts. Other answers discuss specific font families — for example, "using the concrete fonts".

119 Installing a font provided as Metafont source

Installing Metafont fonts is (by comparison with other sorts of font) rather pleasingly simple. Nowadays, they are mostly distributed just as the Metafont source, since modern TeX distributions are able to produce everything the user needs "on the fly"; however, if the distribution *does* include TFM files, install them too, since they save a little time and don't occupy much disc space. Always distrust distributions of PK font bitmap files: there's no way of learning from them what printer they were generated for, and naming schemes under different operating systems are another source of confusion.

"Where to install files" specifies where the files should go.

Further confusion is introduced by font families whose authors devise rules for automatic generation of Metafont sources for generating fonts at particular sizes; the installation has to know about the rules, as otherwise it cannot generate font files. No general advice is available, but most such font families are now obsolescent.

120 ‘Installing’ a PostScript printer built-in font

There is a “standard” set of fonts that has appeared in every PostScript printer since the second generation of the type. These fonts (8 families of four text fonts each, and three special-purpose fonts) are of course widely used, because of their simple availability. The set consists of:

- *Times* family (4 fonts)
- *Palatino* family (4 fonts)
- *New Century Schoolbook* family (4 fonts)
- *Bookman* family (4 fonts)
- *Helvetica* family (4 fonts)
- *Avant Garde* (4 fonts)
- *Courier* family (4 fonts)
- *Utopia* family (4 fonts)
- Zapf *Chancery* (1 font)
- Zapf *Dingbats* (1 font)
- *Symbol* (1 font)

All these fonts are supported, for LaTeX users, by the *psnfss* set of metrics and support files in the file `lw35nfss.zip` on CTAN. Almost any remotely modern TeX system will have some version of *psnfss* installed, but users should note that the most recent version has much improved coverage of maths with *Times* (see package *mathptmx*) and with *Palatino* (see package *mathpazo*, as well as a more reliable set of font metrics.

The archive `lw35nfss.zip` is laid out according to the TDS, so in principle, installation consists simply of “unzipping” the file at the root of a `texmf` tree.

Documentation of the *psnfss* bundle is provided in `psnfss2e.pdf` in the distribution.

`mathpazo.sty`: Part of the *psnfss* bundle

`mathptmx.sty`: Part of the *psnfss* bundle

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

121 Preparing a Type 1 font

The process of installing a Type 1 font set is rather convoluted, and we will deal with it in two chunks: first (in the present answer) preparing the font for installation, and second installing a Type 1 font).

Many fonts are supplied in (La)TeX ready form: such fonts need no preparation, and may be installed immediately.

However, if you purchase a font from a Type foundry (either direct or via one of the web stores), you are likely to need to ‘prepare’ it for use with (La)TeX. The rest of this answer discusses this preparation.

- Acquire the font. A very small set of Type 1 fonts is installed in most PostScript printers you will encounter. For those few (whose use is covered by the basic PSNFSS package), you don’t need the Type 1 font itself, to be able to print using the font.
  For all the myriad other Type 1 fonts, to be able to print using the font you need the Type 1 file itself. Some of these are available for free (they’ve either been donated to the public domain, or were developed as part of a free software project), but the vast majority are commercial products, requiring you to spend real money.
- Acquire the fonts’ AFM files. AFM files contain information from the font foundry, about the sizes of the characters in the font, and how they fit together. One measure of the quality of a font-supplier is that they provide the AFM files by default: if the files are not available, you are unlikely to be able to use the font with (La)TeX.
- Rename the AFM files and the Type 1 files to match the Berry font naming scheme.
- Generate TeX metric files from the AFM files. The commonest tool for this task is *fontinst*; the package documentation helps, but other guides are available (see below). The simplest possible script to pass to *fontinst* is:

```
\latinfamily{xyz}{}
\bye
```

where xyz is the Berry name of the font family. This simple script is adequate for most purposes: its output covers the font family in both T1 and OT1 font encodings. Nevertheless, with fancier fonts, more elaborate things are possible with *fontinst*: see its documentation for details.

Fontinst also generates map files, and LaTeX font definition (`.fd`) files.

Having traversed this list, you have a set of font files ready for installation.

`fontinst.sty`: `fonts/utilities/fontinst`

Type 1 installation guide: `info/Type1fonts/fontinstallationguide/fontinstallationguide.pdf`

122 Installing a Type 1 font

Once you have a prepared Type 1 font, either direct from CTAN or the like, or having 'prepared' it yourself, you can get on with installation.

The procedure is merely an extension of that for packages, etc., so much of what follows will be familiar:

- Install the files, in your local `texmf` tree (the advice about installing non-standard things applies here, too). The following list gives reasonable destinations for the various files related to a font family ⟨*fname*⟩:

  ```
  .pfb,
  .pfa  .../fonts/type1/<foundry>/<fname>
  .tfm  .../fonts/tfm/<foundry>/<fname>
  .vf   .../fonts/vf/<foundry>/<fname>
  .sty,
  .fd   .../tex/latex/<fname>
  .map  .../fonts/map/dvips/<foundry>
  ```

  but if you are lucky, you will be starting from a distribution from CTAN and there is a corresponding `.tds.zip` file: using this TDS-file saves the bother of deciding where to put your files in the TDS tree.
- Regenerate the file indexes (as described in package installation).
- Update the *dvips*, PDFTeX and other maps:
  - On any current TeX Live-based system, or a teTeX v3.0 system, execute the command

    ```
    updmap-sys --enable Map <fname>.map
    ```

    as root. (If you *can* use *updmap-sys* — do; if not — presumably because your (La)TeX system was set up by someone else — you have to fall back on plain *updmap*, but be aware that it's a potent source of confusion, setting up map sets that might be changed behind your back.)
  - On a current MiKTeX system, update the system file `updmap.cfg`, using the shell command

    ```
    initexmf --edit-config-file updmap
    ```

    adding a line at the end:

    ```
    Map <fname>.map
    ```

    for each font family ⟨*fname*⟩ you are adding to the system. Now generate revised maps with the shell command

    ```
    initexmf --mkmaps
    ```

    This, and other matters, are described in MiKTeX "advanced" documentation.

Both processes (preparing and installing a font) are very well (and thoroughly) described in Philipp Lehman's guide to font installation, which may be found on CTAN.

`fontinst.sty`: `fonts/utilities/fontinst`

Type 1 installation guide: `info/Type1fonts/fontinstallationguide/fontinstallationguide.pdf`

123 Installing the Type 1 versions of the CM fonts

This is a specialised case of installing a font, but it is almost never necessary — it's inconceivable that any (even remotely) recent system will *not* have the fonts already installed. You can confirm this (near-inevitable) fact by trying the fonts. On a system that uses *dvips* (almost all do), try the sequence:

```
latex sample2e
dvips -o sample2e.ps sample2e
```

at a "command prompt" (*shell*, in a Unix-style system, "DOS box" in a Windows system).

If the command works at all, the console output of the command will include a sequence of Type 1 font file names, listed as `<path/cmr10.pfb>` and so on; this is *dvips* telling you it's copying information from the Type 1 font, and you need do no more.

If the test has failed, you need to install your own set of the fonts; the distribution (including all the fonts the AMS designed and produced themselves) is now described as *amsfonts*. The bundle contains metric and map files — all you need to install the fonts.

AMS and CM fonts, in Type 1 format: fonts/amsfonts

K Fonts

K.1 Adobe Type 1 ("PostScript") fonts

124 Using Adobe Type 1 fonts with TeX

In order to use any font, TeX needs a *metric* file (TFM file). Several sets of metrics for common Adobe Type 1 fonts are available from the archives; for mechanisms for generating new ones, see metrics for PostScript fonts. You also need the fonts themselves; PostScript printers come with a set of fonts built in, but to extend your repertoire you usually need to buy from one of the many commercial font vendors (see, for example, "choice of fonts").

If you use LaTeX2e, access to your printer's fonts is offered by the PSNFSS package; the LaTeX3 project team declare that PSNFSS is a "required" part of a LaTeX distribution, and bug reports may be submitted via the LaTeX bugs system. PSNFSS gives you a set of packages for changing the default roman, sans-serif and typewriter fonts; *e.g.*, the *mathptmx* package will set up *Times Roman* as the main text font (and introduces mechanisms to typeset mathematics using *Times* and various more-or-less matching fonts), while package *avant* changes the sans-serif family to *AvantGarde*, and *courier* changes the typewriter font to *Courier*. To go with these packages, you need the font metric files and font description (`.fd`) files for each font family you want to use. For convenience, metrics for the 'common 35' PostScript fonts found in most PostScript 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*.

Some common problems encountered are discussed elsewhere (see problems with PS fonts).

Metrics for the 'Laserwriter' set of 35 fonts: macros/latex/required/psnfss/lw35nfss.zip

psnfss: macros/latex/required/psnfss

125 Previewing files using Type 1 fonts

Originally, free TeX previewers were only capable of displaying bitmap (PK) fonts, but free Type 1 font rendering software has been available for some time, and many previewers now use such facilities.

The alternative, for previewers, is automatic generation of the requisite PK files (using *gsftopk*, or similar, behind the scenes).

In the unlikely event that your previewer isn't capable of either, you have a couple options:

- 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 *gv* or shareware offering *GSview*.
- 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 (a process similar to the way some browsers fo the job 'automatically') This can produce excellent results, also suitable for printing with non-PostScript devices. (Note: if you purchased the fonts, it is advisable to check that their licence permits you to convert them, for private use, in this way.)

ghostscript: Browse support/ghostscript/GPL

gsftopk: fonts/utilities/gsftopk

GSview: Browse support/ghostscript/ghostgum

gv: Browse support/gv

ps2pk: fonts/utilities/ps2pk

126 TeX font metric files for Type 1 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. Most modern distributions 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 to *pretend* to TeX that the encoding is the same as it is used to. LaTeX2e has facilities for dealing with fonts in different encodings; read the *LaTeX Companion* 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' encoding.

An alternative favoured by some is Y&Y's "private" LY1 encoding, which is designed to sit well with "Adobe standard" encoded fonts. Basic macro support of LY1 is available: note that the "relation with Adobe's encoding" means that the LY1 user needs no virtual fonts.

Alan Jeffrey's *fontinst* package is an AFM to TFM converter written in TeX; it is used to generate the files used by LaTeX2e'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.

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 LaTeX2e compatible.

For the Macintosh (classic), 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.

dvips: dviware/dvips

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

fontinst: fonts/utilities/fontinst

LY1 support: fonts/psfonts/ly1

127 Deploying Type 1 fonts

For the LaTeX user trying to use the PSNFSS 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 previewing type 1 fonts.

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. 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 Computer Modern produces; but CM is extremely generous.

128 Choice of scalable outline fonts

If you are interested in text alone, you can in principle use any of the huge numbers of text fonts in Adobe Type 1, TrueType or OpenType formats. The constraint is, of course, that your previewer and printer driver should support such fonts (TeX itself *only* cares about metrics, not the actual character programs).

If you also need mathematics, then you are severely limited by the demands that TeX makes of maths fonts (for details, see the papers by B.K.P. Horn in *TUGboat* **14**(3), or by Thierry Bouche in *TUGboat* **19**(2)). For mathematics, 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. "Free" font families that will support TeX mathematics include:

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).

mathpazo version 1.003 (5 fonts) 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 LaTeX2e is available in PSNFSS; the fonts are licensed under the GPL, with legalese permitting the use of the fonts in published documents.

Fourier/Utopia (15 fonts) Michel Bovani
Fourier is a family built on Adobe *Utopia* (which has been released for usage free of charge by Adobe). The fonts provide the basic Computer Modern set of mathematical symbols, and add many of the AMS mathematical symbols (though you are expected to use some from the AMS fonts themselves). There are also several other mathematical and decorative symbols. The fonts come with a *fourier* package for use with LaTeX; text support of OT1 encoding is not provided — you are expected to use T1.

For a sample, see `http://www.tug.dk/FontCatalogue/utopia/`

Fourier/New Century Schoolbook Michael Zedler
Fouriernc is a configuration using the Fourier fonts in the context of New Century Schoolbook text fonts.

For a sample, see `http://www.tug.dk/FontCatalogue/newcent/`

KP-fonts The Johannes Kepler project
The *kp-fonts* family provides a comprehensive set of text and maths fonts. The set includes replacement fixed-width and sans fonts (though some reports have suggested that these are less successful, and their use may be suppressed when loading the fonts' *kpfonts* LaTeX support package).

For an example, see `http://www.tug.dk/FontCatalogue/kpserif/`

MathDesign (4 entire families. . . so far) Paul Pichaureau
This set so far offers mathematics fonts to match Adobe Utopia, URW Garamond, Bitstream Charter (all of which are separately available, on CTAN, in Type 1 format) and Fontsite Garamond (which is a commercial font, available from the http://www.fontsite.com/Pages/FFDownloads.html). There has been a little comment on these fonts, but none from actual users posted to the public forums. Users, particularly those who are willing to discuss their experiences, would obviously be welcome. Browse the CTAN directory and see which you want: there is a wealth of documentation and examples.

For samples of the free variants, see `http://www.tug.dk/FontCatalogue/charter/` for URW Charter, `http://www.tug.dk/FontCatalogue/garamond/` for URW Garamond and `http://www.tug.dk/FontCatalogue/utopia-md/` for Adobe Utopia.

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 is set with Bitmap fonts, and can be difficult on-screen in Acrobat reader 5 or earlier).

MTPro2 Lite Pubish or Perish (Michael Spivak)
A (functional) subset of the MathTime Pro 2 font set, that is made available, free, for general use. While it does not offer the full power of the commercial product (see below), it is nevertheless a desirable font set.

mathptmx 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 entirely acceptable, but can pass in many circumstances.

The real advantage is that the mathptm fonts are (effectively) free, and the resulting PostScript files can be freely exchanged. Support under LaTeX2e is available in PSNFSS.

For a sample, see `http://www.tug.dk/FontCatalogue/times/`

Computer Modern Bright Free scalable outline versions of these fonts do exist; they are covered below together with their commercial parallels.

URW Classico (4 fonts) LaTeX support by Bob Tennent
These are clones of Zapf's Optima available from CTAN (for non-commercial use only). Mathematics support can be provided by using packages *eulervm* or *sansmath*. As a sans-serif font family, Optima is especially suitable for presentations.

The excellent *font catalogue* keeps an up-to-date list which describes the fonts by giving names and short examples, only. (At the time of writing — June 2008 — the list has several that are only scheduled for inclusion here.

Another useful document is Stephen Hartke's "Free maths font survey", which is available on CTAN in both PDF and HTML formats. The survey covers most of the fonts mentioned in the font catalogue, but also mentions some (such as *Belleek* that the catalogue omits.

Fonts capable of setting TeX mathematics, that are available commercially, include:

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`

CH Math (15 fonts) MicroPress Inc.
CH Math is a family of slab serif fonts, designed as a maths companion for Bitstream Charter. (The distribution includes four free Bitstream text fonts, in addition to the 15 hand-hinted MicroPress fonts.) For further details (including samples) see
`http://www.micropress-inc.com/fonts/chmath/chmain.htm`

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. The *hfbright* bundle offers free Type 1 fonts for text using the OT1 encoding — the *cm-super* fonts provide the fonts in T1 text encoding but don't support CM bright mathematics.

For further details of Micropress' offering (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`

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. Support under LaTeX is available under the auspices of the PSNFSS, and pre-built metrics are also provided.

TUG has the right to distribute these fonts; the web site "Lucida and TUG" has details.

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, which are not currently available.)

MathTime Pro2 Publish or Perish (Michael Spivak)
This latest instance of the MathTime family covers all the weights (medium, bold and heavy) and symbols of previous versions of MathTime. In addition it has a much extended range of symbols, and many typographic improvements that make for high-quality documents. The fonts are supported under both Plain TeX and LaTeX2e, and are exclusively available for purchase from Personal TeX Inc.

For further details and samples and fliers, see `http://www.pctex.com/mtpro2.html`

Minion Pro and MnSymbol Adobe, LaTeX support and packaging by Achim Blumensath *et al*.
Minion Pro derives from the widely-available commercial OpenType font of the same name by Adobe; scripts are provided to convert relevant parts of it to Adobe Type 1 format. The *MinionPro* package will set up text and maths support using *Minion Pro*, but a separate (free) font set *MnSymbol* greatly extends the symbol coverage.

PA Math PA Math is a family of serif fonts loosely based on the Palatino (TM) typeface. PA Math 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`

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`

Two other font sets should be mentioned, even though they don't currently produce satisfactory output — their author is no longer working on them, and several problems have been identified:

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 *Palatino*;
- 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 fonts are not perfect; the widths assigned to the characters in the `.tfm` file are wrong for some glyphs; this can cause sequences of characters to "look wrong", or in some cases even to overlap.

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.

The fonts are not perfect; the widths assigned to the characters in the `.tfm` file are wrong for some glyphs; this can cause sequences of characters to "look wrong", or in some cases even to overlap.

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

txfontsb set version 1.00 by Young Ryu and Antonis Tsolomitis
The *txfontsb* bundles *txfonts*, extended to provide a Small Caps set, Old-Style numbers and Greek text (from the GNU Freefont set). Documentation is available for this variant, too.

Finally, one must not forget:

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.

We observe a very limited selection of commercial maths font sets; a maths font has to be explicitly designed for use with TeX, which is an expensive business, and is of little appeal in other markets. Furthermore, the TeX market for commercial fonts is minute by comparison with the huge sales of other font sets.

Text fonts in Type 1 format are available from many vendors including Adobe, Monotype and Bitstream. However, be careful with cheap font "collections"; many of them dodge copyright restrictions by removing (or crippling) parts of the font programs such as hinting. Such behaviour is both unethical and bad for the consumer. 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 (which you need to make TFM files).

TrueType was for a long time the "native" format for Windows, but MicroSoft joined the development of the OpenType specification, and 'modern' windows will work happily with fonts in either format. Some TeX implementations such as TrueTeX 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 used on a system running Windows XP or later, TrueTeX can also use Adobe Type 1 fonts.)

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 (PostScript or PDF) for their output device.

CM family collection: Distributed as part of `fonts/amsfonts`

AMS font collection: Distributed as part of `fonts/amsfonts`

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

URW Classico fonts: `fonts/urw/classico`

`CM-super` collection: `fonts/ps-type1/cm-super`

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

`fourier` (including metrics and other support for *utopia*: `fonts/fourier-GUT`

`fouriernc`: `fonts/fouriernc`

`hfbright` collection: `fonts/ps-type1/hfbright`

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

`kpfonts` family: `fonts/kpfonts`

`Lucida Bright/Math metrics`: `fonts/psfonts/bh/lucida`

Lucida PSNFSS support: macros/latex/contrib/psnfssx/lucidabr

MathDesign collection: fonts/mathdesign

Maths Font Survey: info/Free_Math_Font_Survey/en/survey.pdf (PDF) or info/Free_Math_Font_Survey/en/survey.html (HTML)

MathTime Pro 2 Lite: fonts/mtp2lite

Minion Pro support: fonts/minionpro

MnSymbol family: fonts/mnsymbol

pxfonts: fonts/pxfonts

sansmath.sty: macros/latex/contrib/sansmath

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

txfonts: fonts/txfonts

utopia fonts: fonts/utopia

129 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 outlined in generating PostScript for PDF. 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* v 5.90 (and later versions), which should be available in any recent TeX distribution.

K.2 Macros for using fonts

130 Using non-standard fonts in Plain TeX

Plain TeX (in accordance with its description) doesn't do anything fancy with fonts: it sets up the fonts that Knuth found he needed when writing the package, and leaves you to do the rest.

To use something other than Knuth's default, you can use Knuth's mechanism, the `\font` primitive:

```
\font\foo=nonstdfont
...
\foo
Text set using nonstdfont ...
```

The name you use (`nonstdfont`, above) is the name of the `.tfm` file for the font you want.

If you want to use an italic version of `\foo`, you need to use `\font` again:

```
\font\fooi=nonstdfont-italic
...
\fooi
Text set using nonstdfont italic...
```

This is all very elementary stuff, and serves for simple use of fonts. However, there are wrinkles, the most important of which is the matter of font encodings. Unfortunately, many fonts that have appeared recently simply don't come in versions using Knuth's eccentric font encodings — but those encodings are built into Plain TeX, so that some macros of Plain TeX need to be changed to use the fonts. LaTeX gets around all these problems by using a "font selection scheme" — this 'NFSS' ('N' for 'new', as opposed to what LaTeX 2.09 had) carries around with it separate information about the fonts you use, so the changes to encoding-specific commands happen automagically.

If you only want to use the EC fonts, you can in principle use the *ec-plain* bundle, which gives you a version of Plain TeX which you can run in the same way that you run Plain TeX using the original CM fonts, by invoking *tex*. (*Ec-plain* also extends the EC fonts, for reasons which aren't immediately clear, but which might cause problems if you're hoping to use Type 1 versions of the fonts.)

The *font_selection* package provides a sort of halfway house: it provides font face and size, but not family selection. This gives you considerable freedom, but leaves you stuck with the original CM fonts. It's a compact solution, within its restrictions.

Other Plain TeX approaches to the problem (packages *plnfss*, *fontch* and *ofs*) break out of the Plain TeX model, towards the sort of font selection provided by ConTeXt and LaTeX — font selection that allows you to change family, as well as size and face. The remaining packages all make provision for using encodings other than Knuth's OT1.

Plnfss has a rather basic set of font family details; however, it is capable of using font description (`.fd`) files created for LaTeX. (This is useful, since most modern mechanisms for integrating outline fonts with TeX generate `.fd` files in their process.)

Fontch has special provision for T1 and TS1 encodings, which you select by arcane commands, such as:

```
\let\LMTone\relax
\input fontch.tex
```

for T1.

Ofs seems to be the most thoroughly thought-through of the alternatives, and can select more than one encoding: as well as T1 it covers the encoding IL2, which is favoured in the Czech Republic and Slovakia. *Ofs* also covers mathematical fonts, allowing you the dubious pleasure of using fonts such as the *pxfonts* and *txfonts*.

The *pdcmac* Plain TeX macro package aims to be a complete document preparation environment, like Eplain. One of its components is a font selection scheme, *pdcfsel*, which is rather simple but adequately powerful for many uses. The package doesn't preload fonts: the user is required to declare the fonts the document is going to use, and the package provides commands to select fonts as they're needed. The distribution includes a configuration to use Adobe 'standard' fonts for typesetting text. (Eplain itself seems not to offer a font selection scheme.)

The *font-change* collection takes a rather different approach — it supplies what are (in effect) a series of templates that may be included in a document to change font usage. The package's documentation shows the effect rather well.

Simply to change font *size* in a document (i.e., not changing the default font itself), may be done using the rather straightforward *varisize*, which offers font sizes ranging from 7 points to 20 points (nominal sizes, all). Font size commands are generated when any of the package files is loaded, so the `11pt.tex` defines a command `\elevenpoint`; each of the files ensures there's a "way back", by defining a `\tenpoint` command.

ec-plain: `macros/ec-plain`

font-change: `macros/plain/contrib/font-change`

fontch: `macros/plain/contrib/fontch`

font_selection: `macros/plain/contrib/font_selection`

ofs: `macros/generic/ofs`

pdcmac: `macros/plain/contrib/pdcmac`

plnfss: `macros/plain/plnfss`

varisize: `macros/plain/contrib/varisize`

K.3 Particular font families

131 Using the "Concrete" fonts

The Concrete Roman fonts were designed by Don Knuth for a book called "Concrete Mathematics", which he wrote with Graham and Patashnik (*the* Patashnik, of BibTeX fame). Knuth only designed text fonts, since the book used the Euler fonts for mathematics. The book was typeset using Plain TeX, of course, with additional macros that may be viewed in a file `gkpmac.tex`, which is available on CTAN.

The packages *beton*, *concmath*, and *ccfonts* are LaTeX packages that change the default text fonts from Computer Modern to Concrete. Packages *beton* and *ccfonts* also slightly increase the default value of `\baselineskip` to account for the rather heavier weight of the Concrete fonts. If you wish to use the *Euler* fonts for mathematics, as Knuth did, there's the *euler* package which has been developed from Knuth's own Plain TeX-based set: these macros are currently deprecated (they clash with many things, including AMSLaTeX). The independently-developed *eulervm* bundle is therefore preferred to the *euler* package. (Note that installing the *eulervm* bundle involves installing a series of virtual fonts. While most modern distributions seem to have the requisite files installed by default, you may find you have to install them. If so, see the file `readme` in the *eulervm* distribution.)

A few years after Knuth's original design, Ulrik Vieth designed the Concrete Math fonts. Packages *concmath*, and *ccfonts* also change the default math fonts from Computer Modern to Concrete and use the Concrete versions of the AMS fonts (this last behaviour is optional in the case of the *concmath* package).

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

```
\renewcommand{\bfdefault}{sbc}
```

to the preamble of your document.

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

beton.sty: `macros/latex/contrib/beton`

ccfonts.sty: `macros/latex/contrib/ccfonts`

CM-Super fonts: `fonts/ps-type1/cm-super`

concmath.sty: `macros/latex/contrib/concmath`

Concmath fonts: `fonts/concmath`

Concrete fonts: `fonts/concrete`

euler.sty: `macros/latex/contrib/euler`

eulervm bundle: `fonts/eulervm`

gkpmac.tex: `systems/knuth/local/lib/gkpmac.tex`

132 Using the Latin Modern fonts

The *lm* fonts are an exciting addition to the armoury of the (La)TeX user: high quality outlines of fonts that were until recently difficult to obtain, all in a free and relatively compact package. However, the spartan information file that comes with the fonts remarks "It is presumed that a potential user knows what to do with all these files". This answer aims to fill in the requirements: the job is really not terribly difficult.

Note that teTeX distributions, from version 3.0, already have the *lm* fonts: all you need do is use them. The fonts may also be installed via the package manager, in a current MiKTeX system. The remainder of this answer, then, is for people who don't use such systems.

The font (and related) files appear on CTAN as a set of single-entry TDS trees — `fonts`, `dvips`, `tex` and `doc`. The `doc` subtree really need not be copied (it's really a pair of sample files), but copy the other three into your existing Local `$TEXMF` tree, and update the filename database.

Now, incorporate the fonts in the set searched by PDFLaTeX, *dvips*, *dvipdfm*/*dvipdfmx*, your previewers and Type 1-to-PK conversion programs, by

- On a teTeX system earlier than version 2.0, edit the file `$TEXMF/dvips/config/updmap` and insert an absolute path for the `lm.map` just after the line that starts `extra_modules="` (and before the closing quotes).
- On a teTeX version 2.0 (or later), execute the command

```
updmap --enable Map lm.map
```

- On a MiKTeX system earlier than version 2.2, the "Refresh filename database" operation, which you performed after installing files, also updates the system's "PostScript resources database".
- On a MiKTeX system, version 2.2 or later, update `updmap.cfg` as described in the MiKTeX online documentation. Then execute the command `initexmf --mkmaps`, and the job is done.

To use the fonts in a LaTeX document, you should

```
\usepackage{lmodern}
```

this will make the fonts the default for all three LaTeX font families ("roman", "sans-serif" and "typewriter"). You also need

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

for text, and

```
\usepackage{textcomp}
```

if you want to use any of the TS1-encoding symbols. There is no support for using fonts according to the OT1 encoding.

Latin Modern fonts: `fonts/lm`

K.4 Metafont fonts

133 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 — modern (La)TeX systems contain rather few Metafont fonts of any significance, and when Metafont output is needed the font generation is done, automatically, "on the fly".

If you find you have some special requirement that the system doesn't satisfy, you need to know about Metafont in rather more detail. Metafont, unlike TeX, requires customisation for each output device: such customisation is conventionally held in a "mode" associated with the device. Modes are commonly defined using the `mode_def` convention described on page 94 of *The Metafontbook* (see TeX-related books). Your distribution should provide a file, conventionally called `local.mf`, containing all the `mode_def`s you will be using. In the unlikely event that `local.mf` doesn't already exist, Karl Berry's collection of modes (`modes.mf`) is a good starting point (it can be used as a '`local.mf`' without modification in a modern implementation of Metafont). Settings for new output devices are added to `modes.mf` as they become available.

Now create a `plain` base file using *mf* (in "initialisation" mode), `plain.mf`, and `local.mf`:

```
% mf -ini
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). Move the file to the directory containing the base files on your system, and run *texhash* as necessary.

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 TeX Live (and earlier) distributions this does indeed happen, but we could for instance define a command *plainmf*[2] which executes '`mf -base=plain`' (or, in more traditional style '`mf &plain`') which loads the `plain` base file.

The usual way to create a font with Metafont (with an appropriate base file loaded) is to start Metafont's input 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' mode and Metafont will produce an output file called `<font file name>.2602gf`) The `<magnification>` is a floating point number or a 'magstep' (magsteps define sizes by stating how many times you need to multiply a base size by `1.2`, so for a base size of `10`, `magstep 1` is `12`, `magstep 2` is `14.4` If `mag=<magnification>` is omitted, then the default is `1` (`magstep 0`). For example, to generate *cmr10* at `12pt` for an Epson, printer you might 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 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.

Other sources of help are discussed in our list of Metafont and Metapost Tutorials.

`modes.mf`: `fonts/modes/modes.mf`

[2]On the grounds that a command *plain* could be misconstrued as a reference to Plain TeX

134 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` (the file name may be mangled if you are using an operating system which doesn't permit long file names) and `cmr10.log`).

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

To preview or to produce printed output, the DVI processor will need a font raster file; this is what the GF file provides. However, while there used (once upon a time) to be DVI processors that could use GF files, modern processors use packed raster (PK) files (incidentally, PDFTeX also uses PK files if nothing "better" is available, but see fuzzy fonts in PDF). 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 be needed again, unless there was some sort of problem in the Metafont run, and need not therefore be kept.

135 Acquiring bitmap fonts

When CTAN was young, 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. (There are separate sets for write-black and write-white printers, as well as sets at 120 dpi and 240 dpi.)

There used to regular requests that CTAN should hold a wider range of resolutions, but they were resisted for two reasons:

- The need to decide which printers to generate fonts for. The broad-brush approach taken for 300 dpi printers was (more or less) justified back then, given the dominance of certain printer 'engines', but nowadays one could not make any such assumption.
- Given the above, it has been near-impossible to justify the space that would be required by a huge array of bitmap fonts.

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 TeX Live, at least; otherwise such a person should consult "the use of Metafont)".

If your output is to a PostScript-capable device, or if your output is destined to be converted to PDF, you should switch to using Type 1 versions of the CM fonts. Two free sets are available; the older (*bakoma*) is somewhat less well produced than the *bluesky* fonts, which were originally professionally produced and sold, but were then released for general public use by their originators Y&Y and Bluesky Research, in association with the AMS and other scientific publishers (they are nowadays available under the SIL's Open Fonts Licence). The two sets contain slightly different ranges of fonts, but you are advised to use the *bluesky* set except when *bakoma* is for some reason absolutely unavoidable. In recent years, several other 'Metafont' fonts have been converted to Type 1 format; it's uncommon ever to need to generate bitmap fonts for any purpose other than previewing — see "previewing documents with Type 1 fonts" — if even then.

More modern fonts may be used in place of the Computer Modern set. The EC fonts and the Latin Modern fonts are both close relatives with wider ranges of glyphs to offer.

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

Bluesky fonts: Distributed as part of `fonts/amsfonts`

CM fonts (write-black printers): `fonts/cm/pk/pk300.zip`

CM fonts (write-white printers): `fonts/cm/pk/pk300w.zip`

EC fonts (Type 1 format): fonts/ps-type1/cm-super

Latin Modern fonts: fonts/lm

L Hypertext and PDF

136 Making PDF documents from (La)TeX

There are three general routes to PDF output: Adobe's original 'distillation' route (via PostScript output), direct conversion of a DVI file, and the use of a direct TeX-like PDF generator such as PDFTeX.

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 appropriate tools installed) pass the output through a PDFwriter in place of a printer driver. This route is only appropriate for simple documents: PDF writers cannot create hyperlinks;
- process the document with "vanilla" LaTeX and generate PDF direct from the DVI using *dvipdfm*/*dvipdfmx*; or
- process the document direct to PDF with PDFTeX, LuaTeX, XeTeX or VTeX. PDFTeX, LuaTeX and XeTeX have the advantage of availability for a wide range of platforms, but VTeX is available only for Windows (as a commercial product), or free of charge, but unsupported, for Linux or OS/2.

To translate all the LaTeX cross-referencing into Acrobat links, you need a LaTeX package to redefine the internal commands. There are two of these for LaTeX, both capable of conforming to the HyperTeX specification: Heiko Oberdiek's *hyperref*, and Michael Mehlich's *hyper*. (In practice, almost everyone uses *hyperref*; *hyper* hasn't been updated since 2000.) *Hyperref* can often determine how it should generate hypertext from its environment, but there is a wide set of configuration options you can give via `\usepackage`. The package can operate using PDFTeX primitives, the hyper-TeX `\special`s, or DVI driver-specific `\special` commands. Both *dvips* and Y&Y's *DVIPSONE* can translate the DVI with these `\special` commands into PostScript acceptable to Distiller, and *dvipdfm* and *dvipdfmx* have `\special` commands of their own.

If you use Plain TeX, the Eplain macros can help you create PDF documents with hyper-references. It can operate using PDFTeX primitives, or `\special` commands for the *dvipdfm*/*dvipdfmx* DVI drivers.

While there is no free implementation of all of *Adobe Distiller*'s functionality, any but the very oldest versions of *Ghostscript* provide pretty reliable distillation (but beware of the problems with *dvips* output for distillation).

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* combined with *gv* or *GSview* can display and print PDF files, as can *xpdf*.

In many circumstances, *Ghostscript* combined with a viewer application is actually preferable to Acrobat Reader. For example, on Windows Acrobat Reader locks the `.pdf` file it's displaying: this makes the traditional (and highly effective) (La)TeX development cycle of "Edit→Process→Preview" become rather clumsy — *GSview* doesn't make the same mistake.

Acrobat Reader: download from http://get.adobe.com/reader

dvipdfm: dviware/dvipdfm

dvipdfmx: dviware/dvipdfmx

ghostscript: Browse support/ghostscript/GPL

GSview: Browse support/ghostscript/ghostgum

gv: Browse support/gv

hyper.sty: macros/latex/contrib/hyper

hyperref.sty: macros/latex/contrib/hyperref

137 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:

- start from (La)TeX, and use one of a number of techniques to translate (more or less) directly to HTML;
- Start from *texinfo* source, and use the *info* viewer, or convert the *texinfo* source to HTML using *texi2html*;
- Start from (La)TeX; use PDFTeX, XeTeX or LuaTeX to produce PDF, using *hyperref* to construct hyperlinks.
- Start from (unconventional) (La)TeX which use the hyperTeX conventions.

texinfo: macros/texinfo/texinfo

138 The *hyperTeX* project

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

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

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

139 Quality of PDF from PostScript

Any reasonable PostScript, including any output of *dvips*, may be converted to PDF, using (for example) a sufficiently recent version of *ghostscript*, Frank Siegert's (shareware) *PStill*, or Adobe's (commercial) *Distiller*.

But, although the job may (almost always) be done, the results are often not acceptable: the most frequent problem is bad presentation of the character glyphs that make up the document. The following answers offer solutions to this (and other) problems of bad presentation. Issues covered are:

- Wrong type of fonts used, which is the commonest cause of fuzzy text.
- *Ghostscript* too old, which can also result in fuzzy text.
- Switching to font encoding T1 encoding, which is yet another possible cause of fuzzy text.
- Another problem — missing characters — arises from an aged version of *Adobe Distiller*.
- Finally, there's the common confusion that arises from using the *dvips* configuration file `-Ppdf`, the weird characters.

It should be noted that *Adobe Reader* 6 (released in mid-2003, and later versions) does not exhibit the "fuzziness" that so many of the answers below address. This is of course good news: however, it will inevitably be a long time before every user in the world has this (or later) versions, so the remedies below are going to remain for some time to come.

The problems are also discussed, with practical examples, in Mike Shell's *testflow* package, which these FAQs recommend as a "specialised tutorial.

testflow: `macros/latex/contrib/IEEEtran/testflow`

140 The wrong type of fonts in PDF

This is far the commonest problem: the symptom is that text in the document looks "fuzzy".

Most people use *Adobe Acrobat Reader* to view their PDF: *Reader* is distributed free of charge, and is widely available, for all its faults. One of those faults is its failure to deal with bitmap fonts (at least, in all versions earlier than version 6, all of which copies are pretty old, now ... but some are occasionally found).

So we don't want bitmap fonts in our PostScript: with them, characters show up in *Reader*'s display as blurred blobs which are often not even recognisable as the original letter, and are often not properly placed on the line. Nevertheless, even now, most TeX systems have *dvips* configured to use `.pk` files in its output. Even PDFTeX will use `.pk` files if it can see no alternative for a font in the document it is processing.

Our remedy is to use "Adobe Type 1" versions of the fonts we need. Since Adobe are in the business of selling Type 1 fonts, *Reader* was of course made to deal with them really rather well, from the very beginning.

Of course, if your document uses nothing but fonts that came from Adobe in the first place — fonts such as *Times* that appear in pretty much every PostScript printer, or such as Adobe *Sabon* that you pay extra for — then there's no problem.

But most people use *Computer Modern* to start with, and even those relative sophisticates who use something as exotic as *Sabon* often find themselves using odd characters from CM without really intending to do so. Fortunately, rather good versions of the CM fonts are available from the AMS (who have them courtesy of Blue Sky Research and Y&Y).

Most modern systems have the fonts installed ready to use; and any system installed less than 3 years ago has a *dvips* configuration file 'pdf' that signals the use of the CM fonts, and also sets a few other parameters to improve *dvips*' output. Use this configuration as:

```
dvips -Ppdf myfile -o myfile.ps
```

This may produce a warning message about failing to find the configuration file:

```
dvips: warning: no config file for 'pdf'
```

or something similar, or about failing to find a font file:

```
dvips: ! Couldn't find header file cmr10.pfb
```

Either of these failures signals that your system doesn't have the fonts in the first place.

A way of using the fonts that doesn't involve the sophistication of the `-Ppdf` mechanism is simply to load maps:

```
dvips -Pcmz -Pamz myfile -o myfile.ps
```

You may encounter the same warning messages as listed above.

If your system does not have the fonts, it won't have the configuration file either; however, it might have the configuration file without the fonts. In either case, you need to install the fonts.

141 Fuzzy fonts because *Ghostscript* too old

So you've done everything the FAQ has told you that you need, correct fonts properly installed and appearing in the *dvips* output, but *still* you get fuzzy character output after distilling with *ghostscript*.

The problem could arise from too old a version of *ghostscript*, which you may be using directly, or via a script such as *ps2pdf* (distributed with *ghostscript* itself), *dvipdf*, or similar. Though *ghostscript* was capable of distillation from version 5.50, that version could only produce bitmap Type 3 output of any font other than the fundamental 35 fonts (*Times*, *Helvetica*, etc.). Later versions added 'complete' distillation, but it wasn't until version 6.50 that one could rely on it for everyday work.

So, if your PDF output still looks fuzzy in *Acrobat Reader*, upgrade *ghostscript*. The new version should be at least version 6.50, of course, but it's usually good policy to go to the most recent version (version 8.12 at the time of writing — 2003).

142 Fonts go fuzzy when you switch to T1

You've been having problems with hyphenation, and someone has suggested that you should use "`\usepackage[T1]{fontenc}`" to help sort them out. Suddenly you find that your final PDF has become fuzzy. The problem may arise whether you are using PostScript output and then distilling it, or you are using PDFTeX for the whole job.

In fact, this is the same problem as most others about the quality of PDF: you've abandoned your previous setup using Type 1 versions of the CM fonts, and *dvips* has inserted Type 3 versions of the EC fonts into your document output. (See "Adobe font types for details of these font types; also, note that the font *encoding* T1 has nothing directly to do with the font *format* Type 1).

However, as noted in 8-bit Type 1 fonts, Type 1 versions of CM-like fonts in T1 (or equivalent) encoding are now available, both as "real" fonts, and as virtual font sets. One solution, therefore, is to use one of these alternatives.

The alternative is to switch font family altogether, to something like *Times* (as a no-thought default) or one of the many more pleasing Adobe-encoded fonts. The default action of *fontinst*, when creating metrics for such a font, is to create settings for both OT1 and T1 encodings, so there's little change in what goes on (at the user level) even if you have switched to T1 encoding when using the fonts.

143 Characters missing from PDF output

If you're using *Acrobat Distiller* to create your PDF output, you may find characters missing. This may manifest itself as messed-up maths equations (missing "−" signs, for example), or bits missing from large symbols. Early versions of *Distiller* used to ignore character positions 0–31 and 128–159 of every font: Adobe's fonts never use such positions, so why should *Distiller*?

Well, the answer to this question is "because Adobe don't produce all the world's fonts" — fonts like *Computer Modern* were around before Adobe came on the scene, and *they* use positions 0–31. Adobe don't react to complaints like that in the previous sentence, but they do release new versions of their programs; and *Distiller*, since at least version 4.0, *has* recognised the font positions it used to shun.

Meanwhile, TeX users with old versions of *Distiller* need to deal with their fonts. *Dvips* comes to our aid: the switch `-G1` ("remap characters"), which moves the offending characters out of the way. The PDF configuration file (`-Ppdf`), recommended above, includes the switch.

The switch is not without its problems; pre-2003 versions of *dvips* will apply it to Adobe fonts as well, causing havoc, but fortunately that problem is usually soluble. However, a document using both CM and Adobe-specified fonts is stuck. The only real solution is either to upgrade *dvips*, or to spend money to upgrade *Distiller*.

144 Finding '8-bit' Type 1 fonts

Elsewhere, answers to these FAQs recommend that you use an '8-bit' font to permit accentuation of inflected languages, and also recommend the use of Type 1 fonts to ensure that you get good quality PDF. These recommendations used to be contradictory: one could not just "switch" from the free CM fonts to free Cork- (or similarly) encoded Type 1 fonts. The first approach that started to alleviate these problems, was the development of virtual fonts that make a good approach to the Cork encoding (see below). Now, however, we have "true" Type 1 fonts available: as always, we have an embarrassment of riches with three free alternatives, and one commercial and one shareware version.

CM-super is an auto-traced set which encompasses all of the T1 and TS1 encodings as well as the T2* series (the family of encodings that cover languages based on Cyrillic alphabets). These fonts are pretty easy to install (the installation instructions are clear), but they are huge: don't try to install them if you're short of disc space.

CM-LGC is a similar "super-font" set, but of much more modest size; it covers T1, TS1 and T2A encodings (as does *CM-super*, and also covers the LGR encoding (for typesetting Greek, based on Claudio Beccari's Metafont sources). *CM-LGC* manages to be small by going to the opposite extreme from *CM-super*, which includes fonts at all the sizes supported by the original EC (a huge range); *CM-LGC* has one font per font shape, getting other sizes by scaling. There is an inevitable loss of quality inherent in this approach, but for the disc-space-challenged machine, *CM-LGC* is an obvious choice.

Tt2001 is a simple scan of the EC and TC fonts, and has some virtues — it's noticeably smaller than *CM-super* while being less stark than *CM-LGC*.

Latin Modern is produced using the program *MetaType1*. The *Latin Modern* set comes with T1, TS1 LY1 encoded variants (as well as a variant using the Polish QX encoding); for the glyph set it covers, its outlines seem rather cleaner than those of *CM-super*. *Latin Modern* is more modest in its disc space demands than is *CM-super*, while not being nearly as stark in its range of design sizes as is *CM-LGC* — *Latin Modern*'s fonts are offered in the same set of sizes as the original *CM* fonts. It's hard to argue with the choice: Knuth's range of sizes has stood the test of time, and is one of the bases on which the excellence of the TeX system rests.

Virtual fonts 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 and straightforward 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). Neither offers characters such as `eth` and `thorn` (used in, for example, in Icelandic), but the *aecompl* package works with the *ae* fonts to provide the missing characters from the EC fonts (i.e., as bitmaps).

The sole remaining commercial CM-like 8-bit font comes from Micropress, who 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 "commercial distributions".

The shareware BaKoMa TeX distribution 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.)

Finally, 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). However, 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 — "choice of scalable fonts".

ae fonts: `fonts/ae`

`aecompl.sty`: Distributed with `fonts/ae`

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

BaKoMa fonts: Browse `systems/win32/bakoma/fonts`

CM-LGC fonts: `fonts/ps-type1/cm-lgc`

CM-super fonts: `fonts/ps-type1/cm-super` (beware: very large download)

Latin Modern fonts: fonts/lm

tt2001 fonts: fonts/ps-type1/tt2001

zefonts: fonts/zefonts

145 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.

If your source file is older than *pkfix* can deal with, there's still a modicum of hope: *pkfix-helper* examines the bitmap fonts in a document, compares them with the metric (`.tfm`) fonts on your system and comes to a view of which font might be which. The program reports on "poor" matches, and there are options for confirming, or replacing, its guesses. The technique (which sounds implausible) is successful enough to be worth a try.

A further option is Frank Siegert's (shareware) PStill, which is capable of processing the PostScript it is distilling, and one option is to replace bitmap fonts in the file with Type 1 versions.

pkfix: support/pkfix

pkfix-helper: support/pkfix-helper

146 *Hyperref* and repeated page numbers

The *book* class (and its friends and relations) automatically changes the display of page numbers in the frontmatter of the document to lower-case roman. This is fine for human readers, but if *hyperref* has been misconfigured, the existence of pages have the same page number can cause problems. Fortunately, the configuration options to make *hyperref* "do the right thing" are (by default) set up to avoid problems.

The two options in question are:

`plainpages=false` Make page anchors using the formatted form of the page number. With this option, *hyperref* writes different anchors for pages 'ii' and '2'. (This is the default value for the option, which is a *good thing*...) If the option is set '`true`' *hyperref* writes page anchors as the arabic form of the page number, rather than the formatted form that gets printed; this is not usually appropriate.

`pdfpagelabels` Set PDF page labels; i.e., write the value of `\thepage` to the PDF file so that *Acrobat Reader* can display the page number as (say) 'ii (4 of 40)' rather than simply '4 of 40'.

The two should be used whenever page numbering is not just '1..*n*'; they may be used independently, but usually are not.

The recipe isn't perfect: it relies on `\thepage` being different for every page in the document. A common problem arises when there is an unnumbered title page, after which page numbers are reset: the PDFTeX warning of "duplicate destinations" will happen in this case, regardless of the options.

hyperref.sty: macros/latex/contrib/hyperref

147 Searching PDF files

In principle, you can search a PDF file: the text of the file is available to the viewer, and at least some viewers provide a search facility. (It's not the fastest thing in the world, but it does help in some circumstances.)

However, there is a problem: the viewer wants to look at Unicode text, but no ordinary TeX-based system deals in Unicode text. Fortunately for us Anglophones, this is is hardly ever a problem for our text, since even Knuth's "OT1" encoding matches ASCII (and hence the lowest 128 characters of Unicode) for most things printable. However, using the inflected characters of Continental European languages, or anything that doesn't use a Latin alphabet, there is potential for problems, since TeX's view of what a font is doesn't map PDF's and the reader won't understand…

… Unless you use the *cmap* package with PDFLaTeX, that is. The package will instruct PDFTeX to load character maps into your PDF for output fonts encoded according to the T1 (Western European Languages), T2A, T2B, or T2C (Cyrillic Languages), or T5 (Vietnamese) encodings. If your document uses such encodings, viewers that can search will use the maps to interpret what they find in the file.

Unfortunately, the package only works with fonts that are directly encoded, such as the *cm-super* distribution. Fonts like Adobe Times Roman (which are encoded for (La)TeX use via virtual fonts) are not amenable to this treatment.

`cmap.sty`: `macros/latex/contrib/cmap`

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

M Graphics

148 Importing graphics into (La)TeX documents

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

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

Over the years, therefore, "graphics inclusion" packages have sprung up; most were designed for inclusion of Encapsulated PostScript graphics (Encapsulated PostScript graphics) — which has become the *lingua franca* of graphics inclusion over the last decade or so.

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

The obvious next step was to make the code configurable dynamically. The LaTeX standard *graphics* package and its derivatives made this step: it is strongly preferred for all current work.

Users of Plain TeX have two options allowing them to use *graphicx*: the *miniltx* "LaTeX emulator" and the *graphicx.tex* front-end allow you to load *graphicx*, and Eplain allows you to load it (using the full LaTeX syntax) direct.

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

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

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

The *epsfig* package provides the same facilities as *graphicx*, but via a `\psfig` command (also known as `\epsfig`), capable of emulating the behaviour (if not the bugs) the old *psfig* package. *Epsfig* also supplies homely support for former users of the *epsf* package. However, there's a support issue: if you declare you're using *epsfig*, any potential mailing list or usenet helper has to clear out of the equation the possibility that you're using "old" *epsfig*, so that support is slower coming than it would otherwise be.

There is no rational reason to stick with the old packages, which have never been entirely satisfactory in the LaTeX context. (One irrational reason to leave them behind is that their replacement's name tends not to imply that it's exclusively related to PostScript graphics. The reasoning also excludes *epsfig*, of course.)

A wide variety of detailed techniques and tricks have been developed over the years, and Keith Reckdahl's *epslatex* outlines them in compendious detail: this highly recommendable document is available from CTAN. An invaluable review of the practicalities of exchanging graphics between sites, "Graphics for Inclusion in Electronic Documents" has been written by Ian Hutchinson; the document isn't on CTAN, but may also be browsed on the Web.

epsf.tex: `macros/generic/epsf/epsf.tex`

epsfig.sty: Part of the `macros/latex/required/graphics` bundle

epslatex.pdf: `info/epslatex`

graphics.sty: `macros/latex/required/graphics`

graphicx.sty: Part of the `macros/latex/required/graphics` bundle

miniltx.tex: `macros/plain/graphics`

psfig.sty: `graphics/psfig`

149 Imported graphics in *dvips*

Dvips, as originally conceived, can only import a single graphics format: encapsulated PostScript (`.eps` files, encapsulated PostScript). *Dvips* also deals with the slightly eccentric EPS that is created by Metapost.

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

To include bitmap graphics, you need some means of converting them to PostScript; in fact many standard image manipulators (such as *ImageMagick*'s *convert*) make a good job of creating EPS files (but be sure to ask for output at PostScript level 2 or higher). (*Unix* users should beware of *xv*'s claims: it has a tendency to downsample your bitmap to your screen resolution.)

Special purpose applications *jpeg2ps* (which converts JPEG files using PostScript level 2 functionality), *bmeps* (which converts both JPEG and PNG files) and *a2ping*/*sam2p* (which convert a bewildering array of bitmap formats to EPS or PDF files; *sam2p* is one of the engines that *a2ping* uses) are also considered "good bets".

Bmeps comes with patches to produce your own version of *dvips* that can cope with JPEG and PNG direct, using *bmeps*'s conversion library. *Dvips*, as distributed by MiKTeX, comes with those patches built-in, but assuming that capability destroys portability, and is only recommendable if you are sure you will never want to share your document.

a2ping: `graphics/a2ping`

bmeps: Distributed as part of `support/dktools`

jpeg2ps: `support/jpeg2ps`

sam2p: graphics/sam2p

150 Imported graphics in PDFLaTeX

PDFTeX itself has a rather wide range of formats that it can "natively" incorporate into its output PDF stream: JPEG (`.jpg` files) for photographs and similar images, PNG files for artificial bitmap images, and PDF for vector drawings. Old versions of PDFTeX (prior to version 1.10a) supported TIFF (`.tif` files) format as an alternative to PNG files; don't rely on this facility, even if you *are* running an old enough version of PDFTeX...

In addition to the 'native' formats, the standard PDFLaTeX *graphics* package setup causes Hans Hagen's `supp-pdf` macros to be loaded: these macros are capable of translating the output of Metapost to PDF "on the fly"; thus Metapost output (`.mps` files) may also be included in PDFLaTeX documents.

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

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

The package *pst-pdf* permits other things than 'mere' graphics files in its argument. *Pst-pdf* operates (the authors suggest) "like BibTeX" — you process your file using PDFLaTeX, then use LaTeX, *dvips* and *ps2pdf* in succession, to produce a secondary file to input to your next PDFLaTeX run. (Scripts are provided to ease the production of the secondary file.)

A further extension is *auto-pst-pdf*, which generates PDF (essentially) transparently, by spawning a job to process output such as *pst-pdf* uses. If your PDFLaTeX installation doesn't automatically allow it — see spawning a process — then you need to start PDFLaTeX with:

```
pdflatex -shell-escape <file>
```

for complete 'automation'.

An alternative solution is to use *purifyeps*, a *Perl* script which uses the good offices of *pstoedit* and of Metapost to convert your Encapsulated PostScript to "Something that looks like the encapsulated PostScript that comes out of Metapost", and can therefore be included directly. Sadly, *purifyeps* doesn't work for all `.eps` files.

Good coverage of the problem is to be found in Herbert Voß's PDF support page, which is targeted at the use of *pstricks* in PDFLaTeX, and also covers the *pstricks*-specific package *pdftricks*.

auto-pst-pdf.sty: macros/latex/contrib/auto-pst-pdf

epstopdf: Browse support/epstopdf

epstopdf.sty: Distributed with Heiko Oberdiek's packages macros/latex/contrib/oberdiek

pdftricks.sty: macros/latex/contrib/pdftricks

pst-pdf.sty: macros/latex/contrib/pst-pdf

pstoedit: support/pstoedit

purifyeps: support/purifyeps

151 Imported graphics in *dvipdfm*

Dvipdfm (and *dvipdfmx*) translates direct from DVI to PDF (all other available routes produce PostScript output using *dvips* and then convert that to PDF with *ghostscript* or *Acrobat Distiller*).

Dvipdfm/*Dvipdfmx* are particularly flexible applications. They permit the inclusion of bitmap and PDF graphics, as does PDFTeX, but are also capable of employing *ghostscript* "on the fly" so as to be able to permit the inclusion of encapsulated PostScript (`.eps`) files by translating them to PDF. In this way, they combine the good qualities of *dvips* and of PDFTeX as a means of processing illustrated documents.

Unfortunately, "ordinary" LaTeX can't deduce the bounding box of a binary bitmap file (such as JPEG or PNG), so you have to specify the bounding box. This may be done explicitly, in the document:

```
\usepackage[dvipdfm]{graphicx}
...
\includegraphics[bb=0 0 540 405]{photo.jpg}
```

It's usually not obvious what values to give the "bb" key, but the program *ebb* will generate a file containing the information; the above numbers came from an *ebb* output file `photo.bb`:

```
%%Title: /home/gsm10/photo.jpg
%%Creator: ebb Version 0.5.2
%%BoundingBox: 0 0 540 405
%%CreationDate: Mon Mar  8 15:17:47 2004
```

If such a file is available, you may abbreviate the inclusion code, above, to read:

```
\usepackage[dvipdfm]{graphicx}
...
\includegraphics{photo}
```

which makes the operation feel as simple as does including `.eps` images in a LaTeX file for processing with *dvips*; the *graphicx* package knows to look for a `.bb` file if no bounding box is provided in the `\includegraphics` command.

The one place where usage isn't quite so simple is the need to quote *dvipdfm* explicitly, as an option when loading the *graphicx* package: if you are using *dvips*, you don't ordinarily need to specify the fact, since the default graphics configuration file (of most distributions) "guesses" the `dvips` option if you're using TeX.

dvipdfm: `dviware/dvipdfm`

dvipdfmx: `dviware/dvipdfmx`

ebb: Distributed as part of `dviware/dvipdfm`

152 "Modern" graphics file names

TeX was designed in a world where file names were very simple indeed, typically strictly limited both in character set and length. In modern systems, such restrictions have largely disappeared, which leaves TeX rather at odds with its environment. Particular problems arise with spaces in file names, but things like multiple period characters can seriously confuse the *graphics* package.

The specification of TeX leaves some leeway for distributions to adopt file access appropriate to their operating system, but this hasn't got us very far. Many modern distributions allow you to specify a file name as `"file name.tex"` (for example), which helps somewhat, but while this allows us to say

```
\input "foo bar.tex"
```

the analogous usage

```
\includegraphics{"gappy graphics.eps"}
```

using "ordinary" LaTeX causes confusion in *xdvi* and *dvips*, even though it works at compilation time. Sadly, even within such quotes, multiple dots give `\includegraphics` difficulties. Note that

```
\includegraphics{"gappy graphics.pdf"}
```

works in a similar version of PDFTeX.

If you're using the *graphics* package, the *grffile* package will help. The package offers several options, the simplest of which are `multidot` (allowing more than one dot in a file name) and `space` (allowing space in a file name). The `space` option requires that you're running on a sufficiently recent version of PDFTeX, in PDF mode — and even then it won't work for Metapost files, which are read as TeX input, and therefore use the standard input mechanism).

`grffile.sty`: Distributed as part of `macros/latex/contrib/oberdiek`

153 Importing graphics from "somewhere else"

By default, graphics commands like `\includegraphics` look "wherever TeX files are found" for the graphic file they're being asked to use. This can reduce your flexibility if you choose to hold your graphics files in a common directory, away from your (La)TeX sources.

The simplest solution is to patch TeX's path, by changing the default path. On most systems, the default path is taken from the environment variable `TEXINPUTS`, if it's present; you can adapt that to take in the path it already has, by setting the variable to

```
TEXINPUTS=.:<graphics path(s)>:
```

on a Unix system; on a Windows system the separator will be "`;`" rather than "`:`". The "`.`" is there to ensure that the current directory is searched first; the trailing "`:`" says "patch in the value of `TEXINPUTS` from your configuration file, here".

This method has the merit of efficiency ((La)TeX does *all* of the searches, which is quick), but it's always clumsy and may prove inconvenient to use in Windows setups (at least).

The alternative is to use the *graphics* package command `\graphicspath`; this command is of course also available to users of the *graphicx* and the *epsfig* packages. The syntax of `\graphicspath`'s one argument is slightly odd: it's a sequence of paths (typically relative paths), each of which is enclosed in braces. A slightly odd example (slightly modified from one given in the *graphics* bundle documentation) is:

```
\graphicspath{{eps/}{png/}}
```

which will search for graphics files in subdirectories `eps` and `png` of the directory in which LaTeX is running. (Note that the trailing "`/`" *is* required.)

(Note that some (La)TeX systems will only allow you to use files in the current directory and its sub-directories, for security reasons. However, `\graphicspath` imposes no such restriction: as far as *it* is concerned, you can access files anywhere.)

Be aware that `\graphicspath` does not affect the operations of graphics macros other than those from the graphics bundle — in particular, those of the outdated *epsf* and *psfig* packages are immune.

The slight disadvantage of the `\graphicspath` method is inefficiency. The package will call (La)TeX once for each entry in the list to look for a file, which of course slows things. Further, (La)TeX remembers the name of any file it's asked to look up, thus effectively losing memory, so that in the limit a document that uses a huge number of graphical inputs could be embarrassed by lack of memory. (Such "memory starvation" is pretty unlikely with any ordinary document in a reasonably modern (La)TeX system, but it should be borne in mind.)

If your document is split into a variety of directories, and each directory has its associated graphics, the *import* package may well be the thing for you; see the discussion of "bits of document in other directories" (bits of document in other directories).

`graphics` bundle: `macros/latex/required/graphics`

`import.sty`: `macros/latex/contrib/import`

154 Portable imported graphics

A regular need is a document to be distributed in more than one format: commonly both PostScript and PDF. The following advice is based on a post by one with much experience of dealing with the problem of dealing with EPS graphics in this case.

- Don't specify a driver when loading loading whichever version of the *graphics* package you use. The scheme relies on the distribution's ability to decide which driver is going to be used: the choice is between *dvips* and PDFTeX, in this case. Be sure to exclude options `dvips`, `pdftex` and `dvipdfm` (*dvipdfm* is not used in this scheme, but the aspirant PDF-maker may be using it for his output, before switching to the scheme).
- Use `\includegraphics[...]{filename}` without specifying the extension (i.e., neither `.eps` nor `.pdf`).
- For every `.eps` file you will be including, produce a `.pdf` version, as described in Graphics in PDFLaTeX. Having done this, you will have two copies of each graphic (a `.eps` and a `.pdf` file) in your directory.
- Use PDFLaTeX (rather than LaTeX–*dvips*–distillation or LaTeX–*dvipdfm*) to produce your PDF output.

Dvipdfm's charms are less than attractive here: the document itself needs to be altered from its default (*dvips*) state, before *dvipdfm* will process it.

155 Repeated graphics in a document

A logo or "watermark" image, or any other image that is repeated in your document, has the potential to make the processed version of the document unmanageably large. The problem is, that the default mechanisms of graphics usage add the image at every point it's to be used, and when processed, the image appears in the output file at each such point.

Huge PostScript files are embarrassing; explaining *why* such a file is huge, is more embarrassing still.

The `epslatex` graphics tutorial describes a technique for avoiding the problem: basically, one converts the image that's to be repeated into a PostScript subroutine, and load that as a *dvips* prologue file. In place of the image, you load a file (with the same bounding box as the image) containing no more than an invocation of the subroutine defined in the prologue.

The `epslatex` technique is tricky, but does the job. Trickier still is the neat scheme of converting the figure to a one-character Adobe Type 3 outline font. While this technique is for the "real experts" only (the author of this answer has never even tried it), it has potential for the same sort of space saving as the `epslatex` technique, with greater flexibility in actual use.

More practical is Hendri Adriaens' *graphicx-psmin*; you load this *in place* of *graphicx*, so rather than:

```
\usepackage[<options>]{graphicx}
```

you will write:

```
\usepackage[<options>]{graphicx-psmin}
```

and at the start of your document, you write:

```
\loadgraphics[<bb>]{<list of graphics>}
```

and each of the graphics in the list is converted to an "object" for use within the resulting PostScript output. (This is, in essence, an automated version of the `epslatex` technique described above.)

Having loaded the package as above, whenever you use `\includegraphics`, the command checks if the file you've asked for is one of the graphics in `\loadgraphics`' list. If so, the operation is converted into a call to the "object" rather than a new copy of the file; the resulting PostScript can of course be *much* smaller.

Note that the package requires a recent *dvips*, version 5.95b (this version isn't — yet — widely distributed).

If your PostScript is destined for conversion to PDF, either by a *ghostscript*-based mechanism such as *ps2pdf* or by (for example) *Acrobat Distiller*, the issue isn't so pressing, since the distillation mechanism will amalgamate graphics objects whether or not the PostScript has them amalgamated. PDFTeX does the same job with graphics, automatically converting multiple uses into references to graphics objects.

graphicx-psmin.sty: macros/latex/contrib/graphicx-psmin

156 Limit the width of imported graphics

Suppose you have graphics which may or may not be able to fit within the width of the page; if they will fit, you want to set them at their natural size, but otherwise you want to scale the whole picture so that it fits within the page width.

You do this by delving into the innards of the graphics package (which of course needs a little LaTeX internals programming):

```
\makeatletter
\def\maxwidth{%
  \ifdim\Gin@nat@width>\linewidth
    \linewidth
  \else
    \Gin@nat@width
  \fi
}
\makeatother
```

This defines a "variable" width which has the properties you want. Replace `\linewidth` if you have a different constraint on the width of the graphic.

Use the command as follows:

```
\includegraphics[width=\maxwidth]{figure}
```

157 Top-aligning imported graphics

When TeX sets a line of anything, it ensures that the base-line of each object in the line is at the same level as the base-line of the final object. (Apart, of course, from `\raisebox` commands. . .)

Most imported graphics have their base-line set at the bottom of the picture. When using packages such as *subfig*, one often wants to align figures by their tops. The following odd little bit of code does this:

```
\vtop{%
  \vskip0pt
  \hbox{%
    \includegraphics{figure}%
  }%
}
```

The `\vtop` primitive sets the base-line of the resulting object to that of the first "line" in it; the `\vskip` creates the illusion of an empty line, so `\vtop` makes the very top of the box into the base-line.

In cases where the graphics are to be aligned with text, there is a case for making the base-line one ex-height below the top of the box, as in:

```
\vtop{%
  \vskip-1ex
  \hbox{%
    \includegraphics{figure}%
  }%
}
```

A more LaTeX-y way of doing the job (somewhat inefficiently) uses the *calc* package:

```
\usepackage{calc}
...
\raisebox{1ex-\height}{\includegraphics{figure}}
```

(this has the same effect as the text-align version, above).

The fact is, *you* may choose where the base-line ends up. This answer merely shows you sensible choices you might make.

158 Displaying Metapost output 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 experience an error such as

```
Error: /undefined in cmmi10
```

There is provision in Metapost for avoiding this problem: issue the command `prologues := 2;` at the start of the `.mp` file.

Unfortunately, the PostScript that Metapost inserts in its output, following this command, is incompatible with ordinary use of the PostScript 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. This procedure may be automated using the *Perl* script *mps2eps*, thus saving a certain amount of tedium.

The Plain TeX user may use an adaptation of a jiffy of Knuth's, by Dan Luecking. Dan's version *mpsproof.tex* will work under TeX to produce a DVI file for use with *dvips*, or under PDFTeX to produce a PDF file, direct. The output is set up to look like a proof sheet.

A script application, *mptopdf*, is available in recent (La)TeX distributions: it seems fairly reliably to produce PDF from Metapost, so may reasonably be considered an answer to the question…

mps2eps: support/mps2eps

mpsproof.tex: Distributed as part of the Metapost distribution graphics/metapost

mptopdf: Part of graphics/metapost/contrib/tools/mptopdf

159 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 *eepic*, to sophisticated (but slow) drawing with PiCTeX. Depending on your type of drawing, and setup, here are a few systems you may consider:

- The `picture` environment provides rather primitive drawing capabilities (anything requiring more than linear calculations is excluded, unless a font can come to your help). The environment's tedious insistence on its own `\unitlength`, as the basic measurement in a diagram, may be avoided by use of the *picture* package, which detects whether a length is quoted as a number or as a length, and acts accordingly.
- *epic* was designed to make use of the LaTeX `picture` environment somewhat less agonising; *eepic* extends it, and is capable of using *tpic* `\special` commands to improve printing performance. (If the `\special`s aren't available, the *eepicemu* will do the business, far less efficiently.
- *pict2e*; this was advertised in the LaTeX manual, but didn't appear for nearly ten years after publication of the book! It removes all the petty restrictions that surround the use of the `picture` environment. It therefore suffers *only* from the rather eccentric drawing language of the environment, and is a far more useful tool than the original environment has ever been. (Note that *pict2e* supersedes David Carlisle's stop-gap *pspicture*.)
- PiCTeX is a venerable, and very powerful, system, that draws by placing dots on the page to give the effect of a line or curve. While this has the potential of great power, it is (of course) much slower than any of the other established packages. What's more, there are problems with its documentation.
- *PSTricks*; this gives you access to all the power of PostScript from TeX itself, by sophisticated use of `\special` commands. Since PostScript is itself a pretty powerful programming language, many astounding things can in principle be achieved using *PSTricks* (a wide range of contributed packages, ranging from world mapping to lens design diagrams, is available). *Pstricks*' `\special`s are by default specific to *dvips*, but VTeX (both in its commercial and in its free versions) understands them, and there is a *Pstricks* 'driver' that allow *Pstricks* to operate under XeTeX. PDFTeX users may use *pst-pdf*, which (like *epstopdf* — see PDFLaTeX graphics) generates PDF files using an auxiliary program, from *PSTricks* commands (*pst-pdf* also requires a recent installation of the *preview* package).
  There is a *PSTricks* mailing list (`pstricks@tug.org`) which you may join, or you may just browse the list archives.
- *pgf*: while *pstricks* is very powerful and convenient, using it with PDFLaTeX is an awful fidget: if you simply want the graphical capabilities, *pgf*, together with its rather pleasing "user-oriented" interface *tikz*, may be a good bet for you. While PDF has (in essence) the same graphical capabilities as PostScript, it isn't programmable; *pgf* provides common LaTeX commands that will utilise the graphical capabilities of both PostScript and PDF equally. The *pgf* manual is enormous, but a simple introduction which allows the user to get a feel for the capabilities of the system, is available at `http://cremeronline.com/LaTeX/minimaltikz.pdf`
- Metapost; you liked Metafont, but never got to grips with font files? Try Metapost — 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...
  Note that you can "embed" Metapost source in your document (i.e., keep it in-line with your LaTeX code).
- You liked Metafont (or Metapost), but find the language difficult? *Mfpic* makes up Metafont or Metapost code for you using familiar-looking (La)TeX macros. Not *quite* the full power of Metafont or Metapost, but a friendlier interface, and with Metapost output the results can be used equally well in either LaTeX or PDFLaTeX.
- You liked PiCTeX but don't have enough memory or time? Look at the late Eitan Gurari's *dratex*: it is just as powerful, but is an entirely new implementation which is not as hard on memory, is much more readable and is (admittedly sparsely) doc-

umented at http://www.cse.ohio-state.edu/~gurari/tpf/html/README.html, as well as in the author's book "TeX and LATeX: Drawing and Literate Programming", which remains available from on-line booksellers.

In addition, there are several means of generating code for your graphics application (*asymptote*, *gnuplot* and Metapost, at least) in-line in your document, and then have them processed in a command spawned from your (La)TeX run. For details, see question.

dratex.sty: graphics/dratex

epic.sty: macros/latex/contrib/epic

eepic.sty: macros/latex/contrib/eepic

eepicemu.sty: macros/latex/contrib/eepic

mfpic: graphics/mfpic

preview.sty: macros/latex/contrib/preview

pspicture.sty: macros/latex/contrib/pspicture

pst-pdf.sty: macros/latex/contrib/pst-pdf

pgf.sty: graphics/pgf

pict2e.sty: macros/latex/contrib/pict2e

pictex.sty: graphics/pictex

picture.sty: Distributed as part of macros/latex/contrib/oberdiek

pstricks: graphics/pstricks

tikz.sty: Distributed as part of graphics/pgf

160 In-line source for graphics applications

Some of the free-standing graphics applications may also be used (effectively) in-line in LaTeX documents; examples are

asymptote The package *asymptote* (provided in the *asymptote* distribution) defines an environment `asy` which arranges that its contents are available for processing, and will therefore be typeset (after "enough" runs, in the 'usual' LaTeX way).

Basically, you write

```
\begin{asy}
  ⟨asymptote code⟩
\end{asy}
```

and then execute

```
latex document
asy document-*.asy
latex document
```

egplot Allows you to incorporate *GNUplot* instructions in your document, for processing outside of LaTeX. The package provides commands that enable the user to do calculation in *GNUplot*, feeding the results into the diagram to be drawn.

gmp Allows you to include the source of MetaPost diagrams, with parameters of the diagram passed from the environment call.

emp An earlier package providing facilities similar to those of *gmp* (*gmp*'s author hopes that his package will support the facilities *emp*, which he believes is in need of update.)

mpgraphics Again, allows you to program parameters of Metapost diagrams from your LaTeX document, including the preamble details of the LaTeX code in any recursive call from Metapost.

In all cases (other than *asymptote*), these packages require that you can run external programs from within your document.

asymptote.sty: graphics/asymptote

egplot.sty: macros/latex/contrib/egplot

emp.sty: macros/latex/contrib/emp

gmp.sty: macros/latex/contrib/gmp

mpgraphics.sty: macros/latex/contrib/mpgraphics

161 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 for use with current LaTeX, and works in combination with Metafont (or, in its *feynmp* incarnation, with 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

feyn font bundle: fonts/feyn

feynman bundle: macros/latex209/contrib/feynman

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

162 Labelling graphics

"Technical" graphics (such as graphs and diagrams) are often labelled with quite complex mathematical expressions: there are few drawing or graphing tools that can do such things (the honourable exception being Metapost, which allows you to program the labels, in (La)TeX, in the middle of specifying your graphic).

Placing labels on graphics produced by all those *other* tools is what we discuss here.

The time-honoured *psfrag* package can help, if your image is included as an (encapsulated) PostScript file. Place an unique text in your graphic, using the normal text features of your tool, and you can ask *psfrag* to replace the text with arbitrary (La)TeX material. *Psfrag*'s "operative" command is `\psfrag{`*`Orig text`*`}{`*`Repl text`*`}`, which instructs the system to replace the original ("unique") text with the TeX-typeset replacement text. Optional arguments permit adjustment of position, scale and rotation; full details may be found in `pfgguide` in the distribution.

Since *psfrag* works in terms of (encapsulated) PostScript files, it needs extra work for use with PDFLaTeX. The *pst-pdf* package can support such usage. In fact, the *pst-pdf* support package *auto-pst-pdf* offers a configuration setting precisely for use with *psfrag*.

On the other hand, VTeX's GeX processor explicitly deals with *psfrag*, both in its free and commercial instances.

The *psfragx* package goes one step further than *psfrag*: it provides a means whereby you can put the *psfrag* commands into the preamble of your EPS file itself. *Psfrag* has such a command itself, but deprecates it; *psfragx* has cleaned up the facility, and provides a script *laprint* for use with *Matlab* to produce appropriately tagged output. (In principle, other graphics applications could provide a similar facility, but apparently none does.)

Emacs users may find the embedded editor *iTe* a useful tool for placing labels: it's a (La)TeX-oriented graphical editor written in *Emacs Lisp*. You create `iteblock` environments containing graphics and text, and may then invoke *iTe* to arrange the elements relative to one another.

Another useful approach is *overpic*, which overlays a `picture` environment on a graphic included by use of `\includegraphics`. This treatment lends itself to ready placement of texts and the like on top of a graphic. The package can draw a grid for planning your "attack"; the distribution comes with simple examples.

The *lpic* package is somewhat similar to *overpic*; it defines an environment `lpic` (which places your graphic for you): within the environment you may use the command `\lbl` to position LaTeX material at appropriate places over the graphic.

Pinlabel is another package whose author thought in the same sort of way as that of *overpic*; the documentation explains in detail how to plan your 'labelling attack' — in this case by loading your figure into a viewer and taking measurements from it. (The package discusses direct use of *ghostscript* as well as customised viewers such as *gsview* or *gv*.)

Pstricks can of course do everything that *overpic*, *lpic* or *pinlabel* can, with all the flexibility of PostScript programming that it offers. This capability is exemplified by the *pst-layout* package, which seems to be a superset of both *overpic* and *lpic*.

Similarly, *pgf/TikZ* has all the power needed, but no explicit package has been released.

The *pstricks* web site has a page with several examples of labelling which will get you started; if *pstricks* is an option for you, this route is worth a try.

The confident user may, of course, do the whole job in a picture environment which itself includes the graphic. I would recommend *overpic* or the *pstricks* approach, but such things are plainly little more than a convenience over what is achievable with the do-it-yourself approach.

auto-pst-pdf.sty: `macros/latex/contrib/auto-pst-pdf`

ghostscript: `support/ghostscript/GPL`

gsview: `support/ghostscript/ghostgum`

gv: `support/gv`

iTe: `support/ite`

laprint: Distributed with `macros/latex/contrib/psfragx`

lpic.sty: `macros/latex/contrib/lpic`

overpic.sty: `macros/latex/contrib/overpic`

pinlabel.sty: `macros/latex/contrib/pinlabel`

pgf.sty: `graphics/pgf`

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

psfragx.sty: `macros/latex/contrib/psfragx`

pstricks.sty: `graphics/pstricks`

pst-layout.sty: `graphics/pstricks/contrib/pst-layout`

pst-pdf.sty: `macros/latex/contrib/pst-pdf`

N Bibliographies and citations

N.1 Creating bibliographies

163 Creating a BibTeX bibliography file

A BibTeX bibliography file may reasonably be compared to a small database, the entries in which are references to literature that may be called up by citations in a document.

Each entry in the bibliography has a *type* and a unique *key*. The bibliography is read, by BibTeX, using the details specified in a *bibliography style*. From the style, BibTeX finds what entry types are permissible, what *fields* each entry type has, and how to format the whole entry.

The type specifies the type of document you're making reference to; it may run all the way from things like "Book" and "Proceedings" (which may even contain other citations of type "InBook" or "InProceedings") through dissertation styles like "PhdThesis" to otherwise-uncategorisable things such as "Misc". The unique key is something you choose yourself: it's what you use when you want to cite an entry in the file. People commonly create a key that combines the (primary) author's name and the year of publication, possibly with a marker to distinguish publications in the same year. So, for example, the Dyson, Eddington, Davidson paper about deflection of starlight appears in my experimental `.bib` file as `Dyson20.1`.

So, noting the rules of the style, you have 'simply' to write a bibliography database. Fortunately, there are several tools to help in this endeavour:

- Most of the better (La)TeX-oriented editors have "BibTeX modes".
- If you have an existing `thebibliography` environment, the *Perl* script *tex2bib* will probably help.
- There are a number of BibTeX bibliography management systems available, some of which permit a graphical user interface to the task. Sadly, none seems to be available with the ordinary TeX distributions.
  Tools such as *Xbibfile* (a graphical user interface), *ebib* (a database application written to run 'inside' *emacs*) and *btOOL* (a set of *perl* tools for building BibTeX database handlers) are available from CTAN.
  Other systems, such as *RefDB*, BibORB, *BibDesk*, *pybliographer* and the *Java*-based *Bibkeeper* and *JabRef* (which claims to supersede *Bibkeeper*) are only available from their development sites.
- Some commercial citation-management systems will export in BibTeX format; an example is EndNote.
- Data from on-line citation databases may often be translated to BibTeX format by utilities to be found on CTAN. For example, the *Perl* script *isi2bibtex* will translate citations from ISI "Web of knowledge" (a subscription service, available to UK academics via BIDS). UK academics may translate BIDS downloads using *bids.to.bibtex*
- Google Scholar provides an "Import into BibTeX" tab for each reference it finds for you: that tab gives you a page containing a BibTeX entry for the reference.

bids.to.bibtex: `biblio/bibtex/utils/bids/bids.to.bibtex`

btOOL: `biblio/bibtex/utils/btOOL`

ebib: `biblio/bibtex/utils/ebib`

isi2bibtex: `biblio/bibtex/utils/isi2bibtex`

tex2bib: `biblio/bibtex/utils/tex2bib/tex2bib`

tex2bib.readme: `biblio/bibtex/utils/tex2bib/README`

xbibfile: `biblio/bibtex/utils/xbibfile`

164 Creating a bibliography style

It *is* possible to write your own: the standard bibliography styles are distributed in a form with many comments, and there is a description of the language in the BibTeX distribution (see BibTeX documentation). 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 offer entirely new styles of document, but the system's "master BibTeX styles" already offer significantly more than the BibTeX standard set.

An alternative, which is increassingly often recommended, to use *biblatex*. *Biblatex* offers many hooks for adjusting the format of the output of your 'basic' BibTeX style, and a collection of 'contributed' styles have also started to appear.

biblatex.sty: macros/latex/contrib/biblatex

biblatex contributed styles: macros/latex/contrib/biblatex-contrib

BibTeX documentation: biblio/bibtex/base

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

165 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, chemical formulae, etc.).

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},
```

If your document design specification requires a different style of capitalisation, you should acquire a bibliography style that doesn't enforce BibTeX's default rules. It is definitely *not* a good idea to enclose an entire title in braces, as in

```
title = {{TeXniques and tips}},
```

though that does ensure that the capitalisation is not changed. Your BibTeX database should be a general-purpose thing, not something tuned to the requirements of a particular document or bibliography style, or to the way you are thinking today — for example, on a future occasion, you might find yourself using a different BibTeX style with different capitalisation rules.

There's more on the subject in the BibTeX documentation.

166 Accents in bibliographies

BibTeX not only has a tendency (by default) to mess about with the case of letters in your bibliography, also makes a hash of accent commands: "`ma\~nana`" comes out as "ma nana" (!). The solution is similar that of the letter case problem: enclose the troublesome sequence in braces, as "`{\~n}`", in this example.

167 '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/utils/bibtools/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$
  ```

  Finally,
- delete the function's 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.

168 BibTeX doesn't understand lists of names

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

Names should be expressed in one of the forms

```
First Last
Last, First
Last, Suffix, First
```

and lists of names should be separated with "`and`". For example:

```
AUTHOR = {Fred Q. Bloggs, John P. Doe \&
          Another Idiot}
```

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 \& Another Idiot Fred Q. Bloggs
```

because "John P. Doe & Another Idiot 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
          Another Idiot}
```

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}
```

169 URLs in BibTeX bibliographies

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

The actual information that need be available in a citation of an URL is discussed at some length in the publicly available on-line extracts of ISO 690–2; the techniques below do *not* satisfy all the requirements of ISO 690–2, but they offer a solution that is at least available to users of today's tools.

Until the new version of BibTeX arrives, the simplest technique is to use the `howpublished` field of the standard styles' `@misc` function. Of course, the strictures about typesetting URLs still apply, so the entry will look like:

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

A possible alternative approach is to use BibTeX styles other than the standard ones, that already have URL entry types. Candidates are:

- The *natbib* styles (*plainnat*, *unsrtnat* and *abbrevnat*), which are extensions of the standard styles, principally for use with *natbib* itself. However, they've acquired URLs and other "modern" entries along the way. The same author's *custom-bib* is also capable of generating styles that honour URL entries.
- The *babelbib* bundle, which offers multilingual bibliographies, similarly provides a set of standard-style equivalents that have URL entries.
- More modern styles such as the *harvard* package (if the 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).

You can also acquire new BibTeX styles by use of Norman Gray's *urlbst* system, which is based on a *Perl* script that edits an existing BibTeX style file to produce a new style. The new style thus generated has a `webpage` entry type, and also offers support for `url` and `lastchecked` fields in the other entry types. The *Perl* script comes with a set of converted versions of the standard bibliography styles.

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: 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. Any current version of the *url* or *hyperref* package detects this "`%`–end-of-line" structure in its argument, and removes it.

`babelbib` bundle: `biblio/bibtex/contrib/babelbib`

`custom-bib` bundle: `macros/latex/contrib/custom-bib`

`harvard.sty`: `macros/latex/contrib/harvard`

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

`natbib` styles: `macros/latex/contrib/natbib`

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

`urlbst`: `biblio/bibtex/contrib/urlbst`

170 Using BibTeX with Plain TeX

The file `btxmac.tex` (which is part of the Eplain system) contains macros and documentation for using BibTeX with Plain TeX, either directly or with Eplain. See the use of BibTeX for more information about BibTeX itself.

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

`eplain` system: `macros/eplain`

171 Reconstructing .bib files

Perhaps you've lost the .bib file you generated your document from, or have been sent a document without one. Or even, you've realised the error of building a substantial document without the benefit of BibTeX...

The *Perl* script, *tex2bib* makes a reasonable job of regenerating .bib files from thebibliography environments, provided that the original (whether automatically or manually generated) doesn't deviate too far from the "standard" styles.

You are well-advised to check the output of the script. While it will not usually destroy information, it can quite reasonably mislabel it.

Documentation of the script is to be found in the file tex2bib.readme

tex2bib: biblio/bibtex/utils/tex2bib/tex2bib

tex2bib.readme: biblio/bibtex/utils/tex2bib/README

172 BibTeX sorting and name prefixes

BibTeX recognises a bewildering array of name prefixes (mostly those deriving from European language names); it ignores the prefixes when sorting the bibliography — you want "Ludwig van Beethoven" sorted under "Beethoven", not under "van". (Lamport made a witty deliberate mistake with Beethoven's name, in the first edition of his LaTeX manual.)

However, a recurring issue is the desire to quote Lord Rayleigh's publications ("Lord" isn't an acceptable prefix), or names from languages that weren't considered when BibTeX was designed such as "al-Wakil" (transcribed from the Arabic). What's needed is a separate "sort key", but BibTeX only allows such a thing in citations of items that have no author or editor.

The solution is to embed the sort key in the author's name, but to prevent it from being typeset. Patashnik recommends a command \noopsort (no-output-sortkey), which is defined and used as follows:

```
@PREAMBLE{ {\providecommand{\noopsort}[1]{}} }
...
@ARTICLE{Rayleigh1,
AUTHOR = "{\noopsort{Rayleigh}}{Lord Rayleigh}",
...
}
```

Note that this \noopsort applies to the last name in this kind of construct, so an author with an Arabic name might be rendered:

```
...
AUTHOR = "Ali {\noopsort{Hadiidii}}{al-Hadiidii}",
...
```

A further use might deal with word order games, as in the famous Vietnamese name:

```
...
AUTHOR = "\noopsort{Thanh Han The}{Han The Thanh}",
...
```

though that author seems well-acquainted with Western confusion about the significance of the parts of his name (even to the extent of missing out the accentuation, as above...).

173 'Multi-letter' initials in BibTeX

If your bibliographic style uses initials + surname, you may encounter a problem with some transcribed names (for example, Russian ones). Consider the following example from the real world:

```
@article{epifanov1997,
   author = {Epifanov, S. Yu. and Vigasin, A. A.},
   title  = ...
}
```

Note that the "Yu" is the initial, not a complete name. However, BibTeX's algorithms will leave you with a citation — slightly depending on the bibliographic style — that reads: "S. Y. Epifanov and A. A. Vigasin, . . . ". instead of the intended "S. Yu. Epifanov and A. A. Vigasin, . . . ".

One solution is to replace each affected initial by a command that prints the correct combination. To keep your bibliography portable, you need to add that command to your bibliography with the `@preamble` directive:

```
@preamble{ {\providecommand{\BIBYu}{Yu} } }

@article{epifanov1997,
   author   = {Epifanov, S. {\BIBYu}. and Vigasin, A. A.},
   title    = ...
}
```

If you have many such commands, you may want to put them in a separate file and `\input` that LaTeX file in a `@preamble` directive.

An alternative is to make the transcription look like an accent, from BibTeX's point of view. For this we need a control sequence that does nothing:

```
@article{epifanov1997,
   author   = {Epifanov, S. {\relax Yu}. and Vigasin, A. A.},
   title    = ...
}
```

Like the solution by generating extra commands, this involves tedious extra typing; which of the two techniques is preferable for a given bibliography will be determined by the names in it. It should be noted that a preamble that introduces lots of odd commands is usually undesirable if the bibliography is a shared one.

"Compound" initials (for single names made up of two or more words) may be treated in the same way, so one can enter Forster's rather complicated name as:

```
@article{forster2006,
  author   = {Forster, P.M. {\relax de F.} and Collins, M.},
  title    = ...
```

The same trick can be played if you're entering whole names:

```
...
  author   = {Epifanov, Sasha {\relax Yu}ri and
...
```

(though no guarantee, that either of those names is right, is offered!) However, if you're typing the names in the "natural" (Western) way, with given names first, the trick:

```
...
  author   = {P.M. {\relax de F.} Forster and
...
```

doesn't work — "de F. Forster" is treated as a compound family names.

N.2 Creating citations

174 "Normal" use of BibTeX from LaTeX

To create a bibliography for your document, you need to perform a sequence of steps, some of which seem a bit odd. If you choose to use BibTeX, the sequence is:

First: you need a BibTeX bibliography file (a `.bib` file) — see "creating a BibTeX file" ().

Second: you must write your LaTeX document to include a declaration of the 'style' of bibliography, citations, and a reference to the bibliography file mentioned in the step 1. So we may have a LaTeX file containing:

```
\bibliographystyle{plain}
...
Pooh is heroic~\cite{Milne:1926}.
...
Alice struggles~\cite{Carroll:1865}.
...
\bibliography{mybooks}
```

Note: we have bibliography style *plain*, above, which is nearly the simplest of the lot: a sample text, showing the sorts of style choices available, can be found on Ken Turner's web site: http://www.cs.stir.ac.uk/~kjt/software/latex/showbst.html

Third: you must process the file.

```
latex myfile
```

As LaTeX processes the file, the `\bibliographystyle` command writes a note of the style to the `.aux` file; each `\cite` command writes a note of the citation to the `.aux` file, and the `\bibliography` command writes a note of which `.bib` file is to be used, to the `.aux` file.

Note that, at this stage, LaTeX isn't "resolving" any of the citations: at every `\cite` command, LaTeX will warn you of the undefined citation, and when the document finishes, there will be a further warning of undefined references.

Fourth: you must run BibTeX:

```
bibtex myfile
```

Don't try to tell BibTeX anything but the file name: say `bibtex myfile.aux` (because you know it's going to read the `.aux` file) and BibTeX will blindly attempt to process `myfile.aux.aux`.

BibTeX will scan the `.aux` file; it will find which bibliography style it needs to use, and will "compile" that style; it will note the citations; it will find which bibliography files it needs, and will run through them matching citations to entries in the bibliography; and finally it will sort the entries that have been cited (if the bibliography style specifies that they should be sorted), and outputs the resulting details to a `.bbl` file.

Fifth: you run LaTeX again. It warns, again, that each citation is (still) undefined, but when it gets to the `\bibliography` command, it finds a `.bbl` file, and reads it. As it encounters each `\bibitem` command in the file, it notes a definition of the citation.

Sixth: you run LaTeX yet again. This time, it finds values for all the citations, in its `.aux` file. Other things being equal, you're done… until you change the file.

If, while editing, you change any of the citations, or add new ones, you need to go through the process above from steps 3 (first run of LaTeX) to 6, again, before the document is once again stable. These four mandatory runs of LaTeX make processing a document with a bibliography even more tiresome than the normal two runs required to resolve labels.

To summarise: processing to resolve citations requires: LaTeX; BibTeX; LaTeX; LaTeX.

175 Choosing a bibliography style

A large proportion of people are satisfied with one of Patashnik's original "standard" styles, *plain*, *unsrt*, *abbrv* and *alpha*. However, no style in that set supports the "author-date" citation style that is popular in many fields; but there are a very large number of contributed styles available, that *do* support the format.

(Note that author-date styles arose because the simple and clear citation style that *plain* produces is so awkward in a traditional manuscript preparation scenario. However, TeX-based document production does away with all those difficulties, leaving us free once again to use the simple option.)

Fortunately, help is at hand, on the Web, with this problem:

- a sample text, showing the sorts of style choices available, can be found on Ken Turner's web site;

- an excellent survey, that lists a huge variety of styles, sorted into their nominal topics as well as providing a good range of examples, is the Reed College "Choosing a BibTeX style".

Of course, these pages don't cover everything; the problem the inquisitive user faces, in fact, is to find what the various available styles actually do. This is best achieved (if the links above don't help) by using *xampl.bib* from the BibTeX documentation distribution: one can get a pretty good feel for any style one has to hand using this "standard" bibliography. For style *my-style.bst*, the simple LaTeX document:

```
\documentclass{article}
\begin{document}
\bibliographystyle{my-style}
\nocite{*}
\bibliography{xampl}
\end{document}
```

will produce a representative sample of the citations the style will produce. (Because *xampl.bib* is so extreme in some of its "examples", the BibTeX run will also give you an interesting selection of BibTeX's error messages. . .)

`xampl.bib`: `biblio/bibtex/base/xampl.bib`

176 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 `\include`d 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`: distributed as part of `macros/latex/contrib/cite`

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

177 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 bibliographies per chapter.

For more than one bibliography, there are three options.

The *multibbl* package offers a very simple interface: you use a command `\newbibliography` to define a bibliography "tag". The package redefines the other bibliography commands so that each time you use any one of them, you give it the tag for the bibliography where you want the citations to appear. 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` — `\refname` and `\bibname` — in a *babel*-safe way). So one might write:

```
\usepackage{multibbl}
\newbibliography{bk}
\bibliographystyle{bk}{alpha}
\newbibliography{art}
\bibliographystyle{art}{plain}
...
\cite[pp.~23--25]{bk}{milne:pooh-corner}
...
\cite{art}{einstein:1905}
...
\bibliography{bk}{book-bib}{References to books}
\bibliography{art}{art-bib}{References to articles}
```

(Note that the optional argument of `\cite` appears *before* the new tag argument, and that the `\bibliography` commands may list more than one `.bib` file — indeed all `\bibliography` commands may list the same set of files.)

The `\bibliography` data goes into files whose names are ⟨*tag-name*⟩*.aux*, so you will need to run

```
bibtex bk
bibtex art
```

after the first run of LaTeX, to get the citations in the correct place.

The *multibib* package allows you to define a series of “additional topics”, each of which comes with its own series of bibliography commands. So one might write:

```
\usepackage{multibib}
\newcites{bk,art}%
         {References from books,%
          References from articles}
\bibliographystylebk{alpha}
\bibliographystyleart{plain}
...
\citebk[pp.~23--25]{milne:pooh-corner}
...
\citeart{einstein:1905}
...
\bibliographybk{book-bib}
\bibliographyart{art-bib}
```

Again, as for *multibbl*, any `\bibliography...` command may scan any list of `.bib` files.

BibTeX processing with *multibib* is much like that with *multibbl*; with the above example, one needs:

```
bibtex bk
bibtex art
```

Note that, unlike *multibbl*, *multibib* allows a simple, unmodified bibliography (as well as the “topic” ones).

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 a file `diss.tex` containing:

```
\usepackage{bibtopic}
\bibliographystyle{alpha}
...
\cite[pp.~23--25]{milne:pooh-corner}
...
\cite{einstein:1905}
...
\begin{btSect}{book-bib}
\section{References from books}
\btPrintCited
\end{btSect}
\begin{btSect}[plain]{art-bib}
\section{References from articles}
\btPrintCited
\end{btSect}
```

Note the different way of specifying a bibliographystyle: if you want a different style for a particular bibliography, you may give it as an optional argument to the `btSect` environment.

Processing with BibTeX, in this case, uses `.aux` files whose names are derived from the name of the base document. So in this example you need to say:

```
bibtex diss1
bibtex diss2
```

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{*}`).

However, the *real* difference from *multibbl* and *multibib* is that selection of what appears in each bibliography section is determined in *bibtopic* by what's in the `.bib` files.

An entirely different approach is taken by the *splitbib* package. You provide a `category` environment, in the preamble of your document, for each category you want a separate citation list for. In each environment, you list the `\cite` keys that you want listed in each category. The `\bibliography` command (or, more precisely, the `thebibliography` environment it uses) will sort the keys as requested. (Keys not mentioned in a `category` appear in a "misc" category created in the sorting process.) A code example appears in the package documentation (a PDF file in the CTAN directory, see the file list, below).

bibtopic.sty: macros/latex/contrib/bibtopic

multibbl.sty: macros/latex/contrib/multibbl

multibib.sty: macros/latex/contrib/multibib

splitbib.sty: macros/latex/contrib/splitbib

178 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`. *Inlinebib* was actually designed for footnote citations: its *expected* use is that you place a citation inline as the argument of a `\footnote` command.
- The package *jurabib*, which was originally designed for 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
- Packages *jurabib* and *inlinebib*, again.

Note that *jurabib* does the job using LaTeX's standard footnotes, whereas *footbib* creates its own sequence of footnotes. Therefore, in a document which has other footnotes, it may be advisable to use *jurabib* (or of course *inlinebib*), to avoid confusion of footnotes and foot-citations.

The *usebib* package offers a 'toolbox', which allows the user to place exactly what is needed, in the text (that is, rather than a full citation). The package's command, that does the actual typesetting, is `\usebibdata{⟨key⟩}{⟨field⟩}`; it typesets the *field* item from the entry *key* in the bibliography; the user then formats the entry as desired — obviously one could construct one's own bibliography, altogether, from this command, but it would quickly become tedious.

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

footbib.sty: macros/latex/contrib/footbib *inlinebib.sty*: biblio/bibtex/contrib/inlinebib

jurabib.sty: macros/latex/contrib/jurabib

usebib.sty: macros/latex/contrib/usebib

179 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`).

The package *biblatex* has a built-in style *numeric-comp* for its bibliographies.

biblatex.sty: macros/latex/contrib/biblatex

cite.sty: macros/latex/contrib/cite

hypernat.sty: macros/latex/contrib/hypernat

hyperref.sty: macros/latex/contrib/hyperref

plainnat.bst: Distributed with macros/latex/contrib/natbib

unsrtnat.bst: Distributed with macros/latex/contrib/natbib

180 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* and *mciteplus* packages deal with the problem.

mcite overloads the `\cite` command to recognise a "*" at the start of a key, 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're not limited to collapsing just two references. You can mix "collapsed" references with "ordinary" ones, as in

```
\cite{paper0,paper1,*paper2,paper3}
```

Which will appear in the document as 3 citations "[4,7,11]" (say) — citation '4' will refer to paper 0, '7' will refer to a combined entry for paper 1 and paper 2, and '11' will refer to paper 3.

You need to make a small change to the bibliography style (`.bst`) file you use; the *mcite* package documentation tells you how to do that.

Most recent versions of *REVTeX* (version 4.1 and later), in conjunction with recent versions of *natbib*, already contain support for combined citations and so no longer even need *mciteplus* (but *mciteplus* is more general and will work with many other class and package combinations).

The *mciteplus* package adresses many of the infelicites of *mcite*. Again, 'ordinary' `.bst` files will not work with *mciteplus*, but the package documentation explains how to patch an existing BibTeX style.

The *collref* package takes a rather different approach to the problem, and will work with most (if not all) BibTeX packages. *Collref* spots common subsets of the references, so if it sees a sequence

```
\cite{paper0,paper1,paper2,paper3}
...
\cite{some_other_paper,paper1,paper2,and_another}
```

it will collect `paper1` and `paper2` as a multiple reference.

collref.sty: macros/latex/contrib/collref

mcite.sty: macros/latex/contrib/mcite

mciteplus.sty: macros/latex/contrib/mciteplus

natbib.sty: macros/latex/contrib/natbib

revtex 4.1: macros/latex/contrib/revtex

181 References from the bibliography to the citation

A link (or at least a page reference), from the bibliography to the citing command, is often useful in large documents.

Two packages support this requirement, *backref* and *citeref*. *Backref* is part of the *hyperref* bundle, and supports hyperlinks back to the citing command.

Citeref is the older, and seems to rely on rather simpler (and therefore possibly more stable) code; it produces a list of page references, only. It doesn't interact well with other citation packages (for example, *cite*), which probably reflects its antiquity (it's derived from a LaTeX 2.09 package).

Neither collapses lists of pages ("`5, 6, 7`" comes out as such, rather than as "`5-7`"), but neither package repeats the reference to a page that holds multiple citations. (The failure to collapse lists is of course forgiveable in the case of the *hyperref*-related *backref*, since the concept of multiple hyperlinks from the same anchor is less than appealing.)

backref.sty: Distributed with macros/latex/contrib/hyperref

citeref.sty: macros/latex/contrib/citeref

182 Sorting lists of citations

BibTeX has a sorting function, and most BibTeX styles sort the citation list they produce; most people find this desirable.

However, it is perfectly possible to write a `thebibliography` environment that *looks* as if it came from BibTeX, and many people do so (in order to save time in the short term).

The problem arises when `thebibliography`-writers decide their citations need to be sorted. A common misapprehension is to insert `\bibliographystyle{alpha}` (or similar) and expect the typeset output to be sorted in some magical way. BibTeX doesn't work that way! — if you write `thebibliography`, you get to sort its contents. BibTeX will only sort the contents of a `thebibliography` environment when it creates it, to be inserted from a `.bbl` file by a `\bibliography` command.

183 Reducing spacing in the bibliography

Bibliographies are, in fact, implemented as lists, so all the confusion about reducing list item spacing also applies to bibliographies.

If the *natbib* package 'works' for you (it may not if you are using some special-purpose bibliography style), the solution is relatively simple — add

```
\usepackage{natbib}
\setlength{\bibsep}{0.0pt}
```

to the preamble of your document.

The *compactbib* package has a similar effect. Its primary purpose is to produce two bibliographies, and it seems to preclude use of BibTeX (though the package documentation, in the package file itself, isn't particularly clear).

Otherwise, one is into unseemly hacking of something or other. The *mdwlist* package actually does the job, but it doesn't work here, because it makes a different-named list, while the name "`thebibliography`" is built into LaTeX and BibTeX. Therefore, we need to patch the underlying macro:

```
\let\oldbibliography\thebibliography
\renewcommand{\thebibliography}[1]{%
  \oldbibliography{#1}%
```

```
    \setlength{\itemsep}{0pt}%
  }
```

The *savetrees* package performs such a patch, among a plethora of space-saving measures: you can, in principle, suppress all its other actions, and have it provide you a compressed bibliography *only*.

compactbib.sty: macros/latex/contrib/compactbib/compactbib.sty

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

natbib.sty: macros/latex/contrib/natbib

savetrees.sty: macros/latex/contrib/savetrees

184 Table of contents rearranges "*unsrt*" ordering

If you're using the *unsrt* bibliography style, you're expecting that your bibliography will *not* be sorted, but that the entries will appear in the order that they first appeared in your document.

However, if you're unfortunate enough to need a citation in a section title, and you also have a table of contents, the citations that now appear in the table of contents will upset the "natural" ordering produced by the *unsrt* style. Similarly, if you have citations in captions, and have a list of figures (or tables).

There's a pretty simple "manual" method for dealing with the problem — when you have the document stable:

1. Delete the `.aux` file, and any of `.toc`, `.lof`, `.lot` files.
2. Run LaTeX.
3. Run BibTeX for the last time.
4. Run LaTeX often enough that things are stable again.

Which is indeed simple, but it's going to get tedious when you've found errors in your "stable" version, often enough.

The package *notoccite* avoids the kerfuffle, and suppresses citations while in the table of contents, or lists of figures, tables (or other floating things: the code is quite general).

notoccite.sty: macros/latex/contrib/notoccite

185 Non-english bibliographies

Like so much of early (La)TeX software, BibTeX's assumptions were firmly rooted in what its author knew well, viz., academic papers in English (particularly those with a mathematical bent). BibTeX's standard styles all address exactly that problem, leaving the user who writes in another language (or who deal with citations in the style of other disciplines than maths) to strike out into contributed software.

For the user whose language is not English, there are several alternatives. Possibly most straightforward is to switch to using *biblatex*, which can produce a bibliography appropriate to several languages. However, *biblatex* is large and has correspondingly large documentation (though it is well-written and pleasingly typeset), so its adoption takes time.

Otherwise, the simplest procedure is to provide translations of BibTeX styles into the required language: the solitary *finplain.bst* does that for Finnish; others one can find are for Danish (*dk-bib*), French (*bib-fr*), German (*bibgerm*), Norwegian (*norbib*) and Swedish (*swebib*) bundles (of which the *bib-fr* set is the most extensive). The *spain* style implements a traditional Spanish citation style.

These static approaches solve the problem, for the languages that have been covered by them. Unfortunately, with such an approach, a lot of work is needed for every language involved. Two routes to a solution of the "general" problem are available — that offered by *babelbib*, and the *custom-bib* mechanism for generating styles.

Babelbib (which is a development of the ideas of the *bibgerm* package) co-operates with *babel* to control the language of presentation of citations (potentially at the level of individual items). The package has a built-in set of languages it 'knows about', but the documentation includes instructions on defining commands for other languages. *Babelbib* comes with its own set of bibliography styles, which could be a restriction if there wasn't also a link from *custom-bib*.

The *makebst* menu of *custom-bib* allows you to choose a language for the BibTeX style you're generating (there are 14 languages to choose; it looks as if *spain.bst*, mentioned above, was generated this way). If, however, you opt not to specify a language, you are asked whether you want the style to interact with *babelbib*; if you do so, you're getting the best of both worlds — formatting freedom from *custom-bib* and linguistic freedom via the extensibility of *babelbib*

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

bib-fr bundle: `biblio/bibtex/contrib/bib-fr`

bibgerm bundle: `biblio/bibtex/contrib/germbib`

biblatex.sty: `macros/latex/contrib/biblatex`

custom-bib bundle: `macros/latex/contrib/custom-bib`

finplain.bst: `biblio/bibtex/contrib/misc/finplain.bst`

norbib bundle: `biblio/bibtex/contrib/norbib`

spain: `biblio/bibtex/contrib/spain`

swebib bundle: `biblio/bibtex/contrib/swebib`

186 Format of numbers in the bibliography

By default, LaTeX makes entries in the bibliography look like:

> [1] Doe, Joe et al. Some journal. 2004.
> [2] Doe, Jane et al. Some journal. 2003.

while many documents need something like:

> 1. Doe, Joe et al. Some journal. 2004.
> 2. Doe, Jane et al. Some journal. 2003.

This sort of change may be achieved by many of the "general" citation packages; for example, in *natbib*, it's as simple as:

```
\renewcommand{\bibnumfmt}[1]{#1.}
```

but if you're not using such a package, the following internal LaTeX commands, in the preamble of your document, will do the job:

```
\makeatletter
\renewcommand*{\@biblabel}[1]{\hfill#1.}
\makeatother
```

natbib.sty: `macros/latex/contrib/natbib`

N.3 Manipulating whole bibliographies

187 Listing all your 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.

A couple of packages are available, that aim to reduce the impact of `\nocite{*}` of a large citation database. *Biblist* was written for use under LaTeX 2.09, but seems to work well enough; *listbib* is more modern. Both provide their own `.bst` files. (The impact of large databases was significant in the old days of LaTeX systems with very little free memory; this problem is less significant now than it once was.)

biblist.sty: `macros/latex209/contrib/biblist`

listbib.sty: `macros/latex/contrib/listbib`

188 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 thorough solution is offered by *bib2xhtml*; using it, you make use of one of its modified versions of many common BibTeX styles, and post-process the output so produced using a *perl* script.

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/misc/bbl2html.awk`

bib2xhtml: `biblio/bibtex/utils/bib2xhtml`

noTeX.bst: `biblio/bibtex/utils/misc/noTeX.bst`

O Adjusting the typesetting

O.1 Alternative document classes

189 Replacing the standard classes

People are forever concocting classes that replace the standard ones: the present author produced an *ukart* class that used the *sober* package, and a few British-specific things (such as appear in the *babel* package's British-english specialisation) in the 1980s, which is still occasionally used.

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

The *KOMA-script* bundle (classes named *scr...*) are a strong current contender. They are actively supported and are subject to sensitive development; they are comprehensive in their coverage of significant typesetting issues; they produce good-looking output and they are well documented in both English (*scrguien* in the distribution) and German (*scrguide* in the distribution).

The other comparable class is *memoir*. This aims to replace *book* and *report* classes directly, and (like *KOMA-script*) is comprehensive in its coverage of small issues. *Memoir*'s documentation (*memman*) is very highly spoken of, and its lengthy introductory section is regularly recommended as a tutorial on typesetting.

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

memoir.cls: `macros/latex/contrib/memoir`

NTGclass bundle: `macros/latex/contrib/ntgclass`

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

Lamport's original LaTeX had a separate program (SliTeX) for producing slides; it dates from the age when colour effects were produced by printing separate slides in different-coloured inks, and overlaying them, and was just about acceptable back then. When LaTeX2e came along, the reason SliTeX had to be a separate program went away, and its functionality was supplied by the *slides* class. While this makes life a little easier for system administrators, it does nothing for the inferior functionality of the class: no-one who "knows" uses *slides* nowadays.

The 'classic' alternatives have been *seminar* and *foils* (originally known as FoilTeX). Both were originally designed to produce output on acetate foils, though subsequent work has provided environments in which they can be used with screen projectors (see below).

The advent of Microsoft *PowerPoint* (feeble though early versions of it were) has created a demand for "dynamic" slides — images that develop their content in a more elaborate fashion than by merely replacing one foil with the next in the way that was the norm when *slides*, *foils* and *seminar* were designed.

The *prosper* class builds on *seminar* to provide dynamic effects and the like; it retains the ability to provide PDF for a projected presentation, or to print foils for a foil-based presentation. The add-on package *ppr-prv* adds "preview" facilities (that which is commonly called "hand-out printing"). The *HA-prosper* package, which you load with *prosper*, mends a few bugs, and adds several facilities and slide design styles. The (more recent) *powerdot* class is designed as a replacement for *prosper* and *HA-prosper*, co-authored by the author of *HA-prosper*.

Beamer is a relatively easy-to-learn, yet powerful, class that (as its name implies) was designed for use with projection displays. It needs the *pgf* package (for graphics support), which in turn requires *xcolor*; while this adds to the tedium of installing *beamer* "from scratch", both are good additions to a modern LaTeX installation. *Beamer* has reasonable facilities for producing printed copies of slides.

Talk is another highly functional, yet easy-to-learn class which claims to differ from the systems mentioned above, such as *beamer*, in that it doesn't impose a slide style on you. You get to specify a bunch of slide styles, and you can switch from one to the other between slides, as you need. (The class itself provides just the one style, in the package *greybars*: the author hopes users will contribute their own styles, based on *greybars*.)

Lecturer is a *generic* solution (it works with Plain TeX, LaTeX and ConTeXt mk ii, but not — yet — with ConTeXt mk iv). By separating the functionality needed for a presentation (using TeX for typesetting, and PDF functions for layering and dynamic effects) a clear structure emerges. While it doesn't have the range of "themes" (presentation styles) of *beamer* it seems a useful alternative candidate.

Present is designed for use with Plain TeX only; its design is simple, to the extent that its author hopes that users will themselves be able to tune its macros.

Ppower4 (commonly known as *pp4*) is a *Java*-based support program that will postprocess PDF, to 'animate' the file at places you've marked with commands from one of the *pp4* packages. The commands don't work on PDF that has come from *dvips* output; they work with PDF generated by PDFLaTeX, VTeX LaTeX, or *dvipdfm* running on LaTeX output.

Pdfscreen and *texpower* are add-on packages that permit dynamic effects in documents formatted in "more modest" classes; *pdfscreen* will even allow you to plug "presentation effects" into an *article*-class document.

A more detailed examination of the alternatives (including examples of code using many of them) may be found at Michael Wiedmann's fine `http://www.miwie.org/presentations/presentations.html`

ConTeXt users will find that much (if not all) of what they need is already in ConTeXt itself; there's a useful summary of what's available, with examples, in `http://wiki.contextgarden.net/Presentation_Styles`

`beamer.cls`: Download all of `macros/latex/contrib/beamer`

`foils.cls`: `macros/latex/contrib/foiltex`

greybars.sty: distributed with macros/latex/contrib/talk

HA-prosper.sty: macros/latex/contrib/ha-prosper

lecturer.sty: macros/generic/lecturer

seminar.cls: macros/latex/contrib/seminar

pdfscreen.sty: macros/latex/contrib/pdfscreen

pgf.sty: graphics/pgf

powerdot.cls: macros/latex/contrib/powerdot

pp4: support/ppower4

ppr-prv.sty: macros/latex/contrib/ppr-prv

present.tex: macros/plain/contrib/present

prosper.cls: macros/latex/contrib/prosper

talk.cls: macros/latex/contrib/talk

texpower: macros/latex/contrib/texpower

xcolor.sty: macros/latex/contrib/xcolor

191 Creating posters with LaTeX

There is no complete "canned solution" to creating a poster (as, for example, classes like *seminar*, *powerdot* and *beamer* serve for creating presentations in a variety of styles).

The nearest approach to the complete solution is the *sciposter* class, which provides the means to produce really rather good posters according to the author's required style. A complete worked example is provided with the distribution

Otherwise, there is a range of tools, most of which are based on the *a0poster* class, which sets up an appropriately-sized piece of paper, sets font sizes appropriately, and leaves you to your own devices.

Having used *a0poster*, you can of course slog it out, and write all your poster as an unadorned LaTeX document (presumably in multiple columns, using the *multicol* package), but it's not really necessary: the (straightforward) *textpos* package provides a simple way of positioning chunks of text, or tables or figures, on the poster page.

More sophisticated is the *flowfram* package, whose basic aim in life is flowing text from one box on the page to the next. One of the package's design aims seems to have been the production of posters, and a worked example is provided. The author of *flowfram* has an experimental tool called JpgfDraw, which allows you to construct the outline of frames for use with *flowfram*.

The *beamerposter* package is added to a *beamer* document to enable the user to work as if in a *a0poster* class. Thus *beamer*'s neat provisions for layout may be used when creating the poster. Documentation of *beamerposter* is sparse, but an example file allows the user to get a grip on what's available.

Despite the relative shortage of tools, there are a fair few web pages that explain the process (mostly in terms of the *a0poster* route):

- from Norman Gray, Producing posters using LaTeX;
- from Nicola Talbot, Creating technical posters with LaTeX
- from "*awf*" and "*capes*" Preparing conference posters in LaTeX (the page discusses a sample, which you can download);
- From Rob Clark Advanced LaTeX Posters (which has links to code samples);
- from Brian Wolven, LaTeX Poster Macros, Examples, and Accessories (this page also provides macros and other support suggestions); and
- from "*pjh*" Making and printing a poster with LaTeX, which covers the specific issue of dealing with University of Florida styled poster (offering supporting material as necessary), but has hints which are generally useful.

a0poster.cls: macros/latex/contrib/a0poster

beamer.cls: macros/latex/contrib/beamer *beamerposter.sty*: macros/latex/contrib/beamerposter

flowfram.sty: macros/latex/contrib/flowfram

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

sciposter.cls: macros/latex/contrib/sciposter

textpos.sty: macros/latex/contrib/textpos

192 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-spaced thesis text, you may also need separately to set up double spacing.

If you want to write a new thesis class of your own, a good place to start is the University of California style, but remember that it's often difficult to produce a thesis that both looks good and conforms with the style that your Univeristy demands.

UC thesis style: macros/latex/contrib/ucthesis

193 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 searching CTAN) — 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.

194 A 'report' from lots of 'article's

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

The nearest things to canned solutions are Peter Wilson's *combine* and Federico Garcia's *subfiles* classes, but many approaches have been proposed. Each of of the offerings has its own advantages; in particular, several distinctly light-weight solutions (for example, *includex* and *docmute*) are available, well-suited to less formal documents.

Combine defines the means to '`\import`' entire documents, and provides means of specifying significant features of the layout of the document, as well as a global table of contents, and so on. The complete set of facilities is pretty complex. 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. The basic structure of a combined document would be:

```
\documentclass[...]{combine}
...
\begin{document}
...
<introductory materiel>
...
\begin{papers}
% title and author of first article,
% to go the the main ToC
```

```
\coltoctitle{...}
\coltocauthor{...}
\label{art1}
\import{art1}
...
\end{papers}
...
<acknowledgements, etc.>
...
\end{document}
```

The *subfiles* class is used in the component files of a multi-file project, and the corresponding *subfiles* package is used in the master file; so the structure of the master file looks like:

```
\documentclass{<whatever>}
...
\usepackage{subfiles}
...
\begin{document}
...
\subfile{subfile_name}
...
\end{document}
```

while a subfile has the structure:

```
\documentclass[mainfile_name]{subfiles}
\begin{document}
...
\end{document}
```

Arrangements may be made so that the component files will be typeset using different page format, etc., parameters than those used when they are typeset as a part of the main file.

A more 'raw' toolkit is offered by Matt Swift's *includex* and *newclude* packages, both part of the *frankenstein* bundle. Note that Matt believes *includex* is obsolete (though it continues to work for this author); furthermore, its replacement, *newclude* remains "in development", as it has been since 1999.

Both *includex* and *newclude* enable you to '`\includedoc`' complete articles (in the way that you '`\include`' chapter files in an ordinary report). The preamble (everything up to `\begin{document}`), and everything after `\end{document}`, is ignored by both packages. Thus the packages don'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. (Both packages require *moredefs*, which is also part of the bundle.)

A neat (and simple) toolkit is offered by the *docmute* package; once the package is loaded, anything between `\documentclass[...]{...}` and `\begin{document}` in an `\input`'ed or `\include`'d document is ignored, and then the input is processed up to `\end{document}` in the input file. The package does nothing about `\usepackage` (or anything else) in the preamble of the included document; it's up to the user to ensure that any packages needed are loaded, and any other necessary configuration is done, in the parent document.

The *standalone* package develops on the ideas of *docmute*; it was designed to meet the needs of users who are developing images from one of the more extreme new graphics packages (notably *pgf/tikz*) where the compile time of the graphics is such that separate compilation is very desirable. *Standalone* provides a means of developing the graphics in a convenient way, detached from the development of the document as a whole; its value for use in multiple documents is clear.

The user includes the *standalone* package in the main document, and each subfile uses the *standalone* class. (*Standalone* uses *article* for the "real" work in stand-alone mode, but it may be asked to use another).

The real difference from the *docmute* package is flexibility. In particular, you can ask that the preambles of the included documents be gathered up, so that you can construct a good preamble for the master document.

A final "compile-together" approach comes from the *subdocs* package. The driver file contains a `\subdocuments` command:

```
\subdocuments[options]{file1, file2, ...}
```

(the optional arguments provide layout options, such as control over whether `\clearpage` or `\cleardoublepage` are used between the files). Each of the sub-files will execute

```
\usepackage[master]{subdocs}
```

to declare the name, *master*, of the calling file; each of the subfiles reads all the `.aux` files, so that tables of contents may be produced.

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/combine

combinet.sty: macros/latex/contrib/combine

docmute.sty: macros/latex/contrib/docmute

includex.sty: Distributed in the "unsupported" part of macros/latex/contrib/frankenstein

moredefs.sty: Distributed as part of macros/latex/contrib/frankenstein

newclude.sty: Distributed as part of macros/latex/contrib/frankenstein

pdfpages.sty: macros/latex/contrib/pdfpages

standalone.cls, standalone.sty: macros/latex/contrib/standalone

subdocs.sty: Distributed as part of macros/latex/contrib/bezos

subfiles.cls, etc.: macros/latex/contrib/subfiles

195 *Curriculum Vitae* (Résumé)

Andrej Brodnik's class, *vita*, offers a framework for producing a *curriculum vitae*. The class may be customised both for subject (example class option files support 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.

Xavier Danaux offers a class *moderncv* which supports typesetting modern *curricula vitarum*, both in a classic and in a casual style. It is fairly customizable, allowing you to define your own style by changing the colours, the fonts, etc.

The European Commission has recommended a format for *curricula vitarum* within Europe, and Nicola Vitacolonna has developed a class *europecv* to produce it. While (by his own admission) the class doesn't solve all problems, it seems well-thought out and supports all current official EU languages (together with a few non-official languages, such as Catalan, Galician and Serbian).

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

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

currvita.sty: macros/latex/contrib/currvita

curve.cls: macros/latex/contrib/curve

europecv.cls: macros/latex/contrib/europecv

moderncv.cls: macros/latex/contrib/moderncv

resume.sty: obsolete/macros/latex209/contrib/resume/resume.sty

vita.cls: macros/latex/contrib/vita

196 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. (The latest version of Knuth's macros appear in his "local library" dump on the archive, which is updated in parallel with new versions of TeX — so not very often. . .)

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.

Other classes recommended for inclusion in this FAQ are *akletter* and *isodoc*.

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

There are letter classes in each of the excellent *KOMA-script* (*scrlttr2*: 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/akletter

brief.cls: Distributed as part of macros/latex/contrib/ntgclass

dinbrief.cls: macros/latex/contrib/dinbrief

isodoc.cls: macros/latex/contrib/isodoc

Knuth's letter.tex: systems/knuth/local/lib/letter.tex

newlfm.cls: macros/latex/contrib/newlfm

scrlttr2.cls: Distributed as part of macros/latex/contrib/koma-script

197 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.

A simple solution is to use 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 may not be optimal — but they're certainly practical.

More satisfactory are the *KOMA-script* classes, which are designed to work properly with the class option files that come with *extsizes*, and the *memoir* class that has its own options for document font sizes 9pt–12pt, 14pt, 17pt, 20pt, 25pt, 30pt, 36pt, 48pt and 60pt.

Many classes, designed to produce typeset results other than on "ordinary" paper, will have their own font size mechanisms and ranges of sizes. This is true, for example, of poster classes (such as *a0poster*), and of presentation and lecturing classes (such as *beamer*.

a0poster.cls: `macros/latex/contrib/a0poster`

beamer.cls: `macros/latex/contrib/beamer`

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

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

memoir.cls: `macros/latex/contrib/memoir`

O.2 Document structure

198 The style of document titles

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.

The memoir class includes all the functionality of the *titling* package, while the *KOMA-script* classes have their own range of different titling styles.

Finally, the indefatigable Vincent Zoonekynd supplies examples of how to program alternative title styles. The web page is not useful to users unless they are willing to do their own LaTeX programming.

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

memoir.cls: `macros/latex/contrib/memoir`

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

199 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 section 2.2 of *The LaTeX Companion* (see LaTeX books; the programming itself is discussed elsewhere in this FAQ). The following hack will probably satisfy your editor. Define yourself new commands

```
\newcommand{\ssection}[1]{%
  \section[#1]{\centering\normalfont\scshape #1}}
\newcommand{\ssubsection}[1]{%
  \subsection[#1]{\raggedright\normalfont\itshape #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.

The *titlesec* package offers a structured approach to the problem, based on redefinition of the sectioning and chapter commands themselves. This approach allows it to offer radical adjustment: its options provide (in effect) a toolbox for designing your own sectioning commands' output.

The *sectsty* package provides a more simply structured set of tools; while it is less powerful than is *titlesec*, it is perhaps preferable for minor adjustments, since you can use it after having read a smaller proportion of the manual.

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.

The *memoir* class includes facilities that match *sectsty* and *titlesec*, as well as a bundle of chapter heading styles (including an *anonchap*-equivalent). The *KOMA-script* classes also have sets of tools that provide equivalent functionality, notably formatting specifications `\partformat`, `\chapterformat`, `\sectionformat`, . . . , as well as several useful overall formatting specifications defined in class options.

Finally, the indefatigable Vincent Zoonekynd supplies examples of how to program alternative chapter heading styles and section heading styles. The web pages provide programming examples, and expect users to adapt them to their own LaTeX use.

anonchap.sty: `macros/latex/contrib/anonchap`

fncychap.sty: `macros/latex/contrib/fncychap`

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

memoir.cls: `macros/latex/contrib/memoir`

sectsty.sty: `macros/latex/contrib/sectsty`

titlesec.sty: `macros/latex/contrib/titlesec`

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

200 Appendixes

LaTeX provides an exceedingly simple mechanism for appendixes: the command `\appendix` switches the document from generating sections (in *article* class) or chapters (in *report* or *book* classes) to producing appendixes. Section or chapter numbering is restarted and the representation of the counter switches to alphabetic. So:

```
\section{My inspiration}
...

\section{Developing the inspiration}
...

\appendix
\section{How I became inspired}
...
```

would be typeset (in an *article* document) something like:

1 My inspiration

. . .

2 Developing the inspiration

. . .

A How I became inspired

. . .

which is quite enough for many ordinary purposes. Note that, once you've switched to typesetting appendixes, LaTeX provides you with no way back — once you've had an appendix, you can no longer have an "ordinary" `\section` or `\chapter`.

The *appendix* provides several ways of elaborating on this simple setup. Straightforward use of the package allows you to have a separate heading, both in the body of the document and the table of contents; this would be achieved by

```
\usepackage{appendix}
...
\appendix
\appendixpage
\addappheadtotoc
```

The `\appendixpage` command adds a separate title "Appendices" above the first appendix, and `\addappheadtotoc` adds a similar title to the table of contents. These simple modifications cover many people's needs about appendixes.

The package also provides an `appendices` environment, which provides for fancier use. The environment is best controlled by package options; the above example would be achieved by

```
\usepackage[toc,page]{appendix}
...
\begin{appendices}
...
\end{appendices}
```

The great thing that the `appendices` environment gives you, is that once the environment ends, you can carry on with sections or chapters as before — numbering isn't affected by the intervening appendixes.

The package provides another alternative way of setting appendixes, as inferior divisions in the document. The `subappendices` environment allows you to put separate appendixes for a particular section, coded as `\subsections`, or for a particular chapter, coded as `\sections`. So one might write:

```
\usepackage{appendix}
...
\section{My inspiration}
...
\begin{subappendices}
\subsection{How I became inspired}
...
\end{subappendices}

\section{Developing the inspiration}
...
```

Which will produce output something like:

1 My inspiration

...

1.A How I became inspired

...

2 Developing the inspiration

...

There are many other merry things one may do with the package; the user is referred to the package documentation for further details.

The *memoir* class includes the facilities of the *appendix* package. The *KOMA-script* classes offer a `\appendixprefix` command for manipulating the appearance of appendixes.

appendix.sty: macros/latex/contrib/appendix

KOMA script bundle: macros/latex/contrib/koma-script

memoir.cls: macros/latex/contrib/memoir

201 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

202 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* package provides a sensible set of macros for 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`), or to make use of the LaTeX Companion, which discusses the use of `\@startsection` for this sort of thing.

You will note that Lamport didn't go on adding "sub" to the names of sectioning commands, when creating commands for the lowest levels of a document. This would seem sensible to any but the most rigorous stickler for symmetry — it would surely challenge pretty much anyone's reading of the source of a document, if there was a need to distinguish `\subsubsubsection` and `\subsubsubsubsection`

LaTeX source: `macros/latex/base`

`titlesec.sty`: `macros/latex/contrib/titlesec`

203 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, in a package of your own, or between `\makeatletter`–`\makeatother`:

```
\renewcommand{\fnum@figure}{\textbf{Fig.~\thefigure}}
```

which will cause the number to be typeset in bold face. (Note that the original definition used `\figurename` — `\figurename`.) 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 *caption* and *ccaption* (note the double "*c*") packages provide a range of different formatting options.

ccaption also provides 'continuation' captions and captions that can be placed outside of float environments. The (very simple) *capt-of* package also allows captions outside a float environment. Note that care is needed when doing things that assume the sequence of floats (as in continuation captions), or potentially mix non-floating captions with floating ones.

The *memoir* class includes the facilities of the *ccaption* package; the *KOMA-script* classes also provide a wide range of caption-formatting commands.

The documentation of *caption* is available by processing a file `manual.tex`, which is created when you unpack `caption.dtx`

Note that the previously-recommended package *caption2* has now been overtaken again by *caption*; however, *caption2* remains available for use in older documents.

`caption.sty`: `macros/latex/contrib/caption`

`capt-of.sty`: `macros/latex/contrib/capt-of`

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

float.sty: macros/latex/contrib/float

KOMA script bundle: macros/latex/contrib/koma-script

memoir.cls: macros/latex/contrib/memoir

204 Alternative head- and footlines in LaTeX

The standard LaTeX document classes define a small set of 'page styles' which specify head- and footlines for your document (though they can be used for other purposes, too). The standard set is very limited, but LaTeX is capable of much more. The internal LaTeX coding needed to change page styles is not particularly challenging, but there's no need — there are packages that provide useful abstractions that match the way we typically think about these things.

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

While *fancyhdr* will work with *KOMA-script* classes, an alternative package, *scrpage2*, eases integration with the classes. *Scrpage2* may also be used as a *fancyhdr* replacement, providing similar facilities. The *KOMA-script* classes themselves permit some modest redefinition of head- and footlines, without the use of the extra package.

Memoir also contains the functionality of *fancyhdr*, and has several predefined styles.

Documentation of *fancyhdr* is distributed with the package, in a separate file; documentation of *scrpage2* is integrated with the `scrgui*` documentation files that are distributed with the *KOMA-script* classes.

fancyhdr.sty: macros/latex/contrib/fancyhdr

KOMA script bundle: macros/latex/contrib/koma-script

memoir.cls: macros/latex/contrib/memoir

205 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/ccaption

float.sty: macros/latex/contrib/float

206 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 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} % ditto abstract
    ... text
  \end{onecolabstract}
]
\saythanks             % typeset any \thanks
```

The *memoir* class offers all the facilities of *abstract*.

abstract.sty: macros/latex/contrib/abstract

memoir.cls: macros/latex/contrib/memoir

207 Really blank pages between chapters

If you're using the standard classes, you need to take special action; the *memoir* class and the *Koma-Script* classes provide their own support for this — see below.

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`: instructions on macro programming patching techniques explain the problem and why this is a solution.

The *emptypage* package does this sort of thing for you; all you need do is load the package, and it does the rest.

The *KOMA-Script* replacements for the *book* and *report* classes (*scrbook* and *scrreprt* offers class options `cleardoubleempty`, `cleardoubleplain` and `cleardoublestandard` (using the running page style, as normal) that control the appearance of these empty pages. The classes also offer do-it-yourself commands `\cleardoubleempty` (etc.).

The *memoir* class (and the *nextpage* package) provide commands `\cleartooddpage` and `\cleartoevenpage`, which both take an optional argument (the first, with no argument, being an equivalent of `\cleardoublepage`). One can achieve 'special' effects by putting commands in the optional argument: the `\clearemptydoublepage` we're after would be achieved by `\cleartooddpage[\thispagestyle{empty}]`. The commands will also serve if you want the surreal effect of "This page intentionally left blank" in the centre of an otherwise empty page.

emptypage.sty: `macros/latex/contrib/emptypage`

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

memoir.cls: `macros/latex/contrib/memoir`

nextpage.sty: `macros/latex/contrib/misc/nextpage.sty`

scrbook.cls, scrrept.cls: Part of `macros/latex/contrib/koma-script`

208 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 patching commands. 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/preprint`

flushend.sty: Distributed as part of `macros/latex/contrib/sttools`

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

209 My section title is too wide for the page header

By default, LaTeX sectioning commands make the chapter or section title available for use by page headers and the like. Page headers operate in a rather constrained area, and it's common for titles too be too big to fit: the LaTeX sectioning commands therefore take an optional argument:

```
\section[short title]{full title}
```

If the ⟨*short title*⟩ is present, it is used both for the table of contents and for the page heading. The usual answer to people who complain that their title is too big for the running head is to suggest that they the optional argument.

However, using the same text for the table of contents as for the running head may also be unsatisfactory: if your chapter titles are seriously long (like those of a Victorian novel), a valid and rational scheme is to have a shortened table of contents entry, and a really terse entry in the running head.

One of the problems is the tendency of page headings to be set in capitals (which take up more space); so why not set headings as written for "ordinary" reading? It's not possible to do so with unmodified LaTeX, but the *fancyhdr* package provides a command `\nouppercase` for use in its header (and footer) lines to suppress LaTeX's uppercasing tendencies. Classes in the *KOMA-script* bundle don't uppercase in the first place.

In fact, the sectioning commands use 'mark' commands to pass information to the page headers. For example, `\chapter` uses `\chaptermark`, `\section` uses `\sectionmark`, and so on. With this knowledge, one can achieve a three-layer structure for chapters:

```
\chapter[middling version]{verbose version}
\chaptermark{terse version}
```

which should supply the needs of every taste.

Chapters, however, have it easy: hardly any book design puts a page header on a chapter start page. In the case of sections, one has typically to take account of the nature of the `\*mark` commands: the thing that goes in the heading is the first mark on the page (or, failing any mark, the last mark on any previous page). As a result the recipe for sections is more tiresome:

```
\section[middling version]{verbose version%
                \sectionmark{terse version}}
\sectionmark{terse version}
```

(the first `\sectionmark` deals with the header of the page the `\section` command falls on, and the second deal with subsequent pages; note that here, you need the optional argument to `\section`, even if "*middling version*" is in fact the same text as "*long version*".)

A similar arrangement is necessary even for chapters if the class you're using is odd enough that it puts a page header on a chapter's opening page.

Note that the *titlesec* package manages the running heads in a completely different fashion; for example, you can use the optional argument of sectioning commands for page headers, only, by loading the package as:

```
\usepackage[toctitles]{titlesec}
```

The package documentation offers other useful techniques in this area.

The *memoir* class avoids all the silliness by providing an extra optional argument for chapter and sectioning commands, for example:

```
\section[middling version][terse version]{verbose version}
```

As a result, it is always possible for users of *memoir* to tailor the header text to fit, with very little trouble.

fancyhdr.sty: macros/latex/contrib/fancyhdr

KOMA script bundle: macros/latex/contrib/koma-script

memoir.cls: macros/latex/contrib/memoir

titlesec.sty: macros/latex/contrib/titlesec

210 Page numbering "⟨*n*⟩ of ⟨*m*⟩"

Finding the page number of the last page of a document, from within the document, is somewhat tricky. The *lastpage* package is therefore supplied to make life easy for us all; it defines a label `LastPage` whose number is *right* (after sufficiently many passes through LaTeX, of course). The *memoir* class also defines a "last page" label.

The documentation of the *fancyhdr* package spells out exactly how one might actually use this information to produce page numbering as suggested in the question.

fancyhdr documentation: `macros/latex/contrib/fancyhdr`

`lastpage.sty`: `macros/latex/contrib/lastpage`

211 Page numbering by chapter

When I was a young man, a common arrangement for loose bound technical manuals is to number pages by chapter. (It's quite a good scheme, in those situations: even if your corrections add a whole page to the chapter, the most you have to redistribute is that chapter.)

The problem, at first sight, seems pretty much the same as that in another answer on running numbers within a chapter, and the basic technique is indeed pretty similar.

However, tidying-up loose ends, making sure the page number gets reset to the correct value at the start of each chapter, and so on, is slightly more challenging. This is why the *chappg* package was written: it does the obvious things, and more.

Users have been known to ask for running page numbers within a section, but this really doesn't make sense: you need to run page numbers within document objects that always start on a fresh page.

`chappg.sty`: `macros/latex/contrib/chappg`

O.3 Page layout

212 The size of printed output

The final product of a (La)TeX run is something for a person to read. Often, nowadays, that product will be read "on-screen", but the printed page remains a principal output form.

When we come to print our output, it is important that the output fits on the paper; in some cases, for the output to "fit" is good enough. However, there are circumstances where the actual size of the printed output, on the page, is crucial to the acceptance of the output. (This might happen when the output is a book to be published, or when it's a dissertation which must match the fancies of some bureaucrat even to be considered.)

Sadly, we often find that the printed output doesn't conform to our expectations…

The check-list for such problems has two entries:

- Your output is generated via Adobe *Reader* (or possibly "*Acrobat Reader*" — older versions of the program had the qualified name). In this case, it may be that *Reader* is willfully changing the size of your output: read *Reader* antics.
- Something in your TeX system is producing the wrong size (or shape) of output: read paper geometry.

An alternative approach is to use the (excellent) *testflow* suite, that provides a detailed outline of the potential problems together with a sample document and prototype outputs.

testflow bundle: `macros/latex/contrib/IEEEtran/testflow`

213 Adobe *Reader* messing with print size

Printing from Adobe *Reader* shrinks the page "to fit" (*by default*). Unfortunately, its calculation doesn't consider the existing margins of the document, so that it shrinks what it believes is your whole page onto what it believes is its output page. The effect typically looks as if your margins have expanded.

Solve this problem by adjusting the *Reader*'s default in the print dialogue; unfortunately, this dialogue varies from one version to the next:

- *Reader* version 7:
  Page Scaling (default: "Fit to printer margins") — change to "None", and
  Scale (default 95% of Normal size) — change to "100%".
- Adobe Reader 6:
  in the print dialogue, on the "copies & pages" pane, you'll find a popup menu/drop-down list titled "Page Scaling" — change to "None".
- Windows, Linux Acrobat (Reader) 5.0:
  In the print dialog, make sure the "Shrink oversized pages to fit" checkbox is unchecked. It may also be useful to uncheck the "Expand small pages to fit paper size" checkbox as well.

214 Getting the right paper geometry from (La)TeX

If your output is the wrong size, and you've checked that it's not due to the ministrations of Adobe *Reader*, the problem is probably that your (La)TeX system is producing output that specifies the wrong paper size. Paper sizes can be a pain: they're a forgotten backwater — Knuth seems not to have considered paper size as something the TeX engine needs to know about. As a result, there is no DVI command to specify the paper on which the document should be printed, which has led a dichotomy where macros shape the text according to the needs of the author's chosen paper size, and device drivers' choice happens independently of the macros' ideas.

In practice, one usually finds that macro packages (such as Plain TeX and LaTeX) assume American "letter" paper size, by default; and since most distributions nowadays originate in Europe, the drivers usually default to ISO "A4" paper size.

This is (of course) pretty unsatisfactory. Users may select a different paper size for their document (current LaTeX offers a range of sizes as options in the standard classes), pretty easily. Nevertheless, the user also has to be sure that each time *xdvi*, *dvips* (or whatever) runs, it uses the paper size the document was designed for.

The default paper size for DVI drivers may be changed by a distribution management command (*texconfig* for TeX Live, the *Options* application for MiKTeX), but this still doesn't provide for people using the "wrong" sort of paper for some reason.

A different issue arises for users of PDFTeX — the PDF format *does* have the means of expressing paper size and PDFTeX has system variables `\pdfpagewidth` and `\pdfpageheight`, that are written into the output PDF file. Unfortunately, most of the core software predates PDFTeX, so not even PDFLaTeX sets the correct values into those variables, to match the paper size specified in a `\documentclass` option.

The DVI drivers *dvips*, *dvipdfm* and its extensions (*dvipdfmx* and *xdvipdfmx*) define `\special` commands for the document to specify its own paper size; so in those cases, as when PDFTeX or VTeX is being used, the paper size can be programmed by the document. Users who wish to, may of course consult the manuals of the various programs to write the necessary code.

The *geometry* and *zwpagelayout* packages (whose main business includes defining typeset page areas), also takes notice the size of the paper that the document is going to be printed on, and can issue the commands necessary to ensure the correct size of paper is used. If *geometry* is used when a document is being processed by PDFLaTeX, it can set the necessary dimensions "in the output". If the document is being processed by LaTeX on a TeX or e-TeX engine, there are package options which instruct *geometry* which `\special` commands to use. (Note that the options are ignored if you are using either PDFLaTeX or VTeX.)

So, one resolution of the problem, when you are using LaTeX, is to add

```
\usepackage[processor-option,...]{geometry}
```

Where `processor-option` tells the package what will produce your (PostScript or PDF output — *geometry* knows about `dvips` and `dvipdfm` (`dvipdfm` also serves for the extension *dvipdfmx* and *xdvipdfmx*).

If you're using PDFLaTeX, XeTeX or VTeX, load with

```
\usepackage[program-option,...]{geometry}
```

where `program-option` is `pdftex`, `xetex` or `vtex`.

The alternative, *zwpagelayout* requires a `driver` option:

```
\usepackage[driver=value,...]{zwpagelayout}
```

(permissible ⟨*values*⟩ are `pdftex`, `xetex` and `dvips`; the default value is `unknown`).

Needless to say, both the "big" classes (*koma-script* and *memoir*) provide their own ways to get the paper size "right".

The *typearea* package is the *Koma-script* distribution's way of providing page layout functionality. Load it with the `pagesize` option and it will ensure the correct paper is selected, for PDF output from PDFLaTeX, and for PostScript output from LaTeX via *dvips*.

Memoir has the standard classes' paper-size selections (`a4paper`, `letterpaper` and so on), but also permits the user to choose an arbitrary paper size, by setting the length registers `\stockheight` and `\stockwidth`. The commands `\fixdvipslayout` (for LaTeX processing), and `\fixpdflayout` (for PDFLaTeX processing) then instruct the processor to produce output that specifies the necessary paper size.

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

`memoir.cls`: `macros/latex/contrib/memoir`

`typearea.sty`: Distributed as part of `macros/latex/contrib/koma-script`

`zwpagelayout.sty`: `macros/latex/contrib/zwpagelayout`

215 Changing the margins in LaTeX

Changing the layout of a document's text on the page involves several subtleties not often realised by the beginner. 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.

LaTeX's defaults sometimes seem excessively conservative, but there are sound reasons behind how Lamport designed the layouts themselves, whatever one may feel about his overall design. For example, the common request for "one-inch margins all round on A4 paper" is 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).

So Lamport's warning to beginners in his section on 'Customizing the Style' — "don't do it" — should not lightly be ignored.

This set of FAQs 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, but remember — the packages only provide consistent, working, mechanisms: they don't analyse the quality of what you propose to do.

The following answers deal with the ways one may choose to proceed:

- Choose which package to use.
- Find advice on setting up page layout by hand.

There is a related question — how to change the layout temporarily — and there's an answer that covers that, too:

- Change the margins on the fly.

216 Packages to set up page designs

There are two trustworthy tools for adjusting the dimensions and position of the printed material on the page are *geometry* and the *zwpagelayout* packages; a very wide range of adjustments of the layout may be relatively straightforwardly programmed with either, and package documentation is good and comprehensive.

As is usual, users of the *memoir* class have built-in facilities for this task, and users of the *KOMA-script* classes are recommended to use an alternative package, *typearea*. In either case it is difficult to argue that users should go for *geometry*: both alternatives are good.

The documentation both of *geometry* and of *zwpagelayout* is rather overwhelming, and learning all of of either package's capabilities is likely to be more than you ever need. The *vmargin* package is somewhat simpler to use: it has a canned set of paper sizes (a superset of that provided in LaTeX2e), provision for custom paper, margin adjustments and provision for two-sided printing.

geometry.sty: macros/latex/contrib/geometry

KOMA script bundle: macros/latex/contrib/koma-script

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

memoir.cls: macros/latex/contrib/memoir

typearea.sty: Distributed as part of macros/latex/contrib/koma-script

vmargin.sty: macros/latex/contrib/vmargin

zwpagelayout.sty: macros/latex/contrib/zwpagelayout

217 How to set up page layout "by hand"

So you're eager to do it yourself, notwithstanding the cautions outlined in "changing margins".

It's important that you first 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 LaTeX books).

LaTeX controls the page layout with a number of parameters, which allow you to change the distance from the edges of a page to the left and top edges of your typeset text, the width and height of the text, and the placement of other text on the page. However, 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.

Even changing the text height and width, `\textheight` and `\textwidth`, requires more care than you might expect: the height should be set to fit a whole number of text lines (in terms of multiples of `\baselinskip`), and the width should be constrained by the number of characters per line, as mentioned in "changing margins".

Margins are controlled by two parameters: `\oddsidemargin` and `\evensidemargin`, whose names come from the convention that odd-numbered pages appear on the right-hand side ('recto') of a two-page spread and even-numbered pages on the left-hand side ('verso'). Both parameters actually refer to the left-hand margin of the relevant pages; in each case the right-hand margin is specified by implication, from the value of `\textwidth` and the width of the paper. (In a one-sided document, which is the default in many classes, including the standard *article* and *report* classes, `\oddsidemargin` stands for both.)

The "origin" (the zero position) on the page 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, the parameters `\evensidemargin`, `\oddsidemargin` and `\topmargin`, should be set to be 1 inch less than the true margin; for margins closer to the left and top edges of the page than 1 inch, the margin parameters must be set to negative values.

218 Changing margins "on the fly"

One of the surprises characteristic of TeX use is that you cannot change the width or height of the text within the document, simply by modifying the text size parameters; TeX can't change the text width on the fly, and LaTeX only ever looks at text height when starting a new page.

So the simple rule is that the parameters should only be changed in the preamble of the document, i.e., before the `\begin{document}` statement (so before any typesetting has happened.

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}}
```

The 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 1 centimetre.

Given that TeX can't do this, how does it work? — well, the environment (which is a close relation of the LaTeX `quote` environment) *doesn't* change the text width as far as TeX is concerned: it merely moves text around inside the width that TeX believes in.

The *changepage* 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 (*recto*) or an even (*verso*) page of a two-sided document. *Changepage*'s structure matches that of the *memoir* class.

The (earlier) package *chngpage* provides the same facilities, but it uses rather different syntax. *Changepage*'s structure matches that of the *memoir* class, and it should be used for any new work.

Changing the vertical dimensions of a page is more clumsy still: the LaTeX command `\enlargethispage` adjusts the size of the current page by the size of its argument. Common uses are

```
\enlargethispage{\baselineskip}
```

to make the page one line longer, or

```
\enlargethispage{-\baselineskip}
```

to make the page one line shorter. The process is (to an extent) simplified by the *addlines* package: its `\addlines` command takes as argument the *number* of lines to add to the page (rather than a length): the package documentation also provides a useful analysis of when the command may (or may not) be expected to work.

addlines.sty: macros/latex/contrib/addlines

changepage.sty: macros/latex/contrib/changepage

219 How to get rid of page numbers

Very occasionally, one wants a document with no page numbers. For such occasions, the package *nopageno* will make `\pagestyle{plain}` have the same effect as `\pagestyle {empty}`; in simple documents, this will suppress all page numbering (it will not work, of course, if the document uses some other pagestyle than `plain`).

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 deal with the page numbers on pages containing a `\maketitle`, `\part` or `\chapter` command, since the standard classes; deal with those separately, as described below.

To suppress page numbers on a single page, use `\thispagestyle{empty}` somewhere within the text of the page. Note that, in the standard classes, `\maketitle` and `\chapter` use `\thispagestyle` internally, so your call must be *after* those commands.

Unfortunately, `\thispagestyle` doesn't work for *book* or *report* `\part` commands: they set the page style (as do `\chapter` commands), but then they advance to the next page so that you have no opportunity to change the style using `\thispagestyle`. The present author has proposed solving the problem with the following "grubby little patch", on `comp.text.tex`:

```
\makeatletter
\let\sv@endpart\@endpart
\def\@endpart{\thispagestyle{empty}\sv@endpart}
\makeatother
```

Fortunately, that patch has now been incorporated in a small package *nonumonpart* (a difficult name. . .)

Both the *KOMA-script* classes and *memoir* have separate page styles for the styles of various "special" pages, so, in a *KOMA* class document one might say:

```
\renewcommand*{\titlepagestyle}{empty}
```

while *memoir* will do the job with

```
\aliaspagestyle{title}{empty}
```

An alternative (in all classes) 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), so it is unlikely to be helpful other than at the beginning of a document.

The *scrpage2* package separates out the representation of the page number (it typesets the number using the `\pagemark` command) from the construction of the page header and footer; so one can say

```
\renewcommand*{\pagemark}{}
```

which will also suppress the printing of the page number.

Neither of these "suppress the page number" techniques touches the page style in use; in practice this means they don't make sense unless you are using `\pagestyle {plain}`

fancyhdr.sty: macros/latex/contrib/fancyhdr

KOMA script bundle: macros/latex/contrib/koma-script

memoir.cls: macros/latex/contrib/memoir

nonumonpart.sty: macros/latex/contrib/nonumonpart

nopageno.sty: macros/latex/contrib/nopageno

scrpage2.sty: Distributed as part of macros/latex/contrib/koma-script

220 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. Users of the *memoir* class don't need the package, since *memoir* has its own facilities for programming crop marks.

crop.sty: macros/latex/contrib/crop

memoir.cls: macros/latex/contrib/memoir

221 '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). Unfortunately, however, the package relies on PostScript specials, and will therefore fail if you are viewing your document with *xdvi*, and won't even compile if you're using PDFLaTeX. (PDFLaTeX users need one of the other solutions below.)

The *wallpaper* package builds on *eso-pic* (see below). Apart from the single-image backdrops described above ("wallpapers", of course, to this package), the package provides facilities for tiling images. All its commands come in pairs: one for "general" use, and one applying to the current page only.

The *draftwatermark* package uses the same author's *everypage* package to provide a simple interface for adding textual ('DRAFT'-like) watermarks.

The *xwatermark* package provides very flexible watermarking, with a "modern" (key-value) interface.

More elaborate watermarks may be achieved using the *eso-pic* package, which in turn uses the package *everyshi*, or by using *everypage*. *Eso-pic* attaches a `picture` environment to every page as it is shipped out; you can put things into that environment. The package provides commands for placing things at certain useful points (like "text upper left" or "text centre") in the picture, but you're at liberty to do what you like.

Eso-pic is, in turn, built upon the package *atbegshi*. That package has the capability to produce watermarks *on top of* the other material on the page; this doesn't sound very "watermark-like", but can be useful on pages where the watermark would otherwise be hidden by graphics or the like. The *atbegshi* command that *eso-pic* uses is `\AtBeginShipoutUpperLeft`; `\AtBeginShipoutUpperLeftForeground` is what's needed instead to place the material on top of the rest of the content of the page.

Everypage allows you to add "something" to every page, or to a particular page; you therefore need to construct your own apparatus for anything complicated.

Finally, one can use the *pdftk* untility; with it, the command:

```
pdftk a.pdf background b.pdf output c.pdf
```

will recreate `a.pdf` as `c.pdf`, having used the first page of `b.pdf` as background on every page. If you have a standard background ("DRAFT" or "SECRET", or whatever) used in several files, *pdftk* might well be attractive.

Pdftk is available as a command line tool; it is available in most linux distritbutions, but may be downloaded from its home site

atbegshi.sty: Distributed as part of `macros/latex/contrib/oberdiek`

draftcopy.sty: `macros/latex/contrib/draftcopy`

draftwatermark.sty: `macros/latex/contrib/draftwatermark`

eso-pic.sty: `macros/latex/contrib/eso-pic`

everypage.sty: `macros/latex/contrib/everypage`

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

wallpaper.sty: `macros/latex/contrib/wallpaper`

222 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.
Note that *rotating* has problems in a document that also loads the *float* package, which recommended in other answers in these FAQs, for example that on float placement. The *rotfloat* package loads *rotating* for you, and smooths the interaction with *float*.

- 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 (or *pdflscape* if you're generating PDF output, whether using PDFLaTeX or *dvips* and generating PDF from that). Both packages define 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 might use the techniques described in "flowing text around figures", 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. The *memoir* class has the same facilities, in this respect, as does *geometry*.

A word of warning: most current TeX previewers do not honour rotation requests in DVI files. 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/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

memoir.cls: macros/latex/contrib/memoir

pdflscape.sty: Distributed with Heiko Oberdiek's packages macros/latex/contrib/oberdiek

rotating.sty: macros/latex/contrib/rotating

rotfloat.sty: macros/latex/contrib/rotfloat

supertabular.sty: macros/latex/contrib/supertabular

223 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.

There *are* places on the page, from which things may be hung, and two LaTeX packages allow you position things relative to such points, thus providing a means of absolute positioning.

The *textpos* package aids the construction of pages from "blobs", dotted around over the page (as in a poster); you give it the location, and it places your typeset box accordingly.

The *eso-pic* defines a "shipout picture" that covers the page. The user may add picture-mode commands to this picture, which of course can include box placements as well as the other rather stilted commands of picture-mode. (*Eso-pic* requires the services of *everyshi*, which must therefore also be available.)

eso-pic.sty: macros/latex/contrib/eso-pic

everyshi.sty: Distributed as part of macros/latex/contrib/ms

textpos.sty: macros/latex/contrib/textpos

224 Preventing page breaks between lines

One commonly requires that a block of typeset material be kept on the same page; it turns out to be surprisingly tricky to arrange this.

LaTeX provides a `samepage` environment which claims it does this sort of thing for you. It proceeds by setting infinite penalties for all sorts of page-break situations; but in many situations where you want to prevent a page break, `samepage` doesn't help. If you're trying to keep running text together, you need to end the paragraph inside the environment (see preserving paragraph parameters). Also, if the things you are trying to keep together insert their own pagebreak hints, `samepage` has no power over them (though list items' attempts — they suggest page breaks between items — are subverted by `samepage`). Naturally, if `samepage` *does* work, it is capable of leaving stuff jutting out at the bottom of the page.

Another convenient trick is to set all the relevant stuff in a `\parbox` (or a `minipage` if it contains things like verbatim text that may not be used in the argument of a `\parbox`). The resulting box certainly *won't* break between pages, but that's not to say that it will actually do what you want it to do: again, the box may be left jutting out at the bottom of the page.

Why do neither of these obvious things work? — Because TeX can't really distinguish between infinitely awful things. `Samepage` will make any possible break point "infinitely bad" and boxes don't even offer the option of breaks, but if the alternative is the leave an infinitely bad few centimetres of blank paper at the bottom of the page, TeX will take the line of least resistance and do nothing.

This problem still arises even if you have `\raggedbottom` in effect: TeX doesn't notice the value of *that* until it starts actually shipping a page out. One approach is to set:

```
\raggedbottom
\addtolength{\topskip}{0pt plus 10pt}
```

The `10pt` offers a hint to the output routine that the column is stretchable; this will cause TeX to be more tolerant of the need to stretch while building the page. If you're doing this as a temporary measure, cancel the change to `\topskip` by:

```
\addtolength{\topskip}{0pt plus-10pt}
```

as well as resetting `\flushbottom`. (Note that the `10pt` never actually shows up, because it is overwhelmed when the page is shipped out by the stretchability introduced by `\raggedbottom`; however, it could well have an effect if `\flushbottom` was in effect.)

An alternative (which derives from a suggestion by Knuth in the TeXbook) is the package *needspace* or the *memoir* class, which both define a command `\needspace` whose argument tells it what space is needed. If the space isn't available, the current page is cleared, and the matter that needs to be kept together will be inserted on the new page. For example, if 4 lines of text need to be kept together, the sequence

```
\par
\needspace{4\baselineskip}
% the stuff that must stay together
<text generating lines 1-4>
% now stuff we don't mind about
```

Yet another trick by Knuth is useful if you have a sequence of small blocks of text that need, individually, to be kept on their own page. Insert the command `\filbreak` before each small block, and the effect is achieved. The technique can be used in the case of sequences of LaTeX-style sections, by incorporating `\filbreak` into the definition of a command (as in patching commands). A simple and effective patch would be:

```
\let\oldsubsubsection=\subsubsection
\renewcommand{\subsubsection}{%
  \filbreak
  \oldsubsubsection
}
```

While the trick works for consecutive sequences of blocks, it's slightly tricky to get out of such sequences unless the sequence is interrupted by a forced page break (such as `\clearpage`, which may be introduced by a `\chapter` command, or the end of the document). If the sequence is not interrupted, the last block is likely to be forced onto a new page, regardless of whether it actually needs it.

If one is willing to accept that not everything can be accomplished totally automatically, the way to go is to typeset the document and to check for things that have the potential to give trouble. In such a scenario (which has Knuth's authority behind it, if one is to believe the rather few words he says on the subject in the TeXbook) one can decide, case by case, how to deal with problems at the last proof-reading stage. At this stage, you can manually alter page breaking, using either `\pagebreak` or `\clearpage`, or you can place a `\nopagebreak` command to suppress unfortunate breaks. Otherwise, you can make small adjustments to the page geometry, using `\enlargethispage`. Supposing you have a line or two that stray: issue the command `\enlargethispage {2\baselineskip}` and two lines are added to the page you're typesetting. Whether this looks impossibly awful or entirely acceptable depends on the document context, but the command remains a useful item in the armoury.

Note that both `\pagebreak` and `\nopagebreak` take an optional number argument to adjust how the command is to be interpreted. Thus `\pagebreak[0]`, the command 'suggests' that a page break might be worth doing, whereas `\pagebreak[4]` 'demands' a page break. Similarly `\nopagebreak[0]` makes a suggestion, while `\nopagebreak [4]` is a demand. In both commands, the default value of the optional argument is 4.

memoir.cls: macros/latex/contrib/memoir

needspace.sty: macros/latex/contrib/needspace

225 Parallel setting of text

It's commonly necessary to present text in two languages 'together' on a page, or on a two-page spread. For this to be satisfactory, one usually needs some sort of alignment between the two texts.

The *parallel* package satisfies the need, permitting typesetting in two columns (not necessarily of the same width) on one page, or on the two opposing pages of a two-page spread. Use can be as simple as

```
\usepackage{parallel}
...
\begin{Parallel}{<left-width>}{<right-width}
  \ParallelLText{left-text}
  \ParallelRText{right-text}
  \ParallelPar
  ...
\end{Parallel}
```

Parallel can get terribly confused with colour changes, in PDFTeX; the indefatigable Heiko Oberdiek has a patch for this issue — the *pdfcolparallel* package, which maintains separate colour stacks for the columns.

The *parcolumns* package can (in principle) deal with any number of columns: the documentation shows its use with three columns. Usage is rather similar to that of *parallel*, though there is of course a "number of columns to specify":

```
\usepackage{parcolumns}
...
\begin{parcolumns}[<options>]{3}
```

```
    \colchunk{<Column 1 text>}
    \colchunk{<Column 2 text>}
    \colchunk{<Column 3 text>}
    \colplacechunks
    ...
  \end{parcolumns}
```

The ⟨*options*⟩ can specify the widths of the columns, whether to place rules between the columns, whether to set the columns sloppy, etc. Again, there are issues with colours, which are addressed by the *pdfcolparcolumns* package.

The *ledpar* package is distributed with (and integrated with) the *ledmac* package. It provides parallel setting carefully integrated with the needs of a scholarly text, permitting translation, or notes, or both, to be set in parallel with the 'base' text of the document.

ledpar.sty: Distributed with macros/latex/contrib/ledmac

parallel.sty: macros/latex/contrib/parallel

parcolumns.sty: Distributed as part of macros/latex/contrib/sauerj

pdfcolparallel.sty: Distributed as part of macros/latex/contrib/oberdiek

pdfcolparcolumns.sty: pdfcolparcolumns]

226 Typesetting epigraphs

Epigraphs are those neat quotations that authors put at the start of chapters (or even at the end of chapters: Knuth puts things at the ends of chapters of the TeXbook).

Typesetting them is a bit of a fiddle, but not impossible to do for yourself. Fortunately, there are two packages that do the job, to some extent; there are also facilities in the two "big" classes (*memoir* and *koma-script*.

The *epigraph* package defines an \epigraph command, for creating a single epigraph (as at the top of a chapter):

```
\chapter{The Social Life of Rabbits}
\epigraph{Oh!  My ears and whiskers!}%
         {Lewis Carroll}
```

and an epigraphs environment, for entering more than one epigraph consecutively, in a sort of list introduced by \qitem commands:

```
\begin{epigraphs}
\qitem{What I tell you three times is true}%
      {Lewis Carroll}
\qitem{Oh listen do, I'm telling you!}%
      {A.A. Milne}
\end{epigraphs}
```

The \epigraphhead command enables you to place your epigraph *above* a chapter header:

```
\setlength{\unitlength}{1pt}
...
\chapter{The Social Life of Rabbits}
\epigraphhead[<distance>]{%
  \epigraph{Oh!  My ears and whiskers!}%
           {Lewis Carroll}%
}
```

The ⟨*distance*⟩ says how far above the chapter heading the epigraph is to go; it's expressed in terms of the \unitlength that's used in the picture environment; the package's author recommends 70pt.

The package also offers various tricks for adjusting the layout of chapter header (necessary if you've found a hugely long quotation for an \epigraphhead), for patching the bibliography, for patching `\part` pages, and so on. (Some of these suggested patches lead you through writing your own package. . .)

The *quotchap* package redefines chapter headings (in a moderately striking way), and provides an environment `savequotes` in which you can provide one (or more) quotations to use as epigraphs. The facilities seem not as flexible as those of *epigraph*, but it's probably easier to use.

The *memoir* class offers all the facilities of the *epigraph* package. The *Koma-script* classes have commands `\setchapterpreamble` and `\dictum` to provide these facilities.

epigraph.sty: macros/latex/contrib/epigraph

KOMA script bundle: macros/latex/contrib/koma-script

memoir.cls: macros/latex/contrib/memoir

quotchap.sty: macros/latex/contrib/quotchap

O.4 Spacing of characters and lines

227 Double-spaced documents in LaTeX

A quick and easy way of getting inter-line space for copy-editing is to change `\baselinestretch` — `\linespread{1.2}` (or, equivalently `\renewcommand{\baselinestretch}{1.2}`) may be adequate. 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 so that results will be inconsistent.

For preference (and certainly for a production document, such as a dissertation or an article submission), 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* switches off double-spacing at places where even the most die-hard official would doubt its utility (footnotes, figure captions, and so on); it's very difficult to do this consistently if you're manipulating `\baselinestretch` yourself.

Of course, the real solution (other than for private copy editing) is *not* to use double-spacing at all. Universities, in particular, have no excuse for specifying double-spacing in submitted dissertations: LaTeX is a typesetting system, not a typewriter-substitute, and can (properly used) make single-spaced text even more easily readable than double-spaced typewritten text. If you have any influence on your university's system (for example, through your dissertation supervisor), it may be worth attempting to get the rules changed (at least to permit a "well-designed book" format).

Double-spaced submissions are also commonly required when submitting papers to conferences or journals. Fortunately (judging by the questions from users in this author's department), this demand is becoming less common.

Documentation of *setspace* appears as TeX comments in the package file itself.

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

228 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).

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 another question.)

Possibly the 'ultimate' in this field is the *microtype*, which uses the micro-typography capabilities of current PDFTeX to provide a `\textls` command, which operates according to parameters declared in a `\SetTracking` command. *Microtype*'s 'tracking' facility expands the natural spacing font itself, rather than inserting space between characters. Ordinarily, letter-spacing will destroy ligatures; however, this is *wrong* if the font is of a fraktur style, and the package provides a means of protecting the ligatures in a letter-spaced text.

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

microtype.sty: `macros/latex/contrib/microtype`

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

229 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 — 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 significance of the various parameters of *ragged2e*'s operation.

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

230 Cancelling \ragged commands

LaTeX provides commands `\raggedright` and `\raggedleft`, but none to cancel their effect. The `\centering` command is implemented in the same way as the `\ragged*` commands, and suffers in the same way.

The following code (to be inserted in a package of your own, or as internal LaTeX code —internal LaTeX code) 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.

Any but a really old copy of Martin Schröder's *ragged2e* package has a `\justifying` command to match its versions of the LaTeX 'ragged' commands. The package also provides a `justify` environment, which permits areas of justified text in a larger area which is ragged.

raggedze.sty: Distributed as part of macros/latex/contrib/ms

O.5 Typesetting specialities

231 Including a file verbatim in LaTeX

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

```
\usepackage{verbatim}
...
\verbatiminput{verb.txt}
```

Another way is to use the `alltt` environment, which requires the *alltt* package. The environment interprets its contents 'mostly' verbatim, but executes any (La)TeX commands it finds:

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

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

The *moreverb* package extends the *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 package's `listing` environment and the `\listinginput` command also both honour TAB).

The *sverb* package provides verbatim input (without recourse to the facilities of the *verbatim* package):

```
\usepackage{sverb}
...
\verbinput{verb.txt}
```

The *fancyvrb* package offers configurable implementations of everything *verbatim*, *sverb* 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.

The *memoir* class includes the relevant functionality of the *verbatim* and *moreverb* packages.

alltt.sty: Part of the LaTeX distribution.

fancyvrb.sty: macros/latex/contrib/fancyvrb

memoir.cls: macros/latex/contrib/memoir

moreverb.sty: macros/latex/contrib/moreverb

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

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

232 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*. (*Numline* is considered obsolete, but remains available from the archive.)

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 including files verbatim) may be used. Class *memoir* also provides the necessary facilities.

One common use of line numbers is in critical editions of texts, and for this the *edmac* package offers comprehensive support; *ledmac* is a LaTeX port of *edmac*.

The *vruler* package sidesteps many of the problems associated with line-numbering, by offering (as its name suggests) a rule that numbers positions on the page. The effect is good, applied to even-looking text, but is poor in texts that involve breaks such as interpolated mathematics or figures. Documentation of the package, in the package itself, is pretty tough going, though there is an example (also inside the package file).

edmac: macros/plain/contrib/edmac

fancyvrb.sty: macros/latex/contrib/fancyvrb

ledmac.sty: macros/latex/contrib/ledmac

lineno.sty: macros/latex/contrib/lineno

memoir.cls: macros/latex/contrib/memoir

moreverb.sty: macros/latex/contrib/moreverb

numline.sty: obsolete/macros/latex/contrib/numline/numline.sty

vruler.sty: macros/latex/contrib/vruler

233 Code listings in LaTeX

'Pretty' code listings are sometimes considered worthwhile by the "ordinary" 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 of programs are commonly useful, as well.

Verbatim listings are dealt with elsewhere, as is the problem of typesetting algorithm specifications.

The *listings* package is widely regarded as the best bet for formatted output (it is capable of parsing program source, within the package itself), but there are several well-established packages that rely on a pre-compiler of some sort. You may use *listings* to typeset snippets that you include within your source:

```
\usepackage{listings}
\lstset{language=C}
...
\begin{document}
\begin{lstlisting}
#include <stdio.h>

int main(int argc, char ** argv)
{
  printf("Hello world!\n");
  return 0;
}
\end{lstlisting}
\end{document}
```

or you can have it typeset whole files:

```
\usepackage{listings}
\lstset{language=C}
...
\begin{document}
\lstinputlisting{main.c}
\end{document}
```

These very simple examples may be decorated in a huge variety of ways, and of course there are other languages in the package's vocabulary than just *C*...

For a long time, advice on (La)TeX lists seemed to regard *listings* as the be-all and end-all on this topic. In the last few years, viable alternatives have appeared

Highlight is attractive if you need more than one output format for your program: as well as (La)TeX output, *highlight* will produce (X)HTML, RTF and XSL-FO representations of your program listing. The manual leads you through the details of defining a parameter file for a "new" language, as well as the presentation details of a language.

The *minted* package is another alternative that offers the means of creating new language definitions. It requires that documents be pre-processed using an external (*python*) script, *Pygments*. *Pygments*, in turn, needs a "lexer" that knows the language you want to process; lots of these are available, for the more commonly-used languages, and there is advice on "rolling your own" on the <a href='http://pygments.org/docs/lexerdevelopment/'>*Pygments* site</a>

Longer-established, and variously less "powerful" systems include:

- The *lgrind* system is a well-established pre-compiler, with all the facilities one might need and a wide repertoire of languages; it is derived from the even longer-established *tgrind*, whose output is based on Plain TeX.
- The *tiny_c2l* system is slightly more recent: users are again encouraged to generate their own driver files for languages it doesn't already deal with, but its "tiny" name correctly hints that it's not a particularly elaborate system.
- The *C++2LaTeX* system comes with strong recommendations for use with C and C++.
- An extremely simple system is *c2latex*, for which you write LaTeX source in your C program comments. The program then converts your program into a LaTeX document for processing. The program (implicitly) claims to be "self-documenting".

c2latex: `support/c2latex`

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

highlight: `support/highlight`

lgrind: `support/lgrind`

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

minted.sty: `macros/latex/contrib/minted`

tgrind: `support/tgrind`

tiny_c2l: `support/tiny_c2l`

234 Typesetting pseudocode in LaTeX

There is no consensus on the 'right' way to typeset pseudocode. Consequently, there are a variety of LaTeX packages to choose from for producing æsthetically pleasing pseudocode listings.

Pseudocode differs from actual program listings in that it lacks strict syntax and semantics. Also, because pseudocode is supposed to be a clear expression of an algorithm it may need to incorporate mathematical notation, figures, tables, and other LaTeX features that do not appear in conventional programming languages. Typesetting program listings is described elsewhere.

You can certainly create your own environment for typesetting pseudocode using, for example, the `tabbing` or `list` environments — it's not difficult, but it may prove boring. So it's worth trying the following packages, all designed specifically for typesetting pseudocode.

The *algorithms* bundle (which contains packages *algorithm* and *algorithmic*, both of which are needed for ordinary use) has a simple interface and produces fairly nice output. It provides primitives for statements, which can contain arbitrary LaTeX commands, comments, and a set of iterative and conditional constructs. These primitives can easily be redefined to produce different text in the output. However, there is no support for adding new primitives. Typesetting the pseudocode itself is performed in *algorithmic*; the *algorithms* package uses the facilities of the *float* package to number algorithms sequentially, enable algorithms to float like figures or tables, and support including a List of Algorithms in a document's front matter.

Packages in the *algorithmicx* bundle are similar both in concept and output form to *algorithmic* but additionally provide support for adding new keywords and altering the formatting. It provides the *algpseudocode* package which is (almost) a drop-in replacement for *algorithmic*. Another package in the bundle, *algpascal*, uses Pascal-like keywords, indents differently from *algpseudocode*, and puts command arguments in maths mode instead of text mode. There is no floating environment but *algorithmicx*, like *algorithmic*, is compatible with the *algorithm* package. (There have been reports of difficulty defining new commands to fit with the package; unfortunately, the author is not available to comment.)

The *alg* package, like *algorithms*, offers a floating algorithm environment with all of the ensuing niceties. *alg*, however, can caption its floats in a variety of (natural) languages. In addition, *alg* unlike *algorithms*, makes it easy to add new constructs.

The *newalg* package has a somewhat similar interface to *algorithms*, but its output is designed to mimic the rather pleasant typesetting used in the book "*Introduction to Algorithms*" by Corman, Leiserson, Rivest and Stein. Unfortunately, *newalg* does not support a floating environment or any customisation of the output.

"*Bona fide*" use of the style of "Introduction to Algorithms" may be achieved with Cormen's own *clrscode*: this is the package as used in the second edition of the book.

Similarly, the style of "*Combinatorial Algorithms: Generation, Enumeration and Search*" is supported by the *pseudocode* package, written by the authors of the book. It has the common 'Pascal-like' style, and has some interesting constructs for what one thinks of as Pascal blocks.

The *algorithm2e* is of very long standing, and is widely used and recommended. It loads the *float* package to provide the option of floating algorithm descriptions, but you can always use the "`H`" option of *float* to have the algorithm appear "where you write it".

The usage of the *program* package is a little different from that of the other packages. It typesets programs in maths mode instead of text mode; and linebreaks are significant. *program* lacks a floating environment but does number algorithms like *alg* and *algorithms*. Customisation and extension are not supported. Documentation of the *program* package (such as it is) appears in a file `program.msg` in the distribution.

None of the above are perfect. The factors that should influence your choice of package include the output style you prefer, how much you need to extend or modify the set of keywords, and whether you require algorithms to float like figures and tables.

`algorithm2e.sty`: `macros/latex/contrib/algorithm2e`

`algorithmicx` bundle: `macros/latex/contrib/algorithmicx`

`algorithms` bundle: `macros/latex/contrib/algorithms`

`alg.sty`: `macros/latex/contrib/alg`

`clrscode.sty`: `macros/latex/contrib/clrscode`

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

`newalg.sty`: `macros/latex/contrib/newalg`

`program.sty`: `macros/latex/contrib/program`

`pseudocode.sty`: `macros/latex/contrib/pseudocode`

235 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 post-processing 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 (Eplain), and TeX (whose macros can be used independently with Plain TeX).

idxtex for LaTeX under VMS; *idxtex* comes with a glossary-maker *glotex*.

texindex(1) A witty little shell-script using *sed* and *awk*; designed for LaTeX under Unix.

texindex(2) The Texinfo system also offers a program *texindex*, whose source is to be found in the *texinfo* distribution. The *ltxindex* package provides macros that enable LaTeX users to use this *texindex*.

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`

ltxindex.sty: `macros/latex/contrib/ltxindex`

makeindex: `indexing/makeindex`

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

texindex (the script): `indexing/texindex`

texindex (the program): Distributed with `macros/texinfo/texinfo`

texsis (system): `macros/texsis`

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

xindy: `indexing/xindy`

236 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 — define a similar `\path` command.
  Path, though it works in simple situations, makes no attempt to work with LaTeX (it is irremediably fragile). Despite its long and honourable history, it is no longer recommended for LaTeX use.
- 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 (use of `\protect`) for use within moving arguments.
  The package ordinarily ignores spaces in the URL, but unfortunately URLs that contain spaces do exist; to typeset them, call the package with the `obeyspaces` option. Two other useful options allow line breaks in the URL in places where they are ordinarily suppressed to avoid confusion: `spaces` to allow breaks at spaces (note, this requires `obeyspaces` as well, and `hyphens` to allow breaks after hyphens. (Note that the package *never* does "ordinary" hyphenation of names inside an URL.)
  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.)

Note that neither `\path` (from package *path*) nor `\url` (from package *url*) is robust (in the LaTeX sense). If you need a URL to go in a moving argument, you need the command `\urldef` from the *url* package. So one might write:

```
\urldef\faqhome\url{http://www.tex.ac.uk/faq}
```

after which, `\faqhome` is robust.

Documentation of both package *path* and package *url* is in the package files.

hyperref.sty: macros/latex/contrib/hyperref

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

path.sty: macros/generic/path

url.sty: macros/latex/contrib/url

237 Typesetting music in TeX

The current best bet for music typesetting is to use *musixtex*. *Musixtex* is a three-pass system that has a TeX-based pass, a processing pass and then a further TeX pass. The middle pass, a program called *musixflx*, optimises the spacing and sorts out slurs and ties. *Musixtex* is demanding of TeX resources, and any significant score requires that typesetting is done using e-TeX, whose expanded variable- and box-register ranges allow for more of the "parallel" activities that abound in a music score. Of course, *musixtex* also requires music fonts; those are available in a separate package on the archive.

Musixtex requires pretty arcane input; most people using it actually prepare (less obscure) input for *pmx*, whose output is TeX input suitable for *musixtex*.

A further preprocessor, *M-Tx*, allows preparation of music with lyrics; *M-Tx*'s output is fed into *pmx*, and thence to *musixtex*.

An alternative path to music examples within a (La)TeX document is *Lilypond*. *Lilypond* is (at heart) a batch music typesetting system with plain text input that does most of its work without TeX. *Lilypond*'s input syntax is less cryptic than is MusiXTeX's, though similar quality is achieved. The *lilypond* FAQ mentions development of graphical user interfaces.

Another alternative in the production of music is the ABC notation, which was developed to notate the traditional music of Western Europe (which can be written on a single stave), though it can be used much more widely. A front end to *musictex* (see below), *abc2mtex*, makes ABC typesetting possible.

The program *midi2tex* can also generate *musictex* output, from MIDI files.

The history of music in TeX goes back some time; the earliest "working" macros were MuTeX, by Angelika Schofer and Andrea Steinbach. MuTeX was very limited, but it was some time before 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 no longer recommended.

The first version of MusixTeX was developed by Andreas Egler, but he withdrew from the project to work on another package, OpusTeX. That never reached the mainstream, and is no longer maintained. The current recommended way of doing OpusTeX's job is *gregorio*, which can in principle feed into a TeX-based document.

Once Andreas Egler had withdrawn (his last version of *musixtex* is preserved on the archive), Daniel Taupin took up the development, leading to the *musixtex* used today.

abc2mtex: support/abc2mtex

M-Tx: support/mtx

midi2tex: support/midi2tex

musictex: macros/musictex

musixtex (Taupin's version): macros/musixtex

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

musixtex fonts: fonts/musixtex-fonts

mutex: macros/mtex

pmx: support/pmx

238 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/ntgclass

parskip.sty: macros/latex/contrib/parskip

239 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. Unfortunately, *dropping* has an intimate relation to the set of device drivers available in an early version of the LaTeX graphics package, and it cannot be trusted to work with modern offerings such as PDFTeX, VTeX or DVIpdfm, let alone such modernisms as XeTeX. As a result, the package is widely deprecated, nowadays.

The more recent *lettrine* package is generally more reliable. It has a well-constructed array of options, and an impressive set of examples adds to the package's document.

dropping: macros/latex/contrib/dropping

lettrine: macros/latex/contrib/lettrine

240 The comma as a decimal separator

TeX embodies the British/American cultural convention of using a period as the separator between the whole number and the decimal fraction part of a decimal number. Other cultures use a comma as separator, but if you use a comma in maths mode you get a small space after it; this space makes a comma that is used as a decimal separator look untidy.

A simple solution to this problem, in maths mode, is to type `3{,}14` in place of `3,14`. While such a technique may produce the right results, it is plainly not a comfortable way to undertake any but the most trivial amounts of typing numbers.

Therefore, if you need to use commas as decimal separator, you will probably welcome macro support. There are two packages that can help relieve the tedium: *icomma* and *ziffer*.

Icomma ensures that there will be no extra space after a comma, unless you type a space after it (as in `f(x, y)` — in the absence of the package, you don't need that space), in which case the usual small space after the comma appears. *Ziffer* is specifically targeted at the needs of those typesetting German, but covers the present need, as well as providing the double-minus sign used in German (and other languages) for the empty 'cents' part of an amount of currency.

The *numprint* package provides a command `\numprint{`*number*`}` that prints its argument according to settings you give it, or according to settings chosen to match the language you have selected in *babel*. The formatting works equally well in text or maths. The command is very flexible (it can also group the digits of very 'long' numbers), but is inevitably less convenient than *icomma* or *ziffer* if you are typing a lot of numbers.

icomma.sty: Distributed as part of macros/latex/contrib/was

numprint.sty: macros/latex/contrib/numprint

ziffer.sty: macros/latex/contrib/ziffer

241 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 imposes a serious restriction. 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 *memoir* class includes the functionality of the *framed* package.

The *mdframed* package does the same job, using a different algorithm. The package also provides means of defining 'private' framed environments, whose parameters are set at definition time, thus saving considerable effort in documents with many framed boxes. Its restrictions seem much the same as those of *framed*; this is as one would expect, but the list is noticeably different in the two packages' documentation.

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/misc/backgrnd.tex

boites.sty: macros/latex/contrib/boites *framed.sty*: macros/latex/contrib/framed

mdframed.sty: macros/latex/contrib/mdframed

memoir.cls: macros/latex/contrib/memoir

shade.tex: macros/generic/shade

242 Overstriking characters

This may be used, for example, to indicate text deleted in the course of editing. Both the *ulem* and the *soul* packages provide the necessary facilities.

Overstriking for cancellation in maths expressions is achieved by a different mechanism.

Documentation of *ulem* is in the package file.

soul.sty: macros/latex/contrib/soul

ulem.sty: macros/latex/contrib/ulem

243 Realistic quotes for verbatim listings

The `cmtt` font has "curly" quotes (`‘thus’`), which are pleasing on the eye, but don't really tally with what one sees on a modern *xterm* (which look like `` `this' ``).

The appearance of these quotes is critical in program listings, particularly in those of Unix-like shell scripts. The *upquote* package modifies the behaviour of the `verbatim` environment so that the output is a clearer representation of what the user must type.

upquote.sty: macros/latex/contrib/upquote/upquote.sty

244 Printing the time

TeX has a primitive register that contains "the number of minutes since midnight"; with this knowledge it's a moderately simple programming job to print the time (one that no self-respecting Plain TeX user would bother with anyone else's code for).

However, LaTeX provides no primitive for "time", so the non-programming LaTeX user needs help.

Two packages are available, both providing ranges of ways of printing the date, as well as of the time: this question will concentrate on the time-printing capabilities, and interested users can investigate the documentation for details about dates.

The *datetime* package defines two time-printing functions: `\xxivtime` (for 24-hour time), `\ampmtime` (for 12-hour time) and `\oclock` (for time-as-words, albeit a slightly eccentric set of words).

The *scrtime* package (part of the compendious *KOMA-Script* bundle) takes a package option (`12h` or `24h`) to specify how times are to be printed. The command `\thistime` then prints the time appropriately (though there's no *am* or *pm* in `12h` mode). The `\thistime` command also takes an optional argument, the character to separate the hours and minutes: the default is of course `:`.

datetime.sty: macros/latex/contrib/datetime

scrtime.sty: Distributed as part of macros/latex/contrib/koma-script

245 Redefining counters' \the-commands

Whenever you request a new LaTeX counter, LaTeX creates a bunch of behind-the-scenes commands, as well as defining the counter itself.

Among other things, `\newcounter{`*fred*`}` creates a command `\the`*fred*, which expands to "the value of *fred*" when you're typesetting.

The definition of `\the`*fred* should express the value of the counter: it is almost always always a mistake to use the command to produce anything else. The value may reasonably be expressed as an arabic, a roman or a greek number, as an alphabetic expression, or even as a sequence (or pattern of) symbols. If you need a decision process on whether to re-define `\the`*fred*, consider what might happen when you do so.

So, for example, if you want your section numbers to be terminated by a period, you could make `\thesection` expand with a terminating period. However, such a change to `\thesection` makes the definition of `\thesubsection` look distinctly odd: you are going to find yourself redefining things left, right and centre. Rather, use the standard techniques for adjusting the presentation of section numbers.

Or, suppose you want the page number to appear at the bottom of each page surrounded by dashes (as in "`--~nnn~--`"). If you try to achieve this by redefining `\thepage`, problems will arise from the use of the page number in the table of contents (each number will have the dashes attached), and `\pageref` references will be oddly modified. In this case, the change of appearance is best done by redefining the page style itself, perhaps using package *fancyhdr*.

O.6 Tables of contents and indexes

246 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 2.3 of *The LaTeX Companion*. The commands `\@pnumwidth`, `\@tocrmarg` and `\@dotsep` control the space for page numbers, the indentation of the right-hand margin, and the separation of the dots in the dotted leaders, respectively. The series of commands named `\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.

The *KOMA-Script* classes provides an optional variant structure for the table of contents, and calculates the space needed for the numbers automatically. The *memoir* class includes the functionality of *tocloft*.

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

`memoir.cls`: `macros/latex/contrib/memoir`

`tocloft.sty`: `macros/latex/contrib/tocloft`

247 Unnumbered sections in the Table of Contents

The way the relevant parts of sectioning commands work is exemplified by the way the `\chapter` command uses the counter `secnumdepth` (described in Appendix C of the LaTeX manual):

1. put something in the `.aux` file, which will appear in the `.toc`;
2. if `secnumdepth` ≥ 0, increase the counter for the chapter and write it out.
3. write the chapter title.

Other sectioning commands are similar, but with other values used in the test.

So a simple way to get headings of funny 'sections' such as prefaces in the table of contents is to use the counter:

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

Unfortunately, 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.

The value of the counter `tocdepth` controls which headings will be finally printed in the table of contents; it is normally set in the preamble and is a constant for the document. The package *tocvsec2* package takes the tedium out of changing the `secnumdepth` and/or the `tocdepth` counter values at any point in the body of the document; the commands (respectively) `\setsecnumdepth` and `\settocdepth` make the changes based on the *name* of the sectional unit (`chapter`, `section`, etc.).

The package *abstract* (see one-column abstracts) 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.

The *KOMA-Script* classes have commands `\addchap` and `\addled`, which work like `\chapter` and `\section` but aren't numbered. The *memoir* class incorporates the facilities of all three of the *abstract*, *tocbibind* and *tocvsec2* packages.

abstract.sty: `macros/latex/contrib/abstract`

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

memoir.cls: `macros/latex/contrib/memoir`

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

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

248 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, lists of figures and tables, the bibliography and the index). As a result, these items aren't numbered (which most people don't mind), and (more importantly) they don't appear in the table of contents.

The correct solution (as always) is to have a class of your own that formats your document according to your requirements. The macro to do the job (`\addcontentsline`) is fairly simple, but there is always an issue of ensuring that the contents entry quotes the correct page. Supposing that our the document is chapter-based (class *report* or *book*, for example), the text:

```
\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.

If you are using *hyperref* (which will link entries in the table of contents to the relevant place in the file), a slight adjustment is necessary:

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

The extra command (`\phantomsection`) gives *hyperref* something to "hold on to" when making the link.

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.

Classes of the *KOMA-script* bundle provide this functionality as a set of class options (e.g., `bibtotoc` to add the bibliography to the table of contents); the *memoir* class includes *tocbibind* itself.

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

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

memoir.cls: `macros/latex/contrib/memoir`

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

249 Table of contents, etc., per chapter

The common style, of a “small” table of contents for each part, chapter, or even section, is supported by the *minitoc* package. The package also supports mini-lists of tables and figures; but, as the documentation observes, mini-bibliographies are a different problem — see bibliographies per chapter.

The package’s basic scheme is to generate a little `.aux` file for each chapter, and to process that within the chapter. Simple usage would be:

```
\usepackage{minitoc}
...
\begin{document}
...
\dominitoc \tableofcontents
\dominilof \listoffigures
...
\chapter{blah blah}
\minitoc \mtcskip \minilof
...
```

though a lot of elaborations are possible (for example, you don’t need a `\minitoc` for every chapter).

Babel doesn’t know about *minitoc*, but *minitoc* makes provision for other document languages than English — a wide variety is available. Fortunately, the current version of the *hyperref* package does know about *minitoc* and treats `\minitoc` tables in the same way as “real” tables of contents.

babel.sty: `macros/latex/required/babel`

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

minitoc.sty: `macros/latex/contrib/minitoc`

250 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 an old version of the LaTeX 2.09 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` is completely ignored.

To create a “general” and an “authors” index, one might write:

```
\usepackage{multind}
\makeindex{general}
\makeindex{authors}
...
\index{authors}{Eccentric Professor}
...
\index{general}{FAQs}
...
\printindex{general}{General index}
\printindex{authors}{Author index}
```

To complete the job, run LaTeX on your file enough times that labels, etc., are stable, and then execute the commands

```
makeindex general
makeindex authors
```

before running LaTeX again. Note that the names of the index files to process are not necessarily related to the name of the LaTeX file you're processing, at all. (There's no documentation that comes with the package: what you see above is as good as you will get…)

The *multind* package doesn't work with the AMSLaTeX classes, but the AMS provide a substitute, the *amsmidx* package. You use the *amsmidx* package pretty much the same as you would *multind*, but if things aren't clear, there *is* documentation (unlike *multind*).

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).

Using *index*, to create an author index together with a "normal" index, one would start with preamble commands:

```
\usepackage{index}
\makeindex
\newindex{aut}{adx}{and}{Name Index}
```

which load the package, define a "main" (original-style) index, and then define an author index. Then, in the body of the document, we might find commands like:

```
\index[aut]{Another Idiot}
...
\index{FAQs}
```

Which place an entry in the author index, and then one in the main index. At the end of the document, we have two commands:

```
\printindex
\printindex[aut]
```

Which will print the main index and then the author index. Supposing this lot to be in `myfile.tex`, after enough runs through LaTeX that labels are stable, execute the following commands (Unix-style shell commands shown here, but the principle is the same whatever system you're using):

```
makeindex myfile
makeindex myfile.adx -o myfile.and
```

and rerun LaTeX. The *makeindex* commands process `myfile.idx` to `myfile.ind` (the default action), and then `myfile.adx` to `myfile.and`, the two files needed as input by the two `\printindex` commands in `myfile.tex`.

The *splitidx* package can operate in the same way as the others: load the package with the `split` option, and declare each index with a `\newindex` command:

```
\newindex[⟨index name⟩]{⟨shortcut⟩}
```

and *splitidx* will generate a file `\jobname.⟨shortcut⟩` to receive index entries generated by commands like `\sindex[⟨shortcut⟩]{⟨item⟩}`. As with the other packages, this method is limited by TeX's total number of output files. However, *splitindex* also comes with a small executable *splitindex* (available for a variety of operating systems); if you use this auxiliary program (and don't use `split`), there's no limit to the number of indexes. Apart from this trick, *splitidx* supports the same sorts of things as does *index*. An example of use appears in the documentation.

The *imakeidx* can do a wide range of things (in particular, it can run an index-builder — via `\write18` commands — so as to simplify business of making the final copy of a document). The package can also make multiple indexes; it can do the job in the conventional (*multind*) way, or by using the external *splitindex* script provided with the *splitindex* package. (This arrangement allows efficient operation with small numbers of indexes, while retaining the flexibility of permitting large numbers of indexes without hitting the restriction of numbers of active output streams.)

The *memoir* class has its own multiple-index functionality (as well as its own index layout options, which other packages delegate to the index style used by *makeindex*).

`amsmidx.sty`: Part of the AMS class distribution `macros/latex/required/amslatex/amscls`

`imakeidx.sty`: `macros/latex/contrib/imakeidx`

`index.sty`: `macros/latex/contrib/index`

`makeidx.sty`: Part of the LaTeX distribution

`memoir.cls`: `macros/latex/contrib/memoir`

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

`splitidx.sty` and `splitindex`: `macros/latex/contrib/splitindex`

O.7 Labels and references

251 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 — don't use it in a document in which you also need to use *hyperref*.

The *byname* package is part of the *smartref* bundle and works well with *smartref*, and works (to an extent) with *hyperref*, but the links it defines are not hyperlinks.

The *memoir* class incorporates the functionality of *titleref*, but doesn't work with *byname* (though a search of `comp.text.tex` on `groups.google.com` will find a patch to *byname* to remedy the problem).

The *hyperref* bundle includes a package *nameref*, which will work standing alone (i.e., without *hyperref*: of course, in this mode its references are not hyperlinked). If you load *hyperref* itself, *nameref* is automatically loaded. *Memoir* requires the *memhfixc* when running with *hyperref*; following the sequence:

```
\documentclass[...]{memoir}
...
\usepackage[...]{hyperref}
\usepackage{memhfixc}
```

nameref commands may be used in a *memoir* document.

Zref defines a proposed replacement for all of the LaTeX reference mechanisms, and among other things provides name-referencing mechanisms:

```
\usepackage[user,titleref]{zref}
...
\section{hello}\zlabel{sec:one}
The section name is: \ztitleref{sec:one}.
```

(One might hope that something of this sort would be the "way of the future", but things move slowly in the LaTeX world: don't hold your breath.)

Each of *titleref*, *byname* and *nameref* defines a reference command with the same name as the package: `\titleref`, `\byname` and `\nameref`. The *nameref* package also defines a command `\byshortnameref`, which uses the optional 'short' title argument to the chapter and section commands. (Although it comes from the same author, *zref doesn't* define a short-name variant.)

`byname.sty`: Distributed with `macros/latex/contrib/smartref` *hyperref.sty*: macros/latex/contrib/hyperref

memoir.cls: macros/latex/contrib/memoir

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

smartref.sty: macros/latex/contrib/smartref

titleref.sty: macros/latex/contrib/titleref

zref.sty: Distributed as part of macros/latex/contrib/oberdiek

252 Referring to labels in other documents

When producing a set of inter-related documents, you'll often want to refer to labels in another document of the set; but LaTeX, of its own accord, doesn't permit this.

So the package *xr* was written: if you say

```
\usepackage{xr}
\externaldocument{volume1}
```

will load all the references from `volume1` into your present document.

But what if the documents both have a section labelled "`introduction`" (likely enough, after all)? The package provides a means to transform all the imported labels, so you don't have to change label names in either document. For example:

```
\usepackage{xr}
\externaldocument[V1-]{volume1}
```

loads the references from `volume1`, but prefixes every one with the string `V1-`. So you would refer to the introduction to volume 1 as:

```
\usepackage{xr}
\externaldocument[V1-]{volume1}
...
... the introduction to volume1 (\ref{V1-introduction})...
```

To have the facilities of *xr* working with *hyperref*, you need *xr-hyper*. For simple hyper-cross-referencing (i.e., to a local PDF file you've just compiled), write:

```
\usepackage{xr-hyper}
\usepackage{hyperref}
\externaldocument[V1-]{volume1}
...
... the \nameref{V1-introduction})...
```

and the name reference will appear as an active link to the "introduction" chapter of `volume1.pdf`.

To link to a PDF document on the Web, for which you happen to have the .aux file, write:

```
\usepackage{xr-hyper}
\usepackage{hyperref}
\externaldocument[V1-]{volume1}[http://mybook.com/volume1.pdf]
...
... the \nameref{V1-introduction})...
```

Heiko Oberdiek's experimental *zref* bundle includes a hyper-crossreferencing mechanism using its *zref-xr* module. Usage is closely modelled on *xr* and *xr-hyper*; a trivial example (from a `comp.text.tex` posting) is

```
\usepackage{zref-xr,zref-user}
\zexternaldocument*{xr02}
...
\zref{foo}
```

The module provides all the facilities of the older packages, and can deal both with "traditional" LaTeX labels and with *zref*'s style of labels.

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

xr-hyper.sty: Distributed with macros/latex/contrib/hyperref

zref bundle: Distributed as part of macros/latex/contrib/oberdiek

P How do I do. . . ?

P.1 Mathematics

253 Proof environment

It was long 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 (apparently) original 'automatic' solution came with Paul Taylor's *QED*.

Nowadays, the *ntheorem* package now solves the problem for LaTeX users: it provides an automatic way of signalling the end of a proof.

The AMSLaTeX 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:

```
\begin{proof}
  text...
  \begin{equation*}
    maths... \tag*{\qedhere}
  \end{equation*}
\end{proof}
```

The `\tag*{\qedhere}` construction may be used in any of AMSLaTeX's numbering environments.

amsthm.sty: Distributed as part of the AMSLaTeX bundle macros/latex/required/amslatex

ntheorem.sty: macros/latex/contrib/ntheorem

254 Theorem bodies printed in a roman font

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, you may want to use it to create remarks, examples, proofs, . . .) then you can use the AMSLaTeX *amsthm* package (which now supersedes the *theorem* package previously recommended in these answers). Alternatively, the following sets up an environment `remark` whose content is in the default roman font.

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

The *ntheorem* package provides control of the fonts used by theorems, directly.

amsthm.sty: Distributed as part of macros/latex/required/amslatex

ntheorem.sty: macros/latex/contrib/ntheorem

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

255 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 below and above the function name, as they are on \lim. If you want your subscripts and superscripts always placed to the right, do:

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

AMSLaTeX (in its *amsopn* package, which is automatically loaded by *amsmath*) provides a command \DeclareMathOperator that takes does the same job as the first definition above. To create our original \diag command, one would say:

```
\DeclareMathOperator{\diag}{diag}
```

\DeclareMathOperator* declares the operator always to have its sub- and superscripts in the "\limits position".

The *amsopn* command \operatorname allows you to introduce *ad hoc* operators into your mathematics, so

```
\[ \operatorname{foo}(bar) \]
```

typesets the same as

```
\DeclareMathOperator{\foo}{foo}
...
\[ \foo(bar) \]
```

As with \DeclareMathOperator there's a starred version \operatorname* for sub- and superscripts in the limits position.

(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: Distributed as part of the AMSLaTeX distribution macros/latex/required/amslatex

256 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 \mathbb{N} \middle| x \mbox{ even} \right\}
```

to specify the set of even natural numbers. The e-TeX system 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 e-TeX built-in command if it finds itself running under e-TeX.

braket.sty: macros/latex/contrib/braket

257 Cancelling terms in maths expressions

A technique used when explaining the behaviour of expressions or equations (often for pedagogical purposes). The *cancel* package provides several variants of cancellation marks ("\", "/" and "X"), and a means of cancelling 'to' a particular value.

Documentation of *cancel* is in the package file.

cancel.sty: macros/latex/contrib/cancel

258 Adjusting maths font sizes

In Plain TeX, when you introduce a new font size you must also declare what size fonts are to be used in mathematics with it. This is done by declaring `\textfont`, `\scriptfont` and `\scriptscriptfont` for the maths families you're using; all such things are described in chapter 17 of the TeXbook and in other books and tutorials that discuss Plain TeX in sufficient detail.

In LaTeX, of course, all this stuff is automated: there is a scheme that, for each (text) font size, determines what maths font sizes are to be used. The scheme first checks a set of "known" text sizes, for each of which maths sizes are declared in advance. If the text size isn't "known", the script- and scriptscriptfont sizes are calculated as fixed ratios of the tex font size. (The values used are `\defaultscriptratio`=0.7, and `\defaultscriptscriptratio`=0.5.)

The fixed-ratio formula is capable of producing inconvenient results (particularly if you are using fonts which LaTeX believes are only available in a fixed set of sizes). You may also want to replace LaTeX's ideas altogether, for example by setting maths noticeably larger or smaller than its surrounding text. For this purpose, the LaTeX command `\DeclareMathSizes{`⟨*tfs*⟩`}{`⟨*ts*⟩`}{`⟨*ss*⟩`}{`⟨*sss*⟩`}` may be used (this is the same command that LaTeX itself uses to define its own set of sizes). This establishes (or re-establishes) the maths font sizes to be used when the surrounding text font size is ⟨*tfs*⟩; (⟨*ts*⟩ being the size used for `\textfont`, ⟨*ss*⟩ for `\scriptfont` and ⟨*sss*⟩ for `\scriptscriptfont`).

For example, you might want to use a font with a smaller body height than Computer Modern, but still prefer CM math to any of the alternatives. In this case, you might use:

```
\DeclareMathSizes{10}{9}{7}{5}
```

to get 9pt maths when the surrounding body text is (nominal) 10pt.

`\DeclareMathSizes` may only be used in the preamble of the document: only one association is available for each text font size for the whole document. The default settings are specified in `fontdef.dtx` in the latex distribution, and are compiled into `fontmath.ltx`; the arguments to the command are just numbers ('`pt`' is assumed), but some of them are written using LaTeX abbreviations for standard font sizes. Beware simply copying (parts of) the LaTeX definitions — since they contain those internal abbreviations, they need to be treated as internal commands.

fontdef.dtx: `macros/latex/base/fontdef.dtx`

fontmath.ltx: `macros/latex/unpacked/fontmath.ltx`

259 Ellipses

Ellipses are commonly required, and LaTeX natively supplies a fair range (`\dots`, `\cdots`, `\vdots` and `\ddots`). By using the *graphics* package, one can change the slope of the `\ddots` command, as in

```
$ ... \reflectbox{$\ddots$} ... $
```

While this works, it is not a recommended way of achieving the desired result (see below). Moreover, LaTeX's range is not adequate to everyone's requirements, and at least three packages provide extensions to the set.

The AMSLaTeX bundle provides a range of "semantically-named" ellipses, for use in different situations: `\dotsb` for use between pairs of binary operators, `\dotsc` for use between pairs of commas, and so on.

The *yhmath* package defines an `\adots` command, which is the analogue of `\ddots`, sloping forwards rather than backwards. The *yhmath* package comes with a rather interesting font that extends the standard *cmex*; details are in the documentation. The disadvantage of this setup is, that although `\adots` is merely a macro, the package tries to load its own font and produces a "missing font" substitution warning message if you haven't installed the font.

The *mathdots* package (besides fixing up the behaviour of (La)TeX `\ddots` and `\vdots` when the font size changes) provides an "inverse diagonal" ellipsis `\iddots` (doing the same job as *yhmath*'s `\adots`, but better).

Documentation of *yhmath* appears, processed, in the distribution (thus saving you the bother of installing the package before being able to read the documentation). Documentation of *mathdots* appears at the end the package file itself.

amslatex: macros/latex/required/amslatex

graphics.sty: Part of the macros/latex/required/graphics bundle

mathdots.sty: macros/generic/mathdots

yhmath (fonts): fonts/yhmath

yhmath (macros): macros/latex/contrib/yhmath

260 Sub- and superscript positioning for operators

The commonest hand-written style for expressions is to place the limit expressions on operators such as `\sum` and `\int` physically above and below the operator. In (La)TeX, we write these limit expressions using sub- and superscripts applied to the operator, but they don't always appear in the "handwritten" way in TeX's output.

The reason is, that when an expression appears in non-display maths, in running text (and is therefore in TeX `\textstyle`), placing the limits thus could lead to ragged line spacing (and hence difficult-to-read text). It is therefore common (in `\textstyle`) to place the limits as one would sub- and superscripts of variables.

This is not universally satisfactory, so the primitive `\limits` is provided:

```
$\sum\limits_{n=1}^{m} ...$
```

which will place the limits right above and below the symbol (and be blowed to the typography...).

Contrariwise, you may wish to change the arrangement of the limits when in `\displaystyle`. For this purpose, there's a corresponding `\nolimits`:

```
\[\sum\nolimits_{n=1}^{m} ...\]
```

which will place the limits as they would be in `\textstyle`.

Alternatively, one can manipulate the `\textstyle`/`\displaystyle` state of the mathematics. To get "`\limits` placement" in inline maths,

```
$\displaystyle\sum_{n=1}^{m} ...$
```

and for "`\nolimits` placement" in display maths, `\nolimits`:

```
\[\textstyle\sum_{n=1}^{m} ...\]
```

will serve. Either of these forms may have effects other than on the operator you're considering, but there are still those who prefer this formulation.

Remember, if you're declaring a special operator of your own, the AMSLaTeX functions (that you ought to be using) allow you to choose how limits are displayed, at definition time.

(Note that the macro `\int` normally has `\nolimits` built in to its definition. There is an example in the TeXbook to show how odd `\int\limits` looks when typeset.)

261 Text inside maths

When we type maths in (La)TeX, the letters from which we make up ordinary text assume a special significance: they all become single-letter variable names. The letters appear in italics, but it's not the same sort of italics that you see when you're typing ordinary text: a run of maths letters (for example "$here$") looks oddly "lumpy" when compared with the word written in italic text. The difference is that the italic text is kerned to make the letters fit well together, whereas the maths is set to look as if you're multiplying h by e by r by e. The other way things are odd in TeX maths typing is that spaces are ignored: at best we can write single words in this oddly lumpy font.

So, if we're going to have good-looking text in amongst maths we're writing, we have to take special precautions. If you're using LaTeX, the following should help.

The simplest is to use `\mbox` or `\textrm`:

```
$e = mc^2 \mbox{here we go again}$
```

The problem is that, with either, the size of the text remains firmly at the surrounding text size, so that

```
$z = a_{\mbox{other end}}$
```

can look quite painfully wrong.

The other simple technique, `\textrm`, is no more promising:

```
$z = a_{\textrm{other end}}$
```

does the same as `\mbox`, by default.

(The maths-mode instance of your roman font (`\mathrm`) gets the size right, but since it's intended for use in maths, its spaces get ignored — use `\mathrm` for upright roman alphabetic variable names, but not otherwise.)

You can correct these problems with size selectors in the text, as:

```
$z = a_{\mbox{\scriptsize other end}}$
```

which works if your surrounding text is at default document size, but gives you the wrong size otherwise.

The `\mbox` short cut is (just about) OK for "occasional" use, but serious mathematics calls for a technique that relieves the typist of the sort of thought required. As usual, the AMSLaTeX system provides what's necessary — the `\text` command. (The command is actually provided by the *amstext* package, but the "global" *amsmath* package loads it.) Thus anyone using AMSLaTeX proper has the command available, so even this author can write:

```
\usepackage{amsmath}
...
$z = a_{\text{other end}}$
```

and the text will be at the right size, and in the same font as surrounding text. (The *amstext* package also puts `\textrm` to rights — but `\text` is easier to type than `\textrm`!)

AMSLaTeX also makes provision for interpolated comments in the middle of one of its multi-line display structures, through the `\intertext` command. For example:

```
\begin{align}
  A_1&=N_0(\lambda;\Omega')-\phi(\lambda;\Omega'),\\
  A_2&=\phi(\lambda;\Omega')-\phi(\lambda;\Omega),\\
  \intertext{and} A_3&=\mathcal{N}(\lambda;\omega).
\end{align}
```

places the text "and" on a separate line before the last line of the display. If the interjected text is short, or the equations themselves are light-weight, you may find that `\intertext` leaves too much space. Slightly more modest is the `\shortintertext` command from the *mathtools* package:

```
\begin{align}
  a =& b
  \shortintertext{or}
  c =& b
\end{align}
```

To have the text on the same line as the second equation, one can use the `flalign` environment (from *amsmath*) with lots of dummy equations (represented by the double & signs):

```
\begin{flalign}
            && a =& b && \\
  \text{or} && c =& b &&
\end{flalign}
```

amsmath.sty: Distributed as part of AMSLaTeX macros/latex/required/amslatex

amstext.sty: Distributed as part of AMSLaTeX macros/latex/required/amslatex

mathtools.sty: Distributed as part of the *mh* bundle macros/latex/contrib/mh

262 Re-using an equation

To repeat an existing equation, one wants not only to have the same mathematics in it, one also wants to re-use the original label it had. The *amsmath* package comes to our help, here:

```
\usepackage{amsmath}
...
\begin{equation}
  a=b
  \label{eq1}
\end{equation}
...
Remember that
\begin{equation}
  a=b
  \tag{\ref{eq1}}
\end{equation}
```

Here, the second instance of $a = b$ will be typeset with a copy, made by the `\tag` command, of the label of the first instance.

amsmath.sty: Distributed as part of AMSLaTeX macros/latex/required/amslatex

263 Line-breaking in in-line maths

TeX, by default, allows you to split a mathematical expression at the end of the line; it allows breaks at relational operators (like "=", "<", etc.) and at binary operators (like "+", "-", etc.). In the case of large expressions, this can sometimes be a life-saver.

However, in the case of simple expressions like $a = b + c$, a break can be really disturbing to the reader, and one would like to avoid it.

Fortunately, these breaks are controllable: there are "penalties" associated with each type of operator: the penalty says how undesirable a break at each point is. Default values are:

```
\relpenalty   = 500
\binoppenalty = 700
```

You make the break progressively less attractive by increasing these values. You can actually forbid all breaks, everywhere, by:

```
\relpenalty   = 10000
\binoppenalty = 10000
```

If you want just to prevent breaks in a single expression, write:

```
{%
  \relpenalty   = 10000
  \binoppenalty = 10000
  $a=b+c$
}
```

and the original values will remain undisturbed outside the braces. This is tedious: there is often value in an alternative approach, in which you say which parts of the expression may not break whatever happens, and fortunately this is surprisingly easy. Suppose we want to defer a break until after the equality, we could write:

```
${a+b+c+d} = z+y+x+w$
```

The braces say "treat this subformula as one atom" and (in TeX at least) atoms don't get split: not a `\binoppenalty` change in sight.

264 Numbers for referenced equations only

There are those who consider that papers look untidy with numbers on every equation; there is also a school of thought that claims that there should be numbers everywhere, in case some reader wants to make reference an equation to which the author made no cross-reference.

If you hold to the "only cross-referenced" school of thought, you can (using the `\nonumber` command on the relevant equations, or by using the AMSLaTeX unnumbered environments such as `equation*`) mark those of your equations to which you make no reference. In a long or complex paper, this procedure could well become deeply tedious.

Fortunately, help is at hand: the *mh* bundle's *mathtools* package offers a '`showonlyrefs`' switch through its `\mathtoolsset` command; when that's in operation, only those equations to which you make reference will be numbered in the final output. See the package's documentation for details of how to make references when the switch is in effect.

`mathtools.sty`: Distributed as part of `macros/latex/contrib/mh`

265 Even subscript height

Other things being equal, TeX will aim to position subscripts and superscripts in places that "look good". Unfortunately, it only does this for the sub- and superscripts of each atom at a time, so if you have

```
$ X^{1}_{2} X_{2} $
```

the second subscript will appear higher, since the first has moved down to avoid the superscript; the effect can be noticeably distracting:

$X^{1}_{2} X_{2}$

You can avoid the problem, for a single instance, by

```
$ X^{1}_{2} X^{}_{2} $
```

here, the dummy superscript has the requisite "pushing down" effect:

$X^{1}_{2} X^{}_{2}$

While this technique does what is necessary, it is tedious and potentially error-prone. So, for more than one or two equations in a document, the LaTeX user is advised to use the *subdepth* package, which forces the lower position for all subscripts.

`subdepth.sty`: `macros/latex/contrib/subdepth`

P.2 Lists

266 Fancy enumeration lists

The *enumerate* package allows you to control the display of the enumeration counter. The package adds an optional parameter to the `enumerate` environment, which is used to specify the layout of the labels. The layout parameter contains an enumeration type ('`1`' for arabic numerals, '`a`' or '`A`' for alphabetic enumeration, and '`i`' or '`I`' for Roman numerals), and things to act as decoration of the enumeration. So, for example

```
\usepackage{enumerate}
...
\begin{enumerate}[(a)]
\item ...  ...
\end{enumerate}
```

starts a list whose labels run (a), (b), (c), . . . ; while

```
\usepackage{enumerate}
...
\begin{enumerate}[I/]
\item ...  ...
\end{enumerate}
```

starts a list whose labels run I/, II/, III/, . . .

The *paralist* package, whose primary purpose is compaction of lists, provides the same facilities for its `enumerate`-like environments.

If you need non-stereotyped designs, the *enumitem* package gives you most of the flexibility you might want to design your own. The silly roman example above could be achieved by:

```
\usepackage{enumitem}
...
\begin{enumerate}[label=\Roman{*}/]
\item ...  ...
\end{enumerate}
```

Note that the '`*`' in the key value stands for the list counter at this level. You can also manipulate the format of references to list item labels:

```
\usepackage{enumitem}
...
\begin{enumerate}[label=\Roman{*}/, ref=(\roman{*})]
\item ...  ...
\end{enumerate}
```

to make references to the list items format appear as (i), (ii), (iii), etc.

The *memoir* class includes functions that match those in the *enumerate* package, and has similar functionality for `itemize` lists.

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

`enumitem.sty`: `macros/latex/contrib/enumitem`

`memoir.cls`: `macros/latex/contrib/memoir`

`paralist.sty`: `macros/latex/contrib/paralist`

267 How to adjust list spacing

Lamport's book 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. This happens because each list executes a command `\@list`⟨*depth*⟩ (the depth appearing as a lower-case roman numeral); what's more, the top-level `\@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. The *memoir* class wasn't designed for more compact lists *per se*, but offers the user control over the list spacing.

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{\parskip}{0pt}}%
  {\end{itemize}}
```

The *paralist* package provides several approaches to list compaction:

- its `asparaenum` environment formats each item as if it were a paragraph introduced by the enumeration label (which saves space if the item texts are long);
- its `compactenum` environment is the same sort of compact list as is provided in *expdlist* and *mdwlist*; and
- its `inparaenum` environment produces a list "in the paragraph", i.e., with no line break between items, which is a great space-saver if the list item texts are short.

The package will manipulate its `enumerate` environment labels just like the *enumerate* package does.

Paralist also provides `itemize` equivalents (`asparaitem`, etc.), and `description` equivalents (`asparadesc`, etc.).

The *multenum* package offers a more regular form of *paralist*'s `inparaenum`; you define a notional grid on which list entries are to appear, and list items will always appear at positions on that grid. The effect is somewhat like that of the 'tab' keys on traditional typewriters; the package was designed for example sheets, or lists of answers in the appendices of a book.

The ultimate in compaction (of every sort) is offered by the *savetrees* package; compaction of lists is included. The package's prime purpose is to save space at every touch and turn: don't use it if you're under any design constraint whatever!

The *expdlist*, *mdwlist* and *paralist* packages all offer other facilities for list configuration: you should probably not try the "do-it-yourself" approaches outlined below if you need one of the packages for some other list configuration purpose.

For ultimate flexibility (including manipulation of `\topsep`), the *enumitem* package permits adjustment of list parameters using a "*key*=⟨*value*⟩" format. For example, one might write

```
\usepackage{enumitem}
...
\begin{enumerate}[topsep=0pt, partopsep=0pt]
\item ...
\item ...
\end{enumerate}
```

to suppress all spacing above and below your list, or

```
\usepackage{enumitem}
...
\begin{enumerate}[itemsep=2pt,parsep=2pt]
\item ...
\item ...
\end{enumerate}
```

to set spacing between items and between paragraphs within items. *Enumitem* also permits manipulation of the label format in a more "basic" (and therefore more flexible) manner than the *enumerate* package does.

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

enumitem.sty: `macros/latex/contrib/enumitem`

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

memoir.cls: `macros/latex/contrib/memoir`

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

multenum.sty: `macros/latex/contrib/multenum`

paralist.sty: `macros/latex/contrib/paralist`

savetrees.sty: `macros/latex/contrib/savetrees`

268 Interrupting enumerated lists

It's often convenient to have commentary text, 'outside' the list, between successive entries of a list. In the case of `itemize` lists this is no problem, since there's never anything to distinguish successive items, while in the case of `description` lists, the item labels are under the user's control so there's no automatic issue of continuity.

For `enumerate` lists, the labels are generated automatically, and are context-sensitive, so the context (in this case, the state of the enumeration counter) needs to be preserved.

The belt-and-braces approach is to remember the state of the enumeration in your own counter variable, and then restore it when restarting enumerate:

```
\newcounter{saveenum}
 ...
\begin{enumerate}
  ...
  \setcounter{saveenum}{\value{enumi}}
\end{enumerate}
<Commentary text>
\begin{enumerate}
  \setcounter{enumi}{\value{saveenum}}
  ...
\end{enumerate}
```

This is reasonable, in small doses... Problems (apart from sheer verbosity) are getting the level right ("should I use counter `enumi`, `enumii`, ...") and remembering not to nest the interruptions (i.e., not to have a separate list, that is itself interrupted) in the "commentary text").

The *mdwlist* package defines commands `\suspend` and `\resume` that simplify the process:

```
\begin{enumerate}
  ...
\suspend{enumerate}
<Commentary text>
\resume{enumerate}
  ...
\end{enumerate}
```

The package allows an optional name (as in `\suspend[id]{enumerate}`) to allow you to identify a particular suspension, and hence provide a handle for manipulating nested suspensions.

If you're suspending a fancy-enumeration list, you need to re-supply the optional "item label layout" parameters required by the *enumerate* package when resuming the list, whether by the belt-and-braces approach, or by the *mdwlist* `\resume{enumerate}` technique. The task is a little tedious in the *mdwlist* case, since the optional argument has to be encapsulated, whole, inside an optional argument to `\resume`, which requires use of extra braces:

```
\begin{enumerate}[\textbf{Item} i]
  ...
\suspend{enumerate}
<comment>
\resume{enumerate}[{[\textbf{Item} i]}]
...
\end{enumerate}
```

The *enumitem* package, in its most recent release, will also allow you to resume lists:

```
\begin{enumerate}
...
```

```
\end{enumerate}
<comment>
\begin{enumerate}[resume]
...
\end{enumerate}
```

which feels just as “natural” as the *mdwtools* facility, and has the advantage of playing well with the other excellent facilities of *enumitem*.

Expdlist has a neat way of providing for comments, with its `\listpart` command. The command’s argument becomes a comment between items of the list:

```
\begin{enumerate}
\item item 1
\item item 2
  \listpart{interpolated comment}
\item item 3
\end{enumerate}
```

This, you will realise, means it doesn’t even have to think about suspending or resuming the list, and of course it works equally well in any of the list environments (thought it’s not actually *necessary* for any but `enumerate`).

Enumitem also allows multi-level suspension and resumption of lists:

```
\begin{enumerate}
\item outer item 1
\end{enumerate}
<comment>
\begin{enumerate}[resume]
\item outer item 2
% nested enumerate
\begin{enumerate}
\item inner item 1
\end{enumerate}
<nested comment>
% resume nested enumerate
\begin{enumerate}[resume]
\item inner item 2
\end{enumerate}
\item outer item 3
% end outer enumerate
\end{enumerate}
```

However, the ‘nested comment’ interpolated in the nested enumeration appears as if it were a second paragraph to “outer item 2”, which is hardly satisfactory.

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

`enumitem.sty`: `macros/latex/contrib/enumitem`

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

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

P.3 Tables, figures and diagrams

269 The design of tables

In recent years, several authors have argued that the examples, set out by Lamport in his LaTeX manual, have cramped authors’ style and have led to extremely poor table design. It is in fact difficult even to work out what many of the examples in Lamport’s book “mean”.

The criticism focuses on the excessive use of rules (both horizontal and vertical) and on the poor vertical spacing that Lamport’s macros offer.

The problem of vertical spacing is plain for all to see, and is addressed in several packages — see “spacing of lines in tables”.

The argument about rules is presented in the excellent essay that prefaces the documentation of Simon Fear’s *booktabs* package, which (of course) implements Fear’s scheme for ‘comfortable’ rules. (The same rule commands are implemented in the *memoir* class.)

Lamport’s LaTeX was also inflexibly wrong in “insisting” that captions should come at the bottom of a table. Since a table may extend over several pages, traditional typography places the caption at the top of a table float. The `\caption` command will get its position wrong (by `10pt`) if you simply write:

```
\begin{table}
  \caption{Example table}
  \begin{tabular}{...}
    ...
  \end{tabular}
\end{table}
```

The *topcapt* package solves this problem:

```
\usepackage{topcapt}
...
\begin{table}
  \topcaption{Example table}
  \begin{tabular}{...}
    ...
  \end{tabular}
\end{table}
```

The *KOMA-script* classes provide a similar command `\captionabove`; they also have a class option `tablecaptionabove` which arranges that `\caption` *means* `\captionabove`, in table environments. The *caption* package may be loaded with an option that has the same effect:

```
\usepackage[tableposition=top]{caption}
```

or the effect may be established after the package has been loaded:

```
\usepackage{caption}
\captionsetup[table]{position=above}
```

(Note that the two “position” options are different: actually, “above” and “top” in these contexts mean the same thing.)

Doing the job yourself is pretty easy: *topcapt* switches the values of the LaTeX2e parameters `\abovecaptionskip` (default value `10pt`) and `\belowcaptionskip` (default value `0pt`), so:

```
\begin{table}
  \setlength{\abovecaptionskip}{0pt}
  \setlength{\belowcaptionskip}{10pt}
  \caption{Example table}
  \begin{tabular}{...}
    ...
  \end{tabular}
\end{table}
```

does the job (if the length values are right; the package and classes are more careful!).

booktabs.sty: macros/latex/contrib/booktabs

caption.sty: macros/latex/contrib/caption

KOMA script bundle: macros/latex/contrib/koma-script

memoir.cls: macros/latex/contrib/memoir

topcapt.sty: macros/latex/contrib/misc/topcapt.sty

270 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` column specification, `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 *tabulary* package (by the same author) provides a way of "balancing" the space taken by the columns of a table. The package defines column specifications `C`, `L`, `R` and `J`, giving, respectively, centred, left, right and fully-justified versions of space-sharing columns. The package examines how long each column would be "naturally" (i.e., on a piece of paper of unlimited width), and allocates space to each column accordingly. There are "sanity checks" so that really large entries don't cause everything else to collapse into nothingness (there's a "maximum width" that any column can exert), and so that tiny entries can't get smaller than a specified minimum. Of course, all this work means that the package has to typeset each row several times, so things that leave "side-effects" (for example, a counter used to produce a row-number somewhere) are inevitably unreliable, and should not even be tried.

The *ltxtable* package combines the features of the *longtable* and *tabularx* packages. It's important to read the documentation, since usage is distinctly odd; the distribution contains no more than a file `ltxtable.tex`, which you should process using LaTeX. Processing will give you a `.sty` file as well as the `.dvi` or `.pdf` output containing the documentation.

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

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

`tabulary.sty`: `macros/latex/contrib/tabulary`

271 Variable-width columns in tables

This is a slightly different take on the problem addressed in "fixed-width tables" — here we have a column whose size we can't absolutely predict when we design the document.

While the basic techniques (the *tabularx*, *tabulary* and *ltxtable* packages) are the same for this problem as for the fixed-width *table* problem, there's one extra tool that we can call to our aid, which may be preferable in some situations.

Suppose we have data in one column which we read from an external source, and the source itself isn't entirely predictable. The data in the column may end up pretty narrow in every row of the table, or it may be wide enough that the table would run over the edge of the page; however, we don't want to make the column as wide as possible "just in case", by defining a fixed size for the table. We would like the column to be as small as possible, but have the possibility to spread to a maximum width and (if even that width is exceeded) turn into a `p`-style column.

The *varwidth* package, discussed in "automatic sizing of minipages", provides a solution. If you load it together with the LaTeX "required" *array* package, i.e.:

```
\usepackage{array}
\usepackage{varwidth}
```

varwidth defines a new column-type "`V`", which you can use as follows:

```
\begin{tabular}{l V{3.5cm} r}
  foo & blah      & bar \\
  foo & blah blah & bar \\
\end{tabular}
```

when the second column ends up less than 3.5cm wide; or you can use it as follows:

```
\begin{tabular}{l V{3.5cm} r}
  foo & blah      & bar \\
  foo & blah blah & bar \\
  foo & blah blah blah blah blah blah
                  & bar \\
\end{tabular}
```

where the second column will end up noticeably wider, and will wrap to a second line in the third row.

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

varwidth.sty: macros/latex/contrib/varwidth

272 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`. The *bigstrut* package defines a strut command that you can use for this purpose: `\bigstrut` on its own opens up both above and below the current line; `\bigstrut[t]` opens up above the line, `\bigstrut [b]` opens up below the line.

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 *makecell* package provides a command `\gape` that may be used to apply strut expansion for a single cell of a table:

```
\begin{tabular}{lll}
  ... & \gape{cell contents} & ... \\
  ...
\end{tabular}
```

The package's similar `\Gape` command provides the same function, but with optional arguments that allow you to adjust the top and bottom adjustment.

To adjust every cell in whole tables, the `\setcellgapes{⟨value⟩}` sets the adjustment value (an optional argument of "`t`" or "`b`" restricts adjustment to the top or bottom of each cell, respectively). Having issued `\setcellgapes`, the command `\makegapedcells` switches cell expansion on, and `\nomakegapedcells` switches it off again.

The *cellspace* package does a (possibly inferior) job by defining a new table/array column type "S", which you apply to each column specification. So, for example,

```
\cmdinvoke{begin}{tabular}{l l l p{3cm}}
```

would become

```
\cmdinvoke{begin}{tabular}{Sl Sl Sl Sp{3cm}}
```

and so on. This technique shows promise of not interfering so much with other packages, but this author has heard of no reports from the field.

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. The most recent release of *booktabs* sports compatibility with packages such as *longtable*.

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

bigstrut.sty: `macros/latex/contrib/multirow`

booktabs.sty: `macros/latex/contrib/booktabs`

cellspace.sty: `macros/latex/contrib/cellspace`

makecell.sty: `macros/latex/contrib/makecell`

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

273 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 slightly tedious to set up, but which doesn't force the whole alignment onto a single page.

The *longtable* package builds the whole table (in chunks), in a first pass, and then uses information it has written to the `.aux` file during later passes to get the setting "right" (the package ordinarily manages to set tables in just two passes). Since the package has overview of the whole table at the time it's doing "final" setting, the table is set "uniformly" over its entire length, with columns matching on consecutive pages. *longtable* has a reputation for failing to interwork with other packages, but it does work with *colortbl*, and its author has provided the *ltxtable* package to provide (most of) the facilities of *tabularx* (see fixed-width tables) 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.

A seeming alternative to *ltxtable* is *ltablex*; but it is outdated and not fully functional. Its worst problem is its strictly limited memory capacity (*longtable* is not so limited, at the cost of much complication in its code); *ltablex* can only deal with relatively small tables, it doesn't seem likely that support is available; but its user interface is much simpler than *ltxtable*, so if its restrictions aren't a problem for you, it may be worth a try.

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).

The *stabular* package provides a simple-to-use "extension to `tabular`" that allows it to typeset tables that run over the end of a page; it also has usability extensions, but doesn't have the head- and footline capabilities of the major packages.

Documentation of *ltablex* is to be found in the package file.

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

`ltablex.sty`: `macros/latex/contrib/ltablex/ltablex.sty`

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

`stabular.sty`: Distributed as part of `macros/latex/contrib/sttools`

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

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

274 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. With a `p` cell that looks like:

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

one is liable to encounter errors that complain about a "misplaced `\noalign`" or "extra alignment tab", 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`, `\raggedright` and `\raggedleft` 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.

In the old days, the actual value of `\\` that the `tabular` environment uses 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 that's a rather verbose way of doing things).

The *array* package provides a command `\arraybackslash` which restores `\\` to its correct (within table) meaning; the command may be used in *array*'s "field format" preamble specifications:

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

The `\tabularnewline` and `\arraybackslash` commands are (somewhat) modern additions to LaTeX and the *array* package, respectively. In the unlikely event that neither is available, the user may try the (old) solution which preserves the meaning of `\\`:

```
\newcommand\PBS[1]{\let\temp=\\%
  #1%
  \let\\=\temp
}
```

which one uses within a table as:

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

or in the preamble as:

```
\begin{tabular}{...>{\PBS\centering}p{5cm}}
```

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

275 The thickness of rules in LaTeX tables

The rules in a LaTeX table are by default `0.4pt` thick; this is in fact a default built in at the lowest level, and applies to all rules (including those separating blocks of running text).

Sometimes, however, we look at a table and find we want the rules to stand out — perhaps to separate the text from the rest of the body text, or to make the sections of the table stand out from one another. However, a quick review of any LaTeX manual will reveal no technique for making any one rule stand out, and a little experimentation shows that it is indeed pretty difficult to prevent a change "bleeding" out to affect other rules in the same table.

If you look at what we have to say on the design of tables, elsewhere among these FAQs, and you may sense that the design of LaTeX simply skipped the issues surrounding table design: *that's* presumably why there's no facilities to help you.

Specifically, the length `\arrayrulewidth` affects the thickness of the rules (both horizontal and vertical) within both `tabular` and `array` environments. If you change from the default (see above) only as far as

```
\setlength{\arrayrulewidth}{1pt}
```

the change is remarkably striking. However, really quite subtle user level programming proves incapable of changing just *one* rule: it's necessary to delve into the (rather tricky) code of `\hline` and `\cline` themselves.

Fortunately, this job has already been done for the community: the *booktabs* package defines three different classes of rule (`\toprule`, `\midrule` and `\bottomrule`), and the package documentation offers hints on how to use them. You are *strongly* advised to read the documentation pretty carefully.

The *memoir* class includes the *booktabs* package, and repeats the documentation in its compendious manual.

Note that none of the above mentions the issue of the weight of vertical rules (except in passing). For the reasons, see the documentation of the *booktabs* package (again); vertical rules in tables are in any case even more trickily coded than are horizontal rules, and if their lack of configurability makes them still less attractive, so much the better for the design of your document.

booktabs.sty: macros/latex/contrib/booktabs

memoir.cls: macros/latex/contrib/memoir

276 Flowing text around figures

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 LaTeX2e) 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,...}[overhang]{width}
  figure, caption, etc.
\end{wrapfigure}
```

The syntax of the `wraptable` environment is similar.
The *`height`* may be omitted, in which case it will be calculated from the size of the figure; the package will use the greater of the specified and the actual width. The `{l,r,`*`etc.`*`}` parameter may also be specified as `i`*(nside)* or `o`*(utside)* for two-sided documents, and uppercase may 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 for inserting a picture at the start of a paragraph 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 a summary in English.

All of the above deal insertions at one or other margin; they are able to take advantage of the TeX `\parshape` primitive that allows you to adjust the margins of the text of a paragraph, by line (Knuth provides an example of such use, with text typeset in a circle, half-overlapping the margin, in chapter 14 of the TeXbook). To place insertions in the middle of a paragraph requires effort of an entirely different sort; the *cutwin* package does this for you. It requires a set of “part line widths” (two per line), and typesets the “cutout” section of the paragraph line by line. The examples in the package documentation look enticing.

Plain TeX users only have one option: *figflow* (which doesn’t work in LaTeX). *Figflow* only offers flowed figures at the start of the paragraph, but it seems perfectly functional. Syntax is

`\figflow{⟨width⟩}{⟨height⟩}{⟨figure⟩}`

(the user is responsible for having the dimensions correct, and for ensuring the figure fits on the page).

`cutwin.sty`: `macros/latex/contrib/cutwin`

`figflow.tex`: `macros/plain/contrib/figflow`

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

`picins.sty`: `macros/latex209/contrib/picins`

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

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

277 Diagonal separation in corner cells of tables

You want to label both the top or bottom row and the left- or rightmost column, somewhere at the corner of the table where the row and column meet. A simple way to achieve the result is to construct the table with an arrangement of rules (and possibly `\multicolumn` entries), to look like:

```
-----------------
x  y
```

```
      --------------
      1  2  3  4  5
   -----------------
   1
   2
   3
   4
   5
   -----------------
```

However, this doesn't satisfy everyone: many want the labelling in a single cell at the top left of the table. It sounds a simple enough requirement, yet it calls for some slightly tricky LaTeX coding. The *diagbox* package does this job for you: it defines a command `\diagbox` whose two arguments provide the texts to be used; an optional argument may be used for fine tuning of the result. It draws a picture with the two labels on either side of a slanting line; the command (and hence the picture) may be placed in the corner cell, where the labelled row and column meet.

The *diagbox* package supersedes *slashbox*; the older package's commands `\slashbox` and `\backslashbox` are provided in a compatible way in the newer package, to ease transition.

diagbox.sty: macros/latex/contrib/diagbox

slashbox.sty: macros/latex/contrib/slashbox

278 How to change a whole row of a table

Each cell of a table is set in a box, so that a change of font style (or whatever) only lasts to the end of the cell. If one has a many-celled table, or a long one which needs lots of rows emphasising, putting a font style change command in every cell will be impossibly tedious.

With the *array* package, you can define column modifiers which will change the font style for a whole *column*. However, with a bit of subtlety, one can make such modifiers affect rows rather than columns. So, we set things up by:

```
\usepackage{array}
\newcolumntype{$}{>{\global\let\currentrowstyle\relax}}
\newcolumntype{^}{>{\currentrowstyle}}
\newcommand{\rowstyle}[1]{\gdef\currentrowstyle{#1}%
  #1\ignorespaces
}
```

Now, we put '$' before the first column specifier; and we put '^' before the modifiers of subsequent ones. We then use `\rowstyle` at the start of each row we want to modify:

```
\begin{tabular}{|$l|^l|^l|}   \hline
  \rowstyle{\bfseries}
  Heading & Big and & Bold \\ \hline
  Meek & mild & entry      \\
  Meek & mild & entry      \\
  \rowstyle{\itshape}
  Strange & and & italic   \\
  Meek & mild & entry      \\ \hline
\end{tabular}
```

The *array* package works with several other `tabular`-like environments from other packages (for example `longtable`), but unfortunately this trick won't always work.

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

279 Merging cells in a column of a table

It's easy to come up with a table design that requires a cell that spans several rows. An example is something where the left-most column labels the rest of the table; this can be done (in simple cases) by using diagonal separation in corner cells, but that technique rather strictly limits what can be used as the content of the cell.

The *multirow* package enables you to construct such multi-row cells, in a very simple manner. For the simplest possible use, one might write:

```
\begin{tabular}{|c|c|}
\hline
\multirow{4}{*}{Common g text}
      & Column g2a\\
      & Column g2b \\
      & Column g2c \\
      & Column g2d \\
\hline
\end{tabular}
```

and *multirow* will position "Common g text" at the vertical centre of the space defined by the other rows. Note that the rows that don't contain the "multi-row" specification must have empty cells where the multi-row is going to appear.

The "*" may be replaced by a column width specification. In this case, the argument may contain forced line-breaks:

```
\begin{tabular}{|c|c|}
\hline
\multirow{4}{25mm}{Common\\g text}
      & Column g2a\\
      & Column g2b \\
      & Column g2c \\
      & Column g2d \\
\hline
\end{tabular}
```

A similar effect (with the possibility of a little more sophistication) may be achieved by putting a smaller table that lines up the text into a *-declared `\multirow`.

The `\multirow` command may also used to write labels vertically down one or other side of a table (with the help of the *graphics* or *graphicx* package, which provide the `\rotatebox` command):

```
\begin{tabular}{|l|l|}
\hline
\multirow{4}{*}{\rotatebox{90}{hi there}}
      & Column g2a\\
      & Column g2b \\
      & Column g2c \\
      & Column g2d \\
\hline
\end{tabular}
```

(which gives text going upwards; use angle -90 for text going downwards, of course).

To make a `\multicolumn` multi-row "cell" in a table, you have to enclose a `\multirow` inside a `\multicolumn` — the other way around does not work, so:

```
\begin{tabular}{|c|c|c|}\hline
\multicolumn{2}{|c|}{\multirow{2}{*}{combined cells}}
     &top right\\ \cline{3-3}
\multicolumn{2}{|c|}{}
     &middle right\\ \hline
bottom left
```

```
        &bottom center
        &bottom right\\ \hline
    \end{tabular}
```

Multirow is set up to interact with the *bigstrut* package (which is also discussed in the answer to spacing lines in tables). You use an optional argument to the `\multirow` command to say how many of the rows in the multi-row have been opened up with `\bigstrut`.

The documentation of both *multirow* and *bigstrut* is to be found, as comments, in the package files themselves.

bigstrut.sty: macros/latex/contrib/multirow

multirow.sty: macros/latex/contrib/multirow

P.4 Floating tables, figures, etc.

280 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. Moving tables and figures 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
```

281 Extra vertical space in floats

A common complaint is that extra vertical space has crept into `figure` or `table` floating environments. More common still are users who post code that introduces this extra space, and *haven't noticed the problem*!

The trouble arises from the fact that the `center` environment (and its siblings `flushleft` and `flushright`) are actually based on LaTeX's list-handling code; and lists always separate themselves from the material around them. Meanwhile, there are parameters provided to adjust the spacing between floating environments and their surroundings; so if we have:

```
\begin{figure}
 \begin{center}
   \includegraphics{...}
   \caption{...}
 \end{center}
\end{figure}
```

or worse still:

```
\begin{figure}
 \begin{center}
   \includegraphics{...}
 \end{center}
 \caption{...}
\end{figure}
```

unwarranted vertical space is going to appear.

The solution is to let the float and the objects in it position themselves, and to use "generic" layout commands rather than their list-based encapsulations.

```
\begin{figure}
  \centering
  \includegraphics{...}
  \caption{...}
\end{figure}
```

(which even involves less typing).

This alternative code will work with any LaTeX package. It will not work with obsolete (pre-LaTeX2e) packages such as *psfig* or *epsf* — see graphics inclusion for discussion of the genesis of `\includegraphics`.

282 Centring a very wide figure or table

The normal means of centring a figure or table object is to include `\centering` at the top of the float. This doesn't help if the object is wider than `\textwidth` — the object starts at the left margin and juts out into the right margin (which is actually doubly unsatisfactory, since as well as looking bad, the float won't be placed until the next `\clearpage` or the like.)

You can avoid the problem by rescaling the figure or table to fit, but this is often not satisfactory, for several reasons.

Otherwise, if the object is wider than the printable area of the page, you've no choice other than to rotate it. If, however, the object is *just* wider than the text block, you can make it pretend to be the right size by:

```
\begin{figure}
  \noindent
  \makebox[\textwidth]{\includegraphics{my-wide-figure}}%
  \caption{This figure juts out into both margins}
\end{figure}
```

Note the `\noindent`: the `\makebox` starts a paragraph, and you really don't want that indented by `\parindent`.

283 Placing two-column floats at bottom of page

You specified placement '`[htbp]`' for your full-width figure or table, but they always get placed at the top of the page… Well, it *is* what the documentation says: LaTeX, unadorned, only allows full-width floats at the top of a page, or occupying (part of) a float page.

The *stfloats* package ameliorates the situation somewhat, and makes LaTeX honour '[b]' placement as well; the *dblfloatfix* package combines a tidied version of the changes made in *stfloats* with the float ordering corrections defined in *fixltx2e*.

A particular problem with *stfloats* and *dblfloatfix* is that the float will appear, at its earliest, on the page after it is specified. This has two undesirable side-effects: first, there may be no bottom float on the first page of a document, and second, float numbers may become "entangled" (particularly if you're using *dblfloatfix* that ensures that the early-specified bottom float is set *before* any single column floats).

(The FAQ team doesn't know of any package that will make LaTeX honour '[h]' placement of double-column floats, but the *midfloat* package can be pressed into service to provide something approximating the effect it would have.)

`dblfloatfix.sty`: `macros/latex/contrib/dblfloatfix`

`midfloat.sty`: Distributed as part of `macros/latex/contrib/sttools`

`stfloats.sty`: Distributed as part of `macros/latex/contrib/sttools`

284 Floats in multicolumn setting

If you use

```
\begin{figure}
  ...
\end{figure}
```

in a `multicols` environment, the figure won't appear. If instead you use

```
\begin{figure*}
  ...
\end{figure*}
```

the figure will stretch right across the page (just the same as a `figure*` in standard LaTeX's `twocolumn` option).

It's possible to have single-column figures and tables with captions, using the '`[H]`' placement option introduced by the *float* package but you might have to fiddle with the placement because they won't 'float', and exhibit other strange behaviours (such as silently running off the end of the column at the end of the `multicols` environment).

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

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

285 Facing floats on 2-page spread

If a pair of floats need to be forced to form a 2-page spread (in a book, or whatever), the first must lie on the left side of the spread, on an even-numbered page. The *dpfloat* package provides for this: the construction to use is:

```
\begin{figure}[p]
  \begin{leftfullpage}
    <left side figure>
  \end{leftfullpage}
\end{figure}
\begin{figure}[p]
  \begin{fullpage}
    <right side figure>
  \end{fullpage}
\end{figure}
```

The construction has no effect on documents with class option `oneside` (`twoside` is the default for *book* class).

A special case of this requirement places the caption for a float on the next page. (This is useful if you have a float that "only just" fits the page.) You can (with a certain amount of twiddling) make this work with *dpfloat*, but the *fltpage* package is specially designed for the job:

```
\documentclass[twoside]{article}
\usepackage[leftFloats]{fltpage}
\begin{document}
...
\begin{FPfigure}
  \includegraphics{my-huge-figure}
  \caption{Whew!  That was a big one!}
\end{FPfigure}
...
\end{document}
```

That example should produce a caption

Figure *n (facing page)*: Whew! ...

(Note, however, that the package is an old one, and declares itself to be a beta release. I seems to work, but. . .)

dpfloat.sty: macros/latex/contrib/dpfloat

fltpage.sty: macros/latex/contrib/fltpage

286 Vertical layout of float pages

By default, LaTeX vertically centres the floats on a float page; the present author is not alone in not liking this arrangement. Unfortunately, the control of the positioning is "buried" in LaTeX-internal commands, so some care is needed to change the layout.

Float pages use three LaTeX lengths (i.e., TeX skips) to define their layout:

`\@fptop` defines the distance from the top of the page to the top of the first float,
`\@fpsep` defines the separation between floats, and
`\@fpbot` defines the distance from the bottom of the last float on the page to the bottom of the page.

(In fact, the output routine places a skip of `\@fpsep` above each float, so the `\@fptop` skip is always followed by a correction for that.)

The LaTeX defaults are:

```
\@fptop = 0pt + 1fil
\@fpsep = 8pt + 2fil
\@fpbot = 0pt + 1fil
```

so that the gaps expand to fill the space not occupied by floats, but if there is more than one float on the page, the gap between them will expand to twice the space at top and bottom.

Those who understand this stuff will be able to play elaborate games, but the commonest requirement, that the floats start at the top of the page, is a simple thing to do:

```
\makeatletter
\setlength{\@fptop}{0pt}
\makeatother
```

Surprisingly, you may find this setting leaves your floats too high on the page. One can justify a value of `5pt` (in place of `0pt`) — it's roughly the difference between `\topskip` and the height of normal (`10pt`) text.

Note that this is a "global" setting (best established in a class file, or at worst in the document preamble); making the change for a single float page is likely (at the least) to be rather tricky.

287 Figure (or table) *exactly* where I want it

This is of course a contradiction: `figure` and `table` are *designed* to float, and will always have the potential to appear away from where you asked for them. Therefore you have to find a means of getting the caption and other effects without allowing the figure or table to float.

The most straightforward way is to use one of the *float* or *here* packages; they give you a `[H]` float placement option that prevents floating:

```
\begin{figure}[H]
  \centering
  \includegraphics{foo}
  \caption{caption text}
  \label{fig:nonfloat}
\end{figure}
```

As the example shows, these `[H]` figures (and correspondingly, tables) offer all you need to cross-reference as well as typeset.

However, you don't actually *have* to use *float* (or *here*) since it is, in fact, doing rather little for you. You can place your figure as you please, with a sequence like

```
\begin{center}
  \includegraphics{foo}
  \captionof{figure}{caption text}
  \label{fig:nonfloat}
\end{center}
```

which relies on the `\captionof` command to place a caption in ordinary running text. That command may be had from the extremely simple-minded package *capt-of* or from the highly sophisticated *caption* package.

Using either method, you have to deal with the possibility of the figure or table being too large for the page. (Floating objects will float away in this circumstance; "doing it by hand", like this, you take upon yourself the responsibility for avoiding '*Overfull \vbox*' errors.

A further problem is the possibility that such "fixed floats" will overtake "real floats", so that the numbers of figures will be out of order: figure 6 could be on page 12, while figure 5 had floated to page 13. It's best, therefore, either to stay with floating figures throughout a document, or to use fixed figures throughout.

If it's really impossible to follow that counsel of perfection, you can use the *perpage* package's command `\MakeSorted` command:

```
...
\usepackage{float}
\usepackage{perpage}
\MakeSorted{figure}
\MakeSorted{table}
...
```

and the sequence of float numbers is all correct.

capt-of.sty: macros/latex/contrib/capt-of

caption.sty: macros/latex/contrib/caption

float.sty: macros/latex/contrib/float

here.sty: macros/latex/contrib/here

perpage.sty: Distributed as part macros/latex/contrib/bigfoot

P.5 Footnotes

288 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. As a result, you get footnote marks in the table, and nothing else.

This accords with common typographic advice: footnotes and tables are reckoned not to mix.

The solution, if you accept the advice, is to use "table notes". The package *threeparttable* provides table notes, and *threeparttablex* additionally supports them in `longtable`s. *Threeparttable* works happily in ordinary text, or within a `table` float.

The *ctable* package extends the model of *threeparttable*, and also uses the ideas of the *booktabs* package. The `\ctable` command does the complete job of setting the table, placing the caption, and defining the notes. The "table" may consist of diagrams, and a parameter in `\ctable`'s optional argument makes the float that is created a "figure" rather than a "table".

If you really want "real" footnotes in tables, despite the expert advice, you can:

- 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.
- Stick the table in a `minipage`. Footnotes in the table then "work", in the `minipage`'s style, with no extra effort. (This is, in effect, somewhat like table notes, but the typeset appearance isn't designed for the job.)
- Use *tabularx* or *longtable* from the LaTeX tools distribution; they're noticeably less efficient than the standard `tabular` environment, but they do allow footnotes.
- Use *tablefootnote*; it provides a `\tablefootnote`, which does the job without fuss.
- Use *footnote*, which provides an `savenotes` which collects all footnotes and emits them at the end of the environment; thus if you put your `tabular` environment inside the environment, the footnotes will appear as needed. Alternatively, you may use `\makesavenoteenv{tabular}` in the preamble of your document, and tables will all behave as if they were inside a `savenotes` environment.
- Use *mdwtab* from the same bundle; it will handle footnotes as you might expect, and has other facilities to increase the beauty of your tables. Unfortunately, it may be incompatible with other table-related packages, though not those in the standard 'tools' bundle.

All the techniques listed will work, to some extent, whether in a float or in ordinary text. The author of this FAQ answer doesn't actually recommend any of them, believing that table notes are the way to go…

ctable.sty: macros/latex/contrib/ctable

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

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

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

tablefootnote.sty: macros/latex/contrib/tablefootnote

threeparttable.sty: macros/latex/contrib/threeparttable

threeparttablex.sty: macros/latex/contrib/threeparttablex

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

289 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 headers. While footnotes will work in the table of contents, it's generally not thought a "good thing" to have them there; in the page header, footnotes will simply fail. Whatever the desirability of the matter, there's no mechanism to suppress the footnote in the page header while allowing it in the table of contents, so the footnote may only appear in the section heading itself.

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. With this, you simply need:

```
% in the document preamble
\usepackage[stable]{footmisc}
...
% in the body of the document
\section{title\footnote{title ftnt}}
```

footmisc.sty: macros/latex/contrib/footmisc

290 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 (footnotes migrating to to the list of figures or tables, or apparently random errors because `\footnote` is a fragile command), and with footnotes in tables (typically, the footnote simply disappears). Fortunately, the requirement for footnotes in captions is extremely rare: if you are experiencing problems, it is worth reviewing what you are trying to say by placing this footnote: other options are to place text at the bottom of the float, or to place a footnote at the point where you refer to the float.

Note that the *threeparttable* scheme (see, again, footnotes in tables) also applies to notes in captions, and may very well be preferable to whatever you were thinking of.

If you *are* going to proceed:

- use an optional argument in your `\caption` command, that doesn't have the footnote in it; this prevents the footnote appearing in the "List of . . . ", and
- put your whole float in a `minipage` so as to keep the footnotes with the float.

so we have:

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

However, *as well as* all of the above, 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 (as in the example above) can produce *two* copies of the footnote body "blah". (In fact, the effect only occurs with captions that are long enough to require two lines to be typeset, and so wouldn't appear with such a short caption as that in the example above.)

The documentation of the *ccaption* package describes a really rather awful workaround for this problem.

ccaption.sty: macros/latex/contrib/ccaption

threeparttable.sty: macros/latex/contrib/threeparttable

291 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}]
```

Alternatively, one may use the `\footref` command, which has the advantage of working even when the footnote mark isn't expressed as a number. The command is defined in the *footmisc* package and in the *memoir* class (at least); `\footref` reduces the above example to:

```
...\footnote{Text to repeat\label{fn:repeat}}
...
...\footref{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/fixfoot

footmisc.sty: macros/latex/contrib/footmisc

memoir.cls: macros/latex/contrib/memoir

292 More than one sequence of footnotes

The need for more than one series of footnotes is common in critical editions (and other literary criticism), but occasionally arises in other areas.

Of course, the canonical critical edition package, *edmac*, offers the facility, as does its LaTeX port, the *ledmac* package.

Multiple ranges of footnotes are offered to LaTeX users by the *manyfoot* package. The package provides a fair array of presentation options, as well. Another critical edition *ednotes* package is built upon a basis that includes *manyfoot*, as its mechanism for multiple sets of footnotes.

The *bigfoot* package also uses *manyfoot* as part of its highly sophisticated structure of footnote facilities, which was also designed to support typesetting critical editions.

bigfoot: macros/latex/contrib/bigfoot

edmac: macros/plain/contrib/edmac

ednotes: macros/latex/contrib/ednotes

ledmac: macros/latex/contrib/ledmac

manyfoot.sty: Distributed as part of macros/latex/contrib/ncctools

293 Footnotes numbered "per page"

The obvious solution is to make the footnote number reset whenever the page number is stepped, using the LaTeX internal mechanism. Sadly, the place in the document where the page number is stepped is unpredictable, not ("tidily") at the end of the printed page; so changing the footnote number only ever works by 'luck'.

As a result, resetting footnotes is inevitably a complicated process, using labels of some sort. It's nevertheless important, given the common requirement for footnotes marked by symbols (with painfully small symbol sets). There are four packages that manage it, one way or another.

The *perpage* and *zref-perpage* packages provide a general mechanism for resetting counters per page, so can obviously be used for this task. The interface is pretty simple: `\MakePerPage{footnote}` (in *perpage*) or `\zmakeperpage{footnote}` (in *zref-perpage*). If you want to restart the counter at something other than 1 (for example to avoid something in the LaTeX footnote symbol list), you can use: `\MakePerPage [2]{footnote}` (in *perpage*) or `\zmakeperpage[2]{footnote}` (in *zref-perpage*). Note that you can also load *zref-perpage*

Perpage is a compact and efficient package; *zref-perpage*, being a *zref* "module", comes with *zref*'s general mechanism for extending the the `\label`—`\[page]ref` of LaTeX, which can offer many other useful facilities.

The *footmisc* package provides a variety of means of controlling footnote appearance, among them a package option `perpage` that adjusts the numbering per page; if you're doing something else odd about footnotes, it means you may only need the one package to achieve your ends.

The *footnpag* package also does per-page footnotes (and nothing else). With the competition from *perpage*, it's probably not particularly useful any more.

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

footnpag.sty: `macros/latex/contrib/footnpag`

perpage.sty: Distributed as part `macros/latex/contrib/bigfoot`

zref-perpage.sty: Distributed as part of *zref* in `macros/latex/contrib/oberdiek`

294 Not resetting footnote numbers per chapter

Some classes (for example, *book* and *report*) set up a different set of footnotes per chapter, by resetting the footnote number at the start of the chapter. This is essentially the same action as that of equation, figure and table numbers, except that footnote numbers don't get "decorated" with the chapter number, as happens with those other numbers.

The solution is the same: use the *chngcntr* package; since the numbers aren't "decorated" you can use the `\counterwithout*` variant; the code:

```
\counterwithout*{footnote}{chapter}
```

is all you need

chngcntr.sty: `macros/latex/contrib/chngcntr`

P.6 Document management

295 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 primitive `\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 can be used on them. (Note that latex's `\input` command is itself a patch on top of the Plain TeX command. Our patches apply to the LaTeX version of the command, which is used as `\input{`*`file`*`}`)

It is possible to keep track of the name of the file currently being processed, but it's surprisingly difficult (these FAQs offered code, for a long time, that just didn't work in many cases).

The *currfile* package provides a regular means of keeping track of the details of the current file (its name in `\currfilename`, directory in `\currfiledir`, as well as the file 'base' name (less its extension) and its extension). *Currfile* does this with the help of a second package, *filehook*, which spots file operations that use `\input`, `\InputIfFileExists` and `\include`, as well as package and class loading.

The *FiNK* ("File Name Keeper") package keeps track of the file name and extension, in a macro `\finkfile`. *FiNK* is now deprecated, in favour of *currfile*, but remains available for use in old documents. The *FiNK* bundle includes a `fink.el` that provides support under *emacs* with AUC-TeX.

currfile.sty: `macros/latex/contrib/currfile`

filehook.sty: `macros/latex/contrib/filehook`

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

296 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`, `\include`, `\includegraphics` and so on). The `\input` command, as modified by LaTeX and used, with LaTeX syntax, as:

```
\input{mymacros}
```

records file details for `mymacros.tex`, but if you use TeX primitive syntax for `\input`, as:

```
\input mymacros
```

`mymacros.tex` *won't* be recorded, and so won't listed by `\listfiles` — 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 the *snapshot* 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.

The *mkjobtexmf* finds which files are used in a 'job', either via the `-recorder` option of TeX, or by using the (Unix) command *strace* to keep an eye on what TeX is doing. The files thus found are copied (or linked) to a directory which may then be saved for transmission or archiving.

bundledoc: support/bundledoc

mkjobtexmf: support/mkjobtexmf

snapshot.sty: macros/latex/contrib/snapshot

297 Marking changed parts of your document

One often needs clear indications of how a document has changed, but the commonest technique, "change bars" (also known as "revision bars"), requires surprisingly much trickery of the programmer. The problem is 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 that information, but no PDFTeX-based changebar package has been published, that takes advantage of that.

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 LaTeX-specific 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 you're using (in the same way that you need to tell the *graphics* package); the list of available drivers is pretty wide, but does not include *dvipdfm*. 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 shareware *WinEDT* editor has a macro that will generate *changebar* (or other) macros to show differences from an earlier version of your file, stored in an *RCS*-controlled repository — see http://www.winedt.org/Macros/LaTeX/RCSdiff.php

The *vertbars* package uses the techniques of the *lineno* package (which it loads, so the *lineno* itself must be installed); it's thus the smallest of the packages for change bar marking, since it leaves all the trickery to another package. *Vertbars* defines a `vertbar` environment to create changebars.

The *framed* package is another that provides bars as a side-effect of other desirable functionality: its `leftbar` environment is simply a stripped-down frame (note, though, that the environment makes a separate paragraph of its contents, so it is best used when the convention is to mark a whole changed paragraph.

Finally, the *memoir* class allows marginal editorial comments, which you can obviously use to delimit areas of changed text.

An even more comprehensive way to keep track of changes is employed by some word-processors — to produce a document that embodies both "old" and "new" versions.

To this end, the package *changes* allows the user to manually markup changes of text, such as additions, deletions, or replacements. Changed text is shown in a different colour; deleted text is crossed out. The package allows you to define additional authors and their associated colour; it also allows you to define a markup for authors or annotations. The documentation (very clearly) demonstrates how the various functions work.

The *Perl* script *latexdiff* may also be used to generate such markup for LaTeX documents; you feed it the two documents, and it produces a new LaTeX document in which the changes are very visible. An example of the output is embedded in the documentation, latexdiff-man.pdf (part of the distribution). A rudimentary revision facility is provided by another *Perl* script, *latexrevise*, which accepts or rejects all changes. Manual editing of the difference file can be used to accept or reject selected changes only.

backgrnd.tex: macros/generic/misc/backgrnd.tex

changebar.sty: macros/latex/contrib/changebar

changes.sty: macros/latex/contrib/changes *framed.sty*: macros/latex/contrib/framed

latexdiff, *latexrevise*: support/latexdiff

lineno.sty: macros/latex/contrib/lineno

memoir.cls: macros/latex/contrib/memoir

vertbars.sty: macros/latex/contrib/vertbars

winedt: systems/win32/winedt

298 Conditional compilation and "comments"

While LaTeX (or any other TeX-derived package) isn't really like a compiler, people regularly want to do compiler-like things using it. Common requirements are conditional 'compilation' and 'block comments', and several LaTeX-specific means to this end are available.

The simple `\newcommand{\gobble}[1]{}` and `\iffalse ... \fi` aren't really satisfactory (as a general solution) for comments, since the matter being skipped is nevertheless scanned by TeX, not always as you would expect. The scanning imposes restrictions on what you're allowed to skip; this may not be a problem in *today's* job, but could return to bite you tomorrow. For an example of surprises that may come to bite you, consider the following example (derived from real user experience):

```
\iffalse % ignoring this bit
consider what happens if we
use \verb+\iftrue+ -- a surprise
\fi
```

The `\iftrue` is spotted by TeX as it scans, ignoring the `\verb` command; so the `\iffalse` isn't terminated by the following `\fi`. Also, `\gobble` is pretty inefficient at consuming anything non-trivial, since all the matter to be skipped is copied to the argument stack before being ignored.

If your requirement is for a document from which whole chapters (or the like) are missing, consider the LaTeX `\include`/`\includeonly` system. If you '`\include`' your files (rather than `\input` them — see What's going on in my `\include` commands?), 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 `\include`d are skipped entirely, but the document processing continues as if they *were* there, and page, footnote, and other numbers are not disturbed. Note that you can choose which sections you want included interactively, using the *askinclude* package.

The inverse can be done using the *excludeonly* package: this allows you to exclude a (list of) `\include`d files from your document, by means of an `\excludeonly` command.

If you want to select particular pages of your document, use Heiko Oberdiek's *pagesel* or the *selectp* packages. You can do something similar with an existing PDF document (which you may have compiled using *pdflatex* in the first place), using the *pdfpages* package. The job is then done with a document looking like:

```
\documentclass{article}
\usepackage[final]{pdfpages}
\begin{document}
\includepdf[pages=30-40]{yoursource.pdf}
\end{document}
```

(To include all of the document, you write

```
\includepdf[pages=-]{yoursource.pdf}
```

omitting the start and end pages in the optional argument.)

If you want flexible facilities for including or excluding small portions of a file, consider the *comment*, *version* or *optional* packages.

The *comment* package allows you to declare areas of a document to be included or excluded; you make these declarations in the preamble of your file. The command `\includecomment{`*version-name*`}` declares an environment `version-name` whose content will be included in your document, while `\excludecomment{`*version-name*`}` defines an environment whose content will be excluded from the document. The package uses a method for exclusion that is pretty robust, and can cope with ill-formed bunches of text (e.g., with unbalanced braces or `\if` commands).

These FAQs employ the *comment* package to alter layout between the printed (two-column) version and the PDF version for browsing; there are `narrowversion` and `wideversion` for the two versions of the file.

version offers similar facilities to `comment.sty` (i.e., `\includeversion` and `\excludeversion` commands); it's far "lighter weight", but is less robust (and in particular, cannot deal with very large areas of text being included/excluded).

A significant development of *version*, confusingly called *versions* (i.e., merely a plural of the old package name). *Versions* adds a command `\markversion {`*version-name*`}` which defines an environment that prints the included text, with a clear printed mark around it.

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.

And, not least of this style of conditional compilation, *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. The *memoir* class offers the same environment.

An interesting variation is the *xcomment* package. This defines an environment whose body is all excluded, apart from environments named in its argument. So, for example:

```
\begin{xcomment}{figure,table}
  This text is not included
  \begin{figure}
    This figure is included
  \end{figure}
  This is not included, either
  \begin{table}
    This table also included
  \end{table}
  ...
\end{xcomment}
```

A further valuable twist is offered by the *extract* package. This allows you to produce a "partial copy" of an existing document: the package was developed to permit production of a "book of examples" from a set of lecture notes. The package documentation shows the following usage:

```
\usepackage[
  active,
  generate=foobar,
  extract-env={figure,table},
  extract-cmd={chapter,section}
]{extract}
```

which will cause the package to produce a file `foobar.tex` containing all the `figure` and `table` environments, and the `\chapter` and `\section` commands, from the document being processed. The new file `foobar.tex` is generated in the course of an otherwise ordinary run on the 'master' document. The package provides a good number of other facilities, including (numeric or labelled) ranges of environments to extract, and an `extract` environment which you can use to create complete ready-to-run LaTeX documents with stuff you've extracted.

askinclude.sty: Distributed as part of macros/latex/contrib/oberdiek

comment.sty: macros/latex/contrib/comment

excludeonly.sty: macros/latex/contrib/excludeonly

extract.sty: macros/latex/contrib/extract

memoir.cls: macros/latex/contrib/memoir

optional.sty: macros/latex/contrib/optional

pagesel.sty: Distributed as part of macros/latex/contrib/oberdiek

pdfpages.sty: macros/latex/contrib/pdfpages

selectp.sty: macros/latex/contrib/selectp

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

version.sty: macros/latex/contrib/version

versions.sty: macros/latex/contrib/versions/versions.sty

xcomment.sty: macros/generic/xcomment

299 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 used in temporary installations 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.

The *chapterfolder* package provides commands to deal with its (fixed) model of file inclusion in a document. It provides commands `\cfpart`, `\cfchapter`, `\cfsection` and `\cfsubsection`, each of which takes directory and file arguments, e.g.:

```
\cfpart[pt 1]{Part One}{part1}{part}
```

which command will issue a 'normal' command `\part[pt 1]{Part One}` and then input the file `part1/part.tex`, remembering that `part1/` is now the "current folder". There are also commands of the form `\cfpartstar` (which corresponds to a `\part*` command).

Once you're "in" a *chapterfolder*-included document, you may use `\cfinput` to input something relative to the "current folder", or you may use `\input`, using

`\cfcurrentfolder` to provide a path to the file. (There are also `\cfcurrentfolderfigure` for a `figure/` subdirectory and `\cfcurrentfolderlistings` for a `listings/` subdirectory.)

Documentation of *chapterfolder* is in French, but the `README` in the directory is in English.

chapterfolder.sty: macros/latex/contrib/chapterfolder

import.sty: macros/latex/contrib/import

300 Version control using RCS, CVS or the like

If you use RCS, CVS, *Subversion*, *Bazaar* or *Git* to maintain your (La)TeX documents under version control, you may need some mechanism for including the version details 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 for RCS and CVS 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. The package *rcsinfo* is simpler, but does most of what you want, and some people prefer it; it is explicitly compatible with *LaTeX2HTML*.

If, however, you need a solution which works without using external packages, or which will work in Plain TeX, then you can use the following minimal solution:

```
\def\RCS$#1: #2 ${\expandafter\def\csname RCS#1\endcsname{#2}}
\RCS$Revision: 1.37 $ % or any RCS keyword
\RCS$Date: 2012/02/07 21:46:08 $
...
\date{Revision \RCSRevision, \RCSDate}
```

If you are a user of *Subversion*, the package *svn* may be for you. It has explicit cleverness about dealing with dates:

```
\documentclass{⟨foo⟩}
...
\usepackage{svn}
\SVNdate $Date$
\author{...}
\title{...}
...
\begin{document}
\maketitle
...
\end{document}
```

will (once *subversion* has committed a copy of the document) cause `\maketitle` use the date that has been written into the `$Date$` keyword.

Another alternative for *Subversion* users is the *svninfo* package, which has much the same mechanisms as does *svn* but with a rather different focus. *Svninfo* does the date trick that *svn* performs (controlled by a package option), and can set up page foot-lines using package *fancyhdr*. There isn't much to choose between the two packages: you should read the packages' documentation to see which suits you best.

An alternative script-based approach to version control has been taken by the *vc* bundle, that in certain situations might work more reliably than any of the packages mentioned above. The *vc* bundle supports *Bazaar*, *Git* and *Subversion* usage and works with both LaTeX and Plain TeX. Note that *vc* is the only option that currently claims to support *Bazaar*-controlled repositories.

Finally, for now, the *gitinfo* package supports *Git*-controlled documents.

gitinfo.sty: macros/latex/contrib/gitinfo

rcs.sty: macros/latex/contrib/rcs

rcsinfo.sty: macros/latex/contrib/rcsinfo *svn.sty*: macros/latex/contrib/svn

svninfo.sty: macros/latex/contrib/svninfo

vc: support/vc

301 Makefiles for LaTeX documents

LaTeX is a tricky beast for running *make* on: the need to instruct LaTeX to run several times for essentially different reasons (for example, "get the table of contents stable", "get the labels stable", "add the bibliography", "add the index") is actually rather difficult to express in the 'ordinary' sort of dependency graph that one constructs for *make*.

The *latex-make* package offers help with this task (far more sophisticated techniques than in the FAQ building script); it looks good, but reports of its use (other than by its author) are scarce.

For a long time, the only *make*-like package on CTAN was *latexmk*, which is a *Perl* script that analyses your LaTeX source for its dependencies, runs BibTeX or *makeindex* as and when it notices that those programs' input (parts of the `.aux` file, or the `.idx` file, respectively) has changed, and so on. *Latexmk* is a fine solution (and was used in generating printable versions of these FAQs for a long time); it has recently been upgraded and has many bells and whistles that allow it to operate as if it were a poor man's WYSIWYG system.

Apparently along the same lines, is AutoLaTeX. The `README` of the distribution is actual a Unix-type man-page output, and shows great attention to the details of the document production process.

The *mk* (also, apparently, known as *latex_maker*) is a Ruby script for the task in hand; it works well with another of the author's scripts script called *vpp* (View and Print PostScript/PDF).

Windows users of the MiKTeX system may use that system's *texify* application. *Texify* deals with basic LaTeX features, including generating a bibliography and an index; it makes no claim to deal with other things (such as multiple bibliographies or indexes, or lists of terminology, etc.), which AutoLaTeX can be configured to process.

The *texinfo* system comes with a similar utility called *texi2dvi*, which is capable of "converting" either LaTeX or *texinfo* files into DVI (or into PDF, using PDFTeX).

A later contribution is the bundle *latexmake*, which offers a set of *make* rules that invoke *texi2dvi* as necessary.

The curious may examine the rules employed to run the present FAQ through LaTeX: we don't present them as a complete solution, but some of the tricks employed are surely re-usable.

AutoLaTeX: support/autolatex

FAQ distribution: help/uk-tex-faq

latexmake: support/latexmake

latex-make: support/latex-make

latex_make: support/latex_maker

latexmk: support/latexmk

texi2dvi: Distributed as part of macros/texinfo/texinfo

302 How many pages are there in my document?

Simple documents (those that start at page 1, and don't have any breaks in their page numbering until their last page) present no problem to the seeker after this truth. The number of pages is reported by the *lastpage* package in its `LastPage` label.

For more complicated documents (most obviously, books with frontmatter in a different series of page numbers) this simple approach will not do.

The *count1to* package defines a label `TotalPages`; this is the value of its copy of `\count1` (a reserved TeX count register) at the end of the document.

Package *totpages* defines a label `TotPages`, but it also makes the register it uses available as a LaTeX counter, `TotPages`, which you can also reference via `\theTotPages`.

Of course, the counter `TotPages` is asynchronous in the same way that page numbers are, but snapshots may safely be taken in the output routine.

The *memoir* class defines two counters `lastpage` and `lastsheet`, which are set (after the first run of a document) to the equivalent of the `LastPage` label and the `TotalPages` labels.

Both *count1to* and *totpages* need the support of the *everyshi* package.

count1to.sty: Distributed as part of `macros/latex/contrib/ms`

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

lastpage.sty: `macros/latex/contrib/lastpage`

memoir.cls: `macros/latex/contrib/memoir`

totpages.sty: `macros/latex/contrib/totpages`

303 Including Plain TeX files in LaTeX

LaTeX, though originally based on Plain TeX, does not contain all of Plain TeX's commands. Worse, some Plain TeX command names appear in LaTeX, with different semantics. As a result, special measures need to be taken to allow general Plain TeX documents (or parts of documents) to be typeset within LaTeX.

The truly reliable way is to translate the Plain TeX commands, to produce an equivalent of the original's semantics. However, this is not practical in many circumstances, and for those occasions, the *plain* package will often come to your aid. The package defines a `plain` environment, in which a Plain TeX document may be processed:

```
\begin{plain}
  \input{plain-doc}
\end{plain}
```

The package is known to fail, for example, with documents that use AMSTeX; no doubt it would also fail if asked to load Eplain. All these things can be overcome (although it's not often easy), but the environment saves a lot of work on many occasions.

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

P.7 Hyphenation

304 My words aren't being hyphenated

Let's assume you've selected the right TeX 'language' — as explained in "how hyphenation works", 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 "using a new language" 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 "weird hyphenation"), 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 accents and hyphens.
- 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/hyphenat`

305 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 LaTeX2e. 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 LaTeX2e, 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, which introduced an alternative hyphen character. The LaTeX2e 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 LaTeX2e 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.gz`

plain.tex: `macros/plain/base`

306 (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 "using a new language" for instructions on adding this new "language").

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

307 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 the EC fonts) 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, once promoted by Y&Y inc) may be used in place of T1. Indeed, most current 8-bit TeX font encodings will 'work' with the relevant sets of hyphenation patterns.

One might hope that, with the many aspirant successors to TeX such as Omega, LuaTeX and ExTeX, all of which base their operations on Unicode, that the whole basis of encodings will change.

308 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
(babel)                the language 'Catalan'
(babel)                I will use the patterns loaded for \language=0 instead.
```

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 installing a new package).

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 It's possible to do the whole operation in one go, by using the *texconfig* command:

```
texconfig hyphen latex
```

which first enters an editor for you to edit `language.dat`, and then regenerates the format you specify (*latex* in this case).
Otherwise, 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{}* distribution v2.0 or later, the whole procedure can be done via the GUI. To select the new language, do:
`Start→Programs→MiKTeX 2→MiKTeX Options`, and select the `Languages` tab. Select your language from the list, press the `Apply` button, and then the `OK` button. Then select the `General` tab and press the `Update Now` button.
Otherwise, edit the `language.dat` file (as outlined above), and then run:
`initexmf --dump`
just as for a pre-v2.0 system.

***Caveat*:** It is (just) possible that your TeX system may run out of "pattern memory" while generating the new format. Most TeX implementations have fixed-size arrays for storing the details of hyphenation patterns, but although their size is adjustable in most modern distributions, actually changing the size is a fiddle. If you *do* find you've run out of memory, it may be worth scanning the list of languages in your `language.dat` to see whether any could reasonably be removed.

babel: macros/latex/required/babel

hyphenation patterns: language/hyphenation

309 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 output 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. You can load *hyphenat* with the command

```
\usepackage[none]{hyphenat}
```

to prevent any hyphenation in a single-language document. The technique cannot work in a document in which *babel* controls language selection, since *babel* incorporates hyphenation change into its language change facilities.

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 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.

The simplest route is to use `\sloppy` (or its environment version `sloppypar`), and have TeX stretch what would otherwise be underfull lines to fill the space offered, while prematurely wrapping overfull lines and stretching the remainder.

Alternatively, you may set the text ragged right, 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/hyphenat

310 Preventing hyphenation of a particular word

It's quite possible for (*any*) hyphenation of a particular word to seem "completely wrong", so that you want to prevent it being hyphenated.

If the word occurs in just one place, put it in a box:

```
\mbox{oddword}
```

(Plain TeX users should use `\hbox`, and take care at the start of paragraphs.) However, boxing the word is not really advisable unless you are sure it only occurs once.

If the word occurs commonly, the best choice is to assert a non-hyphenation for it:

```
\hyphenation{oddword}
```

This hyphenation exception (with no break points) will be used in preference to what TeX's hyphenation algorithm may come up with.

In a multilingual document, repeat the exception specification for each language the word may appear in. So:

```
\usepackage[french,english]{babel}
\selectlanguage{english}
\hyphenation{oddword}
\selectlanguage{french}
\hyphenation{oddword}
```

(note that *babel* will select the default language for the document — English, in this case — at `\begin{document}`.)

A particular instance of this requirement is avoiding the hyphenation of acronyms; a general rule for those that concoct acronyms seems to be to make the capital-letter sequence read as near as is possible like a "real" word, but hyphenating an acronym often looks silly. The TeX flag `\uchyph` is designed for suppressing such behaviour:

```
\uchyph=0
```

will stop hyphenation of upper-case words. (Note that Plain TeX syntax is needed here: there's no LaTeX alternative for setting this value.)

311 Hyphenation exceptions

While TeX's hyphenation rules are good, they're not infallible: you will occasionally find words TeX just gets *wrong*. So for example, TeX's default hyphenation rules (for American English) don't know the word "*manuscript*", and since it's a long word you may find you need to hyphenate it. You *can* "write the hyphenation out" each time you use the word:

```
... man\-u\-script ...
```

Here, each of the `\-` commands is converted to a hyphenated break, if (*and only if*) necessary.

That technique can rapidly become tedious: you'll probably only accept it if there are no more than one or two wrongly-hyphenated words in your document. The alternative is to set up hyphenations in the document preamble. To do that, for the hyphenation above, you would write:

```
\hyphenation{man-u-script}
```

and the hyphenation would be set for the whole document. Barbara Beeton publishes articles containing lists of these "hyphenation exceptions", in *TUGboat*; the hyphenation 'man-u-script' comes from one of those articles.

What if you have more than one language in your document? Simple: select the appropriate language, and do the same as above:

```
\usepackage[french]{babel}
\selectlanguage{french}
\hyphenation{re-cher-cher}
```

(nothing clever here: this is the "correct" hyphenation of the word, in the current tables). However, there's a problem here: just as words with accents in them won't break, so `\hyphenation` commands with accents in them produce an error:

```
\usepackage[french]{babel}
\selectlanguage{french}
\hyphenation{r\'e-f\'e-rence}
```

tells us that the hyphenation is "improper", and that it will be "flushed". But, just as hyphenation of words is enabled by selecting an 8-bit font encoding, so `\hyphenation` commands are rendered proper again by selecting that same 8-bit font encoding. For the hyphenation patterns provided in the usual distributions, the encoding is Cork, so the complete sequence is:

```
\usepackage[T1]{fontenc}
\usepackage[french]{babel}
\selectlanguage{french}
\hyphenation{r\'e-f\'e-rence}
```

The same sort of performance goes for any language for which 8-bit fonts and corresponding hyphenation patterns are available. Since you have to select both the language and the font encoding to have your document typeset correctly, it should not be a great imposition to do the selections before setting up hyphenation exceptions.

P.8 Odds and ends

312 Typesetting all those TeX-related logos

Knuth was making a particular point about the capabilities of TeX when he defined the logo. Unfortunately (in some people's opinion) he thereby opened floodgates to give the world a whole range of rather silly 'bumpy road' logos for TeX entities such as AMSTeX, 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, with a raised small 'A' and a lowered 'E') and marketing input from Addison-Wesley led to the even stranger current logo for LaTeX2e, which appends a lowered 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 LaTeX2e). The Metafont and Metapost logos can be set in fonts that LaTeX2e 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 — most versions of the logo font distributed nowadays contain the missing letters, and the author himself uses just 'MetaPost'.

A well-designed set of macros is provided by package *hologo*, which defines a command `\hologo`, which one uses as (for example) `\hologo{pdfLaTeX}` for what you might get by typing "`pdf\LaTeX`", as well as a capitalised version `\Hologo{pdfLaTeX}` for "`Pdf\LaTeX`".

The package *metalogo* deals with a problem of these myriad logos, that's often ignored nowadays: the geometry of characters from different fonts is (obviously) different, and they naturally fit together differently. The package makes it possible for you to adjust the spacing between the the letters of one of these odd logos (even the especially weird mirrored "E" in XeTeX).

For those who don't wish to acquire the 'proper' logos, the canonical thing to do is to say `AMS-\TeX{}` for AMSTeX, `Pic\TeX{}` for PiCTeX, `Bib\TeX{}` for BibTeX, and so on.

`hologo.sty`: Distributed as part of `macros/latex/contrib/oberdiek`

`metalogo.sty`: `macros/latex/contrib/metalogo`

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

`texnames.sty`: `info/biblio/texnames.sty`

313 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 CTAN 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 (these are (La)TeX-specific files, so the "*public*" supplier would be an appropriate place). Once you've rebuilt the file indexes as necessary, 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.

Another alternative is to use the EC fonts, which come with bold variants of the small-caps fonts.

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. There are, however, Type 1 distributions of the EC fonts, so you can switch to EC and use them; alternatives are discussed in 8-bit Type 1 fonts.

Of course, commercial fixed-width fonts (even the default *Courier*) almost always come with a bold variant, so that's not a problem. Furthermore PSNFSS will usually provide "faked" small caps fonts, and has no compunctions about providing them in a bold form. *Courier* is (as we all know, to our cost) freely available; a far more presentable monospace font is *LuxiMono*, which is also freely available (monospace text in the typeset version of this FAQ uses *LuxiMono*, with the metrics and LaTeX support available on the archive.

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

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

LuxiMono fonts: `fonts/LuxiMono`

314 Automatic sizing of `minipage`

The `minipage` environment requires you to specify the width of the "page" you're going to create. This is sometimes inconvenient: you would like to occupy less space, if possible, but `minipage` sets a box that is exactly the width you specified.

The *pbox* package defines a `\pbox` whose width is exactly that of the longest enclosed line, subject to a maximum width that you give it. So while `\parbox{2cm}{Hello\\world!}` produces a box of width exactly `2cm`, `\pbox{2cm}{Hello\\world!}` produces one whose width is `1.79cm` (if one's using the default *cmr* font for the text, at least). The package also provides a `\settominwidth[`*`min`*`]{`*`length`*`}{`*`text`*`}` (which looks (almost) like the standard `\settowidth` command), and a `\widthofpbox` function analogous to the `\widthof` command for use with the *calc* package.

The *eqparbox* package extends *pbox*'s idea, by allowing you to set a series of boxes, all with the same (minimised) width. (Note that it doesn't accept a limiting maximum width parameter.) The package documentation shows the following example drawn from a joke *curriculum vitae*:

```
\noindent%
\eqparbox{place}{\textbf{Widgets, Inc.}} \hfill
\eqparbox{title}{\textbf{Senior Widget Designer}} \hfill
\eqparbox{dates}{\textbf{1/95--present}}

...

\noindent%
\eqparbox{place}{\textbf{Thingamabobs, Ltd.}} \hfill
\eqparbox{title}{\textbf{Lead Engineer}} \hfill
\eqparbox{dates}{\textbf{9/92--12/94}}
```

The code makes the three items on each of the heading lines have exactly the same width, so that the lines as a whole produce a regular pattern down the page. A command `\eqboxwidth` allows you to use the measured width of a group: the documentation shows how the command may be used to produce sensible-looking columns that mix `c`-, `r`- or `l`-rows, with the equivalent of a `p{...}` entry, by making the fixed-width rows an *eqparbox* group, and making the last from a `\parbox` using the width that's been measured for the group.

The *varwidth* package defines a `varwidth` environment which sets the content of the box to match a "narrower natural width" if it finds one. (You give it the same parameters as you would give `minipage`: in effect, it is a 'drop-in' replacement.) *Varwidth* provides its own ragged text command: `\narrowragged`, which aims to make narrower lines and to put more text in the last line of the paragraph (thus producing lines with more nearly equal lengths than typically happens with `\raggedright` itself).

The documentation (in the package file) lists various restrictions and things still to be done, but the package is already proving useful for a variety of jobs.

`eqparbox.sty`: `macros/latex/contrib/eqparbox`

`pbox.sty`: `macros/latex/contrib/pbox`

`varwidth.sty`: `macros/latex/contrib/varwidth`

Q Symbols, etc.

315 Using symbols

Most symbol font sets come with a package that defines commands for every symbol in the font. While this is convenient, it can lead to difficulties, particularly with name clashes when you load packages that cover fonts which duplicate symbols — an issue which is discussed in "symbol already defined". Some font sets (for example the related set: *FdSymbol*, *MdSymbol* and *MnSymbol*) are huge, and the accompanying macros cover so many symbols that name clashes are surely a serious problem.

The *pifont* package (originally designed to use the Adobe *Zapf Dingbats* font) avoids this sort of problem: it requires you to know the font position of any symbol you want to use (the documentation provides font tables). The basic command is `\ding{`*`number`*`}` for a single symbol; there are commands for other fancier uses. *Pifont* also allows you to select other fonts, for similar use.

The *yagusylo* describes itself as "an extended version of *pifont*, gone technicolor". It provides all the facilities of *pifont*, but allows you to create your own mnemonic names for symbols. Thus, while you can say `\yagding[`*`family`*`]{`*`symbol number`*`}[`*`colour`*`]`, you can also define symbol names with `\defdingname`, and then use them with `\yagding*{`*`symbol name`*`}` (the defined name carries the font family and colour specified in the arguments of `\defdingname`).

Yagusylo is somewhat complicated, but its documentation is clear; it is probably the best tool to use for picking and choosing symbols from a variety of font families.

`FdSymbol` *fonts*: `fonts/fdsymbol`

`MdSymbol` *fonts*: `fonts/mdsymbol`

`MnSymbol` *fonts*: `fonts/mnsymbol`

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

`yagusylo.sty`: `macros/latex/contrib/yagusylo`

316 Symbols for the number sets

Mathematicians commonly use special lettering for the real numbers and other standard number sets. Traditionally these were typeset in bold. In the ordinary course of events, mathematicians do not have access to bold chalk, so they invented special symbols that are now often used for the number sets. Such symbols are known as "blackboard bold" (or double-stroked) letters; in place of the heavier strokes of a bold font, (some) strokes of the letters are doubled. The minimum useful set is upper-case letters 'I', 'N', 'R', 'Q' and 'Z'; some fonts offer a figure '1' (for a unit matrix — not a number set at all).

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 AMS actually provides a pair of font families (the other is called *msam*), which offer a large number of mathematical symbols to supplement those provided in Knuth's fonts. The fonts are available in Type 1 format in modern distributions. Support for using the fonts under LaTeX is available in packages *amssymb* and *amsfonts*. The font shape is a rather austere sans, which many people don't like (though it captures the essence of quickly-chalked writing rather well).

The *bbold* family is set of blackboard bold fonts written in Metafont. This set offers blackboard bold forms of lower-case letters; 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.

The *bbm* family claims to provide 'blackboard' versions of most of the *cm* fonts . . . including the bold and bold-extended series. Again, the fonts are designed in Metafont and are not available in Type 1 format. LaTeX macro support comes from a package by Torsten Hilbrich.

The *doublestroke* family comes in just roman and sans shapes, at a single weight, and is available both as Metafont sources and as Type 1; the font covers the uppercase latin letters, lowercase 'h' and 'k', and the digit '1'.

A document that shows the *bbm*, *bbold*, *doublestroke* and *msbm* fonts, so that you can get a feel for their appearance, is available (CTAN package *blackboard*).

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 choice of outline fonts). Of the free sets, the *txfonts* and *pxfonts* families both come with replicas of *msam* and *msbm* (though, as noted elsewhere, there are other reasons not to use these fonts). The *mathpazo* family includes a "mathematically significant" choice of blackboard bold characters, and the *fourier* fonts contain blackboard bold upper-case letters, the digit '1', and lower-case 'k'.

The "lazy person's" blackboard bold macros:

```
\newcommand{\R}{{\textsf{R}\hspace*{-0.9ex}%
```

```
  \rule{0.15ex}{1.5ex}\hspace*{0.9ex}}}
\newcommand{\N}{{\textsf{N}\hspace*{-1.0ex}%
  \rule{0.15ex}{1.3ex}\hspace*{1.0ex}}}
\newcommand{\Q}{{\textsf{Q}\hspace*{-1.1ex}%
  \rule{0.15ex}{1.5ex}\hspace*{1.1ex}}}
\newcommand{\C}{{\textsf{C}\hspace*{-0.9ex}%
  \rule{0.15ex}{1.3ex}\hspace*{0.9ex}}}
```

are almost acceptable at normal size if the surrounding text is *cmr10* (the position of the vertical bar can be affected by the surrounding font). However, they are not part of a proper maths font, and do not work in sub- and superscripts. As we've seen, there are plenty of alternatives: that mythical "lazy" person can inevitably do better than the macros, or anything similar using capital 'I' (which looks even worse!). Voluntary (La)TeX effort has redefined the meaning of laziness (in this respect!).

AMS support files: Distributed as part of fonts/amsfonts

AMS symbol fonts: Distributed as part of fonts/amsfonts

bbm fonts: fonts/cm/bbm

bbm macros: macros/latex/contrib/bbm

bbold fonts: fonts/bbold

blackboard evaluation set: info/symbols/blackboard

doublestroke fonts: fonts/doublestroke

fourier fonts: fonts/fourier-GUT

mathpazo fonts: fonts/mathpazo

pxfonts: fonts/pxfonts

txfonts: fonts/txfonts

317 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.

mathabx The *mathabx* bundle provides calligraphic letters (in both upper and lower case); the fonts were developed in MetaFont, but a version in Adobe Type 1 format is available. The bundle's documentation offers a series of comparisons of its calligraphic set with Computer Modern's (both regular mathematical and calligraphic letters); the difference are not large.

mnsymbol The *mnsymbol* bundle provides (among many other symbols) a set of calligraphic letters, though (again) they're rather similar to the default Computer Modern set.

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 (Type 1 versions of the font are also provided, courtesy of Taco Hoekwater). The package creates a new command `\mathscr`.

RSFSO The bundle *rsfso* provides a less dramatically oblique version of the RSFS fonts; the result proves quite pleasing — similar to the effect of the the (commercial) script font in the Adobe Mathematical Pi collection.

Zapf Chancery is the standard PostScript calligraphic font. There is no package but you can easily make it available by means of the command

```
\DeclareMathAlphabet{\mathscr}{OT1}{pzc}{m}{it}
```

in your preamble. You may find the font rather too big; if so, you can use a scaled version of it like this:

```
\DeclareFontFamily{OT1}{pzc}{}
\DeclareFontShape{OT1}{pzc}{m}{it}{<-> s * [0.900] pzcmi7t}{}
\DeclareMathAlphabet{\mathscr}{OT1}{pzc}{m}{it}
```

Adobe Zapf Chancery (which the above examples use) is distributed in any but the most basic PostScript printers. A substantially identical font (to the extent that the same metrics may be used) is available from URW, called URW Chancery L: it is distributed with Ghostscript, and as part of the "URW base35" bundle; the *urwchancal* package (which includes virtual fonts to tweak appearance) provides for its use as a calligraphic font.

Examples of the available styles are linked from the packages' catalogue entries.

eucal.sty: Distributed as part of fonts/amsfonts

euler fonts: Distributed as part of fonts/amsfonts

ghostscript: Browse support/ghostscript/GPL

mathabx as Metafont: fonts/mathabx

mathabx in Type 1 format: fonts/ps-type1/mathabx

mathrsfs.sty: Distributed as part of macros/latex/contrib/jknappen

mnsymbol fonts: fonts/mnsymbol

rsfs fonts: fonts/rsfs

rsfso fonts: fonts/rsfso

Script font examples: info/symbols/math/scriptfonts.pdf

urwchancal: fonts/urwchancal

URW Chancery L: Distributed as part of fonts/urw/base35

318 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 AMSLaTeX *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 AMSLaTeX) 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 AMSLaTeX macros/latex/required/amslatex

amsmath.sty: Distributed as part of AMSLaTeX macros/latex/required/amslatex

319 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.

320 How to typeset an underscore character

The underscore character ‘_’ is ordinarily used in TeX to indicate a subscript in maths mode; if you type _, on its own, in the course of ordinary text, TeX will complain. The “proper” LaTeX command for underscore is `\textunderscore`, but the LaTeX 2.09 command _ is an established alias. Even so, if you’re writing a document which will contain a large number of underscore characters, the prospect of typing _ 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’ (the answer in “defining characters as macros” uses this example to illustrate its techniques). However, the code *is* somewhat tricky, and more importantly there are significant points where it’s easy to get it wrong. There is therefore a package *underscore* which provides a general solution to this requirement.

There is a problem, though: OT1 text fonts don’t contain an underscore character, unless they’re in the typewriter version of the encoding (used by fixed-width fonts such as `cmtt`). In place of such a character, LaTeX (in OT1 encoding) uses a short rule for the command `\textunderscore`, but this poses problems for systems that interpret PDF — for example those PDF-to-voice systems used by those who find reading difficult.

So either you must ensure that your underscore characters only occur in text set in a typewriter font, or you must use a more modern encoding, such as T1, which has the same layout for every font, and thus an underscore in every font.

A stable procedure to achieve this is:

```
% (1) choose a font that is available as T1
% for example:
\usepackage{lmodern}

% (2) specify encoding
\usepackage[T1]{fontenc}

% (3) load symbol definitions
\usepackage{textcomp}
```

which will provide a command `\textunderscore` which robustly selects the right character. The *underscore* package, mentioned above, will use this command.

`underscore.sty`: `macros/latex/contrib/underscore`

321 How to type an ‘@’ sign?

Long ago, some packages used to use the ‘@’ sign as a special character, 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 ‘@’.

322 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. When the currency first appeared, typesetting it was a serious problem for (La)TeX users; things are easier now (most fonts have some way of providing a Euro sign), but this answer provides a summary of methods “just in case”.

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 TC 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 each of the *latin9* and *latin10* input encoding definitions for the *inputenc* package has a euro character defined (character position 164, occupied in other ISO Latin character sets by the “currency symbol” ¤, which ordinary people seldom see except in character-set listings...). The TC encoding file offers the command `\texteuro` for the character; that command is (probably) *only* available from the *textcomp* package.

Use of the TC encoding character may therefore made via `\texteuro` or via the Latin-9 or Latin-10 character in ordinary text.

Note that there is a Microsoft code page position (128), too, and that has been added to *inputenc* tables for CP1252 and CP1257. (There’s another position in CP858, which has it in place of “dotless i” in CP850; the standardisation of these things remains within Microsoft, so one can never tell what will come next...)

Outline fonts which contain nothing but Euro symbols are available (free) from Adobe — the file is packaged as a *Windows* self-extracting executable, but it may be decoded as a `.zip` format archive 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.

The *euro-ce* bundle is a rather pleasing Metafont-only design providing Euro symbols in a number of shapes. The file `euro-ce.tex`, in the distribution, offers hints as to how a Plain TeX user might make use of the fonts.

Euro symbols are found in several other places, which we list here for completeness.

The *marvosym* font contains a Euro symbol (in a number of typographic styles), among many other good things; the font is available in both Adobe Type 1 and TrueType formats.

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/china2e`

EC fonts: `fonts/ec`

euro fonts: `fonts/euro`

euro-ce fonts: `fonts/euro-ce`

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

eurosym fonts: `fonts/eurosym`

marvosym fonts: `fonts/marvosym`

textcomp.sty: Part of the LaTeX distribution.

323 How to get copyright, trademark, etc.

The "Comprehensive symbol list" (Comprehensive symbol list), lists the symbol commands `\textcopyright`, `\textregistered` and `\texttrademark`, which are available in TS1-encoded fonts, and which are enabled using the *textcomp* package.

In fact, all three commands are enabled in default LaTeX, but the glyphs you get aren't terribly beautiful. In particular, `\textregistered` behaves oddly when included in bold text (for example, in a section heading), since it is composed of a small-caps letter, which typically degrades to a regular shape letter when asked to set in a bold font. This means that the glyph becomes a circled "r", whereas the proper symbol is a circled "R".

This effect is of course avoided by use of *textcomp*.

Another problem arises if you want `\textregistered` in a superscript position (to look similar to `\texttrademark`). Using a maths-mode superscript to do this provokes lots of pointless errors: you *must* use

```
\textsuperscript{\textregistered}
```

324 Using "old-style" figures

These numbers are also called medieval or lowercase figures and their use is mostly font-specific. Terminology is confusing since the lining figures (which are now the default) are a relatively recent development (19th century) and before they arrived, oldstyle figures were the norm, even when setting mathematics. (An example is Thomas Harriot's *Artis Analyticae Praxis* published in 1631). In a typical old style 3, 4, 5, 7 and 9 have descenders and 6 and 8 ascend above the x-height; sometimes 2 will also ascend (this last seems to be a variation associated with French typography).

LaTeX provides a command `\oldstylenums{`*digits*`}`, which by default uses an old-style set embedded in Knuth's 'math italic' font. The command isn't sensitive to the font style of surrounding text: the glyphs are only available to match the normal medium weight Computer Modern Roman fonts.

The *textcomp* package changes `\oldstylenums` to use the glyphs in the Text Companion fonts (LaTeX TS1 encoding) when in text mode, and also makes them available using the macros of the form `\text<number>oldstyle`, e.g., `\textzerooldstyle`. (Of course, not all font families can provide this facility.)

Some font packages (e.g., *mathpazo*) make old-style figures available and provide explicit support for making them the default: `\usepackage[osf]{mathpazo}` selects a form where digits are always old-style in text. The *fontinst* package will automatically generate "old-style versions" of commercial Adobe Type 1 font families for which "expert" sets are available.

It's also possible to make virtual fonts, that offer old-style digits, from existing font packages. The *cmolddig* bundle provides a such a virtual version of Knuth's originals, and the *eco* or *hfoldsty* bundles both provide versions of the EC fonts. The *lm* family offers old-style figures to OpenType users (see below), but we have no stable mapping for *lm* with old-style digits from the Adobe Type 1 versions of the fonts.

Originally, oldstyle figures were only to be found the expert sets of commercial fonts, but now they are increasingly widely available. An example is Matthew Carter's Georgia font, which has old-style figures as its normal form (the font was created for inclusion with certain Microsoft products and is intended for on-screen viewing).

OpenType fonts have a pair of axes for number variations — proportional/tabular and lining/oldstyle selections are commonly available. "Full feature access" to OpenType fonts, making such options available to the (La)TeX user, is already supported by

XeTeX using, for example, the *fontspec* package. Similar support is also in the works for LuaTeX.

`cmolddig` *fonts*: `fonts/cmolddig`

`eco` *fonts*: `fonts/eco`

`fontinst`: `fonts/utilities/fontinst`

`fontspec.sty`: `macros/latex/contrib/fontspec`

`mathpazo` *fonts*: `fonts/mathpazo`

R Macro programming

R.1 "Generic" macros and techniques

325 Non-letters in macro names

New LaTeX users are often suprised that macro definitions containing non-letters, such as

```
\newcommand{\cul8r}{Goodbye!}
```

fail to compile. The reason is that the TeX macro language, unlike most programming languages, allows nothing but letters in macro names.

There are a number of techniques for defining a macro with a name like `\cul8r`. Unfortunately, none of the techniques is particularly good:

1. Use `\csname`...`\endcsname` to define and invoke the macro:

   ```
   \expandafter\newcommand\csname cul8r\endcsname{Goodbye!}
   I said, ``\csname cul8r\endcsname''.
   ```

   Pro: No unexpected side effects
   Con: So verbose as to be unusable

2. Define a "special-command generator", and use the resulting commands:

   ```
   \newcommand{\DefineRemark}[2]{%
     \expandafter\newcommand\csname rmk-#1\endcsname{#2}%
   }
   \newcommand{\Remark}[1]{\csname rmk-#1\endcsname}
   ...
   \DefineRemark{cul8r}{Goodbye!}
   ...
   \Remark{cul8r}
   ```

   Pro: Straightforward to use, not too untidy
   Con: It's hardly doing what we set out to do (experts will see that you are defining a macro, but others likely won't)

3. Convince TeX that "8" is a letter:

   ```
   \catcode`8 = 11
   \newcommand{\cul8r}{Goodbye!}
   I said, ``\cul8r''.
   ```

   Pro: `\cul8r` can be used directly
   Con: Likely to break other uses of "8" (such as numbers or dimensions; so `\setlength{\paperwidth}{8in}` tells us:

   ```
   ! Missing number, treated as zero.
   <to be read again>
                      8
   ```

As a general rule, changing category codes is something to use *in extremis*, after detailed examination of options. It is conceivable that such drastic action could be useful for you, but most ordinary users are well advised not even to try such a technique.

4. Define a macro \cul which must always be followed by "8r":

```
\def\cul8r{Goodbye!}
I said, ``\cul8r''.
```

Pro: \cul8r can be used directly

Con #1: Breaks if \cul is followed by anything other than "8r", with a confusing diagnostic — \cul99 produces:

```
! Use of \cul doesn't match its definition.
<*> \cul9
         9
```

(which would confuse someone who hadn't even realised there *was* a definition of \cul in the document).

Con #2: Silently redefines existing \cul, if any; as a result, the technique cannot be used to define both a \cul8r and, say, a \cul123 macro in the same document.

Technique 3 is in fact commonly used — in a limited form — within most LaTeX packages and within LaTeX itself. The convention is to use "@" within the names of internal macros to hide them from the user and thereby prevent naming conflicts. To this end, LaTeX automatically treats "@" as a letter while processing classes and packages and as a non-letter while processing the user's document. The key to this technique is the separation: internally a non-letter is used for macro names, and the user doesn't see anything of it, while the status remains "frozen" in all the definitions created within the class or package. See \@ and @ in macro names for more information.

Note that analogous use of technique 3 in this example would give us

```
\begingroup
  \catcode`8 = 11
  \gdef\cul8r{Goodbye!}
  \gdef\later{\cul8r}
\endgroup
I said, ``\later''.
```

which works, but rather defeats the object of the exercise. (\later has the "frozen" catcode for '8', even though the value has reverted to normal by the time it's used; note, also, the use of the primitive command \gdef, since \newcommand can't make a macro that's available outside the group.)

Recommendation: Either choose another mechanism (such as \DefineRemark above), or choose another name for your macro, one that contains only ordinary letters. A common approach is to use roman numerals in place of arabic ones:

```
\newcommand{\culVIIIr}{Goodbye!}
```

which rather spoils the intent of the joke implicit in the example \cul8r!

326 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.

327 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 'unterminated recursion', 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:

```
\let\OldSmooth\smooth
\renewcommand{\smooth}[2]{\mumble\OldSmooth{#1}{#2}}
```

The situation is more difficult still if `\smooth` takes an optional argument; the structure of the command is so complex that the simple `\let` command does not retain the necessary detail. In this case, we need the *letltxmacro* package which knows about all sorts of LaTeX commands and how to replicate them. Suppose we have a command defined as:

```
\newcommand{\rough}[1][\default]{...}
```

with an optional argument (substituted with `\default` if it's not present) as well as an ordinary one. In this case we copy it using

```
\LetLtxMacro{\NewRough}{\rough}
```

and then repeat the technique we had above, with one extension:

```
\renewcommand{\rough}[1][\newdef]%
             {\mumble\OldRough[{#1}]{#2}}
```

We see here that (for tedious argument-matching reasons) it is necessary to provide braces arround the optional argument that is being passed on.

The general case may be achieved in two ways. First, one can use the LaTeX command `\CheckCommand`; this compares an existing command with the definition you give it, and issues a warning if two don't match. Use is therefore:

```
\CheckCommand{\complex}{⟨original definition⟩}
\renewcommand{\complex}{⟨new definition⟩}
```

This technique is obviously somewhat laborious, but if the original command comes from a source that's liable to change under the control of someone else, it does at least warn you that your patch is in danger of going wrong.

Otherwise, LaTeX users may use one of the *patchcmd*, *ted* or *etoolbox* packages.

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, after

```
\def\b{b}
\patchcmd\b{a}{c}
```

we will have a new version of `\b` defined as "`abc`".

The *ted* package is a 'token list editor', and provides a command `\substitute` which will patch the contents of a macro, putting the result in a token-list, or (in the form `\Substitute*`) using the result to (re)define a macro. The following example defines a simple macro, and then changes its definition:

```
\newcommand{\myfont}%
    {\Large\sffamily\selectfont}
\Substitute*[\renewcommand{\myfont}]{\myfont}%
    {\Large\sffamily}{\huge\itshape}
```

The macro's definition is now:

```
\huge\itshape\selectfont
```

The package also offers a command `\ShowTokens`, which shows the content of its argument, one token to a line, with details of the token's category code, etc. This is recommended as a debugging tool.

The *etoolbox* (which provides user access to e-TeX's programming facilities) provides a command `\patchcmd` which is very similar to `\Substitute`, except that it only replaces a single instance of the token(s) in its search pattern. Since not all commands may be patched in this way, `\patchcmd` takes extra arguments for *success* and *failure* cases. The package also provides commands that prepend (`\pretocmd`) or append (`\apptocmd`) to the definition of a command. Not all commands may be patched in this way; the package provides a command `\ifpatchable` which checks the prerequisites, and checks that the target command's body does indeed include the patch string you propose to use. (The command `\ifpatchable*` omits the test on the patch string.)

Finally, we'll briefly consider a package that is (just about) usable, but not really recommendable. *Patch* gives you an ingenious (and difficult to understand) mechanism, and comes as an old-style LaTeX documented macro file, which can no longer be processed to produce formatted documentation, etc.. Fortunately, the file (`patch.doc`) may be input directly, and "documentation" may found by reading the source of the package. Roughly speaking, one gives the command a set of instructions analogous to *sed* substitutions, and it regenerates the command thus amended. Unless you can't do your job any other way, *patch* is best avoided.

etoolbox.sty: `macros/latex/contrib/etoolbox`

letltxmacro.sty: Distributed as part of `macros/latex/contrib/oberdiek`

patch.doc: `macros/generic/misc/patch.doc`

patchcommand.sty: `macros/latex/contrib/patchcmd`

ted.sty: `macros/latex/contrib/ted`

328 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 `\strip@prefix` that you can use.

329 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\gobbleminus#1{\ifx-#1\else#1\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=0#1 \end}
\def\testresult#1\end{\ifx\end#1\end\isanumtrue\else\isanumfalse\fi}
```

(which technique is due to David Kastrup; the trick for avoiding the errors, noted in an earlier version of this answer, was suggested by Andreas Matthias). 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. Sadly, this technique has the unfortunate property that it accepts simple expressions such as '`1+2-3`'; this could be solved by an initial `\gobbleminus`-like construction.

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
```

The LaTeX *memoir* class has an internal command of its own, `\checkifinteger{`*num*`}`, that sets the conditional command `\ifinteger` according to whether the argument was an integer.

Of course, all this kerfuffle would be (essentially) void if there was a simple means of "catching" TeX errors. Imagining an error-catching primitive `\ifnoerror`, one might write:

```
\def\IsInteger#1{%
  TT%
  \ifnoerror
    \tempcount=#1\relax
% carries on if no error
    \expandafter\iftrue
  \else
% here if there was an error
    \expandafter\iffalse
  \fi
}
```

thus using TeX's own integer-parsing code to do the check. It's a pity that such a mechanism was never defined (it could be that it's impossible to program within TeX!).

memoir.cls: macros/latex/contrib/memoir

330 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:

```
\newcommand\a[1]{+#1+#1+#1+}
```

or (using the TeX primitive `\def`):

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

and use it as `\a{b}`, the macro expansion produces '+b+b+b+', as most people would expect.

However, if we now replace part of the macro:

```
\newcommand\a[1]{+#1+\newcommand\x[1]{xxx#1}}
```

then `\a{b}` will give us the rather odd

```
+b+\newcommand{x}[1]{xxxb}
```

so that the new `\x` ignores its argument.

If we use the TeX primitive:

```
\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 is surely not what was intended!

Actually, to define `\x` to take an argument, we need

```
\newcommand\a[1]{+#1+\newcommand\x[1]{xxx##1}}
```

or, using the TeX primitive definition:

```
\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`, doubling the number of # signs; and at the next level you need 8 #s each time, and so on.

331 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 (unlike consecutive spaces that you type), 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 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

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.

332 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

* faq-mac-prog.tex (q-keyval): tweak words about getoptk

333 Key-value input for macros and package options

When we discussed extending the number of arguments to a macro, we suggested that large numbers of arguments, distinguished only by their position, aren't very kind to the user, and that a package such as *keyval* offers a more attractive user interface. We now consider the packages that the macro programmer might use, to create such a user interface.

The simplest key-value processor (for LaTeX, at least) remains *keyval*; it has a command \define@key to declare a key and a *handler* to process it, and a macro \setkeys to offer values to the handler of one or more keys. Thus:

```
\define@key{my}{foo}{Foo is #1\par}
\define@key{my}{bar}[99]{Bar is #1\par}
...
\setkeys{my}{foo=3,bar}
```

will produce output saying:

Foo is 3

Bar is 99

This has defined two keys '`foo`' and '`bar`' in family '`my`', and then executed them, the first with argument '`3`' and the second with no argument, so that the default value of '`99`' is picked up. In effect, the two calls to `\define@key` are simply defining commands, as (for example):

```
\newcommand{\KV@my@foo}[1]{Foo is #1}
```

(the definition of `\KV@my@bar` is similar, but trickier). The command `\setkeys` knows how to find those commands when it needs to process each key — it is easy to regurgitate the structure of the command name, with family name ('`my`', here) after the first '`@`', and the key name after the second '`@`'. (The '`KV`' part is fixed, in *keyval*.)

These simple commands are enough, in fact, to process the botanical example offered as replacement for multi-argument commands in the question mentioned above, or the optional arguments of the `\includegraphics` command of the *graphicx* package. (The last is, in fact, what *keyval* was designed to do.)

However, we need more if we're to to have package options in 'key-value' form. Packages like *hyperref* have enormously complicated package options which need key-value processing at `\ProcessOptions` time: *keyval* can't do that on its own.

Heiko Oberdiek's *kvoptions* package comes to our help: it enables the programmer to declare class or package options that operate as key and value pairs. The package defines commands `\DeclareBoolOption` for options whose value should be either *true* or *false*, and `\DeclareStringOption` for all other options that have a value. Keys are declared using *keyval* and may remain available for use within the document, or may be 'cancelled' to avoid confusion. If you have loaded *kvoptions*, the LaTeX kernel's `\DeclareOption` becomes `\DeclareVoidOption` (it's an option with no value), and `\DeclareOption*` becomes `\DeclareDefaultOption`.

Heiko also provides *kvsetkeys* which is a more robust version of *setkeys*, with some of the rough edges made smoother.

Hendri Adriaens' *xkeyval* offers more flexibility than the original *keyval* and is more robust than the original, too. Like *kvoptions*, the package also has mechanisms to allow class and package options in key-value form (macros `\DeclareOptionX`, `\ExecuteOptionsX` and `\ProcessOptionsX`. *Pstricks* bundle packages use a *xkeyval* derivative called *pst-xkey* for their own key-value manipulation.

The (widely-respected) *pgf* graphics package has its own key-value package called *pgfkeys*. The documentation of the package (part of the huge *pgf* manual, in part 5, "utilities") contains a useful comparison with other key-value systems; some notable differences are:

- key organisation: *pgfkeys* uses a tree structure, while *keyval* and *xkeyval* both associate keys with a family;
- *pgfkeys* supports multi-argument key code; and
- *pgfkeys* can support call-backs when an unknown key appears (these things are called *handlers*.

Keys are organized in a tree that is reminiscent of the Unix fille tree. A typical key might be, `/tikz/coordinate system/x` or just `/x`. When you specify keys you can provide the complete path of the key, but you usually just provide the name of the key (corresponding to the file name without any path) and the path is added automatically. So a `\pgfkeys` command might be:

```
\pgfkeys{/my key=hello,/your keys/main key=something\strange,
  key name without path=something else}
```

and for each key mentioned, the associated code will be executed. ... and that code is also set up using `\pgfkeys`:

```
\pgfkeys{/my key/.code=The value is '#1'.}
```

after which

```
\pgfkeys{/my key=hi!}
```

will produce just

> The value is 'hi!'.

The manual goes on, showing how to define a key with two arguments, how to provide default value for a key, and how to define aliases for particular key sequences (which are called "styles"). All in all, it seems a well thought-out system, offering a lot of flexibility that isn't available with the other keys packages. However, there seems to be no mechanism for using *pgfkeys* keys as part of the options of another package, in the way that *kvoptions* does.

The *l3kernel* programming layer for LaTeX3 includes the *l3keys* module. Inspired by *pgfkeys*, it provides a keyval-based method for the programmer to create keys. As with keyval and derivatives, *l3keys* uses separate macros for defining and setting keys. The package *l3keys2e* makes it possible for LaTeX2e class and package options to be processed using *l3keys*. *L3kernel* code can be used within existing LaTeX2e documents, so *l3keys* is also available to the LaTeX2e programmer 'direct'.

Another key-value system that's part of larger set of macros is *scrbase*, which uses the facilities of *keyval* to build a larger set of facilities, originally for use within the *KOMA-script* bundle. For English-speaking authors, there are difficulties from the German-only documentation; however, from a partial translation available to the author of this answer, a summary is possible. The package may build on the facilities either of *kyeval* or of *xkeyval*, and builds its functionality on the structure of the 'key family'. The user may define family 'members' and keys are defined relative to the members. (For example, the package *scrbase* is part of the *KOMA-script* bundle; so its keys are all members of the *scrbase.sty* family within the *KOMA* family. The function `\FamilyProcessOptions` allows the programmer to decode the options of the package in terms of the package's key family. Note that there is no special provision made for "traditional" package options, as in the *kvoptions* package.

This brief summary was guided by input from two sources: a draft article for *TUGboat* by Joseph Wright, and the partial translation of the documentation of package *scrbase* prepared by Philipp Stephani.

All the above are (at least) aimed at LaTeX programming; there is one package, *getoptk*, aimed at the Plain TeX programmer. *Getoptk* uses syntax inspired by that offered by TeX primitives such as `\hrule` and `\hbox`, so we are offered syntax such as:

```
\begindisplay file {chapter1} literal offset 20pt
```

(taken from the package manual).

There are (we know) people who would swear that such syntax is wonderful (the present author wouldn't), but the package earns its place as the only stand-alone key-value macros that will work in Plain TeX.

getoptk.tex: macros/plain/contrib/getoptk

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

kvoptions.sty: Distributed as part of macros/latex/contrib/oberdiek

kvsetkeys.sty: Distributed as part of macros/latex/contrib/oberdiek

l3keys.sty: Distributed as part of macros/latex/contrib/l3kernel

l3keys2e.sty: Distributed as part of macros/latex/contrib/l3packages

pgfkeys.sty: Distributed as part of graphics/pgf

scrbase.sty: Distributed as part of macros/latex/contrib/koma-script

xkeyval.sty: macros/latex/contrib/xkeyval

334 Defining characters as macros

Single characters can act as macros (defined commands), and both Plain TeX and LaTeX define the character "~" as a "non-breakable space". A character is made definable, or "active", by setting its *category code* (catcode) to be `\active` (13):

```
\catcode`\_=\active
```

Any character could, in principle, be activated this way and defined as a macro:

```
\def_{\_}
```

which could be characterised as an over-simple answer to using underscores. However, you must be wary: whereas people expect an active tilde, other active characters may be unexpected and interact badly with other macros. Furthermore, by defining an active character, you preclude the character's use for other purposes, and there are few characters "free" to be subverted in this way.

To define the character "z" as a command, one would say something like:

```
\catcode`\z=\active
\def z{Yawn, I'm tired}%
```

and each subsequent "z" in the text would become a yawn. This would be an astoundingly bad idea for most documents, but might have special applications. (Note that, in "`\def z`", "z" is no longer interpreted as a letter; the space is therefore not necessary — "`\defz`" would do; we choose to retain the space, for what little clarity we can manage.) Some LaTeX packages facilitate such definitions. For example, the *shortvrb* package with its `\MakeShortVerb` command.

TeX uses category codes to interpret characters as they are read from the input. *Changing a catcode value will not affect characters that have already been read*. Therefore, it is best if characters have fixed category codes for the duration of a document. If catcodes are changed for particular purposes (the `\verb` command does this), then the altered characters will not be interpreted properly when they appear in the argument to another command (as, for example, in `\verb` in command arguments). An exemplary case is the *doc* package, which processes .dtx files using the *shortvrb* package to define `|...|` as a shorthand for `\verb|...|`. But `|` is also used in the preambles of tabular environments, so that tables in `.dtx` files can only have vertical line separation between columns by employing special measures of some sort.

Another consequence is that catcode assignments made in macros often don't work as expected (Active characters in command arguments). For example, the definition

```
\def\mistake{%
\catcode`_=\active
\def_{\textunderscore\-}%
}
```

does not work because it attempts to define an ordinary _ character: When the macro is used, the category change does not apply to the underscore character already in the macro definition. Instead, one may use:

```
\begingroup
\catcode`_=\active
\gdef\works{%    note the global \gdef
  \catcode`_=\active
  \def_{\textunderscore\-}%
}
\endgroup
```

The alternative ("tricksy") way of creating such an isolated definition depends on the curious properties of `\lowercase`, which changes characters without altering their catcodes. Since there is always *one* active character ("~"), we can fool `\lowercase` into patching up a definition without ever explicitly changing a catcode:

```
\begingroup
  \lccode`\~=`\_
  \lowercase{\endgroup
    \def~{\textunderscore\-}%
  }%
```

The two definitions have the same overall effect (the character is defined as a command, but the character does not remain active), except that the first defines a `\global` command.

For active characters to be used only in maths mode, it is much better to leave the character having its ordinary catcode, but assign it a special active *maths code*, as with

```
\begingroup
  \lccode`~=`x
  \lowercase{\endgroup
    \def~{\times}%
  }%
\mathcode`x="8000
```

The special character does not need to be redefined whenever it is made active — the definition of the command persists even if the character's catcode reverts to its original value; the definition becomes accessible again if the character once again becomes active.

doc.sty: Part of the LaTeX distribution macros/latex/base

shortvrb.sty: Part of the LaTeX distribution macros/latex/base

335 Active characters in command arguments

Occasionally, it's nice to make one or two characters active in the argument of a command, to make it easier for authors to code the arguments.

Active characters *can* be used safely in such situations; but care is needed.

An example arose while this answer was being considered: an aspirant macro writer posted to `comp.text.tex` asking for help to make # and b produce musical sharp and flat signs, respectively, in a macro for specifying chords.

The first problem is that both # and b have rather important uses elsewhere in TeX (to say the least!), so that the characters can only be made active while the command is executing.

Using the techniques discussed in characters as commands, we can define:

```
\begingroup
  \catcode`\#=\active
  \gdef#{$\sharp$}
\endgroup
```

and:

```
\begingroup
  \lccode`\~=`\b
  \lowercase{\endgroup
    \def~{$\flat$}%
  }
```

The second problem is one of timing: the command has to make each character active *before* its arguments are read: this means that the command can't actually "have" arguments itself, but must be split in two. So we write:

```
\def\chord{%
  \begingroup
    \catcode`\#=\active
    \catcode`\b=\active
    \Xchord
}
\def\Xchord#1{%
    \chordfont#1%
  \endgroup
}
```

and we can use the command as `\chord{F#}` or `\chord{Bb minor}`.

Two features of the coding are important:

- `\begingroup` in `\chord` opens a group that is closed by `\endgroup` in `\Xchord`; this group limits the change of category codes, which is the *raison d'être* of the whole exercise.
- Although # is active while `\Xchord` is executed, it's *not* active when it's being defined, so that the use of `#1` doesn't require any special attention.

Note that the technique used in such macros as `\chord`, here, is analogous to that used in such commands as `\verb`; and, in just the same way as `\verb` (see `\verb` doesn't work in arguments), `\chord` won't work inside the argument of another command (the error messages, if they appear at all, will probably be rather odd).

336 Defining a macro from an argument

It's common to want a command to create another command: often one wants the new command's name to derive from an argument. LaTeX does this all the time: for example, `\newenvironment` creates start- and end-environment commands whose names are derived from the name of the environment command.

The (seemingly) obvious approach:

```
\def\relay#1#2{\def\#1{#2}}
```

doesn't work (the TeX engine interprets it as a rather strange redefinition of `\#`). The trick is to use `\csname`, which is a TeX primitive for generating command names from random text, together with `\expandafter`. The definition above should read:

```
\def\relay#1#2{%
  \expandafter\def\csname #1\endcsname{#2}%
}
```

With this definition, `\relay{blah}{bleah}` is equivalent to `\def\blah{bleah}`.

Note that the definition of `\relay` omits the braces round the 'command name' in the `\newcommand` it executes. This is because they're not necessary (in fact they seldom are), and in this circumstance they make the macro code slightly more tedious.

The name created need not (of course) be *just* the argument:

```
\def\newrace#1#2#3{%
  \expandafter\def\csname start#1\endcsname{%
    #2%
  }%
  \expandafter\def\csname finish#1\endcsname{%
    #3%
  }%
}
```

With commands

```
\def\start#1{\csname start#1\endcsname}
\def\finish#1{\csname finish#1\endcsname}
```

these 'races' could behave a bit like LaTeX environments.

337 Transcribing LaTeX command definitions

At several places in this FAQ, questions are answered in terms of how to program a LaTeX macro. Sometimes, these macros might also help users of Plain TeX or other packages; this answer attempts to provide a rough-and-ready guide to transcribing such macro definitions for use in other packages.

The reason LaTeX has commands that replace `\def`, is that there's a general philosophy within LaTeX that the user should be protected from himself: the user has different commands according to whether the command to be defined exists (`\renewcommand`) or not (`\newcommand`), and if its status proves not as the user expected, an error is reported. A third definition command, `\providecommand`, only defines if the target is not already defined; LaTeX has no direct equivalent of `\def`, which ignores the present state of the command. The final command of this sort is `\DeclareRobustCommand`, which creates a command which is "robust" (i.e., will not expand if subjected to LaTeX "protected expansion"); from the Plain TeX user's point of view, `\DeclareRobustCommand` should be treated as a non-checking version of `\newcommand`.

LaTeX commands are, by default, defined `\long`; an optional `*` between the `\newcommand` and its (other) arguments specifies that the command is *not* to be defined `\long`. The `*` is detected by a command `\@ifstar` which uses `\futurelet` to switch between two branches, and gobbles the `*`: LaTeX users are encouraged to think of the `*` as part of the command name.

LaTeX's checks for unknown command are done by `\ifx` comparison of a `\csname` construction with `\relax`; since the command name argument is the desired control sequence name, this proves a little long-winded. Since `#1` is the requisite argument, we have:

```
\expandafter\ifx
  \csname\expandafter\@gobble\string#1\endcsname
  \relax
    ...
```

(`\@gobble` simply throws away its argument).

The arguments of a LaTeX command are specified by two optional arguments to the defining command: a count of arguments (0–9: if the count is 0, the optional count argument may be omitted), and a default value for the first argument, if the defined command's first argument is to be optional. So:

```
\newcommand\foo{...}
\newcommand\foo[0]{...}
\newcommand\foo[1]{...#1...}
\newcommand\foo[2][boo]{...#1...#2...}
```

In the last case, `\foo` may be called as `\foo{goodbye}`, which is equivalent to `\foo [boo]{goodbye}` (employing the default value given for the first argument), or as `\foo [hello]{goodbye}` (with an explicit first argument).

Coding of commands with optional arguments is exemplified by the coding of the last `\foo` above:

```
\def\foo{\futurelet\next\@r@foo}
\def\@r@foo{\ifx\next[%
    \let\next\@x@foo
  \else
    \def\next{\@x@foo[boo]}%
  \fi
  \next
}
\def\@x@foo[#1]#2{...#1...#2...}
```

338 Detecting that something is empty

Suppose you need to know that the argument of your command is empty: that is, to distinguish between `\cmd{}` and `\cmd{blah}`. This is pretty simple:

```
\def\cmd#1{%
  \def\tempa{}%
  \def\tempb{#1}%
  \ifx\tempa\tempb
    <empty case>
  \else
    <non-empty case>
  \fi
}
```

The case where you want to ignore an argument that consists of nothing but spaces, rather than something completely empty, is more tricky. It's solved in the code fragment *ifmtarg*, which defines commands `\@ifmtarg` and `\@ifnotmtarg`, which examine their first argument, and select (in opposite directions) their second or third argument. The package's code also appears in the LaTeX *memoir* class.

Ifmtarg makes challenging reading; there's also a discussion of the issue in number two of the "around the bend" articles by the late lamented Mike Downes.

Around the bend series: `info/challenges/aro-bend`

`ifmtarg.sty`: `macros/latex/contrib/ifmtarg`

`memoir.cls`: `macros/latex/contrib/memoir`

339 Am I using PDFTeX, XeTeX or LuaTeX?

You often need to know what "engine" your macros are running on (the engine is the TeX-derivative or TeX-alike processor that typesets your document). The reason that you need to know is that the set of functions available in each engine is different. Thus, for TeX macros to run on any engine, they need to "know" what they can and cannot do, which depends on the engine in use. Getting the right answer is surprisingly tricky (see below for an elaboration of one apparently simple test).

There is now a comprehensive set of packages that answer the question for you. They all create a TeX conditional command:

- *ifpdf* creates a command `\ifpdf`,
- *ifxetex* creates a command `\ifxetex` and
- *ifluatex* creates a command `\ifluatex`.

These TeX commands may be used within the LaTeX conditional framework, as (for example):

```
\ifthenelse{\boolean{pdf}}{⟨if pdf⟩}{⟨if not pdf⟩}
```

The *ifxetex* package also provides a command `\RequireXeTeX` which creates an error if the code is not running under XeTeX; while the other packages don't provide such a command, it's not really difficult to write one for yourself.

Now for those who want to do the job for themselves: here's a discussion of doing the job for PDFTeX and `\ifpdf` — the eager programmer can regenerate `\ifxetex` or `\ifluatex` in the same fashion. It's not recommended…

Suppose you need to test whether your output will be PDF or DVI. The natural thing is to check whether you have access to some PDFTeX-only primitive; a good one to try (not least because it was present in the very first releases of PDFTeX) is `\pdfoutput`. So you try

```
\ifx\pdfoutput\undefined
  ... % not running PDFTeX
\else
  ... % running PDFTeX
\fi
```

Except that neither branch of this conditional is rock-solid. The first branch can be misleading, since the "awkward" user could have written:

```
\let\pdfoutput\undefined
```

so that your test will falsely choose the first alternative. While this is a theoretical problem, it is unlikely to be a major one.

More important is the user who loads a package that uses LaTeX-style testing for the command name's existence (for example, the LaTeX *graphics* package, which is useful even to the Plain TeX user). Such a package may have gone ahead of you, so the test may need to be elaborated:

```
\ifx\pdfoutput\undefined
  ... % not running PDFTeX
\else
  \ifx\pdfoutput\relax
    ... % not running PDFTeX
  \else
    ... % running PDFTeX
  \fi
\fi
```

If you only want to know whether some PDFTeX extension (such as marginal kerning) is present, you can stop at this point: you know as much as you need.

However, if you need to know whether you're creating PDF output, you also need to know about the value of `\pdfoutput`:

```
\ifx\pdfoutput\undefined
  ... % not running PDFTeX
\else
  \ifx\pdfoutput\relax
    ... % not running PDFTeX
  \else
    % running PDFTeX, with...
    \ifnum\pdfoutput>0
      ... % PDF output
    \else
      ... % DVI output
    \fi
  \fi
\fi
```

ifpdf.sty: Distributed as part Heiko Oberdiek's bundle macros/latex/contrib/oberdiek

ifluatex.sty: Distributed as part of Heiko Oberdiek's bundle macros/latex/contrib/oberdiek

ifxetex.sty: macros/generic/ifxetex

340 Subverting a token register

A common requirement is to "subvert" a token register that other macros may use. The requirement arises when you want to add something to a system token register (`\output` or `\every*`), but know that other macros use the token register, too. (A common requirement is to work on `\everypar`, but LaTeX changes `\everypar` at every touch and turn.)

The following technique, due to David Kastrup, does what you need, and allows an independent package to play the exact same game:

```
\let\mypkg@@everypar\everypar
\newtoks\mypkg@everypar
\mypkg@everypar\expandafter{\the\everypar}
\mypkg@@everypar{\mypkgs@ownstuff\the\mypkg@everypar}
\def\mypkgs@ownstuff{%
```

```
  <stuff to do at the start of the token register>%
}
\let\everypar\mypkg@everypar
```

As you can see, the package (*mypkg*)

- creates an alias for the existing "system" `\everypar` (which is frozen into any surrounding environment, which will carry on using the original);
- creates a token register to subvert `\everypar` and initialises it with the current contents of `\everypar`;
- sets the "old" `\everypar` to execute its own extra code, as well as the contents of its own token register;
- defines the macro for the extra code; and
- points the token `\everypar` at the new token register.

and away we go.

The form `\mypkg@...` is (sort of) blessed for LaTeX package internal names, which is why this example uses macros of that form.

341 Is this command defined?

Macro sets from the earliest days of TeX programming may be observed to test whether commands exist by using

`\ifx \`*`command`* `\undefined` ⟨*stuff*⟩ ...

(which of course actually tests that the command *doesn't* exist). LaTeX programmers can make use of the internal command

`\@ifundefined{`*`cmd name`*`}{`*`action1`*`}{`*`action2`*`}`

which executes `action1` if the command is undefined, and `action2` if it is defined (*cmd name* is the command name only, omitting the '\' character).

The `\@ifundefined` command is based on the sequence

```
\expandafter \ifx \csname cmd name\endcsname \relax
```

which relies on the way `\csname` works: if the command doesn't exist, it simply creates it as an alias for `\relax`.

So: what is wrong with these techniques?

Using `\undefined` blithely assumes that the command is indeed not defined. This isn't entirely safe; one could make the name more improbable, but that may simply make it more difficult to spot a problem when things go wrong. LaTeX programmers who use the technique will typically employ `\@undefined`, adding a single level of obscurity.

The `\@ifundefined` mechanism has the unfortunate property of polluting the name space: each test that turns out undefined adds a name to the set TeX is holding, and often all those "`\relax`" names serve no purpose whatever. Even so (sadly) there are places in the code of LaTeX where the existence of the `\relax` is relied upon, after the test, so we can't get away from `\@ifundefined` altogether.

David Kastrup offers the (rather tricky)

```
{\expandafter}\expandafter\ifx \csname cmd name\endcsname\relax ...
```

which "creates" the `\relax`-command inside the group of the first `\expandafter`, therefore forgets it again once the test is done. The test is about as good as you can do with macros.

The e-TeX system system comes to our help here: it defines two new primitives:

- `\ifdefined`, which tests whether a thing is defined (the negative of comparing with `\undefined`, as it were), and
- `\ifcsname cmd name\endcsname`, which does the negative of `\@ifundefined` without the `\relax`-command side-effect.

So, in an e-TeX-based system, the following two conditional clauses do the same thing:

```
\ifdefined\foo
  \message{\string\foo\space is defined}%
\else
  \message{no command \string\foo}%
\fi
%
\ifcsname foo\endcsname
  \message{\string\foo\space is defined}%
\else
  \message{no command \string\foo}%
\fi
```

However, after using the LaTeX `\@ifundefined{foo}`..., the conditionals will detect the command as "existing" (since it has been `\let` to `\relax`); so it is important not to mix mechanisms for detecting the state of a command.

Since most distributions nowadays use e-TeX as their base executable for most packages, these two primitives may be expected appear widely in new macro packages.

R.2 LaTeX macro tools and techniques

342 Using Plain or primitive commands in LaTeX

It's well-known that LaTeX commands tend to be more complex, and to run more slowly than, any Plain TeX (or primitive) command that they replace. There is therefore great temptation not to use LaTeX commands when macro programming. Nevertheless, the general rule is that you should use LaTeX commands, if there are seeming equivalents. The exception is when you are sure you know the differences between the two commands and you know that you need the Plain TeX version. So, for example, use `\mbox` in place of `\hbox` unless you know that the extras that LaTeX provides in `\mbox` would cause trouble in your application. Similarly, use `\newcommand` (or one of its relatives) unless you need one of the constructs that cannot be achieved without the use of `\def` (or friends).

As a general rule, any LaTeX text command will start a new paragraph if necessary; this isn't the case with Plain TeX commands, a fact which has a potential to confuse.

The commands `\smallskip`, `\medskip` and `\bigskip` exist both in Plain TeX and LaTeX, but behave slightly differently: in Plain TeX they terminate the current paragraph, but in LaTeX they don't. The command `\line` is part of picture mode in LaTeX, whereas it's defined as "`\hbox to \hsize`" in Plain TeX. (There's no equivalent for users of the Plain TeX command in LaTeX: an equivalent appears as the internal command `\@@line`).

Maths setting shows a case where the LaTeX version *is* essentially equivalent to the TeX primitive commands: the LaTeX `\( ... \)` does essentially no different to the TeX `$ ... $` (except for checking that you're not attempting to open a maths environment when you're already in one, or vice versa). However, `\[ ... \]` *isn't* the same as `$$ ... $$`: the TeX version, used in LaTeX, can miss the effect of the class option `fleqn`.

Font handling is, of course, wildly different in Plain TeX and LaTeX. Plain TeX's font loading command (`\font\foo=`⟨*fontname*⟩) and its LaTeX equivalent (`\newfont`) should be avoided wherever possible. They are only safe in the most trivial contexts, and are potential sources of great confusion in many circumstances. Further discussion of this issue may be found in "What's wrong with `\newfont`?".

343 `\@` and `@` in macro names

Macro names containing `@` are *internal* to LaTeX, and without special treatment just don't work in ordinary use. A nice example of the problems caused is discussed in `\@` in vertical mode".

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 some other way (for example, using \input), you can probably 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 first edition of the The LaTeX Companion are full of fragments of packages as examples for you to employ. The second edition of the *Companion* makes clearer how you should use these fragments, and in addition, the code of all the examples is now available on CTAN. To see the technique in practice, look at the example below, from file 2-2-7.ltx in the *Companion* examples directory:

```
\makeatletter
\renewcommand\subsection{\@startsection
  {subsection}{2}{0mm}%name, level, indent
  {-\baselineskip}%             beforeskip
  {0.5\baselineskip}%            afterskip
  {\normalfont\normalsize\itshape}}% style
\makeatother
```

(That example appears on page 29 of *The LaTeX Companion*, second edition.)

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 way you hide your LaTeX internal code somewhere that LaTeX internal code is expected, which often looks 'tidier'.

Examples from the Companion: info/examples/tlc2

344 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.) Candidates 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 headings and figure captions are the most prominent examples; there's a complete list in Lamport's book (see TeX-related books).

What's going on really, behind the scenes? The commands in moving arguments are normally expanded to their internal structure during the process of saving. Sometimes this expansion results in invalid TeX code, which shows either during expansion or when the code is processed again. Protecting a command, using "\protect\cmd" tells LaTeX to save \cmd as \cmd, without expanding it at all.

So, 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.

Lamport's book says in its description of every LaTeX command whether it is 'robust' or 'fragile'; it also says that every command with an optional argument is fragile. The list isn't reliable, and neither is the assertion about optional arguments; the statements may have been true in early versions of LaTeX2e but are not any longer necessarily so:

- Some fragile commands, such as \cite, have been made robust in later revisions of LaTeX.
- Some commands, such as \end and \nocite, are fragile even though they have no optional arguments.
- The "user's way" of creating a command with an optional argument (using \newcommand or \newcommand*) now always creates a robust command (though

macros without optional arguments may still be fragile if they do things that are themselves fragile).
- There is no reason that a package author should not also make robust commands with optional arguments as part of the package.
- Some robust commands are redefined by certain packages to be fragile (the `\cite` command commonly suffers this treatment).

Further, simply "hiding" a fragile command in another command, has no effect on fragility. So, if `\fred` is fragile, and you write:

```
\newcommand{\jim}{\fred}
```

then `\jim` is fragile too. There is, however, the `\newcommand`-replacement `\DeclareRobustCommand`, which *always* creates a robust command (whether or not it has optional arguments). The syntax of `\DeclareRobustCommand` is substantially identical to that of `\newcommand`, and if you do the wrapping trick above as:

```
\DeclareRobustCommand{\jim}{\fred}
```

then `\jim` is robust.

Finally, we have the *makerobust* package, which defines `\MakeRobustCommand` to convert a command to be robust. With the package, the "wrapping" above can simply be replaced by:

```
\MakeRobustCommand\fred
```

Whereafter, `\fred` is robust. Using the package may be reasonable if you have lots of fragile commands that you need to use in moving arguments.

In short, the situation is confusing. No-one believes this is satisfactory; the LaTeX team have removed the need for protection of some things, but the techniques available in current LaTeX mean that this is an expensive exercise. It remains a long-term aim of the team to remove all need for `\protection`.

makerobust.sty: Distributed as part of Heiko Oberdiek's bundle
macros/latex/contrib/oberdiek

345 \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, and fragile commands with `\protection` are only robust 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 '@' in control sequence names.

346 The definitions of LaTeX commands

There are several reasons to want to know the definitions of LaTeX commands: from the simplest "idle curiosity", to the pressing need to patch something to make it "work the way you want it". None of these are *pure* motives, but knowledge and expertise seldom arrive through the purest of motives.

Using a TeX executable of some sort, the simple answer is to try `\show`, in a run that is taking commands from the terminal:

```
*\makeatletter
*\show\protected@edef
> \protected@edef=macro:
->\let \@@protect \protect
  \let \protect \@unexpandable@protect
  \afterassignment \restore@protect \edef .
```

(I've rearranged the output there, from the rather confused version TeX itself produces.)

We may perhaps, now, wonder about `\@unexpandable@protect`:

```
 *\show\@unexpandable@protect
 > \@unexpandable@protect=macro:
->\noexpand \protect \noexpand .
```

and we're starting to see how one part of the `\protect`ion mechanism works (one can probably fairly safely guess what `\restore@protect` does).

Many kernel commands are declared robust:

```
 *\show\texttt
 > \texttt=macro:
->\protect \texttt  .
```

so that `\show` isn't much help. Define a command `\pshow` as shown below, and simply execute the command to find its definition:

```
*\def\pshow#1{{\let\protect\show #1}}
*\pshow\texttt
> \texttt =\long macro:
#1->\ifmmode \nfss@text {\ttfamily #1}%
    \else \hmode@bgroup \text@command {#1}%
          \ttfamily \check@icl #1\check@icr
    \expandafter \egroup \fi .
```

Note that the command name that is protected is the 'base' command, with a space appended. This is cryptically visible, in a couple of places above. (Again, the output has been sanitised.)

The command *texdef* (or *latexdef* — the same command with a different name) will do all that for you and return the results slightly more tidily than LaTeX itself manages. For example:

```
latexdef texttt
```

gives:

```
$ latexdef texttt

\texttt:
macro:->\protect \texttt

\texttt :
#1->\ifmmode \nfss@text {\ttfamily #1}%
    \else \hmode@bgroup \text@command {#1}%
          \ttfamily \check@icl #1\check@icr
    \expandafter \egroup \fi .
```

(again, the output has been sanitised — but we see that *latexdef* has useful 'intelligence' in it, as it has spotted and dealt with the `\protect`.)

If one has a malleable text editor, the same investigation may be conducted by examining the file `latex.ltx` (which is usually to be found, in a TDS system, in directory `tex/latex/base`).

In fact, `latex.ltx` is the product of a *docstrip* process on a large number of `.dtx` files, and you can refer to those instead. The LaTeX distribution includes a file `source2e.tex`, and most systems retain it, again in `tex/latex/base`. `Source2e.tex` may be processed to provide a complete source listing of the LaTeX kernel (in fact the process isn't entirely straightforward, but the file produces messages advising you what to do). The result is a huge document, with a line-number index of control sequences the entire kernel and a separate index of changes recorded in each of the files since the LaTeX team took over.

The printed kernel is a nice thing to have, but it's unwieldy and sits on my shelves, seldom used. One problem is that the comments are patchy: the different modules range from well and lucidly documented, through modules documented only through an automatic process that converted the documentation of the source of LaTeX 2.09, to modules that hardly had any useful documentation even in the LaTeX 2.09 original.

In fact, each kernel module `.dtx` file will process separately through LaTeX, so you don't have to work with the whole of `source2e`. You can easily determine which module defines the macro you're interested in: use your "malleable text editor" to find the definition in `latex.ltx`; then search backwards from that point for a line that starts `%%% From File:` — that line tells you which `.dtx` file contains the definition you are interested in. Doing this for `\protected@edef`, we find:

```
%%% From File: ltdefns.dtx
```

When we come to look at it, `ltdefns.dtx` proves to contain quite a dissertation on the methods of handling `\protection`; it also contains some automatically-converted LaTeX 2.09 documentation.

And of course, the kernel isn't all of LaTeX: your command may be defined in one of LaTeX's class or package files. For example, we find a definition of `\thebibliography` in *article*, but there's no `article.dtx`. Some such files are generated from parts of the kernel, some from other files in the distribution. You find which by looking at the start of the file: in `article.cls`, we find:

```
%% This is file ‘article.cls’,
%% generated with the docstrip utility.
%%
%% The original source files were:
%%
%% classes.dtx  (with options: ‘article’)
```

so we need to format `classes.dtx` to see the definition in context.

All these .dtx files are on CTAN as part of the main LaTeX distribution.

LaTeX distribution: macros/latex/base

`texdef`, aka `latexdef`: support/texdef

347 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:

```
\documentclass{article}
\newcommand\thing[2][\DefaultOpt]{%
  \def\DefaultOpt{#2}%
  optional arg: #1,  mandatory arg: #2%
}
\begin{document}
\thing{manda}% #1=#2

\thing[opti]{manda}% #1="opti"
\end{document}
```

LaTeX itself has a trickier (but less readily understandable) method, using a macro `\@dblarg`; inside LaTeX, the example above would have been programmed:

```
\newcommand\thing{\@dblarg\@thing}
\newcommand\@thing[2][\@error]{%
  optional arg: #1,  mandatory arg: #2%
}
```

In that code, `\@thing` is only ever called with an optional and a mandatory argument; if the default from the `\newcommand` is invoked, a bug in user code has bitten…

348 More than one optional argument

If you've already read "breaking the 9-argument limit". you can probably guess the "simple" 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 without, 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.

Variants of `\newcommand` (and friends), with names like `\newcommandtwoopt`, are available in the *twoopt* package. However, if you can, it's probably better to learn to write the commands yourself, just to see why they're not even a good idea from the programming point of view.

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", which offers an alternative way forward.

If you must, however, consider the *optparams* or *xargs* packages. *Optparams* provides a `\optparams` command that you use as an intermediate in defining commands with up to nine optional arguments. The documentation shows examples of commands with four optional arguments (and this from an author who has his own key-value package!).

The *xargs* package uses a key-value package (*xkeyval*) to *define* the layout of the optional arguments. Thus

```
\usepackage{xargs}
...
\newcommandx{\foo}[3][1=1, 3=n]{...}
```

defines a command `\foo` that has an optional first argument (default 1), a mandatory second argument, and an optional third argument (default n).

An alternative approach is offered by Scott Pakin's *newcommand* program, which takes a command name and a definition of a set of command arguments (in a fairly readily-understood language), and emits (La)TeX macros which enable the command to be defined. The command requires that a *Python* interpreter (etc.) be installed on your computer.

newcommand.py: support/newcommand

optparams.sty: Distributed as part of macros/latex/contrib/sauerj

twoopt.sty: Distributed as part of macros/latex/contrib/oberdiek *xargs.sty*: macros/latex/contrib/xargs

xkeyval.sty: macros/latex/contrib/xkeyval

349 Commands defined with * options

LaTeX commands commonly have "versions" defined with an asterisk tagged onto their name: for example `\newcommand` and `\newcommand*` (the former defines a `\long` version of the command).

The simple-minded way for a user to write such a command involves use of the *ifthen* package:

```
\newcommand{\mycommand}[1]{\ifthenelse{\equal{#1}{*}}%
  {\mycommandStar}%
  {\mycommandNoStar{#1}}%
}
\newcommand{\mycommandStar}{starred version}
\newcommand{\mycommandNoStar}[1]{normal version}
```

This does the trick, for sufficiently simple commands, but it has various tiresome failure modes, and it requires `\mycommandnostar` to take an argument.

Of course, the LaTeX kernel has something slicker than this:

```
\newcommand{\mycommand}{\@ifstar
                     \mycommandStar%
                     \mycommandNoStar%
}
\newcommand{\mycommandStar}[2]{starred version}
\newcommand{\mycommandNoStar}[1]{normal version}
```

(Note that arguments to `\mycommandStar` and `\mycommandNoStar` are independent — either can have their own arguments, unconstrained by the technique we're using, unlike the trick described above.) The `\@ifstar` trick is all very well, is fast and efficient, but it requires the definition to be `\makeatletter` protected.

A pleasing alternative is the *suffix* package. This elegant piece of code allows you to define variants of your commands:

```
\newcommand\mycommand{normal version}
\WithSuffix\newcommand\mycommand*{starred version}
```

The package needs e-LaTeX, but any new enough distribution defines LaTeX as e-LaTeX by default. Command arguments may be specified in the normal way, in both command definitions (after the "`*`" in the `\WithSuffix` version). You can also use the TeX primitive commands, creating a definition like:

```
\WithSuffix\gdef\mycommand*{starred version}
```

For those of an adventurous disposition, a further option is to use the *xparse* package from the *l3packages* distribution. The package defines a bunch of commands (such as `\NewDocumentCommand`) which are somewhat analagous to `\newcommand` and the like, in LaTeX2e. The big difference is the specification of command arguments; for each argument, you have a set of choices in the command specification. So, to create a *-command (in LaTeX2e style), one might write:

```
\NewDocumentCommand \foo { s m } {%
  % #1 is the star indicator
  % #2 is a mandatory argument
  ...
}
```

The "star indicator" (`s`) argument appears as `#1` and will take values `\BooleanTrue` (if there was a star) or `\BooleanFalse` (otherwise); the other (`m`) argument is a normal TeX-style mandatory argument, and appears as `#2`.

While *xparse* provides pleasing command argument specifications, it *is* part of the LaTeX 3 experimental harness, and simply loading the package "pulls in" a large bunch of other package files via the *expl3* package: these packages provide the rest of the harness.

ifthen.sty: Part of the LaTeX distribution

suffix.sty: Distributed as part of `macros/latex/contrib/bigfoot`

xparse.sty: Distributed as part of `macros/latex/contrib/l3packages`

expl3.sty: Distributed as part of `macros/latex/contrib/l3kernel`

350 LaTeX internal "abbreviations", etc.

In the deeps of time, when TeX first happened, computers had extremely limited memory, and were (by today's standards) painfully slow. When LaTeX came along, things weren't much better, and even when LaTeX2e appeared, there was a strong imperative to save memory space (and to a lesser extent) CPU time.

From the very earliest days, Knuth used shortcut macros to speed things up. LaTeX, over the years, has extended Knuth's list by a substantial amount. An interesting feature of the "abbreviations" is that on paper, they may look longer than the thing they stand for; however, to (La)TeX they *feel* smaller...

The table at the end of this answer lists the commonest of these "abbreviations". It is not complete; as always, if the table doesn't help, try the LaTeX source. The table lists each abbreviation's *name* and its *value*, which provide most of what a user needs to know. The table also lists the abbreviation's *type*, which is a trickier concept: if you need to know, the only real confusion is that the abbreviations labelled 'defn' are defined using an `\xxxxdef` command.

Name	Type	Value
`\m@ne`	count	-1
`\p@`	dimen	1pt
`\z@`	dimen	0pt
`\z@skip`	skip	0pt plus 0pt minus 0pt
`\@ne`	defn	1
`\tw@`	defn	2
`\thr@@`	defn	3
`\sixt@@n`	defn	16
`\@cclv`	defn	255
`\@cclvi`	defn	256
`\@m`	defn	1000
`\@M`	defn	10000
`\@MM`	defn	20000
`\@vpt`	macro	5
`\@vipt`	macro	6
`\@viipt`	macro	7
`\@viiipt`	macro	8
`\@ixpt`	macro	9
`\@xpt`	macro	10
`\@xipt`	macro	10.95
`\@xiipt`	macro	12
`\@xivpt`	macro	14.4
`\@xviipt`	macro	17.28
`\@xxpt`	macro	20.74
`\@xxvpt`	macro	24.88
`\@plus`	macro	"`plus`"
`\@minus`	macro	"`minus`"

351 Defining LaTeX commands within other commands

LaTeX command definition is significantly different from the TeX primitive form discussed in an earlier question about definitions within macros.

In most ways, the LaTeX situation is simpler (at least in part because it imposes more restrictions on the user); however, defining a command within a command still requires some care.

The earlier question said you have to double the # signs in command definitions: in fact, the same rule holds, except that LaTeX already takes care of some of the issues, by generating argument lists for you.

The basic problem is that:

```
\newcommand{\abc}[1]{joy, oh #1!%
  \newcommand{\ghi}[1]{gloom, oh #1!}%
}
```

followed by a call:

```
\cmdinvoke{abc}{joy}
```

typesets "joy, oh joy!", but defines a command `\ghi` that takes one parameter, which it ignores; `\ghi{gloom}` will expand to "gloom, oh joy!", which is presumably not what was expected.

And (as you will probably guess, if you've read the earlier question) the definition:

```
\newcommand{\abc}[1]{joy, oh #1!%
  \newcommand{\ghi}[1]{gloom, oh ##1!}%
}
```

does what is required, and `\ghi{gloom}` will expand to "gloom, oh gloom!", whatever the argument to `\abc`.

The doubling is needed whether or not the enclosing command has an argument, so:

```
\newcommand{\abc}{joy, oh joy!%
  \newcommand{\ghi}[1]{gloom, oh ##1!}%
}
```

is needed to produce a replica of the `\ghi` we defined earlier.

352 How to print contents of variables?

It is often useful to print out the values of variables in the log file or on the terminal. Three possible ways to print out the contents of `\textheight` variable are:

1. `\showthe\textheight`
2. `\message{The text height is \the\textheight}`
3. `\typeout{The text height is \the\textheight}`

These techniques use the TeX primitives `\the` (which provides the value of a variable), `\showthe` (print a variable to the terminal and the log, on a line of its own), and `\message`, which interpolates something into the log. The command `\typeout` is LaTeX's general message output mechanism.

In each case, the variable's value is printed as a number of points.

To typeset the value of `\textheight`, just `\the\textheight` is enough, but a more flexible alternative is to use the *printlen* package. *Printlen* allows you to choose the units in which you print a variable; this is useful, given that the most ordinary people don't think in points (particularly Knuth's points, of which there are 72.27 to the inch).

So, using *printlen*, we could say:

```
\newlength{\foo}
\setlength{\foo}{12pt}
\verb|\foo| is \printlength{\foo}
```

and get:

```
\foo is 12pt
```

while, if we say:

```
\newlength{\foo}
\setlength{\foo}{12pt}
\uselengthunit{mm}
\verb|foo| is \printlength{\foo}
```

we get:

```
\foo is 4.21747mm
```

`printlen.sty`: `macros/latex/contrib/printlen`

353 Using labels as counter values

Labels are tempting sources of 'numbers' — their most common use, after all, is simply to typeset a number. However, their seeming simplicity is deceptive; the packages *babel* and *hyperref*, at least, fiddle with the definition of `\ref` and `\pageref` in ways that make

```
\setcounter{foo}{\ref{bar}}
```

(etc.) not work; thus the technique may not be relied upon.

The solution is to use the *refcount* package (incidentally, by the author of *hyperref*). The package provides four commands, all similar to:

```
\usepackage{refcount}
...
\label{bar}
...
\setcounterref{foo}{bar}
```

(the other three are `\addtocounterref`, `\setcounterpageref` and `\addtocounterpageref`).

The package also provides a command `\getrefnumber{`*`label-name`*`}` that may be used where a 'number' value is needed. For example:

```
... \footnote{foo bar ...\label{foofoot}}
...
\footnotemark[\getrefnumber{foofoot}]
```

which gives you a second footnote mark reference the the footnote. (There is also a command `\getpagerefnumber`, of course).

The commands could be used by one determined not to use *changepage* to determine whether the current page is odd, but it's probably no more trouble to use the fully-developed tool in this case.

`refcount.sty`: Distributed as part of `macros/latex/contrib/oberdiek`

R.3 LaTeX macro programming

354 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 changing *babel* names if you're using it.)

The names that are defined in the standard LaTeX classes (and the *makeidx* package) are listed below. Some of the names are only defined in a subset of the classes (and the *letter* class has a set of names all of its own); the list shows the specialisation of each name, where appropriate.

`\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 caption)

355 Changing the words *babel* uses

LaTeX uses symbolic names for many of the automatically-generated text it produces (special-purpose section headings, captions, etc.). As noted in "LaTeX fixed names" (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}%
}
```

356 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 the *chngcntr* package, which provides commands `\counterwithin` (which establishes this nested numbering relationship) and `\counterwithout` (which undoes it).

So if you have figures numbered by chapter as 1.1, 1.2, 2.1, . . . , the command

```
\counterwithout{figure}{chapter}
```

will convert them to figures 1, 2, 3, ... (Note that the command has also removed the chapter number from the counter's definition.)

More elaborate use could change things numbered per section to things numbered per chapter:

```
\counterwithout{equation}{section}
\counterwithin{equation}{chapter}
```

(assuming there was a class that did such a thing in the first place...)

The *chngcntr* approach doesn't involve much programming, and the enthusiastic LaTeX programmer might choose to try the technique that we had to use before the advent of *chngcntr*. Each of the packages *removefr* and *remreset* defines a `\@removefromreset` command, and having included the package one writes something like:

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

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

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

(remember to do the whole job, for every counter you want to manipulate, within `\makeatletter` . . . `\makeatother`).

This technique, too, may 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:

```
\makeatletter
\@removefromreset{figure}{section}
\@addtoreset{figure}{chapter}
\renewcommand{\thefigure}{\thechapter.\@arabic\c@figure}
\makeatother
```

(the command `\@addtoreset` is a part of LaTeX itself).

chngcntr.sty: macros/latex/contrib/chngcntr

memoir.cls: macros/latex/contrib/memoir

removefr.tex: macros/latex/contrib/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/carlisle

357 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.

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

Another question 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 *changepage* 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 `\ifoddpage` true if the command was issued on an odd page. (The *memoir* class has the same command.) LaTeX users who are unfamiliar with TeX's `\if...` commands may use the *ifthen* package:

```
\usepackage{ifthen,changepage}
...
\checkoddpage
\ifthenelse{\boolean{oddpage}}{<odd page stuff>}{<even page stuff>}
```

Of course, the 'label' contributes to LaTeX's "Rerun to get cross-references right" error messages...

The Koma-Script classes have an `addmargin*` environment that also provides the sorts of facilities that the *changepage* offers. Koma-Script's supporting command:

```
\ifthispageodd{<true>}{<false>}
```

executes different things depending on the page number.

`changepage.sty`: `macros/latex/contrib/changepage`

`ifthen.sty`: Part of the LaTeX distribution: `macros/latex/base`

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

`memoir.cls`: `macros/latex/contrib/memoir`

359 How to change the format of labels

By default, when a label is created, it takes on the appearance of the counter labelled, so the label appears as `\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 — change the structure of list items.) 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 supposedly done using the command `\p@enumii{\theenumii}`. Unfortunately, the internal workings of this aren’t quite right, and you need to patch the `\refstepcounter` command:

```
\renewcommand*\refstepcounter[1]{\stepcounter{#1}%
  \protected@edef\@currentlabel{%
    \csname p@#1\expandafter\endcsname
      \csname the#1\endcsname
  }%
}
```

With the patch in place you can now, for example, change the labels on all inner lists by adding the following code in your preamble:

```
\makeatletter
\renewcommand{\p@enumii}[1]{\theenumi(#1)}
\makeatother
```

This would make the labels for second-level enumerated lists appear as “1(a)” (and so on). The analogous change works for any counter that gets used in a `\label` command.

In fact, the *fncylab* package does all the above (including the patch to LaTeX itself). With the package, the code above is (actually quite efficiently) rendered by the command:

```
\labelformat{enumii}{\theenumi(#1)}
```

In fact, the above example, which we can do in several different ways, has been rendered obsolete by the appearance of the *enumitem* package, which is discussed in the answer about decorating enumeration lists.

enumitem.sty: macros/latex/contrib/enumitem

fncylab.sty: macros/latex/contrib/fncylab

360 Adjusting the presentation of section numbers

The general issues of adjusting the appearance of section headings are pretty complex, and are covered in question the style of section headings.

However, people regularly want merely to change the way the section number appears in the heading, and some such people don’t mind writing out a few macros. This answer is for *them*.

The section number is typeset using the LaTeX internal `\@seccntformat` command, which is given the “name” (section, subsection, ...) of the heading, as argument. Ordinarily, `\@seccntformat` merely outputs the section number, and then a `\quad` of space:

```
\renewcommand*{\@seccntformat}[1]{%
  \csname the#1\endcsname\quad
}
```

Suppose you want to put a stop after every section (subsection, subsubsection, ...) number, a trivial change may be implemented by simple modification of the command:

```
\renewcommand*{\@seccntformat}[1]{%
  \csname the#1\endcsname.\quad
}
```

However, many people want to modify section numbers, but not subsection numbers, or any of the others. To do this, one must make `\@seccntformat` switch according to its argument. The following technique for doing the job is slightly wasteful, but is efficient enough for a relatively rare operation:

```
\renewcommand*{\@seccntformat}[1]{%
  \csname the#1\endcsname
  \csname adddot@#1\endcsname\quad
}
```

which uses a second-level command to provide the dot, if it has been defined; otherwise it merely appends `\relax` (which does nothing in this context). The definition of the second-level command (the version for the `section`, here) specifies what to put after a section number, but it could be used to put anything after it:

```
\newcommand*{\adddot@section}{.}
```

Note that all the command definitions above are dealing in LaTeX internal commands, so the above code should be in a package file, for preference.

The *Koma-script* classes have different commands for specifying changes to section number presentation: `\partformat`, `\chapterformat` and `\othersectionlevelsformat`, but otherwise their facilities are similar to those of "raw" LaTeX.

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

361 There's a space added after my environment

You've written your own environment `env`, and it works except that a space appears at the start of the first line of typeset text after `\end{env}`. This doesn't happen with similar LaTeX-supplied environments.

You could impose the restriction that your users always put a "%" sign after the environment ... but LaTeX environments don't require that, either.

The LaTeX environments' "secret" is an internal flag which causes the unwanted spaces to be ignored. Fortunately, you don't have to use the internal form: since 1996, LaTeX has had a user command `\ignorespacesafterend`, which sets the internal flag.

362 Finding if a label is undefined

People seem to want to know (at run time) if a label is undefined (I don't actually understand *why*, particularly: it's a transient state, and LaTeX deals with it quite well).

A resolved label is simply a command: `\r@`⟨*label-name*⟩; determining if the label is set is then simply a matter of detecting if the command exists. The usual LaTeX internal way of doing this is to use the command `\@ifundefined`:

```
\@ifundefined{r@label-name}{undef-cmds}{def-cmds}
```

In which, ⟨*label-name*⟩ is exactly what you would use in a `\label` command, and the remaining two arguments are command sequences to be used if the label is undefined (⟨*undef-cmds*⟩) or if it is defined (⟨*def-cmds*⟩).

Note that any command that incorporates `\@ifundefined` is naturally fragile, so remember to create it with `\DeclareRobustCommand` or to use it with `\protect` in a moving argument.

If you're into this game, you may well not care about LaTeX's warning about undefined labels at the end of the document; however, if you are, include the command `\G@refundefinedtrue` in ⟨*undef-cmds*⟩.

And of course, remember you're dealing in internal commands, and pay attention to the at-signs.

All the above can be avoided by using the *labelcas* package: it provides commands that enable you to switch according to the state of a single label, or the states of a list of labels. The package's definition is a bit complicated, but the package itself is pretty powerful.

`labelcas.sty`: `macros/latex/contrib/labelcas`

363 Master and slave counters

It's common to have things numbered "per chapter" (for example, in the standard *book* and *report* classes, figures, tables and footnotes are all numbered thus). The process of resetting is done automatically, when the "master" counter is stepped (when the `\chapter` command that starts chapter ⟨*n*⟩ happens, the `chapter` counter is stepped, and all the dependent counters are set to zero).

How would you do that for yourself? You might want to number algorithms per section, or corollaries per theorem, for example. If you're defining these things by hand, you declare the relationship when you define the counter in the first place:

```
\newcounter{new-name}[master]
```

says that every time counter ⟨*master*⟩ is stepped, counter ⟨*new-name*⟩ will be reset.

But what if you have an uncooperative package, that defines the objects for you, but doesn't provide a programmer interface to make the counters behave as you want?

The `\newcounter` command uses a LaTeX internal command, and you can also use it:

```
\@addtoreset{new-name}{master}
```

(but remember that it needs to be between `\makeatletter` and `\makeatother`, or in a package of your own).

The *chngcntr* package encapsulates the `\@addtoreset` command into a command `\counterwithin`. So:

```
\counterwithin*{corrollary}{theorem}
```

will make the corollary counter slave to theorem counters. The command without its asterisk:

```
\counterwithin{corrollary}{theorem}
```

will do the same, and also redefine `\thecorollary` as ⟨*theorem number*⟩.⟨*corollary number*⟩, which is a good scheme if you ever want to refer to the corollaries — there are potentially many "corollary 1" in any document, so it's as well to tie its number to the counter of the theorem it belongs to. This is true of pretty much any such counter-within-another; if you're not using the *chngcntr*, refer to the answer to redefining counters' `\the`-commands for the necessary techniques.

Note that the technique doesn't work if the master counter is `page`, the number of the current page. The `page` counter is stepped deep inside the output routine, which usually gets called some time after the text for the new page has started to appear: so special techniques are required to deal with that. One special case is dealt with elsewhere: footnotes numbered per page. One of the techniques described there, using package *perpage*, may be applied to any counter. The command:

```
\MakePerPage{counter}
```

will cause ⟨*counter*⟩ to be reset for each page. The package uses a label-like mechanism, and may require more than one run of LaTeX to stabilise counter values — LaTeX will generate the usual warnings about labels changing.

`chngcntr.sty`: `macros/latex/contrib/chngcntr`

`perpage.sty`: Distributed as part `macros/latex/contrib/bigfoot`

364 Fonts at arbitrary sizes

Almost all fonts, nowadays, are provided with LaTeX control (`.fd`) files, so the temptation to risk the problems of `\newfont` is usually easy to resist.

However, one temptation remains, arising from the way that LaTeX restricts the sizes of fonts. In fact, the restriction only significantly applies to the default (Computer Modern) and the Cork-encoded (T1) EC fonts, but it is widely considered to be anomalous, nowadays. In recognition of this problem, there is a package *fix-cm* which will allow you to use the fonts, within LaTeX, at any size you choose. If you're not using scaleable versions of the fonts, most modern distributions will just generate an appropriate bitmap for you.

So, suppose you want to produce a heading in Computer Modern Roman at 30 points, you might be tempted to write:

```
\newfont{\bigfont}{cmr10 at 30pt}
\begin{center}
  \bigfont Huge text
\end{center}
```

which will indeed work, but will actually produce a worse result than

```
\usepackage{fix-cm}
...
\begin{center}
  \fontsize{30}{36}\selectfont
  Huge text
\end{center}
```

Note that the *fix-cm* package was not distributed until the December 2003 edition of LaTeX; if you have an older distribution, the packages *type1cm* (for CM fonts) and *type1ec* (for EC fonts) are available.

If *fix-cm* doesn't do the job for you, try one of the other two; *fix-cm* has one or two omissions — fonts the LaTeX team did not consider useful, or something; the CM dunhill fonts (as CM, but with stretched ascenders) and the CM fibonacci (which is only available in 8-point design size) are certainly missing.

A further alternative might be to switch to the *Latin Modern* fonts (which provide a close simulacrum of the *Computer Modern* set); these fonts were scaleable from their first distribution, and don't therefore need any such trick as the above.

`fix-cm.sty`: Distributed as part of `macros/latex/base` (an unpacked version is available at `macros/latex/unpacked/fix-cm.sty`)

Latin Modern fonts: `fonts/lm`

`type1cm.sty`: `macros/latex/contrib/type1cm`

`type1ec.sty`: `fonts/ps-type1/cm-super/type1ec.sty` (the package is actually part of the `fonts/ps-type1/cm-super` distribution, but it works happily in the absence of the scaled fonts)

365 The quality of your LaTeX

The *l2tabu* tutorial (mentioned in online introductions) is undoubtedly a good read.

However, it's always difficult to remember the things you should *not* do, when there are so many things to remember that you really must do: some automation is needed. . . .

The nicely-named *nag* allows you to apply a configurable set of checks to your document, as you run it through LaTeX; you get messages like:

```
Package nag Warning: Command \bf is an old LaTeX 2.09 command.
(nag)                Use \bfseries or \textbf instead on input line 30.
```

(the package provides a demo file which contains most of the sorts of errors you might make — the example is one of them).

The *lacheck* program analyses your source and comments on it; its view of what is "bad" is *very* subjective (the documentation says), but it can be useful.

There's also a web site TeXidate which will do a static analysis of your document (unfortunately, you have to paste your document source into a text window). The site doesn't seem as comprehensive as *nag*, but it allows you to download its script, which you can then juggle with to make it more draconian.

l2tabu: Browse `info/l2tabu` for a copy in a language that's convenient for you

lacheck: `support/lacheck`

nag.sty: `macros/latex/contrib/nag`

S Things are Going Wrong…

S.1 Getting things to fit

366 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.

367 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. e-TeX and Omega both do this, as does MicroPress Inc's VTeX.

It's actually quite practical (with most modern distributions) to use e-TeX's extended register set: use package *etex* (which comes with e-TeX distributions) and the allocation mechanism is altered to cope with the larger register set: PiCTeX will now load.

If you're in some situation where you can't use e-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 Schrell'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? Fortunately for us all, 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 available either as a *Basic* program for DOS, or as a *Perl* program for other systems (including Windows, nowadays).

etex.sty: macros/latex/contrib/etex-pkg

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

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

MathsPic: graphics/mathspic

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

S.2 Making things stay where you want them

368 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". 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, look at the recommendations for "non-floating floats"
- Are the placement parameters on your figures right? The default ("`tbp`") is usually satisfactory, 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 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 may 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 your floats can't wander 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 `\FloatBarriers` wherever you choose.
- If you are bothered by floats appearing at the top of the page (before they are specified in your text), try the *flafter* package, which avoids this problem by insisting that floats should always appear after their definition.
- Have a look at the LaTeX2e *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 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 allows you to increase the number of floating inserts that LaTeX can handle at one time (from its original value of 18, in LaTeX2e). Note that, even with e-TeX's greater supply of registers, there is still a modest limit on the total number of floating inserts (e-TeX increases the total number of registers available, but inserts don't work if they are constructed from registers in the extended range).
  Caveat: if you are using *etex* to increase the number of registers available, you need to "reserve" some inserts for *morefloats*: something like:

  ```
  \usepackage{etex}
  \reserveinserts{18}
  \usepackage{morefloats}
  ```
- 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/endfloat

etex.sty: macros/latex/contrib/etex-pkg

flafter.sty: Part of the LaTeX distribution

float.sty: macros/latex/contrib/float

morefloats.sty: macros/latex/contrib/morefloats

placeins.sty: macros/latex/contrib/placeins

369 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 — thesis styles) 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.) The *soul* package defines an `\ul` command (after which the package is, in part, named) that underlines running text.

Beware of *ulem*'s default behaviour, which is to convert the `\emph` command into an underlining command; this can be avoided by loading the package with:

```
\usepackage[normalem]{ulem}
```

ulem.sty: macros/latex/contrib/ulem

soul.sty: macros/latex/contrib/soul

370 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 (La)TeX page maker, when forming a page, takes account of variables `\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. Therefore, for typical layouts, there are only two sensible settings for the penalties: finite (150 or 500, it doesn't matter which) to allow widows and orphans, and infinite (10000 or greater) to forbid them.

The problem 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 should proceed to 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. Options are

- If the previous page contains a long paragraph with a short last line, it may be possible to set it 'tight': write `\looseness=-1` immediately after the last word of the paragraph.
- If that doesn't work, adjusting the page size, using `\enlargethispage{\baselineskip}` to 'add a line' to the page, which may have the effect of getting 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 that `\looseness=1` (which should increase the line length by one) seldom has the right effect — the looser paragraph typically has a one-word final line, which doesn't look much better than the original widow.

S.3 Things have "gone away"

371 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 LaTeX2e; to use the unconverted LaTeX 2.09 packages under LaTeX2e, you need also to include the *rawfonts* package (which is part of the LaTeX2e distribution).

372 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).

AMS fonts: `fonts/amsfonts`

373 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. There may, of course, still be 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).

A partial re-implementation of the blackboard-bold part of the *msy* font (covering C, N, R, S and Z, only) is available in Type 1 format; if your mathematical needs only extend that far, the font could be a good choice.

If you have a DVI file that requests the fonts, there is a package of virtual fonts to map the old to the new series.

`msam` & `msbm` fonts: Distributed as part of `fonts/amsfonts`

`msym` fonts: `fonts/msym`

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

374 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)[3]. 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, the only reasonable approach is to edit the document to replace *am** font names with *cm**.

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.

You therefore have to fool the system into using *cm** fonts where the original author specified *am**.

One option is the font substitutions that many DVI drivers provide via their configuration file — specify that every *am* font should be replaced by its corresponding *cm* font.

Alternatively, one may try DVI editing — packages *dtl* (DVI Text Language) and *dviasm* (DVI assembler) can both provide round trips from DVI to text and back to DVI. One therefore edits font names (throughout the text representation of the file) in the middle of that round trip.

The DTL text is pretty straightforward, for this purpose: fontnames are in single quotes at the end of lines, so:

[3] 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.

```
dv2dt -o ⟨doc.txt⟩ ⟨doc.dvi⟩
(edit the .txt file)
dt2dv -o ⟨edited.dvi⟩ ⟨edited.txt⟩
```

(you have to compile the C programs for this).

Dviasm is a *Python* script; its output has font names in a section near the start of the document, and then dotted about through the body, so:

```
python dviasm.py -o ⟨doc.txt⟩ ⟨doc.dvi⟩
(edit the .txt file)
python dviasm.py -o ⟨edited.dvi⟩ ⟨edited.txt⟩
```

Both routes seem acceptable ways forward; it is a matter of taste which any particular user may choose (it's not likely that it will be necessary very often...).

dviasm.py: dviware/dviasm

dtl: dviware/dtl

375 What's happened to *initex*?

In the beginning, (La)TeX was stretching the capacity of every system it was ported to, so there was a premium on reducing the size of executables. One way of doing this was to have a separate executable, *initex*, that had things in it that aren't needed in ordinary document runs — notably `\patterns` (which builds hyphenation tables), and `\dump` (which writes out a format).

On modern systems, the size of this code is insignificant in comparison to the memory available, and maintaining separate programs has been found sufficiently error-prone that free Unix-style system distributions have abolished *initex* and its friends and relations such as *inipdftex* in favour of a single executable (that is, just *tex* or *pdftex*) that will "do what *initex* (or whatever) used to do" if it detects the command option "`-ini`".

The change happened with the advent of teTeX version 3.0, which appeared at the beginning of 2005. At that time, TeX Live was following teTeX, so that year's TeX Live distribution would also have dropped *initex*.

It would appear that the equation is somewhat different for the MiKTeX developers, since that system continues to offer an *initex* executable.

T Why does it *do* that?

T.1 Common errors

376 LaTeX gets cross-references wrong

Sometimes, however many times you run LaTeX, the cross-references are just wrong. A likely reason is that you have placed the label before the data for the label was set; if the label is recording a `\caption` command, the `\label` command must appear *after* the `\caption` command, or be part of it. For example:

```
\begin{figure}
  <the illustration itself>
  \caption{My figure}
  \label{myfig}
\end{figure}
```

is correct, as is

```
\begin{figure}
  <the illustration itself>
  \caption{My figure%
    \label{myfig}}
\end{figure}
```

whereas, in

```
\begin{figure}
  <the illustration itself>
  \label{myfig}
  \caption{My figure}
\end{figure}
```

the label will report the number of the section (or whatever) in which the surrounding text resides, or the like.

You can, with the same malign effect, shield the `\caption` command from its associated `\label` command, by enclosing the caption in an environment of its own. This effect will be seen with:

```
\begin{figure}
  <the illustration itself>
  \caption{A Figure}
\end{figure}
\label{myfig}
```

where the `\label` definitely *is* after the `\caption`, but because the `figure` environment closed before the `\label` command, the `\caption` is no longer "visible".

In summary, the `\label` must be *after* the command that defines it (e.g., `\caption`), and if the `\caption` is inside an environment, the `\label` must be in there too.

377 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".

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

which is a usage this FAQ would not recommend, anyway: refer to the reason not to use `eqnarray`.

Note that the *amsmath* package modifies the behaviour of `\\` in maths. With *amsmath*, the `eqnarray` example doesn't need any special measures.

378 Why doesn't verbatim 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` and `\begin {verbatim}` have to assume that they are getting the first look at the parameter text; if they aren't, TeX has already assigned category codes so that the verbatim command 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 the rather more helpful '`\verb` illegal in command argument'. The same sorts of thing happen with `\begin{verbatim}` . . . `\end{verbatim}`:

```
\ifthenelse{\boolean{foo}}{%
\begin{verbatim}
foobar
\end{verbatim}
}{%
\begin{verbatim}
barfoo
\end{verbatim}
}
```

provokes errors like 'File ended while scanning use of `\@xverbatim`', as `\begin {verbatim}` fails to see its matching `\end{verbatim}`.

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 "normal" command parameter, regardless of `\protection`. (The `\verb` command tries hard to detect if you're misusing it; unfortunately, it can't always do so, and the error message is therefore not reliable as an indication of problems.)

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* will help: it doesn't suffer from all the problems of `\verb`, though it's still not robust; "typesetting URLs" offers advice here.
- If you're putting `\verb` into the argument of a boxing command (such as `\fbox`), consider using the `lrbox` environmen)t:

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

If you can't avoid verbatim, the `\cprotect` command (from the package *cprotect*) might help. The package manages to make a macro read a verbatim argument in a "sanitised" way by the simple medium of prefixing the macro with `\cprotect`:

```
\cprotect\section{Using \verb|verbatim|}
```

The package (at the time this author tested it) was still under development (though it *does* work in this simple case) and deserves consideration in most cases; the package documentation gives more details.

Another way out is to use one of "argument types" of the `\NewDocumentCommand` command in the experimental LaTeX3 package *xparse*:

```
\NewDocumentCommand\cmd{ m v m }{#1 ‘#2’ #3}
\cmd{Command }|\furble|{ isn’t defined}
```

Which gives us:

Command `\furble` isn't defined

The "`m`" tag argument specifies a normal mandatory argument, and the "`v`" specifies one of these verbatim arguments. As you see, it's implanting a `\verb`-style command argument in the argument sequence of an otherwise "normal" sort of command; that '`|`' may be any old character that doesn't conflict with the content of the argument.

This is pretty neat (even if the verbatim is in an argument of its own) but the downside is that *xparse* pulls in *expl3* (the experimental LaTeX3 programming environment) which is painfully large.

Other than the *cprotect* package, there are three partial solutions to the problem:

- Some packages have macros which are designed to be responsive to verbatim text in their arguments. For example, the *fancyvrb* package defines a command `\VerbatimFootnotes`, which redefines the `\footnotetext` command, and hence also the behaviour of the `\footnote`) command, 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 of the *footmisc* package.
  The *memoir* class defines its `\footnote` command so that it will accept verbatim in its arguments, without any supporting 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, whose `\verbdef` command 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` will make it work in such a case.
  A robust alternative is:

  ```
  \chardef\us=‘\_
  ...
  \section{... \texttt{my\us name}}
  ```

  Such a definition is 'naturally' robust; the construction "⟨*back-tick*⟩\⟨*char*⟩" may be used for any troublesome character (though it's plainly not necessary for things like percent signs for which (La)TeX already provides robust macros).

cprotect.sty: macros/latex/contrib/cprotect

expl3.sty: Distributed as part of macros/latex/contrib/l3kernel

fancyvrb.sty: macros/latex/contrib/fancyvrb

memoir.cls: macros/latex/contrib/memoir

url.sty: macros/latex/contrib/url

verbdef.sty: macros/latex/contrib/verbdef

xparse.sty: Distributed as part of macros/latex/contrib/l3packages

379 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 *LaTeX Companion* — 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/expdlist

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

380 Two-column float numbers out of order

When LaTeX can't place a float immediately, it places it on one of several "defer" lists. If another float of the same type comes along, and the "defer" list for that type still has something in it, the later float has to wait for everything earlier in the list.

Now, standard LaTeX has different lists for single-column floats, and double-column floats; this means that single-column figures can overtake double-column figures (or vice-versa), and you observe later figures appear in the document before early ones. The same is true, of course, for tables, or for any user-defined float.

The LaTeX team recognise the problem, and provides a package (*fixltx2e*) to deal with it. *Fixltx2e* amalgamates the two defer lists, so that floats don't get out of order.

For those who are still running an older LaTeX distribution, the package *fix2col* should serve. This package (also by a member of the LaTeX team) was the basis of the relevant part of *fixltx2e*. The functionality has also been included in *dblfloatfix*, which also has code to place full-width floats at `[b]` placement.

Once you have loaded the package, no more remains to be done: the whole requirement is to patch the output routine; no extra commands are needed.

dblfloatfix.sty: macros/latex/contrib/dblfloatfix

fix2col.sty: macros/latex/contrib/fix2col

fixltx2e.sty: Part of the LaTeX distribution

381 Accents misbehave in tabbing

So you are constructing a `tabbing` environment, and you have the need of some diacriticised text — perhaps something as simple as `\'{e}` — and the accent disappears because it has been interpreted as a `tabbing` command, and everything goes wrong.

This is really a rather ghastly feature of the `tabbing` environment; in order to type accented characters you need to use the `\a` kludge: so `\a'{e}` inside `tabbing` for `\'{e}` outside, and similarly `\a`` for `\`` and `\a=` for `\=`. This whole procedure is of course hideous and error-prone.

The simplest alternative is to type in an encoding that has the diacriticised characters in it, and to use an appropriate encoding definition file in the *inputenc* package. So for example, type:

```
\usepackage[latin1]{inputenc}
...
\begin{tabbing}
...
... \> voilà \> ...
```

for:

... voilà ...

and the internal mechanisms of the *inputenc* package will put the right version of the accent command in there.

A witty reversal of the rôles is introduced by the package *Tabbing* (note the capital "T"): it provides a `Tabbing` environment which duplicates `tabbing`, but all the single-character commands become complicated objects. So `tabbing`'s `\>` becomes `\TAB>`, `\=` becomes `\TAB=`, and so on. The above trivial example would therefore become:

```
\usepackage{Tabbing}
...
\begin{Tabbing}
  ...   ... \TAB> voil\`a \TAB> ...
```

Tabbing.sty: macros/latex/contrib/Tabbing

382 Package reports "command already defined"

You load a pair of packages, and the second reports that one of the commands it defines is already present. For example, both the *txfonts* and *amsmath* define a command `\iint` (and `\iiint` and so on); so

```
...
\usepackage{txfonts}
\usepackage{amsmath}
```

produces a string of error messages of the form:

```
! LaTeX Error: Command \iint already defined.
               Or name \end... illegal, see p.192 of the manual.
```

As a general rule, things that *amsmath* defines, it defines well; however, there is a good case for using the *txfonts* version of `\iint` — the associated *tx* fonts have a double integral symbol that doesn't need to be "faked" in the way *amsmath* does. In the case that you are loading several symbol packages, every one of which defines the same symbol, you are likely to experience the problem in a big way (`\euro` is a common victim).

There are similar cases where one package redefines another's command, but no error occurs because the redefining package doesn't use `\newcommand`. Often, in such a case, you only notice the change because you assume the definition given by the first package. The *amsmath*–*txfonts* packages are just such a pair; *txfonts* doesn't provoke errors.

You may deal with the problem by saving and restoring the command. Macro programmers may care to do this for themselves; for the rest of us, there's the package *savesym*. The sequence:

```
\usepackage{savesym}
\usepackage{amsmath}
\savesymbol{iint}
\usepackage{txfonts}
\restoresymbol{TXF}{iint}
```

does the job; restoring the *amsmath* version of the command, and making the *txfonts* version of the command available as `\TXFiint`.

Documentation of *savesym* doesn't amount to much: the only commands are `\savesymbol` and `\restoresymbol`, as noted above.

amsmath.sty: Part of `macros/latex/required/amslatex`

savesym.sty: `macros/latex/contrib/savesym/savesym.sty`

txfonts.sty: Part of `fonts/txfonts`

383 Why are my sections numbered 0.1 . . . ?

This happens when your document is using the standard *book* or *report* class (or one similar), and you've got a `\section` before your first `\chapter`.

What happens is, that the class numbers sections as "⟨*chapter no*⟩.⟨*section no*⟩", and until the first `\chapter` has appeared, the chapter number is 0. (If you use`\chapter*`, which doesn't number the chapter it produces, the problem still arises.)

If you're doing this, it's possible that the *article* class is for you; try it and see. Otherwise, put a `\chapter` before your sections, or do away with section numbering by using `\section*` instead. An alternative way of avoiding numbering is discussed in "unnumbered sections in the table of contents".

384 Link text doesn't break at end line

When using the *hyperref* package, you make a block of text "active" when you define a hyper-link (when the user clicks on that text, the reader program will divert to the *target* of the link).

The *hyperref* package uses a *driver* (in the same way as the *graphics* package does), to determine how to implement all that hyper-stuff.

If you use the driver for *dvips* output (presumably you want to distill the resulting PostScript), limitations in the way *dvips* deals with the `\special` commands mean that *hyperref* must prevent link anchors from breaking at the end of lines. Other drivers (notably those for PDFTeX and for *dvipdfm*) don't suffer from this problem.

The problem may occur in a number of different circumstances. For a couple of them, there are work-arounds:

First, if you have an URL which is active (so that clicking on it will activate your web browser to "go to" the URL). In this case *hyperref* employs the *url* package to split up the URL (as described in typesetting URLs), but the *dvips* driver then suppresses the breaks. The way out is the *breakurl* package, which modifies the `\url` command to produce several smaller pieces, between each of which a line break is permitted. Each group of pieces, that ends up together in one line, is converted to a single clickable link.

Second, if you have a table of contents, list of figure or tables, or the like, *hyperref* will ordinarily make the titles in the table of contents, or captions in the lists, active. If the title or caption is long, it will need to break within the table, but the *dvips* driver will prevent that. In this case, load *hyperref* with the option `linktocpage`, and only the page number will be made active.

Otherwise, if you have a lengthy piece of text that you want active, you have at present no simple solution: you have to rewrite your text, or to use a different PDF generation mechanism.

breakurl.sty: `macros/latex/contrib/breakurl`

385 Page number is wrong at start of page

This is a long story, whose sources are deep inside the workings of TeX itself; it all derives from the TeX's striving to generate the best possible output.

The page number is conventionally stored in `\count0`; LaTeX users see this as the counter `page`, and may typeset its value using `\thepage`.

The number (that is to say, `\count0`) is only updated when TeX actually outputs a page. TeX only even tries to do this when it detects a hint that it may be a good thing to do. From TeX's point of view, the end of a paragraph is a good time to consider outputting a page; it will output a page if it has *more* than a page's worth of material to output. (Ensuring it always has something in hand makes some optimisations possible.) As a result, `\count0` (`\thepage`) is almost always wrong in the first paragraph of a page (the exception is where the page number has been "forcibly" changed, either by changing its value directly, or by breaking the page where TeX wouldn't necessarily have chosen to break).

LaTeX provides a safe way of referring to the page number, by using label references. So, rather than writing:

```
Here is page \thepage{}.
```

you should write:

```
Here is page \pageref{here}\label{here}.
```

(note: no space between the `\pageref` and the `\label`, since that could potentially end up as a page-break space itself, which rather defeats the purpose of the exercise!).

386 My brackets don't match

(La)TeX has a low-level mechanism for matching braces in document text. This means you can type something like:

```
\section{All \emph{OK} now.}
```

and know that the first brace (for the argument of `\section`) will be matched with the last brace, and the internal pair of braces (for the argument of `\emph`) will be matched with each other. It's all very simple.

However, LaTeX has a convention of enclosing optional arguments in brackets, as in:

```
\section[OK]{All \emph{OK} now.}
```

These brackets are not matched by TeX mechanisms, despite the superficial similarity of their use. As a result, straightforward-looking usages like:

```
\section[All [OK] now]{All \emph{OK} now.}
```

aren't OK at all — the optional argument comes to consist of "All [OK", and `\section` takes the single character "n" (of the first "now") as its argument.

Fortunately, TeX's scanning mechanisms helps us by accepting the syntax "`{]}`" to 'hide' the closing bracket from the scanning mechanism that LaTeX uses. In practice, the commonest way to use this facility is:

```
\section[All {[OK]} now]{All \emph{OK} now.}
```

since bracing the bracket on its own "looks odd".

LaTeX has another argument syntax, even less regular, where the argument is enclosed in parentheses, as in:

```
\put(1,2){foo}
```

(a picture environment command).

This mechanism is also prone to problems with matching closing parentheses, but the issue seldom arises since such arguments rarely contain text. If it were to arise, the same solution (enclosing the confused characters in braces) would solve the problem.

387 Characters disappear from figures in PDFTeX

You have a PDF figure, which you want to use in your PDFLaTeX document. When you compile the document, PDFTeX complains about "missing glyphs", and some (or all) of the labelling text or symbols in the original figure is no longer visible.

What has happened is:

1. Your figure file (say `fig.pdf`) has a font `font.pfb` embedded in it.
2. PDFTeX notes that it has `font.pfb` on disc, and loads that in place of the copy in `fig.pdf`.
3. It turns out that the copy in `fig.pdf` has glyphs that aren't in `font.pfb` on disc, so that you get errors while compiling and you see that characters are missing when you view the output. (PDFTeX can't know that the fonts are different, since they have the same name.)

Which is all very undesirable.

PDFTeX does this to keep file sizes down: suppose you have a document that loads figures `fig1.pdf` and `fig2.pdf`; both of those use font `font.pfb`. If PDFTeX takes no action, there will be *two* copies of `font.pfb` in the output. (If your document also uses the font, there could be three copies.)

A real case is the URW font `NimbusRomNo9L-Regu` (a clone of Times Roman), which is available in a version with Cyrillic letters, while the version in TeX distributions doesn't have those letters. Both versions, as distributed, have the same name.

The simple ("quick and dirty") solution is to add the command

```
\pdfinclusioncopyfonts=1
```

to the preamble of your document.

The "real" solution is that one or other font should be renamed. In either case, this would require that you reconfigure some program's (TeX's or your drawing package's) font tables — inevitably a tiresome job.

388 I asked for "empty", but the page is numbered

If you use `\pagestyle{empty}` and you find some pages are numbered anyway, you are probably encountering one of the style decisions built into the standard LaTeX classes: that certain special pages should always appear with `\pagestyle{plain}`, with a page number at the centre of the page foot. The special pages in question are those (in *article* class) containing a `\maketitle`, or (in *book* and *report* classes) `\chapter` or `\part` commands.

The simple solution is to reissue the page style *after* the command, with effect for a single page, as, for example (in *article*):

```
\maketitle
\thispagestyle{empty}
```

or (in *book* or *report*)

```
\chapter{foo bar}
\thispagestyle{empty}
```

A similar technique doesn't work for a *book* or *report* `\part` command pages. For that, and for other detail, take look at "getting rid of page numbers".

T.2 Common misunderstandings

389 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. (The requirement derives from the difficulties of observing page numbers.) 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).

390 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 impose 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 parameters that `\raggedright` set). Then TeX 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 `\raggedright`'s 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}
```

In fact, there are a number of parameters for which TeX only maintains one value per paragraph. A tiresome one is the set of upper case/lower case translations, which (oddly enough) constrains hyphenation of mutilingual texts. Another that regularly creates confusion is `\baselineskip`.

391 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 LaTeX commands (*not* the TeX primitive commands `\uppercase` and `\lowercase`), so that section headings and the like don't produce broken page headings.

textcase.sty: macros/latex/contrib/textcase

392 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.

The *fnbreak* package detects (and generates warnings about) split footnotes.

fnbreak.sty: macros/latex/contrib/fnbreak

393 Getting `\marginpar` on the right side

In an ideal world, marginal notes would be in "analogous" places on every page: notes on an even-side page would be in the left margin, while those on an odd-side page would be in the right margin. A moment's thought shows that a marginal note on the left needs to be typeset differently from a marginal note on the right. The LaTeX `\marginpar` command therefore takes two arguments in a `twoside` documents: `\marginpar[`*left text*`]{`*right text*`}`. LaTeX uses the "obvious" test to get the `\marginpar`s in the correct margin, but a booby-trap arises because TeX runs its page maker asynchronously. If a `\marginpar` is processed while page *n* is 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 "finding if you're on an odd or an even page".

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 label-like marks stored in the `.aux` file; the *memoir* class does likewise.

memoir.cls: macros/latex/contrib/memoir

mparhack.sty: macros/latex/contrib/mparhack

394 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 provided 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.

395 "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/footmisc`

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

396 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. In any case, be careful with usage of `\xspace` — it changes your input syntax, which can be confusing, notably to a collaborating author (particularly if you create some commands which use it and some which don't). Of course, no command built into LaTeX or into any "standard" class or package will use `\xspace`.

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

397 (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 "my words aren't being hyphenated" (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 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 "ignoring paragraph parameters". LaTeX users could use an environment like:

```
\newenvironment{tolerant}[1]{%
  \par\tolerance=#1\relax
}{%
  \par
}
```

enclosing entire paragraphs (or set of paragraphs) in it.

The value of `\emergencystretch` is added to the assumed stretchability of each line of a paragraph, in a further run of the paragraph formatter in case that the paragraph can't be made to look right any other way. (The extra scan happens if `\emergencystretch>0pt` — if it's zero or negative, no gain could be had from rerunning the paragraph setter.) 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. It is therefore clear that `\emergencystretch` needs to be treated with a degree of caution.

More subtle (but more tricky to manage) are the microtypographic extensions provided by PDFTeX. Since PDFTeX is the default 'engine' for LaTeX and ConTeXt work in all distributions, nowadays, the extensions are available to all. There are two extensions, margin kerning and font expansion; margin kerning only affects the visual effect of the typeset page, and has little effect on the ability of the paragraph setter to "get things right". Font expansion works like a subtler version of the trick that `\emergencystretch` plays: PDFTeX 'knows' that your current font may be stretched (or shrunk) to a certain extent, and will do that "on the fly" to optimise the setting of a paragraph. This is a powerful tool in the armoury of the typesetter.

As mentioned above, the microtypographic extensions are tricky beasts to control; however, the *microtype* package relieves the user of the tedious work of specifying how to perform margin adjustments and how much to scale each font ... for the fonts the package knows about; it's a good tool, and users who can take on the specification of adjustments for yet more fonts are always welcome.

`microtype.sty`: `macros/latex/contrib/microtype`

398 Maths symbols don't scale up

By default, the "large" maths symbols stay at the same size regardless of the font size of the text of the document. There's good reason for this: the *cmex* fonts aren't really designed to scale, so that TeX's maths placement algorithms don't perform as well as they might when the fonts are scaled.

However, this behaviour confounds user expectations, and can lead to slightly odd-looking documents. If you want the fonts to scale, despite the warning above, use the *exscale* package — just loading it is enough.

`exscale.sty`: Part of the LaTeX distribution.

399 Why doesn't `\linespread` work?

The command `\linespread{factor}` is supposed to multiply the current `\baselineskip` by ⟨*factor*⟩; but, to all appearances, it doesn't.

In fact, the command is equivalent to `\renewcommand{\baselinestretch}{factor}`: written that way, it somehow feels less surprising that the effect isn't immediate. The `\baselinestretch` factor is only used when a font is selected; a mere change of `\baselinestretch` doesn't change the font, any more than does the command `\fontsize{size}{baselineskip}` — you have to follow either command with `\selectfont`. So:

```
\fontsize{10}{12}%
\selectfont
```

or:

```
\linespread{1.2}%
\selectfont
```

Of course, a package such as *setspace*, whose job is to manage the baseline, will deal with all this stuff — see "managing double-spaced documents". If you want to avoid *setspace*, beware the behaviour of linespread changes within a paragraph: read "`\baselineskip` is a paragraph parameter".

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

400 Only one `\baselineskip` per paragraph

The `\baselineskip`, which determines the space between lines, is not (as one might hope) a property of a line, but of a paragraph. As a result, in a `10pt` (nominal) document (with a default `\baselineskip` of `12pt`), a single character with a larger size, as:

```
{\Huge A}
```

will be squashed into the paragraph: TeX will make sure it doesn't scrape up against the line above, but won't give it "room to breathe", as it does the text at standard size; that is, its size (`24.88pt`) is taken account of, but its `\baselineskip` (`30pt`) isn't. This problem may be solved by a *strut*: the name comes from movable metal typography, and refers to a spacer that held the boxes (that contained the metal character shapes) apart. Every time you change font size, LaTeX redefines the command `\strut` to provide the equivalent of a metal-type strut for the size chosen. So for the example above, we would type

```
Paragraph text ...
   {\Huge A\strut}
   ... paragraph continues ...
```

This technique *only* works for such very short intrusions; if you need several lines, you should convert your intrusion into a `quote` environment, since it's not possible to provide a `\strut` command for every line of the intrusion, in a sensible way, so proceed by:

```
\begin{quote}
  \Huge A LENGTHY TEXT ...
  SHOUTING AT THE READER!
\end{quote}
```

The contrary case:

```
Paragraph text ...
{\footnotesize Extended interjection ...
    ... into the paragraph.}
       ... paragraph continues ...
```

will look wrong, since the `8pt` interjection will end up set on the `12pt` `\baselineskip` of the paragraph, rather than its preferred `8.5pt`. A `\strut` here is no help: there is no such thing as a "negative strut", that draws lines together, so once more, one falls back on the `quote` to separate the interjection:

```
Paragraph text ...
\begin{quote}
  \footnotesize Extended interjection ...
  ... into the paragraph.
\end{quote}
... paragraph continues ...
```

The same effect is at work when we have something like:

```
Paragraph text ...
  ... paragraph body ends.
{\footnotesize Comment on the paragraph.}

Next paragraph starts...
```

which will set the body of the first paragraph on the constricted `\baselineskip` of the `\footnotesize` comment. Solve this problem by ending the initial paragraph before starting the comment:

```
Paragraph text ...
  ... paragraph body ends.
\par\noindent
{\footnotesize Comment on the paragraph.}

Next paragraph starts...
```

(We suggest `\noindent` to make the comment look as if it is part of the paragraph it discusses; omit `\noindent` if that is inappropriate.)

A variation of the previous issue arises from a paragraph whose size is different from those around it:

```
{\Large (Extended) IMPORTANT DETAILS ...}

Main body of text...
```

Again, the problem is solved by ending the paragraph in the same group as the text with a different size:

```
{\Large (Extended) IMPORTANT DETAILS ...\par}

Main body of text...
```

401 Numbers too large in table of contents, etc.

LaTeX constructs the table of contents, list of figures, tables, and similar tables, on the basis of a layout specified in the class. As a result, they do *not* react to the sizes of things in them, as they would if a `tabular` environment (or something similar) was used.

This arrangement can provoke problems, most commonly with deep section nesting or very large page numbers: the numbers in question just don't fit in the space allowed for them in the class.

A separate answer discusses re-designing the tables — re-designing the tables — and those techniques can be employed to make the numbers fit.

402 Why is the inside margin so narrow?

If you give the standard classes the `twoside` option, the class sets the margins narrow on the left of odd-numbered pages, and on the right of even-numbered pages. This is often thought to look odd, but it is quite right.

The idea is that the typographic urge for symmetry should also apply to margins: if you lay an even numbered page to the left of an odd-numbered one, you will see that you've three equal chunks of un-printed paper: the left margin of the even page, the right margin of the odd page, and the two abutting margins together.

This is all very fine in the abstract, but in practical book(let) production it only works "sometimes".

If your booklet is produced on double-width paper and stapled, the effect will be good; if your book(let) is produced using a so-called "perfect" binding, the effect will again be good.

However, almost any "quality" book-binder will need some of your paper to grab hold of, and a book bound in such a way won't exhibit the treasured symmetry unless you've done something about the margin settings.

The packages recommended in "setting up margins" mostly have provision for a "binding offset" or a "binding correction" — search for "binding" in the manuals (*vmargin* doesn't help, here).

If you're doing the job by hand (see manual margin setup), the trick is to calculate your page and margin dimensions as normal, and then:

- subtract the binding offset from `\evensidemargin`, and
- add the binding offset to `\oddsidemargin`.

which can be achieved by:

```
\addtolength{\evensidemargin}{-offset}
\addtolength{\oddsidemargin}{offset}
```

(substituting something sensible like "`5mm`" for "`offset`", above).

The above may not be the best you can do: you may well choose to change the `\textwidth` in the presence of the binding offset; but the changes do work for constant `\textwidth`.

T.3 Why shouldn't I?

403 Why use *fontenc* rather than *t1enc*?

In the very earliest days of LaTeX2e, the only way to use the T1 encoding was *t1enc*; with the summer 1994 "production" release, the *fontenc* package appeared, and provided comprehensive support for use of the encoding.

Nevertheless, the *t1enc* package remains (as part of the LaTeX 2.09 compatibility code), but it does very little: it merely selects font encoding T1, and leaves to the user the business of generating the character codes required.

Generating such character codes could be a simple matter, *if* the T1 encoding matched any widely-supported encoding standard, since in that case, one might expect one's keyboard to generate the character codes. However, the T1 encoding is a mix of several standard encodings, and includes code points in areas of the table which standard encodings specifically exclude, so no T1 keyboards have been (or ever will be) manufactured.

By contrast, the *fontenc* package generates the T1 code points from ordinary LaTeX commands (e.g., it generates the é character codepoint from the command `\'e`). So, unless you have program-generated T1 input (which is almost inconceivable), use `\usepackage[T1]{fontenc}` rather than `\usepackage{t1enc}`.

404 Why bother with *inputenc* and *fontenc*?

The standard input encoding for Western Europe (pending the arrival of Unicode) is ISO 8859–1 (commonly known by the standard's subtitle 'Latin-1'). Latin-1 is remarkably close, in the codepoints it covers, to the (La)TeX T1 encoding.

In this circumstance, why should one bother with *inputenc* and *fontenc*? Since they're pretty exactly mirroring each other, one could do away with both, and use just *t1enc*, despite its shortcomings.

One doesn't do this for a variety of small reasons:

Confusion You've been happily working in this mode, and for some reason find you're to switch to writing in German: the effect of using "ß" is somewhat startling, since T1 and Latin-1 treat the codepoint differently.

Compatibility You find yourself needing to work with a colleague in Eastern Europe: their keyboard is likely to be set to produce Latin-2, so that the simple mapping doesn't work.

Traditional LaTeX You lapse and write something like `\'{e}` rather than typing é; only *fontenc* has the means to convert this LaTeX sequence into the T1 character, so an `\accent` primitive slips through into the output, and hyphenation is in danger.

The *inputenc–fontenc* combination seems slow and cumbersome, but it's safe.

405 Why not use eqnarray?

The environment `eqnarray` is attractive for the occasional user of mathematics in LaTeX documents: it seems to allow aligned systems of equations. Indeed it *does* supply such things, but it makes a serious mess of spacing. In the system:

```
\begin{eqnarray}
  a & = & b + c \\
  x & = & y - z
\end{eqnarray}
```

the spacing around the "=" signs is *not* that defined in the metrics for the font from which the glyph comes — it's `\arraycolsep`, which may be set to some very odd value for reasons associated with real arrays elsewhere in the document.

The user is far better served by the AMSLaTeX bundle, which provides an `align` environment, which is designed with the needs of mathematicians in mind (as opposed to the convenience of LaTeX programmers). For this simple case (`align` and other AMSLaTeX alignment environments are capable of far greater things), code as:

```
\begin{align}
  a & = b + c \\
  x & = y - z
\end{align}
```

The matter is discussed in more detail in a PracTeX journal paper by Lars Madsen; Stefan Kottwitz offers a TeX blog entry which includes screen shots of the output, convincingly demonstrating the problem.

AMSLaTeX: `macros/latex/required/amslatex`

406 Why use `\[...\]` in place of `$$...$$`?

LaTeX defines inline- and display-maths commands, apparently s those that derive from the TeX primitive maths sequences bracketing maths commands with single (or pairs of) dollar signs.

As it turns out, LaTeX's inline maths grouping, `\( ... \)`, has precisely the same effect as the TeX primitive version `$... $`. (Except that the LaTeX version checks to ensure you don't put `\(` and `\)` the wrong way round.)

In this circumstance, one often finds LaTeX users, who have some experience of using Plain TeX, merely assuming that LaTeX's display maths grouping `\[ ... \]` may be replaced by the TeX primitive display maths `$$... $$`.

Unfortunately, they are wrong: if LaTeX code is going to patch display maths, it can only do so by patching `\[` and `\]`. The most obvious way this turns up, is that the class option `fleqn` simply does not work for equations coded using `$$... $$`, whether you're using the standard classes alone, or using package *amsmath*. Also, the `\[` and `\]` construct has code for rationalising vertical spacing in some extreme cases; that code is not available in `$$... $$`, so if you use the non-standard version, you may occasionally observe inconsistent vertical spacing .

There are more subtle effects (especially with package *amsmath*), and the simple rule is "use `\[ ... \]` whenever displayed maths is needed in LaTeX".

407 What's wrong with `\bf`, `\it`, etc.?

The font-selection commands of LaTeX 2.09 were `\rm`, `\sf`, `\tt`, `\it`, `\sl`, `\em` and `\bf`; they were modal commands, so you used them as:

```
{\bf Fred} was {\it here\/}}.
```

with the font change enclosed in a group, so as to limit its effect; note the italic correction command `\/` that was necessary at the end of a section in italics.

At the release of LaTeX2e in summer 1994, these simple commands were deprecated, but recognising that their use is deeply embedded in the brains of LaTeX users, the commands themselves remain in LaTeX, *with their LaTeX 2.09 semantics*. Those semantics were part of the reason they were deprecated: each `\`*xx* overrides any other font settings, keeping only the size. So, for example,

```
{\bf\it Here we are again\/}
```

ignores `\bf` and produces text in italic, medium weight (and the italic correction has a real effect), whereas

```
{\it\bf happy as can be\/}
```

ignores `\it` and produces upright text at bold weight (and the italic correction has nothing to do). The same holds if you mix LaTeX2e font selections with the old style commands:

```
\textbf{\tt all good friends}
```

ignores the `\textbf` that encloses the text, and produces typewriter text at medium weight.

So why are these commands deprecated? — it is because of confusions such as that in the last example. The alternative (LaTeX2e) commands are discussed in the rest of this answer.

LaTeX2e's font commands come in two forms: modal commands and text-block commands. The default set of modal commands offers weights `\mdseries` and `\bfseries`, shapes `\upshape`, `\itshape`, `\scshape` and `\slshape`, and families `\rmfamily`, `\sffamily` and `\ttfamily`. A font selection requires a family, a shape and a series (as well as a size, of course). A few examples

```
{\bfseries\ttfamily and jolly good company!}
```

produces bold typewriter text (but note the lack of a bold typewriter font in the default Computer Modern fonts), or

```
{\slshape\sffamily Never mind the weather\/}
```

(note the italic correction needed on slanted fonts, too).

LaTeX2e's text block commands take the first two letters of the modal commands, and form a `\textxx` command from them. Thus `\bfseries` becomes `\textbf{text}`, `\itshape` becomes `\textit{text}`, and `\ttfamily` becomes `\texttt{text}`. Block commands may be nested, as:

```
\textit{\textbf{Never mind the rain}}
```

to produce bold italic text (note that the block commands supply italic corrections where necessary), and they be nested with the LaTeX2e modal commands, too:

```
\texttt{\bfseries So long as we're together}
```

for bold typewriter, or

```
{\slshape \textbf{Whoops!  she goes again}\/}
```

for a bold slanted instance of the current family (note the italic correction applied at the end of the modal command group, again).

The new commands (as noted above) override commands of the same type. In almost all cases, this merely excludes ludicrous ideas such as "upright slanted" fonts, or "teletype roman" fonts. There are a couple of immediate oddities, though. The first is the conflict between `\itshape` (or `\slshape`) and `\scshape`: while many claim that an italic small-caps font is typographically unsound, such fonts do exist. Daniel Taupin's *smallcap* package enables use of the instances in the EC fonts, and similar techniques could be brought to bear on many other font sets. The second is the conflict between `\upshape` and `\itshape`: Knuth actually offers an upright-italic font which LaTeX uses for the "£" symbol in the default font set. The combination is sufficiently weird that, while there's a defined font shape, no default LaTeX commands exist; to use the shape, the (eccentric) user needs LaTeX's simplest font selection commands:

```
{\fontshape{ui}\selectfont Tra la la, di dee}
```

smallcap.sty: macros/latex/contrib/smallcap

408 What's wrong with \newfont?

If all else fails, you *can* specify a font using the LaTeX `\newfont` command. The font so specified doesn't fit into the LaTeX font selection mechanism, but the technique can be tempting under several circumstances. The command is merely the thinnest of wrappers around the `\font` primitive, and doesn't really fit with LaTeX at all. A simple, but really rather funny, example of the problems it poses, may be seen in:

```
\documentclass[10pt]{article}
\begin{document}
\newfont{\myfont}{cmr17 scaled 2000}
\myfont
\LaTeX
\end{document}
```

(the reader is encouraged to try this). The "A" of `\LaTeX` pretty much disappears: LaTeX chooses the size on the "A" according to *its* idea of the font size (10pt), but positions it according to the dimensions of "`\myfont`", which is more than three times that size.

Another "`\myfont`" example arises from an entirely different source. The mini-document:

```
\documentclass{article}
\begin{document}
\newfont{\myfont}{ecrm1000}
{\myfont voil\`a}
\end{document}
```

gives you "German low double quotes" (under the "a") in place of the grave accent. This happens because *ecrm1000* is in a different font encoding than LaTeX is expecting — if you use the LaTeX *fontenc* package to select the EC fonts, all these tiresome encoding issues are solved for you, behind the scenes.

There does however remain a circumstance when you will be tempted to use `\newfont` — viz., to get a font size that doesn't fall into the Knuth standard set of sizes: LaTeX (by default) won't allow you to use such a size. Don't despair: see the answer "arbitrary font sizes".

U The joy of TeX errors

409 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 the structure of errors 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 books about TeX) 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 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`

410 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 approaching errors, 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).

411 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.

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 same happens with captions (the following is a simplification of a comp.text.tex post):

```
    \caption{Energy: \[e=mc^2\]}
```

giving us the error message

```
    ! Argument of \@caption has an extra }.
```

The similar (but more sensible):

```
    \caption{Energy: \(e=mc^2\)}
```

is more tiresome, still: there's no error when you first run the job ... but there is on the second pass, when the list of figures (or tables) is generated, giving:

```
    ! LaTeX Error: Bad math environment delimiter.
```

in the \listoffigures processing.

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 \section case would show as

```
    \section{Mumble\protect\footnote{I couldn't think of anything better}}
```

However, in both the \section case and the \caption case, you can separate the moving argument, as in \section[*moving*]{*static*}; this gives us another standard route — simply to omit (or otherwise sanitise) the fragile command in the moving argument. So, one might rewrite the \caption example as:

```
    \caption[Energy: (Einstein's equation)]{Energy: \(E=mc^2\)}
```

In practice, inserting mathematics in a moving argument has already been addressed in LaTeX2e by the robust command \ensuremath:

```
    \caption{Energy: \ensuremath{E=mc^2}}
```

So: always look for alternatives to the \protect route.

Footnotes can be even more complex; "footnotes in LaTeX section headings" deals specifically with that issue.

412 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 *general advice* on errors) *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.

There are in fact two items being consumed when you nest boxes: the other is the grouping level. Whether you exhaust your *semantic nest* or your permitted *grouping levels* first is controlled entirely by the relative size of the two different sets of buffers in your (La)TeX executable.

413 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 testing TeX implementations).

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, e-TeX, Ω and LuaTeX — and respectively — all do this, as does MicroPress Inc's VTeX.

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 epsf problems).

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. The more modern drawing package, *pgf* with its higher-level interface *TikZ* is also a common source of such problems.

In such cases, it is usually possible to use the e-TeX extensions (all modern distributions provide them). The LaTeX package *etex* modifies the register allocation mechanism to make use of e-TeX's extended register sets. *Etex* is a derivative of the Plain TeX macro file *etex.src*, which is used in building the e-TeX Plain format; both files are part of the e-TeX distribution and are available in current distributions.

It is possible that, even with *etex* loaded, you still find yourself running out of things. Problems can be caused by packages that use large numbers of "inserts" (inserts are combinations of counter, box, dimension and skip registers, used for storing floats and footnotes). *Morefloats* does this, of course (naturally enough, allocating new floats), and footnote packages such as *manyfoot* and *bigfoot* (which uses *manyfoot*) can also give problems. The *etex* extensions allow you to deal with these things: the command `\reserveinserts{`*n*`}` ensures there is room for ⟨*n*⟩ more inserts. Hint: by default *morefloats* adds 18 inserts (though it can be instructed to use more), and *manyfoot* seems to be happy with 10 reserved, but there are 'hard' limits that we cannot program around — the discussion of running out of floats has more about this. It is essential that you load *etex* before any other packages, and reserve any extra inserts immediately:

```
\documentclass[...]{...}
\usepackage{etex}
\reserveinserts{28}
```

The e-TeX extensions don't help with `\read` or `\write` objects (and neither will the *etex* package), but the *morewrites* package can provide the *illusion* of large numbers of `\write` objects.

morewrites.sty: macros/latex/contrib/morewrites

414 epsf gives up after a bit

Some copies of the documentation of `epsf.tex` seemed once 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 no room for a new thing).

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.

415 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 "hyphenation failures", 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 to look at it.

416 Option clash for package

So you've innocently added:

```
\usepackage[draft]{graphics}
```

to your document, and LaTeX responds with

```
! LaTeX Error: Option clash for package graphics.
```

The error is a complaint about loading a package *with options*, more than once (LaTeX doesn't actually examine what options there are: it complains because it can't do anything with the multiple sets of options). You can load a package any number of times, with no options, and LaTeX will be happy, but you may only supply options when you first load the package.

So perhaps you weren't entirely innocent — the error would have occurred on the second line of:

```
\usepackage[dvips]{graphics}
\usepackage[draft]{graphics}
```

which could quite reasonably (and indeed correctly) have been typed:

```
\usepackage[dvips,draft]{graphics}
```

But if you've not made that mistake (even with several lines separating the `\usepackage` commands, it's pretty easy to spot), the problem could arise from something else loading the package for you. How do you find the culprit? The "h" response to the error message tells you which options were loaded each time. Otherwise, it's down to the log analysis games discussed in "How to approach errors"; the trick to remember is that that the process of loading each file is parenthesised in the log; so if package *foo* loads *graphics*, the log will contain something like:

```
(<path>/foo.sty ...
...
(<path>/graphics.sty ...
...)
...
)
```

(the parentheses for *graphics* are completely enclosed in those for *foo*; the same is of course true if your class *bar* is the culprit, except that the line will start with the path to `bar.cls`).

If we're dealing with a package that loads the package you are interested in, you need to ask LaTeX to slip in options when *foo* loads it. Instead of:

```
\usepackage{foo}
\usepackage[draft]{graphics}
```

you would write:

```
\PassOptionsToPackage{draft}{graphics}
\usepackage{foo}
```

The command `\PassOptionsToPackage` tells LaTeX to behave as if its options were passed, when it finally loads a package. As you would expect from its name, `\PassOptionsToPackage` can deal with a list of options, just as you would have in the the options brackets of `\usepackage`.

More trickily, instead of:

```
\documentclass[...]{bar}
\usepackage[draft]{graphics}
```

you would write:

```
\PassOptionsToPackage{draft}{graphics}
\documentclass[...]{bar}
```

with `\PassOptionsToPackage` *before* the `\documentclass` command.

However, if the *foo* package or the *bar* class loads *graphics* with an option of its own that clashes with what you need in some way, you're stymied. For example:

```
\PassOptionsToPackage{draft}{graphics}
```

where the package or class does:

```
\usepackage[final]{graphics}
```

sets `final` *after* it's dealt with option you passed to it, so your `draft` will get forgotten. In extreme cases, the package might generate an error here (*graphics* doesn't go in for that kind of thing, and there's no indication that `draft` has been forgotten).

In such a case, you have to modify the package or class itself (subject to the terms of its licence). It may prove useful to contact the author: she may have a useful alternative to suggest.

417 Option clash for package

The error message

```
! LaTeX Error: Option clash for package footmisc
```

means what it says — your document contains a (potentially) clashing pair of options; sadly, it is not always obvious how the error has arisen.

If you simply write:

```
\usepackage[a]{foo}
...
\usepackage{foo}
```

LaTeX is happy, as it is with:

```
\usepackage[a]{foo}
...
\usepackage[a]{foo}
```

since LaTeX can see there's no conflict (in fact, the second load does nothing).

Similarly,

```
\usepackage[a,b]{foo}
...
\usepackage[a]{foo}
```

produces no error and does nothing for the second load.

However

```
\usepackage[a]{foo}
...
\usepackage[b]{foo}
```

produces the error; even if option '`b`' is an alias for option '`a`' — LaTeX doesn't "look inside" the package to check anything like that.

The general rule is: the first load of a package defines a set of options; if a further `\usepackage` or `\RequirePackage` also calls for the package, the options on that call may not extend the set on the first load.

Fortunately, the error (in that sort of case) is easily curable once you've examined the preamble of your document.

Now, suppose package *foo* loads *bar* with option `b`, and your document says:

```
\usepackage{foo}
...
\usepackage[a]{bar}
```

or

```
\usepackage[a]{bar}
...
\usepackage{foo}
```

the error will be detected, even though you have only explicitly loaded *bar* once. Debugging such errors is tricky: it may involve reading the logs (to spot which packages were called), or the documentation of package *foo*.

418 "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 not being placed properly. LaTeX has a limited amount of storage for 'floats' (figures, tables, or floats you've defined yourself with the *float* package); if something you have done has prevented LaTeX from typesetting floats, it will run out of storage space.

This failure usually occurs in extreme cases of floats moving "wrongly"; LaTeX has found it can't place a float, and floats of the same type have piled up behind it.

How does this happen? — LaTeX guarantees that caption numbers are sequential in the document, but the caption number is allocated when the figure (or whatever) is created, and can't be changed. Thus, if floats are placed out of order, their caption numbers would also appear out of order in the body of the document (and in the list of figures, or whatever). As a result, enforcement of the guarantee means that simple failure to place a float means that no subsequent float can be placed; and hence (eventually) the error.

Techniques for solving the problem are discussed in the floats question already referenced.

An alternative *may* be to use the *morefloats* package. The package will allocate more "float skeletons" than LaTeX does by default; each such skeleton may then be used to store a float. Beware that even with *morefloats*, the number you can allocate is limited; even with the *etex* package (which makes available many more registers, etc., than LaTeX does by default; e-TeX can create lots more registers, but none of those "beyond the original TeX default" may be used in float skeletons). Thus, *etex* may offer some relief, but it can *not* be regarded as a panacea

The error also occurs in a long sequence of float 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 prolonged: once the page fills, LaTeX won't place any more floats, leading to the error.

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/float`

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

419 \spacefactor complaints

The errors

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

or

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

or simply

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

bite the LaTeX programmer who uses an internal command without taking "precautions". An internal-style command such as `\@foo` has been defined or used in a private macro, and it is interpreted as `\@`, followed by the 'text' `foo`. (`\@` is used, for real, to set up end-of-sentence space in some circumstances; it uses `\spacefactor` to do that.)

The problem is discussed in detail in "`@` in macro names", together with solutions.

420 \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 one or more repetitions of the "occurred inside a group" message before LaTeX finally exits. The *checkend* package recognises other unclosed `\begin{blob}` commands, and generates an "ended by" error message for each one, rather than producing the "occurred inside a group" message, which is sometimes useful (if you remember to load the package).

In the absence of such information from LaTeX, you need to use "traditional" binary search to find the offending group. Separate the preamble from the body of your file, and process each half on its own with the preamble; 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.

e-TeX (and e-LaTeX — LaTeX run on e-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).

`checkend.sty`: Distributed as part of `macros/latex/contrib/bezos`

421 "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.

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

The commonest arises from attempting to use an example from the *The LaTeX Companion* (first edition), 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, in its first edition, the *Companion*'s examples always assumed 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*. (The problem does not really arise with the second edition; copies of all the examples are available on the accompanying CD-ROM, or on CTAN.)

The other problem, which is increasingly rare nowadays, arises from misconfiguration of a system that has been upgraded from LaTeX 2.09: the document uses the *times* package, and the error appears at `\begin{document}`. The file search paths are wrongly set up, and your `\usepackage{times}` has picked up a LaTeX 2.09 version of the package, which in its turn has invoked another which has no equivalent in LaTeX2e. The obvious solution is to rewrite the paths so that LaTeX 2.09 packages are chosen only as a last resort so that the startlingly simple LaTeX2e *times* package will be picked up. Better still is to replace the whole thing with something more modern still; current *psnfss* doesn't provide a *times* package — the alternative *mathptmx* incorporates *Times*-like mathematics, and a sans-serif face based on *Helvetica*, but scaled to match *Times* text rather better.

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

Examples for LaTeX Companion: `info/examples/tlc2`

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

422 "Please type a command or say \end"

Sometimes, when you are running (La)TeX, it will abruptly stop and present you with a prompt (by default, just a * character). Many people (including this author) will reflexively hit the 'return' key, pretty much immediately, and of course this is no help at all — TeX just says:

```
(Please type a command or say '\end')
```

and prompts you again.

What's happened is that your (La)TeX file has finished prematurely, and TeX has fallen back to a supposed including file, from the terminal. This could have happened simply because you've omitted the `\bye` (Plain TeX), `\end{document}` (LaTeX), or whatever. Other common errors are failure to close the braces round a command's argument, or (in LaTeX) failure to close a verbatim environment: in such cases you've already read and accepted an error message about encountering end of file while scanning something.

If the error is indeed because you've forgotten to end your document, you can insert the missing text: if you're running Plain TeX, the advice, to "say `\end`" is good enough: it will kill the run; if you're running LaTeX, the argument will be necessary: `\end {document}`.

However, as often as not this isn't the problem, and (short of debugging the source of the document before ending) brute force is probably necessary. Excessive force (killing the job that's running TeX) is to be avoided: there may well be evidence in the `.log` file that will be useful in determining what the problem is — so the aim is to persuade TeX to shut itself down and hence flush all log output to file.

If you can persuade TeX to read it, an end-of-file indication (control-`D` under Unix, control-`Z` under Windows) will provoke TeX to report an error and exit immediately. Otherwise you should attempt to provoke an error dialogue, from which you can exit (using the `x` 'command'). An accessible error could well be inserting an illegal character: what it is will depend on what macros you are running. If you can't make that work, try a silly command name or two.

423 "Unknown graphics extension"

The LaTeX graphics package deals with several different types of DVI (or other) output drivers; each one of them has a potential to deal with a different selection of graphics formats. The package therefore has to be told what graphics file types its output driver knows about; this is usually done in the ⟨*driver*⟩`.def` file corresponding to the output driver you're using.

The error message arises, then, if you have a graphics file whose extension doesn't correspond with one your driver knows about. Most often, this is because you're being optimistic: asking *dvips* to deal with a `.png` file, or PDFTeX to deal with a `.eps` file: the solution in this case is to transform the graphics file to a format your driver knows about.

If you happen to *know* that your device driver deals with the format of your file, you are probably falling foul of a limitation of the file name parsing code that the graphics package uses. Suppose you want to include a graphics file `home.bedroom.eps` using the *dvips* driver; the package will conclude that your file's extension is `.bedroom.eps`, and will complain.

The *grffile* package deals with the last problem (and others — see the package documentation); using the package, you may write:

```
\usepackage{graphicx}
\usepackage{grffile}
...
\includegraphics{home.bedroom.eps}
```

or you may even write

```
\includegraphics{home.bedroom}
```

and *graphicx* will find a `.eps` or `.pdf` (or whatever) version, according to what version of (La)TeX you're running.

If for some reason you can't use *grffile*, you have three unsatisfactory alternatives:

- Rename the file — for example `home.bedroom.eps`→`home-bedroom.eps`
- Mask the first dot in the file name:
  ```
  \newcommand*{\DOT}{.}
  \includegraphics{home\DOT bedroom.eps}
  ```
- Tell the graphics package what the file is, by means of options to the `\includegraphics` command:
  ```
  \includegraphics[type=eps,ext=.eps,read=.eps]{home.bedroom}
  ```

`grffile.sty`: Distributed as part of the Oberdiek collection `macros/latex/contrib/oberdiek`

424 "Missing $ inserted"

There are certain things that *only* work in maths mode. If your document is not in maths mode and you have an _ or a ^ character, TeX (and by inheritance, LaTeX too) will say

```
! Missing $ inserted
```

as if you couldn't possibly have misunderstood the import of what you were typing, and the only possible interpretation is that you had committed a typo in failing to enter maths mode. TeX, therefore, tries to patch things up by inserting the `$` you 'forgot', so that the maths-only object will work; as often as not this will land you in further confusion.

It's not just the single-character maths sub- and superscript operators: anything that's built in or declared as a maths operation, from the simplest lower-case `\alpha` through the inscrutable `\mathchoice` primitive, and beyond, will provoke the error if misused in text mode.

LaTeX offers a command `\ensuremath`, which will put you in maths mode for the execution of its argument, if necessary: so if you want an `\alpha` in your running text, say `\ensuremath{\alpha}`; if the bit of running text somehow transmutes into a bit of mathematics, the `\ensuremath` will become a no-op, so it's pretty much always safe.

425 Warning: "Font shape ... not available"

LaTeX's font selection scheme maintains tables of the font families it has been told about. These tables list the font families that LaTeX knows about, and the shapes and series in which those font families are available. In addition, in some cases, the tables list the sizes at which LaTeX is willing to load fonts from the family.

When you specify a font, using one of the LaTeX font selection commands, LaTeX looks for the font (that is, a font that matches the encoding, family, shape, series and size that you want) in its tables. If the font isn't there at the size you want, you will see a message like:

```
LaTeX Font Warning: Font shape `OT1/cmr/m/n' in size <11.5> not available
(Font)              size <12> substituted on input line ...
```

There will also be a warning like:

```
LaTeX Font Warning: Size substitutions with differences
(Font)              up to 0.5pt have occurred.
```

after LaTeX has encountered `\end{document}`.

The message tells you that you've chosen a font size that is not in LaTeX's list of "allowed" sizes for this font; LaTeX has chosen the nearest font size it knows is allowed. In fact, you can tell LaTeX to allow *any* size: the restrictions come from the days when only bitmap fonts were available, and they have never applied to fonts that come in scaleable form in the first place. Nowadays, most of the fonts that were once bitmap-only are also available in scaleable (Adobe Type 1) form. If your installation uses scaleable versions of the Computer Modern or European Computer Modern (EC) fonts, you can tell LaTeX to remove the restrictions; use the *type1cm* or *type1ec* package as appropriate.

If the combination of font shape and series isn't available, LaTeX will usually have been told of a fall-back combination that may be used, and will select that:

```
LaTeX Font Warning: Font shape `OT1/cmr/bx/sc' undefined
(Font)              using `OT1/cmr/bx/n' instead on input line 0.
```

Substitutions may also be "silent"; in this case, there is no more than an "information" message in the log file. For example, if you specify an encoding for which there is no version in the current font family, the 'default family for the encoding' is selected. This happens, for example, if you use command `\textbullet`, which is normally taken from the maths symbols font, which is in `OMS` encoding. My test log contained:

```
LaTeX Font Info:    Font shape `OMS/cmr/m/n' in size <10> not available
(Font)              Font shape `OMS/cmsy/m/n' tried instead on input line ...
```

In summary, these messages are not so much error messages, as information messages, that tell you what LaTeX has made of your text. You should check what the messages say, but you will ordinarily not be surprised at their content.

type1cm.sty: macros/latex/contrib/type1cm

type1ec.sty: fonts/ps-type1/cm-super/type1ec.sty

426 Unable to read an entire line

TeX belongs to the generation of applications written for environments that didn't offer the sophisticated string and i/o manipulation we nowadays take for granted (TeX was written in Pascal, and the original Pascal standard made no mention of i/o, so that anything but the most trivial operations were likely to be unportable).

When you overwhelm TeX's input mechanism, you get told:

```
! Unable to read an entire line---bufsize=3000.
    Please ask a wizard to enlarge me.
```

(for some value of '3000' — the quote was from a `comp.text.tex` posting by a someone who was presumably using an old TeX).

As the message implies, there's (what TeX thinks of as a) line in your input that's "too long" (to TeX's way of thinking). Since modern distributions tend to have tens of thousands of bytes of input buffer, it's somewhat rare that these messages occur "for real". Probable culprits are:

- A file transferred from another system, without translating record endings. With the decline of fixed-format records (on mainframe operating systems) and the increased intelligence of TeX distributions at recognising other systems' explicit record-ending characters, this is nowadays rather a rare cause of the problem.
- A graphics input file, which a package is examining for its bounding box, contains a binary preview section. Again, sufficiently clever TeX distributions recognise this situation, and ignore the previews (which are only of interest, if at all, to a TeX previewer).

The usual advice is to ignore what TeX says (i.e., anything about enlarging), and to put the problem right in the source.

If the real problem is over-long text lines, most self-respecting text editors will be pleased to automatically split long lines (while preserving the "word" structure) so that they are nowhere any longer than a given length; so the solution is just to edit the file.

If the problem is a ridiculous preview section, try using *ghostscript* to reprocess the file, outputting a "plain `.eps`" file. (*Ghostscript* is distributed with a script *ps2epsi* which will regenerate the preview if necessary.) Users of the shareware program *GSview* will find buttons to perform the required transformation of the file being displayed.

ghostscript: Browse support/ghostscript/GPL

GSview: Browse support/ghostscript/ghostgum

427 "Fatal format file error; I'm stymied"

(La)TeX applications often fail with this error when you've been playing with the configuration, or have just installed a new version.

The format file contains the macros that define the system you want to use: anything from the simplest (Plain TeX) all the way to the most complicated, such as LaTeX or ConTeXt. From the command you issue, TeX knows which format you want.

The error message

```
Fatal format file error; I'm stymied
```

means that TeX itself can't understand the format you want. Obviously, this could happen if the format file had got corrupted, but it usually doesn't. The commonest cause of the message, is that a new binary has been installed in the system: no two TeX binaries on the same machine can understand each other's formats. So the new version of TeX you have just installed, won't understand the format generated by the one you installed last year.

Resolve the problem by regenerating the format; of course, this depends on which system you are using.

- On a teTeX-based system, run

  ```
  fmtutil --all
  ```

  or

```
fmtutil --byfmt=<format name>
```

to build only the format that you are interested in.

- On a MiKTeX system, click `Start`→`Programs`→`MiKTeX` *`version`*→`MiKTeX Options`, and in the options window, click `Update now`.

428 Non-PDF special ignored!

This is a PDFTeX error: PDFTeX is running in PDF output mode, and it has encountered a `\special` command (`\special`). PDFTeX is able to generate its own output, and in this mode of operation has no need of `\special` commands (which allow the user to pass information to the driver being used to generate output).

Why does this happen? LaTeX users, nowadays, hardly ever use `\special` commands on their own — they employ packages to do the job for them. Some packages will generate `\special` commands however they are invoked: *pstricks* is an example (it's very raison d'être is to emit PostScript code in a sequence of `\special` commands). *Pstricks* may be dealt with by other means (the *pdftricks* package offers a usable technique).

More amenable to correction, but more confusing, are packages (such as *color*, *graphics* and *hyperref*) that specify a "driver". These packages have plug-in modules that determine what `\special` (or other commands) are needed to generate any given effect: the `pdftex` driver for such packages knows not to generate `\special` commands. In most circumstances, you can let the system itself choose which driver you need; in this case everything will act properly when you switch to using PDFLaTeX. If you've been using *dvips* (and specifying the `dvips` driver) or *dvipdfm* (for which you have to specify the driver), and decide to try PDFLaTeX, you *must* remove the `dvips` or `dvipdfm` driver specification from the package options, and let the system recognise which driver is needed.

`pdftricks.sty`: `macros/latex/contrib/pdftricks`

`pstricks.sty`: `graphics/pstricks`

429 Mismatched mode ljfour and resolution 8000

You're running *dvips*, and you encounter a stream of error messages, starting with "`Mismatched mode`". The mode is the default used in your installation — it's set in the *dvips* configuration file, and `ljfour` is commonest (since it's the default in most distributions), but not invariable.

The problem is that *dvips* has encountered a font for which it must generate a bitmap (since it can't find it in Type 1 format), and there is no proforma available to provide instructions to give to Metafont.

So what to do? The number 8000 comes from the '`-Ppdf`' option to *dvips*, which you might have found from the answer "wrong type of fonts" ("wrong type of fonts"). The obvious solution is to switch to the trivial substitute '`-Pwww`', which selects the necessary type 1 fonts for PDF generation, but nothing else: however, this will leave you with undesirable bitmap fonts in your PDF file. The "proper" solution is to find a way of expressing what you want to do, using type 1 fonts.

430 "Too deeply nested"

This error appears when you start a LaTeX list.

LaTeX keeps track of the nesting of one list inside another. There is a set of list formatting parameters built-in for application to each of the list nesting levels; the parameters determine indentation, item separation, and so on. The `list` environment (the basis for list environments like `itemize` and `enumerate`) "knows" there are only 6 of these sets.

There are also different label definitions for the `enumerate` and `itemize` environments at their own private levels of nesting. Consider this example:

```
\begin{enumerate}
\item first item of first enumerate
  \begin{itemize}
```

```
      \item first item of first itemize
        \begin{enumerate}
        \item first item of second enumerate
        ...
        \end{enumerate}
      ...
      \end{itemize}
    ...
    \end{enumerate}
```

In the example,

- the first `enumerate` has labels as for a first-level `enumerate`, and is indented as for a first-level list;
- the first `itemize` has labels as for a first level `itemize`, and is indented as for a second-level list; and
- the second `enumerate` has labels as for a second-level `enumerate`, and is indented as for a third-level list.

Now, as well as LaTeX *knowing* that there are 6 sets of parameters for indentation, it also *knows* that there are only 4 types of labels each, for the environments `enumerate` and `itemize` (this "knowledge" spells out a requirement for class writers, since the class supplies the sets of parameters).

From the above, we can deduce that there are several ways we can run out of space: we can have 6 lists (of any sort) nested, and try to start a new one; we can have 4 `enumerate` environments somewhere among the set of nested lists, and try to add another one; and we can have 4 `itemize` environments somewhere among the set of nested lists, and try to add another one.

What can be done about the problem? Not much, short of rewriting LaTeX — you really need to rewrite your document in a slightly less labyrinthine way.

431 Capacity exceeded — input levels

The error

```
! TeX capacity exceeded, sorry [text input levels=15].
```

is caused by nesting your input too deeply. You can provoke it with the trivial (Plain TeX) file `input.tex`, which contains nothing but:

```
\input input
```

In the real world, you are unlikely to encounter the error with a modern TeX distribution. TeTeX (used to produce the error message above) allows 15 files open for TeX input at any one time, which is improbably huge for a document generated by real human beings.

However, for those improbable (or machine-generated) situations, some distributions offer the opportunity to adjust the parameter `max_in_open` in a configuration file.

432 PDFTeX destination . . . ignored

The warning:

```
! pdfTeX warning (ext4): destination with the same identifier
(name{page.1}) has been already used, duplicate ignored
```

arises because of duplicate page numbers in your document. The problem is usually soluble: see PDF page destinations — which answer also describes the problem in more detail.

If the identifier in the message is different, for example `name{figure.1.1}`, the problem is (often) due to a problem of package interaction. The `README` in the *hyperref* distribution mentions some of these issues — for example, `equation` and `eqnarray` as supplied by the *amsmath* package; means of working around the problem are typically supplied there.

Some packages are simply incompatible with *hyperref*, but most work simply by ignoring it. In most cases, therefore, you should load your package before you load *hyperref*, and *hyperref* will patch things up so that they work, so you can utilise your (patched) package *after* loading both:

```
\usepackage{your package}
...
\usepackage[opts]{hyperref}
...
⟨code that uses your package⟩
```

For example:

```
\usepackage{float}          % defines \newfloat
...
\usepackage[...]{hyperref}  % patches \newfloat
...
\newfloat{...}{...}{...}
```

You should load packages in this order as a matter of course, unless the documentation of a package says you *must* load it after *hyperref*. (There are few packages that require to be loaded after hyperref: one such is *memoir*'s "*hyperref* fixup" package *memhfixc*.)

If loading your packages in the (seemingly) "correct" order doesn't solve the problem, you need to seek further help.

433 Alignment tab changed to \cr

This is an error you may encounter in LaTeX when a tabular environment is being processed. "Alignment tabs" are the & signs that separate the columns of a `tabular` (or `array` or matrix) environment; so the error message

```
! Extra alignment tab has been changed to \cr
```

could arise from a simple typo, such as:

```
\begin{tabular}{ll}
  hello   & there & jim \\
  goodbye & now
\end{tabular}
```

where the second & in the first line of the table is more than the two-column `ll` column specification can cope with. In this case, an extra "`l`" in that solves the problem. (If you continue from the error in this case, "`jim`" will be moved to a row of his own.) Another simple typo that can provoke the error is:

```
\begin{tabular}{ll}
  hello   & there
  goodbye & now
\end{tabular}
```

where the '`\\`' has been missed from the first line of the table. In this case, if you continue from the error, you will find that LaTeX has made a table equivalent to:

```
\begin{tabular}{ll}
  hello   & there goodbye\\
  now
\end{tabular}
```

(with the second line of the table having only one cell).

Rather more difficult to spot is the occurrence of the error when you're using alignment instructions in a "`p`" column:

```
\usepackage{array}
...
\begin{tabular}{l>{\raggedright}p{2in}}
here & we are again \\
happy & as can be
\end{tabular}
```

the problem here (as explained in tabular cell alignment) is that the `\raggedright` command in the column specification has overwritten `tabular`'s definition of `\\`, so that "`happy`" appears in a new line of the second column, and the following `&` appears to LaTeX just like the second `&` in the first example above.

Get rid of the error in the way described in tabular cell alignment — either use `\tabularnewline` explicitly, or use the `\RBS` trick described there.

The *amsmath* package adds a further twist; when typesetting a matrix (the package provides many matrix environments), it has a fixed maximum number of columns in a matrix — exceed that maximum, and the error will appear. By default, the maximum is set to 10, but the value is stored in counter `MaxMatrixCols` and may be changed (in the same way as any counter):

```
\setcounter{MaxMatrixCols}{20}
```

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

434 Graphics division by zero

While the error

```
! Package graphics Error: Division by 0.
```

can actually be caused by offering the package a figure which claims to have a zero dimension, it's more commonly caused by rotation.

Objects in TeX may have both height (the height above the baseline) and depth (the distance the object goes below the baseline). If you rotate an object by 180 degrees, you convert its height into depth, and vice versa; if the object started with zero depth, you've converted it to a zero-height object.

Suppose you're including your graphic with a command like:

```
\includegraphics[angle=180,height=5cm]{myfig.eps}
```

In the case that `myfig.eps` has no depth to start with, the scaling calculations will produce the division-by-zero error.

Fortunately, the *graphicx* package has a keyword `totalheight`, which allows you to specify the size of the image relative to the sum of the object's `height` and `depth`, so

```
\includegraphics[angle=180,totalheight=5cm]{myfig.eps}
```

will resolve the error, and will behave as you might hope.

If you're using the simpler *graphics* package, use the `*` form of the `\resizebox` command to specify the use of `totalheight`:

```
\resizebox*{!}{5cm}{%
  \rotatebox{180}{%
    \includegraphics{myfig.eps}%
  }%
}
```

graphics.sty,graphicx.sty: Both parts of the macros/latex/required/graphics bundle

435 Missing `\begin{document}`

The *preamble* of your document is the stuff before `\begin{document}`; you put `\usepackage` commands and your own macro definitions in there. LaTeX doesn't like *typesetting* anything in the preamble, so if you have:

- typed the odd grumble,
- created a box with `\newsavebox` and put something in it using `\sbox` (or the like),
- forgotten to put `\begin{document}` into the document, at all, or even
- gave it the wrong file

the error is inevitable and the solution is simple — judicious use of comment markers ('`%`') at the beginning of a line, moving things around, providing something that was missing ... or switching to the correct file.

The error may also occur while reading the `.aux` file from an earlier processing run on the document; if so, delete the `.aux` file and start again from scratch. If the error recurs, it could well be due to a buggy class or package.

However, it may be that none of the above solves the problem.

If so, remember that things that appear before `\documentclass` are also problematical: they are inevitably before `\begin{document}`!

Unfortunately, modern editors are capable of putting things there, and preventing you from seeing them. This can happen when your document is being 'written' in Unicode. The Unicode standard defines "Byte Order Marks" (BOM), that reassure a program (that reads the document) of the way the Unicode codes are laid out. Sadly ordinary LaTeX or PDFLaTeX choke on BOMs, and consider them typesetting requests. The error message you see will look like:

```
! LaTeX Error: Missing \begin{document}.
...
l.1 <?>
       <?><?>\documentclass{article}
```

(Those <?>s are your operating system's representation of an unknown character; on the author's system it's a reverse video '?' sign.)

You can spot the BOM by examining the bytes; for example, the Unix *hexdump* application can help:

```
$ hexdump -C <file>
00000000  ef bb bf 5c 64 6f 63 75 ...
```

The `5c 64 6f 63 75` are the "`\docu`" at the start of (the 'real' part of) your document; the three bytes before it form the BOM.

How to stop your editor from doing this to you depends, of course, on the editor you use; if you are using GNU Emacs, you have to change the encoding from `utf-8-with-signature` to 'plain' `utf-8`; instructions for that are found on the "stack overflow" site

(So far, all instances of this problem that the author has seen have afflicted GNU Emacs users.)

Fortunately XeTeX and LuaTeX know about BOMs and what to do with them, so LaTeX using them is "safe".

436 `\normalsize` not defined

The LaTeX error:

```
The font size command \normalsize is not defined:
there is probably something wrong with the class file.
```

is reporting something pretty fundamental (document base font size not set up). While this *can*, as the message implies, be due to a broken class file, the more common cause is that you have simply forgotten to put a `\documentclass` command in your document.

437 Too many math alphabets

TeX mathematics is one of its most impressive features, yet the internal structure of the mechanism that produces it is painfully complicated and (in some senses) pathetically limited. One area of limitation is that one is only allowed 16 ”maths alphabets”

LaTeX offers the user quite a lot of flexibility with allocating maths alphabets, but few people use the flexibility directly. Nevertheless, there are many packages that provide symbols, or that manipulate them, which allocate themselves one or more maths alphabet.

If you can’t afford to drop any of these packages, there’s still hope if you’re using the *bm* package to support bold maths: *bm* is capable of gobbling alphabets as if there is no tomorrow. The package defines two limiter commands: `\bmmax` (for *bold* symbols; default 4) and `\hmmax` (for *heavy* symbols, if you have them; default 3), which control the number of alphabets to be used.

Any reduction of the `\xxmax` variables will slow *bm* down — but that’s surely better than the document not running at all. So unless you’re using maths fonts (such as *Mathtime Plus*) that feature a heavy symbol weight, suppress all use of heavy families by

```
\newcommand{\hmmax}{0}
```

(before loading *bm*), and then steadily reduce the bold families, starting with

```
\newcommand{\bmmax}{3}
```

(again before loading *bm*), until (with a bit of luck) the error goes away.

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

438 Not in outer par mode

The error:

```
! LaTeX Error: Not in outer par mode.
```

comes when some “main” document feature is shut up somewhere it doesn’t like.

The commonest occurrence is when the user wants a figure somewhere inside a table:

```
\begin{tabular}{|l|}
  \hline
  \begin{figure}
  \includegraphics{foo}
  \end{figure}
  \hline
\end{tabular}
```

a construction that was supposed to put a frame around the diagram, but doesn’t work, any more than:

```
\framebox{\begin{figure}
  \includegraphics{foo}
  \end{figure}%
}
```

The problem is, that the `tabular` environment, and the `\framebox` command restrain the `figure` environment from its natural métier, which is to float around the document.

The solution is simply not to use the `figure` environment here:

```
\begin{tabular}{|l|}
  \hline
  \includegraphics{foo}
  \hline
\end{tabular}
```

What was the float for? — as written in the first two examples, it serves no useful purpose; but perhaps you actually wanted a diagram and its caption framed, in a float.

It's simple to achieve this — just reverse the order of the environments (or of the `figure` environment and the command):

```
\begin{figure}
  \begin{tabular}{|l|}
    \hline
    \includegraphics{foo}
    \caption{A foo}
    \hline
  \end{tabular}
\end{figure}
```

The same goes for `table` environments (or any other sort of float you've defined for yourself) inside tabulars or box commands; you *must* get the float environment out from inside, one way or another.

439 Perhaps a missing `\item`?

Sometimes, the error

```
Something's wrong--perhaps a missing \item
```

actually means what it says:

```
\begin{itemize}
  boo!
\end{itemize}
```

produces the error, and is plainly in need of an `\item` command.

You can also have the error appear when at first sight things are correct:

```
\begin{tabular}{l}
  \begin{enumerate}
  \item foo\\
  \item bar
  \end{enumerate}
\end{tabular}
```

produces the error at the `\\`. This usage is just wrong; if you want to number the cells in a table, you have to do it "by hand":

```
\newcounter{tablecell}
...
\begin{tabular}{l}
  \stepcounter{tablecell}
  \thetablecell. foo\\
  \stepcounter{tablecell}
  \thetablecell. bar
\end{tabular}
```

This is obviously untidy; a command `\numbercell` defined as:

```
\newcounter{tablecell}
...
\newcommand*{\numbercell}{%
  \stepcounter{tablecell}%
  \thetablecell. % **
}
```

could make life easier:

```
\begin{tabular}{l}
  \numbercell foo\\
  \numbercell bar
\end{tabular}
```

Note the deliberate introduction of a space as part of the command, marked with asterisks. Omitted above, the code needs to set the counter `tablecell` to zero (`\setcounter {tablecell}{0}`) before each tabular that uses it.

The error also regularly appears when you would never have thought that a `\item` command might be appropriate. For example, the seemingly innocent:

```
\fbox{%
  \begin{alltt}
    boo!
  \end{alltt}%
}
```

produces the error (the same happens with `\mbox` in place of `\fbox`, or with either of their "big brothers", `\framebox` and `\makebox`). This is because the `alltt` environment uses a "trivial" list, hidden inside its definition. (The `itemize` environment also has this construct inside itself, in fact, so `\begin{itemize}` won't work inside an `\fbox`, either.) The list construct wants to happen between paragraphs, so it makes a new paragraph of its own. Inside the `\fbox` command, that doesn't work, and subsequent macros convince themselves that there's a missing `\item` command.

To solve this rather cryptic error, one must put the `alltt` inside a paragraph-style box. The following modification of the above *does* work:

```
\fbox{%
  \begin{minipage}{0.75\textwidth}
    \begin{alltt}
      hi, there!
    \end{alltt}
  \end{minipage}
}
```

The code above produces a box that's far too wide for the text. One may want to use something that allows variable size boxes in place of the `minipage` environment.

Oddly, although the `verbatim` environment wouldn't work inside a `\fbox` command argument (see verbatim in command arguments), you get an error that complains about `\item`: the environment's internal list bites you before `verbatim` has even had a chance to create its own sort of chaos.

Another (seemingly) obvious use of `\fbox` also falls foul of this error:

```
\fbox{\section{Boxy section}}
```

This is a case where you've simply got to be more subtle; you should either write your own macros to replace the insides of LaTeX's sectioning macros, or look for some alternative in the packages discussed in "The style of section headings".

440 Illegal parameter number in definition

The error message means what it says. In the simple case, you've attempted a definition like:

```
\newcommand{\abc}{joy, oh #1!}
```

or (using TeX primitive definitions):

```
\def\abc{joy, oh #1!}
```

In either of the above, the definition uses an argument, but the programmer did not tell (La)TeX, in advance, that she was going to. The fix is simple — `\newcommand{\abc}[1]`, in the LaTeX case, `\def\abc#1` in the basic TeX case.

The more complicated case is exemplified by the attempted definition:

```
\newcommand{\abc}{joy, oh joy!%
  \newcommand{\ghi}[1]{gloom, oh #1!}%
}
```

will also produce this error, as will its TeX primitive equivalent:

```
\def\abc{joy, oh joy!%
  \def\ghi#1{gloom, oh #1!}%
}
```

This is because special care is needed when defining one macro within the code of another macro. This is explained elsewhere, separately for LaTeX definitions and for TeX primitive definitions

441 Float(s) lost

The error

```
! LaTeX Error: Float(s) lost.
```

seldom occurs, but always seems deeply cryptic when it *does* appear.

The message means what it says: one or more figures, tables, etc., or marginpars has not been typeset. (Marginpars are treated internally as floats, which is how they come to be lumped into this error message.)

The most likely reason is that you placed a float or a `\marginpar` command inside another float or marginpar, or inside a `minipage` environment, a `\parbox` or `\footnote`. Note that the error may be detected a long way from the problematic command(s), so the techniques of tracking down elusive errors all need to be called into play.

This author has also encountered the error when developing macros that used the LaTeX internal float mechanisms. Most people doing that sort of thing are expected to be able to work out their own problems. . .

442 Token not allowed in PDFDocEncoded string

The package *hyperref* produces this error when it doesn't know how to make something into a "character" that will go into one of its PDF entries. For example, the (unlikely) sequence

```
\newcommand{\filled}[2]{%
  #1%
  \hfil
  #2%
}
\section{\filled{foo}{bar}}
```

provokes the error. *Hyperref* goes on to tell you:

```
removing '\hfil' on input line ...
```

It's not surprising: how would *you* put the typesetting instruction `\hfil` into a PDF bookmark?

Hyperref allows you to define an alternative for such things: the command `\texorpdfstring`, which takes two arguments — the first is what is typeset, the second is what is put into the bookmark. For example, what you would probably like in this case is just a single space in the bookmark; if so, the erroneous example above would become:

```
\newcommand{\filled}[2]{%
  #1%
  \texorpdfstring{\hfil}{\space}%
  #2%
}
\section{\filled{foo}{bar}}
```

and with that definition, the example will compile succesfully (*hyperref* knows about the macro `\space`).

443 Checksum mismatch in font

When Metafont generates a font it includes a checksum in the font bitmap file, and in the font metrics file (TFM). (La)TeX includes the checksum from the TFM file in the DVI file.

When *dvips* (or other DVI drivers) process a DVI file, they compare checksums in the DVI file to those in the bitmap fonts being used for character images. If the checksums don't match, it means the font metric file used by (La)TeX was not generated from the same Metafont program that generated the font.

This commonly occurs when you're processing someone else's DVI file.

The fonts on your system may also be at fault: possibilities are that the new TFM was not installed, or installed in a path after an old TFM file, or that you have a personal cache of bitmaps from an old version of the font.

In any case, look at the output – the chances are that it's perfectly OK, since metrics tend not to change, even when the bitmaps are improved. (Indeed, many font designers — Knuth included — maintain the metrics come what may.)

If the output *does* look bad, your only chance is to regenerate things from scratch. Options include: flushing your bitmap cache, rebuild the TFM file locally, and so on.

444 Entering compatibility mode

You run your LaTeX job, and it starts by saying

```
Entering LaTeX 2.09 COMPATIBILITY MODE
```

followed by lines of asterisks and `!!WARNING!!`.

This means that the document is not written in "current" LaTeX syntax, and that there is no guarantee that all parts of the document will be formatted correctly.

If the document is someone else's, and you want no more than a copy to read, ignore the error. The document may fail elsewhere, but as often as not it will provide a `.dvi` or `.pdf` that's adequate for most purposes.

If it's a new document you have just started working on, you have been misled by someone. You have written something like:

```
\documentstyle{article}
```

or, more generally:

```
\documentstyle[options]{class}
```

These forms are (as the warning says) LaTeX 2.09 syntax, and to get rid of the warning, you must change the command.

The simple form is easy to deal with:

```
\documentstyle{article}
```

should become:

```
\documentclass{article}
```

The complex form is more difficult, since LaTeX 2.09 "options" conflate two sorts of things — options for the class (such as `11pt`, `fleqn`), and packages to be loaded. So:

```
\documentstyle[11pt,verbatim]{article}
```

should become:

```
\documentclass[11pt]{article}
\usepackage{verbatim}
```

because `11pt` happens to be a class option, while *verbatim* is a package.

There's no simple way to work out what are class options under LaTeX 2.09; for *article*, the list includes `10pt`, `11pt`, `12pt`, `draft`, `fleqn`, `leqno`, `twocolumn` and `twoside` — anything else must be a package.

Your document may well "just work" after changes like those above; if not, you should think through what you're trying to do, and consult documentation on how to do it — there are lots of free tutorials to help you on your way, if you don't have access to a LaTeX manual of any sort.

445 LaTeX won't include from other directories

You wanted to \include{../bar/xyz.tex}, but LaTeX says:

```
latex: Not writing to ../bar/xyz.aux (openout_any = p).
! I can't write on file '../bar/xyz.aux'.
```

The error comes from TeX's protection against writing to directories that aren't descendents of the one where your document resides. (The restriction protects against problems arising from LaTeXing someone else's malicious, or merely broken, document. If such a document overwrites something you wanted kept, there is obvious potential for havoc.)

Document directory structures that can lead to this problem will look like the fictional mybook:

```
./base/mybook.tex
./preface/Preface.tex
./preface/***
./chapter1/Intro.tex
...
```

With such a structure, any document directory (other than the one where mybook.tex lives), seems "up" the tree from the base directory. (References to such files will look like \include{../preface/Preface}: the ".." is the hint.)

But why did it want to write at all? — "what's going in in my \include" explains how \include works, among other things by writing an .aux file for every \included file.

Solutions to the problem tend to be drastic:

1. Restructure the directories that hold your document so that the master file is at the root of the tree:

   ```
   ./mybook.tex
   ./mybook/preface/Preface.tex
   ./mybook/preface/***
   ./mybook/chapter1/Intro.tex
   ...
   ```

   and so on.
2. Did you actually *need* \include? — if not, you can replace \include by \input throughout. (This only works if you don't need \includeonly.)
3. You *could* patch your system's texmf.cnf — if you know what you're doing, the error message should be enough of a hint; this action is definitely not recommended, and is left to those who can "help themselves" in this respect.

V Current TeX-related projects

446 The LaTeX project

The LaTeX 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. The new system is (provisionally) called LaTeX3; it will remain freely available and it will be fully documented at all levels.

The LaTeX team's first product (LaTeX2e) was delivered in 1994 (it's now properly called "LaTeX", since no other version is current).

LaTeX2e 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 on CTAN; other (published) articles are to be found on the project web site (`http://www.latex-project.org/papers/`).

Some of the project's experimental code is visible on the net:

- via `http://www.latex-project.org/code.html`, which points to the project's SVN repository;
- via the project's *GitHub* mirror;
- from CTAN: snapshots of two major collections from the code, *l3kernel* (supporting LaTeX3 coding conventions in a LaTeX2e environment), *l3packages* (a first cut of a "document designer's interface") as well as *l3experimental* (new stuff that's still under development).

The packages *l3kernel* and *l3packages* provide an "experimental harness" that may be used as a testbed for LaTeX3 work.

Note that *l3kernel* introduces a coding structure quite different from earlier LaTeX code; a few hardy authors, who are not members of the project, are nevertheless using it in their development work.

Anyone may participate in discussions of the future of LaTeX through the mailing list `latex-l`; some development work (outside the project) is discussed on the list. Subscribe to the list by sending a message '`subscribe latex-l <your name>`' to `listserv@urz.Uni-Heidelberg.de`

`l3experimental` *bundle*: `macros/latex/contrib/l3experimental`

`l3kernel` *bundle*: `macros/latex/contrib/l3kernel`

LaTeX project publications: `info/ltx3pub`

`l3packages` *bundle*: `macros/latex/contrib/l3packages`

447 Future WWW technologies and (La)TeX

An earlier answer ("converting to HTML") 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 whose availability designers can count on. Some existing (La)TeX to HTML converters provide maths symbols by hitching them to alternate font face specifications for standard code points in a non-standard way. This does nothing for the universality of the HTML so generated.

Now, however, free Unicode-encoded OpenType fonts, with coverage of mathematical symbols, are starting to appear. The much-heralded *STIX* fonts are now available on CTAN, and a tweaked version (*XITS*) and *Asana Math* are also available. The STIX project has still not released macros for using the fonts, but the *unicode-math* package will do what is necessary under XeTeX and LuaTeX, and the fonts can of course be used in browsers.

XML The core of the range of new standards is XML, which provides a framework for better structured markup; limited support for it has already appeared in some browsers.

Conversion of (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 an XSLT stylesheet or code for an XML library, and then simply using LaTeX; alternatively, one may typeset direct from the XML source.

Direct representation of mathematics MathML is a standard for representing maths on the Web; its original version was distinctly limited, but version 2 of MathML has had major browser support since 2002 with richness of mathematical content for online purposes approaching that of TeX for print. Browser support for MathML is provided by *amaya*, the 'Open Source' browser *mozilla* (and its derivatives including *NetScape*, *Firefox* and *Galeon*) and *Internet Explorer* when equipped with a suitable plug-in such as *MathPlayer*. There's evidence that (La)TeX users are starting to use such browsers. Some believe that XHTML+MathML now provides better online viewing than PDF. Work to produce XHTML+MathML is well advanced in both the TeX4ht and *TtH* projects for (La)TeX conversion.

The *MathJax* engine will process the content of LaTeX `\[ ... \]` and `\( ... \)` 'environments' in an HTML document, to produce mathematical output that may (for example) be cut-and-pasted into other programs.

Incorporation into your document can be as simple as incorporating:

```
<script type="text/javascript"
  src="http://cdn.mathjax.org/mathjax/latest/MathJax.js?config=TeX-AMS_HTML">
</script>
```

into the header of your HTML document, though the MathJax project's site also allows you to download your own copy and install it on one of *your* servers. *MathJax* is open source software, so you could, in principle, extend it to do even more eccentric tasks.

An approach different from (La)TeX conversion is taken by the *GELLMU* Project. Its *article* XML document type, which has a markup vocabulary close to LaTeX that can be edited using LaTeX-like markup (even though it is not LaTeX — so far), comes with translators that make both PDF (via *pdflatex*) and XHTML+MathML. Such an approach avoids the inherent limitations of the "traditional" (La)TeX translation processes, which have traps that can be sprung by unfettered use of (La)TeX markup.

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 wider deployment of MathML).

Browser plug-ins, that deal with SVG are already available (Adobe offer one, for example). More recently, the open source graphics editor *Inkscape* has appeared, and has been reported to be useful for SVG-related work in at least one TeX-related project. Be aware that the developers of *Inkscape* have no illusions about being able to replace commercial software, yet…

Direct use of TeX markup Some time back, IBM developed a browser plug-in called TechExplorer, which would display (La)TeX documents direct in a browser. Over the years, it developed into a MathML browser plug-in, while still retaining its (La)TeX abilities, but it's now distributed (free for Linux and Windows platforms) by Integre Technical Publishing.

The disadvantage of the TechExplorer approach is that it places the onus on the browser user; and however technically proficient *you* are, it's never safe to assume too much of your readers. An interesting alternative is MathTeX, which sits on your server as a CGI script, and you use it to include your TeX, in your HTML, as if it were an image:

```
  <img src="/cgi-bin/mathtex.cgi?f(x)=\int\limits_{-\infty}^xe^{-t^2}dt">
```

(*Mathtex* supersedes the author's earlier *mimetex*.)

Asana Math fonts: `fonts/Asana-Math`

GELLMU: `support/gellmu`

MathTeX: `support/mathtex`

MimeTeX: `support/mimetex`

STIX fonts: `fonts/stix`

tex4ht: `obsolete/support/TeX4ht/tex4ht-all.zip` (but see `http://tug.org/tex4ht/`)

unicode-math.sty: macros/latex/contrib/unicode-math

XITS fonts: fonts/xits

448 Making outline fonts from Metafont

TeXtrace, originally developed by Péter Szabó, is a bundle of Unix scripts that use Martin Weber's freeware boundary tracing package *autotrace* 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's always the *FontForge* font editor to tidy things. Whatever, there remain fonts which many people find useful and which fail to attract the paid experts, and auto-tracing is providing a useful service here. Notable sets of fonts generated using *TeXtrace* are Péter Szabó's own EC/TC font set *tt2001* and Vladimir Volovich's CM-Super set, which covers the EC, TC, and the Cyrillic LH font sets (for details of both of which sets, see "8-bit" type 1 fonts).

Another system, which arrived slightly later, is *mftrace*: this is a small *Python* program that does the same job. *Mftrace* may use either *autotrace* (like *TeXtrace*) or Peter Selinger's *potrace* to produce the initial outlines to process. *Mftrace* is said to be more flexible, and easier to use, than is *TeXtrace*, but both systems are increasingly being used to provide Type 1 fonts to the public domain.

The *MetaType1* system aims to use Metafont font sources, by way of Metapost and a bunch of scripts and so on, to produce high-quality Type 1 fonts. The first results, the *Latin Modern* fonts, are now well-established, and a bunch of existing designs have been reworked in MetaType1 format.

Mf2pt1 is another translator of Metafont font sources by way of Metapost; in addition, available, *mf2pt1* will use *fontforge* (if it's available) to auto-hint the result of its conversion. (*Mf2pt1* is also written in *perl*.)

MetaType1: fonts/utilities/metatype1

mf2pt1: support/mf2pt1

449 The TeX document preparation environment

"Why TeX is not WYSIWYG" outlines the reasons (or excuses) for the huge disparity of user interface between "typical" TeX environments and commercial word processors.

Nowadays, at last, there is a range of tools available that try either to bridge or to close the gap. One range modestly focuses on providing the user with a legible source document. At the other extreme we have *TeXmacs*, a document processor using TeX's algorithms and fonts for both editor display and printing. *TeXmacs* does not use the TeX language itself (though among other formats, LaTeX may be exported and imported). A bit closer to LaTeX is LyX, which has its own editor display and file formats as well, but does its print output by exporting to LaTeX. The editor display merely resembles the printed output, but you have the possibility of entering arbitrary LaTeX code. If you use constructs that LyX does not understand, it will just display them as source text marked red, but will properly export them.

Since a lot of work is needed to create an editor from scratch that actually is good at editing (as well as catering for TeX), it is perhaps no accident that several approaches have been implemented using the extensible *emacs* editor. The low end of the prettifying range is occupied by syntax highlighting: marking TeX tokens, comments and other stuff with special colours. Many free editors (including *emacs*) can cater for TeX in this way. Under Windows, one of the more popular editors with such support is the Shareware product *winedt*. Continuing the range of tools prettifying your input, we have the *emacs* package *x-symbol*, which does the WYSIWYG part of its work by replacing single TeX tokens and accented letter sequences with appropriate-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* which is best used with a previewer by the same author.

Another *emacs* package called *preview-latex* tries to solve the problem of visual correlation between source and previews in a more direct way: it uses a LaTeX package to chop the document source into interesting fragments (like figures, text or display math) which it runs through LaTeX and replaces the source text of those fragments with the corresponding rendered output images. Since it does not know about the structure of the images, at the actual cursor position the source text is displayed while editing rather than the preview. This approach is more or less a hybrid of the source prettifying and fast preview approaches since it works in the source buffer but uses actual previews rendered by LaTeX.

A more ambitious contender is called 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 correspondence of the display to the final output, and the balance they strike between visual aid and visual distraction.

preview-latex: Distributed as part of `support/auctex`

texmacs: Browse `support/TeXmacs`

450 Omega and Aleph

Omega (Ω) was developed as an extension of TeX, to use with multilingual texts, expressed in a variety of input encodings. Omega uses 16-bit, Unicode-encoded, characters. It provides many innovative concepts, notably including the "translation process" that takes a character stream and transforms it according to various processes that may be internally specified, or be a separate program.

While Omega showed a lot of promise at its mid-1990s announcement, progress was slow, and development was essentially dead by the time that one of the original developers withdrew (taking with him a bunch of research students).

Before that distressing event, a separate thread of development had started, to produce a program called Aleph (ℵ), which merged the facilities of e-TeX into a stable Omega codebase and added other extensions. Aleph also proved an attractive platform for many people; but its development, too, has dried up.

A presentation at EuroTeX 2006 claimed that development of Omega was picking up again, in parallel with research into what the (new) developers consider a rational scheme for supporting TeX-style typesetting. The new system was to be known as Omega-2 (Ω_2), and was to be designed in a modular fashion so that support of new facilities (such as use of advanced OpenType fonts) could be added in a relatively straightforward fashion. A quick web search leads to a recommendation that potential users consider LuaTeX instead. Fortunately, lessons learned in Aleph project are being carried forward in the LuaTeX project.

451 The XeTeX project

XeTeX, by Jonathan Kew, is a successor to the shareware TeX/GX program for Macintoshes. It is a Unicode-based TeX implementation which is able to make direct use of the fonts installed on a system. It supports the advanced typographical features available in modern font technology (OpenType, AAT on Mac OS X, Graphite). XeTeX was originally developed for Mac OS X but it is now fully integrated in TeX Live on a wide range of platforms, as well as being available as part of MiKTeX. XeTeX supports Unicode character sets and bidirectional typesetting (via e-TeX primitives). It is popular among linguists and scholars in the humanities, as well as an increasing range of people working in other fields. Support for mathematical Unicode fonts (such as Asana Math and Cambria) is progressing.

The project has an active mailing list; a collection of other information is to be found in a set of links on the TUG web site.

XeTeX reference: `info/xetexref`

452 PDFTeX and LuaTeX

As is said elsewhere in these FAQs, development of PDFTeX is "in essence" complete — no new facilities are being developed at the time of writing. The PDFTeX team has announced that they have frozen PDFTeX's specification in its current state (version 1.40.11), and that nothing but bug corrections will be provided up to the time of the final release, PDFTeX 1.50.0. (The interpretation of the statement seems to allow sensible changes that are beyond any reasonable definition of *bug*. . .)

In parallel with the running-down of PDFTeX development, development of a new system, LuaTeX is under way. *Lua* is a script language, designed to offer an interpreter that may be incorporated into other applications. LuaTeX consists of a TeX-like engine with a *lua* interpreter 'embedded' in it; the *lua* interpreter has access to many of the data structures used for typesetting, and the user may also interpolate chunks of *lua* code into their (La)TeX macros, or as 'call-backs' for use when the TeX-like engine does certain operations.

This arrangement offers the prospect of a "semi-soft" typesetting engine: it will have its basic behaviour, but the user gets to redefine functionality if an idea occurs — there will be no need to persuade the world first, and then find a willing developer to work on the sources of of the distribution.

The LuaTeX project is (with monetary support from various sources) pursuing avenues that many of the other current projects have in their sights, notably Unicode character representations and support for OpenType fonts. The intention is to integrate the extensions pioneered by Aleph. Users may also care to view the LuaTeX documentation page

The current released version (0.70.1, in June 2011) of LuaTeX is supposed at least to demonstrate the final functionality. This stability was declared with version 0.50.0, released near the end of December 2009. Much work remains to be done, and this remains a beta-release. TeX Live 2011 incorporates (at the time of writing) LuaTeX version 0.70.1, and current MiKTeX (version 2.9) offers version 0.70.0.

ConTeXt 'Mark 4' can already make use of LuaTeX; much of its code already appears in two forms — a TeX-based version (`.mkii`) and a '`.mkiv`' version (new functionality typically *only* appears in '`.mkiv`' form), which uses LuaTeX extensions (including *lua* scripting). LaTeX packages that support its use are appearing (some of them providing re-implementations of existing ConTeXt code).

LuaTeX snapshot: `systems/luatex`

PDFTeX distribution: `systems/pdftex`

453 The ANT typesetting system

Achim Blumensath's ANT project aims not to replicate TeX with a different implementation technique, but rather to provide a replacement for TeX which uses TeX-like typesetting algorithms in a very different programming environment. ANT remains under development, but it is now approaching the status of a usable typesetting system.

ANT's markup language is immediately recognisable to the (La)TeX user, but the scheme of implementing design in ANT's own implementation language (presently *OCaml*) comes as a pleasant surprise to the jaded FAQ writer. This architecture holds the promise of a system that avoids a set of serious problems with TeX's user interface: those that derive from the design language being the same as the markup language.

ANT: systems/ant

454 The ExTeX project

The ExTeX project is building on the experience of the many existing TeX development and extension projects, to develop a new TeX-like system. The system is to be developed in Java (like the ill-fated NTS project).

While ExTeX will implement all of TeX's primitives, some may be marked as obsolete, and "modern" alternatives provided (an obvious example is the primitive `\input` command, whose syntax inevitably makes life difficult for users of modern operating system file paths). Desirable extensions from e-TeX, PDFTeX and Ω have been identified.

Usability will be another focus of the work: debugging support and log filtering mechanisms will please those who have long struggled with TeX macros.

ExTeX will accept Unicode input, from the start. In the longer term, drawing primitives are to be considered.

455 Replacing the BibTeX–LaTeX mechanism

Producing a successor to BibTeX has long been a favoured activity among a certain class of TeX-users; the author has seen reports of progress (on several projects), over the years, but few that claim to be ready for "real-world" use.

Few would deny that BibTeX is ripe for renewal: as originally conceived, it was a program for creating bibliographies for technical documents, in English. People have contributed mechanisms for a degree of multilingual use (whose techniques are arcane, and quite likely inextensible), while an extension (*bibtex8*) allows use with 8-bit character codes, thus providing some multilingual capabilities. In addition, specialist BibTeX style files are available for use in non-technical papers.

BibTeX uses a style language whose mechanisms are unfamiliar to most current programmers: it's difficult to learn, but since there are few opportunities to write the language, it's also difficult to become fluent (in the way that so many people fluently write the equally arcane TeX macro language).

Oren Patashnik (the author of BibTeX) summarises the issues as he sees them, in a TUG conference paper from 2003 that seems to suggest that we might expect a BibTeX 1.0 . . . which hasn't (yet) appeared.

In the absence of BibTeX 1.0, what do we need from the bibliography system of the future? — simple: a superset of what BibTeX does (or can be made to do), preferably implementing a simpler style language, and with coherent multilingual capabilities.

There are two parts to a bibliography system; processing the database of citations, and typesetting the results. The existing BibTeX system provides a means of processing the database, and there are macros built into LaTeX, as well as many LaTeX packages, that process the results.

Of the direct BibTeX replacements, only two have been submitted to CTAN: CrossTeX and *biber*.

CrossTeX's language feels familiar to the existing user of BibTeX, but it's redesigned in an object-oriented style, and looks (to a non-user) as if it may well be adequately flexible. It is said to operate as a BibTeX replacement.

CrossTeX's team respond to queries, and seem well aware of the need for multilingual support, though it isn't currently offered.

Biber is intimately associated with the LaTeX package *biblatex*; it is logically a BibTeX replacement, but is also capable of using bibliography databases in its own *biblatexml* (XML-based) format. *Biblatex* can also use BibTeX, but *biber* opens up a far wider range of possibilities, including full Unicode support.

Biblatex is a processor for the output of an application such as *biber* or BibTeX; the style of citations and of the bibliography itself (in your document) is determined by the way your *biblatex* style has been set up, not on some BibTeX-LaTeX package combination. *Biblatex*'s structure thus eliminates the collections of BibTeX styles, at a stroke; it comes with a basic set of styles, and details are determined by options, set at package loading time. The author, Philipp Lehman, evaluated the whole field of bibliography software before starting, and as a result the package provides answers to many of the questions asked in the bibliography sections of these FAQs.

Biblatex was released as experimental software, but it's clear that many users are already using it happily; Lehman is responsive to problem reports, at the moment, but a *de facto* set of expert users is already establishing itself. A set of contributed styles has appeared, which cover some of the trickier bibliography styles. The road map of the project shows that we are now working on the final *beta* releases before the "stable" *biblatex* 1.0.

Finally, *Amsrefs* uses a transformed `.bib` file, which is expressed as LaTeX macros. (The package provides a BibTeX style that performs the transformation, so that a LaTeX source containing a `\nocite{*}` command enables BibTeX to produce a usable *amsrefs* bibliography database.)

Amsrefs is maintained by the AMS as part of its author support programme,

amsrefs.sty: `macros/latex/contrib/amsrefs`

biber: `biblio/biber`

biblatex.sty: `macros/latex/contrib/biblatex`

bibtex8: `biblio/bibtex/8-bit`

biblatex contributions: `macros/latex/contrib/biblatex-contrib`

CrossTeX: `biblio/crosstex`

W You're still stuck?

456 You don't understand the answer

While the FAQ maintainers don't offer a 'help' service, they're very keen that you understand the answers they've already written. They're (almost) written "in a vacuum", to provide something to cover a set of questions that have arisen; it's always possible that they're written in a way that a novice won't understand them.

Which is where you can help the community. Mail the maintainers to report the answer that you find unclear, and (if you can) suggest what we need to clarify. 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.

Note that the FAQ development email address is not for answering questions: it's for you to suggest which questions we should work on, or new questions we should answer in future editions.

Those who simply ask questions at that address will be referred to `texhax@tug.org` or to `comp.text.tex`.

457 Submitting new material for the FAQ

The FAQ will never be complete, and we always expect that there will be people out there who know better than we do about something or other. We always need to be put right about whatever we've got wrong, and suggestions for improvements, particularly covering areas we've missed, are always needed: mail anything you have to the maintainers

If you have actual material to submit, your contribution is more than ever welcome. Submission in plain text is entirely acceptable, but if you're really willing, you may feel free to mark up your submission in the form needed for the FAQ itself. The markup is a strongly-constrained version of LaTeX — the constraints come from the need to translate the marked-up text to HTML on the fly (and hence pretty efficiently). There is a file `markup-syntax` in the FAQ distribution that describes the structure of the markup, but there's no real substitute for reading at least some of the source (`faqbody.tex`) of the FAQ itself. If you understand *Perl*, you may also care to look at the translation code in `texfaq2file` and `sanitize.pl` in the distribution: this isn't the code actually used on the Web site, but it's a close relation and is kept up to date for development purposes.

FAQ distribution: help/uk-tex-faq

458 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 infrequently, 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@ams.org`), or contact TUG.

If you've found a bug in LaTeX2e, report it using mechanisms supplied by the LaTeX team. (Please be careful to ensure you've got a LaTeX bug, or a bug in one of the "required" packages distributed by the LaTeX team.)

If you've found a bug in a contributed LaTeX package, the best starting place is usually to ask in the "usual places for help on-line, or just possibly one of the specialised mailing lists. The author of the package may well answer on-line, but if no-one offers any help, you may stand a chance if you mail the author (presuming that you can find an address...).

If you've found a bug in LaTeX 2.09, or some other such unsupported software, your only real hope is help on-line.

Failing all else, you may need to pay for help — TUG maintains a register of TeX consultants. (This of course requires that you have the resources — and a pressing enough need — to hire someone.)

459 Reporting a LaTeX bug

The LaTeX team supports LaTeX, and will deal with *bona fide* bug reports. Note that the LaTeX team does *not* deal with contributed packages — just the software that is part of the LaTeX distribution itself: LaTeX and the "required" packages. Furthermore, 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 may reject your report.

2. Has your bug already been reported? Browse the LaTeX bugs database, to find any earlier instance of your bug. In many cases, the database will list a work-around.

3. Prepare a "minimum" file that exhibits the problem. Ideally, such a file should contain no contributed packages — the LaTeX team as a whole takes no responsibility for such packages (if they're supported at all, they're supported by their authors). The "minimum" file should be self-sufficient: if a member of the team should run it in a clean directory, on a system with no contributed packages, it should replicate your problem.

4. Run your file through LaTeX: the bug system needs the `.log` file that this process creates.

You now have two possible ways to proceed: either create a mail report to send to the bug processing mechanism (5, below), or submit your bug report via the web (7, below).

5. Process the bug-report creation file, using LaTeX itself:

```
latex latexbug
```

`latexbug` asks you some questions, and then lets you describe the bug you've found. It produces an output file `latexbug.msg`, which includes the details you've supplied, your "minimum" example file, and the log file you got after running the example. (I always need to edit the result before submitting it: typing text into `latexbug` isn't much fun.)

6. Mail the resulting file to `latex-bugs@latex-project.org`; the subject line of your email should be the same as the bug title you gave to `latexbug`. The file `latexbug.msg` should be included into your message in-line: attachments are likely to be rejected by the bug processor.
7. Connect to the latex bugs processing web page and enter details of your bug — category, summary and full description, and the two important files (source and log file); note that members of the LaTeX team *need* your name and email address, as they may need to discuss the bug with you, or to advise you of a work-around.